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

PROYECTO DE GRADO INGENIERÍA EN COMPUTACIÓN

Simulación numérica de flujo para medios granulares aplicando el método de los elementos discretos y computación de alto desempeño

Miguel Da Silva
mdasilva@fing.edu.uy

Setiembre de 2018

Supervisor del proyecto:
Sergio Nesmachnow

Simulación numérica de flujo para medios granulares aplicando el método de los elementos discretos y computación de alto desempeño

Da Silva, Miguel

Proyecto de grado de Ingeniería en Computación

Instituto de Computación - Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, julio de 2018

Simulación numérica de flujo para medios granulares aplicando el método de los elementos discretos y computación de alto desepeño

Resumen

El Método de los Elementos Discretos (*Discrete Element Method*, DEM) es una de las herramientas más utilizadas en aplicaciones que requieren estudiar el comportamiento dinámico de un sistema de partículas. Las industrias minera, agropecuaria y farmacéutica, el área geotécnica y, recientemente, la astrofísica, son ejemplos de campos de aplicación de DEM. Un conjunto muy amplio de propiedades físico-químicas de las partículas y del medio en que están inmersas pueden ser estudiadas a partir de la simulación de la dinámica de las partículas. Un caso de estudio de interés en el ámbito nacional es el análisis de las presiones ejercidas por partículas granulares sobre la superficie interior de un silo de almacenamiento cuando hay movimiento relativo. Múltiples estudios se han dedicado a la investigación de la correlación entre las presiones medidas y la formación de un canal de flujo de partículas en la etapa de vaciado del silo. Para lograr resultados en simulaciones de escala industrial, que involucran un alto número de partículas ($\sim 10^6 - 10^7$) es necesario disponer de un alto poder de cómputo. Estos requerimientos exigen que el sistema informático que implementa DEM trabaje eficientemente en infraestructuras de cómputo distribuido. En esta línea de trabajo, este proyecto de grado propone una implementación de un algoritmo de descomposición dinámica de dominio para el sistema de simulación LIGGGHTS (*LAMMPS Improved for General Granular and Granular Heat Transfer Simulations*) para la simulación de flujo de medios granulares. La implementación permite lograr un mejor aprovechamiento de los recursos de cómputo a través de distribuciones sucesivas de la carga de trabajo entre los recursos ociosos. Para validar la eficiente computacional se consideran dos problemas de validación (*benchmark*) distribuidos junto con el código fuente de LIGGGHTS y un problema más complejo de dinámica granular. Los resultados numéricos obtenidos muestran que el implementado reduce los tiempos de ejecución en hasta un 42 % y un tiene speedup relativo de hasta aproximadamente 12X, sin afectar la calidad de los resultados numéricos.

Palabras clave: Simulación numérica, DEM, medios granulares, computación paralela, computación de alto desempeño, MPI

Índice general

Índice de figuras

Índice de tablas

Lista de algoritmos

Índice de listados

Capítulo 1

Introducción

Los medios granulares (también conocidos como *materiales granulares*) son una familia muy extensa de cuerpos de diferentes forma, dimensión y composición que se encuentran espontáneamente en la naturaleza. Algunos ejemplos de medios granulares son los aglomerados rocosos, los granos vegetales, la arena, los minerales y los cuerpos celestes (por ejemplo, los asteroides y los anillos de Saturno). Más allá de esta gran diversidad, los medios granulares comparten características físico-químicas comunes y frecuentemente son considerados el cuarto estado de la materia [1].

La aplicación de los medios granulares en la industria es muy amplia y son el segundo tipo de material más utilizado en los procesos industriales, después del água [2]. Algunos ejemplos de equipamiento industrial que operan con medios granulares son los reactores químicos de lecho fluidizado, las cintas transportadoras y los silos de almacenamiento. El uso industrial de equipamiento especializado en el manejo de medios granulares exige conocer el comportamiento del material granular durante la operación. Mindlin y Deresiewicz [3] en la década de 1950 difinieron las bases teóricas que permitieron estudiar el comportamiento físico de los aglomerados de material granular durante su manipulación. A fines de la década de 1970, Cundall y Strack [4] propusieron un método computacional que permitió utilizar eficientemente simulaciones en computador para estudiar la dinámica de los medios granulares. Este método es conocido como Método de los Elementos Discretos (DEM).

El principal objetivo de DEM era sustituir los experimentos de laboratorio por simulaciones en computador. Las principales desventajas de los experimentos de laboratorio se relacionaban con la necesidad de construir modelos a escala reducida del sistema real estudiado y ofrecían resultados que tenían imprecisiones numéricas, debido a las escalas utilizadas [5]. Existían antecedentes previos de propuestas de métodos computacionales cuyo objetivo era sustituir experimentos en laboratorio por simulaciones en computador [6, 7]. Sin embargo, estos métodos computacionales eran ineficientes. La propuesta de Cundall y Strack logró plantear un método computacional eficiente al considerar solamente las características físico-químicas necesarias para reproducir con alto grado de precisión el comportamiento de los medios granulares observado en laboratorio.

Desde la aparición de la propuesta original de Cundall y Strack, el método DEM ha sido aplicado en áreas muy diversas de la ingeniería y de las ciencias básicas [8, 9, 10, 11]. Muchos investigadores han realizado aportes importantes a los trabajos de Cundall y Strack, y actualmente se encuentran en pleno desarrollo diferentes líneas de investigación que proponen mejoras a la propuesta original de DEM [12, 13, 14, 15, 16, 17].

El método DEM simula el comportamiento a lo largo del tiempo de cada partícula de un sistema de medios granulares mediante la segunda ley de Newton. Esta característica hace que las simulaciones correspondientes a sistemas que contienen un número elevado de partículas se transformen en tareas de cómputo intensivo. Desde finales de la década de 1980, diferentes trabajos fueron publicados por investigadores que lograron ejecutar eficientemente los algoritmos de DEM en computadores con múltiples procesadores o, más recientemente, múltiples núcleos de procesamiento [18, 10, 19, 12, 20, 21, 22, 23]. Las investigaciones en el área de nuevos algoritmos paralelos para DEM es muy activa y se encuentran en la literatura trabajos que hacen aportes muy importantes al área [24, 25, 10, 12, 26, 20, 11, 21, 27].

La consolidación de DEM en el medio académico como una herramienta válida para la simulación de sistemas de medios granulares impulsó el desarrollo de paquetes de software que ofrecen un amplio conjunto de herramientas para la investigación de dinámica de medios granulares. Estos paquetes de software son desarrollados por centros de investigación o empresas privadas y se convertieron en piezas fundamentales en la investigación científica y en la producción industrial. EDEM [28], LIGGGHTS [22] y MFIX [29] son algunos ejemplos de estos paquetes de software.

Los desarrolladores de los sistemas de simulación numérica para DEM suelen incluir varias funcionalidades que permiten ejecutar una simulación utilizando múltiples unidades de procesamiento. Estas funcionalidades son fundamentales para mejorar el desempeño de un sistema de simulación numérica y pueden ser implementadas a través de diferentes estrategias y algoritmos. Una de las principales estrategias utilizadas para la mejora de desempeño es la capacidad de redistribuir la carga de trabajo entre mútiples unidades de procesamiento a lo largo de una simulación. Se encuentran en la literatura varios algoritmos para la distribución dinámica de carga de trabajo [27, 30, 31].

Habitualmente, los paquetes de software para simulación numérica que implementan el método DEM están disponibles de dos maneras: (1) mediante algún medio de acceso pago o (2) a través de alguna licencia de distribución gratuita de software. LIGGGHTS es un ejemplo de sistema de simulación numérica distribuido a través de ambas maneras. LIGGGHTS es desarrollado por la empresa DCS Computing (Linz, Austria) y según la forma de acceso elegida, el usuario tiene disponible un conjunto más completo de funcionalidades.

La variante LIGGGHTS-PUBLIC de LIGGGHTS es de acceso gratuito y es utilizada por los investigadores del Instituto de Mecánica de los Fluidos e Ingeniería Ambiental (IMFIA) de la Facultad de Ingeniería para ejecutar simulaciones de descarga de medios granulares almacenados en silos. Las simulaciones permiten a los investigadores estudiar la distribución de las presiones ejercidas sobre las paredes de un silo durante el procedimiento de descarga. Esta variante no cuenta con un mecanismo de distribución dinámica de carga de trabajo. La distribución de carga de trabajo es estática y ocurre únicamente al principio de una simulación. Esta característica hace que el desempeño de LIGGGHTS se reduzca, especialmente para simulaciones de gran porte. Las variantes de LIGGGHTS que ofrecen alguna estrategia de distribución dinámica de cargas de trabajo solamente se acceden mediante algún medio de pago.

A partir de un relevamiento de la literatura respecto de estrategias de distribución dinámica de cargas de trabajo adecuadas para DEM [19, 22, 30, 31], este proyecto de grado propuso incorporar a LIGGGHTS-PUBLIC una implementación de un algoritmo de carácter geométrico para la distribución dinámica de cargas de trabajo. El algoritmo

implementado está basado en la estrategia de planos móviles propuesta por Markauskas et al. [30]. Los detalles de la implementación, las pruebas de validación realizadas y los resultados obtenidos se encuentran detalladamente reportados en este informe.

Los resultados que fueron obtenidos mostraron que el algoritmo implementado es capaz de reducir satisfactoriamente los tiempos de ejecución de las simulaciones de dinámica granular. Este algoritmo pudo manejar adecuadamente sistemas de medios granulares con partículas de igual radio y partículas con diferentes valores de radio. Los resultados del algoritmo se vieron beneficiados según aumentón el número de procesos que ejecutaron una simulación.

El resto de este documento está organizado como se describe a continuación. El capítulo 2 describe la aplicación de métodos computacionales al estudio de la dinámica de medios granulares y muestra como las técnicas de computación de alto desempeño pueden ser aplicadas para mejorar su desempeño computacional. Además, el capítulo 2 introduce el problema estudiado en este proyecto de grado. El capítulo 3 introduce el método DEM y presenta sus principales características. El capítulo 4 describe el sistema de simulación numérica LIGGGHTS y introduce sus principaes características y funcionalidades. El capítulo 5 contiene un resumen de trabajos relacionados, relevantes en el área de dinámica granular y simulación numérica. El capítulo 6 describe la solución propuesta en este proyecto de grado para mejorar el desempeño del sistema de simulación numérica LIGGGHTS. El capítulo 7 reporta y analiza los resultados obtenidos en la etapa de validación de la solución propuesta. Finalmente, el capítulo 8 plantea las principales conclusiones de este proyecto de grado y las líneas de trabajo futuro.

Capítulo 2

Métodos computacionales aplicados al estudio de dinámica de medios granulares

Este capítulo presenta una descripción del problema abordado en el proyecto de grado y las principales consideraciones sobre la aplicación de métodos computacionales para su resolución. La sección 2.1 introduce los medios granulares y describe las principales líneas de investigación en dinámica de medios granulares vinculadas con la temática del proyecto. La sección 2.2 introduce el estudio de la correlación entre el patrón de flujo observado y la distribución de presiones en las paredes del silo, y muestra cómo las simulaciones en computador pueden ser utilizadas para el estudio de los patrones de flujo. La sección 2.3 presenta de manera sucinta las principales características de la computación de alto desempeño como técnica de simulación eficiente de dinámica de medios granulares. Finalmente, la sección 2.4 describe cómo pueden ser aplicadas las técnicas de computación de alto desempeño para mejorar el desempeño de simulaciones de medios granulares.

2.1. Medios granulares y simulaciones de silos

Los medios granulares son definidos como una amplia familia de granos de diferentes formas, tamaños y composiciones que comparten características comunes y presentan un comportamiento físico similar. A nivel granular, cada grano o partícula tiene propiedades físico-químicas propias, pero a nivel macroscópico un aglomerado de granos se comporta como un sólido o un fluido [32].

En la naturaleza los medios granulares ocurren espontáneamente. Fenómenos geológicos tales como los deslizamientos de tierra, las avalanchas y la erosión son ejemplos de la presencia de material granular en la naturaleza. En la industria, la utilización de materiales en forma de medios granulares es muy amplia y frecuentemente se considera como el cuarto estado de la materia, pues pueden presentar propiedades de materia en estado líquido, sólido y gaseoso simultáneamente. Después de los líquidos, los medios granulares son el segundo tipo de material más utilizado en los procesos industriales [1, 2]. Algunos ejemplos de medios granulares en la industria son los granos de origen vegetal (arroz, trigo, maíz, etc.), los minerales obtenidos en actividades de minería (oro, cobre, mineral de hierro, etc.) y las burbujas de gas encontradas en un reactor de lecho fluidizado.

Debido a su uso en la industria, el almacenamiento y el tratamiento adecuado de los medios granulares a lo largo de diferentes procesos industriales presentan varios desafíos a la comunidad científica y al personal responsable de la operación del equipamiento en plantas industriales. Es muy común encontrar en fábricas silos de almacenamiento o estructuras similares cuyas dimensiones, forma y modo de operacíon son muy diferentes. La figura 2.1 presenta algunos ejemplos de silos de almacenamiento encontrados normalmente en la industria.

(a) Silo de almacenamiento para granos

(b) Silo de almacenamiento para combustibles

(c) Silo de almacenamiento para minerales

(d) Silo de almacenamiento para granos

Figura 2.1: Ejemplos de silos de almacenamiento - imágemes de uso libre
Fuente: Wikipedia [33]

Cuando el material granular debe ser almacenado en un silo es necesario conocer su comportamiento durante la etapa de carga y descarga. No conocer cómo interactúan entre sí las partículas almacenadas y cómo éstas interactúan con la estructura de almacenamiento, puede llevar a la ocurrencia de efectos no deseados durante la operación del silo. Como consecuencia, la integridad física del silo y del personal que lo opera pueden estar comprometidas. Conocer el comportamiento de las partículas en el interior del silo es fundamental para determinar adecuamente las características estructurales del silo. Las decisiones a tomar estarán influenciadas por las propiedades físico-químicas del material almacenado.

La comunidad científica ha dedicado mucho esfuerzo al estudio del comportamiento de los medios granulares en el interior de un silo de almacenamiento durante las etapas de carga y descarga. Sin embargo, debido a características típicas de los medios granulares y de los silos de almacenamiento resulta muy difícil realizar experimentos a escala real. Por ejemplo, para un silo de dimensiones similares a las encontradas en la industria

y considerando la cantidad de material granular que es manejado, es muy común la generación de polvo que puede impedir el funcionamiento correcto de equipamiento de medición durante las etapas de carga y descarga [34, 35, 36, 37].

Generalmente se utilizan modelos a escala reducida para la experimentación y se emplean componentes que facilitan la realización de los experimentos. Este enfoque es muy difundido, pero tiene una gran desventaja: los componentes utilizados y las técnicas aplicadas para estudiar modelos a escala reducida no se encuentran con frecuencia en la industria. Por ejemplo, los modelos de silos a escala reducida son construidos con materiales transparentes para visualizar facilmente el material almacenado y frecuentemente las paredes son perforadas para la instalación de equipamiento de medición. No se encuentra en la industria silos con paredes transparentes o fabricantes de silos que permitan la perforación de las paredes para instalación de equipamiento de medición [34, 36].

Una línea de investigación en dinámica de medios granulares con aplicabilidad directa en la construcción y operación de silos es el estudio del flujo de medios granulares en la etapa de descarga. Una temática importante que se estudia en esta línea es la relación entre el canal de flujo generado durante la descarga y las presiones ejercidas en las paredes del silo cuando se utiliza un orificio de salida excéntrico (el orificio de salida se encuentra cercano al borde exterior del silo, alejado del centro).

Investigaciones han determinado que el patrón de flujo dentro del silo está relacionado con la distribución de presiones en la etapa de descarga cuando se utiliza un orificio de salida concéntrico (en este caso, el orificio de salida se encuentra en el centro de la circuferencia determinada por la estructura del silo). Asimismo, se ha verificado que los resultados obtenidos en modelos a escala reducida no ofrecen buenas aproximaciones de los resultados esperados en modelos a escara real [5, 38]. Esta última observación proporciona especial interés a experimentos que son realizados a escala real.

Debido a las dificultades presentadas anteriormente, la comunidad científica ha recurrido a simulaciones en computador para realizar los experimentos para modelos de silo a larga escala. El material encontrado en la literatura es muy vasto respecto de la validez de utilización de técnicas de simulación y de su amplia aplicabilidad en la investigación de dinámica de medios granulares [4, 24, 25, 37, 39].

2.2. Patrones de flujo de medios granulares, distribución de presiones en las paredes de un silo y simulaciones numéricas de medios granulares

El estudio de patrones de flujo de medios granulares dentro de un silo tiene como objetivo determinar el comportamiento de las partículas durante las etapas de carga y descarga. En estas etapas, las partículas se mueven en el interior del silo y ocurre la interacción entre partículas y entre las partículas y las paredes del silo. Como se explicó en la sección 2.1, el silo puede ser operado de modo que las partículas son descargadas a través de orificios de salida localizados en diferentes posiciones.

Cuando se usa un silo con orificio de descarga concéntrico, se observan frecuentemente dos patrones de flujo: (a) flujo de masa (*mass flow*) y (b) flujo afunilado (*funnel flow*). En el caso de flujo de masa se observa el movimiento simultáneo de todas las partículas almacenadas en el silo. En el caso de flujo afunilado se observa la formación de una región compuesta por partículas que no se mueven—en la literatura se hace referencia a esa

región como zona de estancamiento (*stagnant zone* o *stagnant region*)—y las partículas restantes se mueven formando un *canal de flujo*.

Es posible construir el silo de modo que el patrón de flujo pueda ser determinado previamente. La elección de un patrón determinado se toma en base a los requerimientos existentes. Por ejemplo, es común la elección de diseños de silos que favorecen la formación de un patrón de flujo de masa cuando es necesario evitar la segregación del material almacenado. En este caso, el movimiento simultáneo de las partículas hace que éstas se mezclen durante la descarga. En la literatura se encuentran referencias que presentan estudios detallados de dinámica de medios granulares para patrones de flujo de masa y flujo afunilado [40, 41, 42].

En el caso de descarga a través de un orificio excéntrico se observa la formación de un patrón de flujo afunilado contiguo a la pared más cercana al orificio de salida (flujo afunilado excéntrico, *eccentric funnel flow*). La figura 2.2 presenta un diagrama del comportamiento de las partículas según los patrones de flujo de masa, flujo afunilado, flujo afunilado excéntrico. Este proyecto de grado se concentra en el estudio de las interacciones que ocurren cuando el movimiento de las partículas es debido a la descarga del silo a través de un orificio de salida excéntrico.

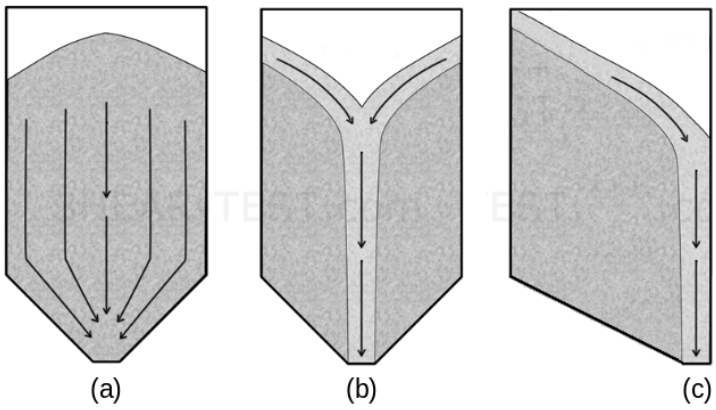


Figura 2.2: Diagrama de patrones de flujo; (a) flujo de masa, (b) flujo afunilado y (c) flujo afunilado excéntrico

Para experimentos a escala reducida es común construir un modelo del silo utilizando componentes que facilitan la realización de los experimentos como se ha ejemplificado en la sección 2.1. Una vez que el silo contiene la cantidad de partículas deseadas, el orificio de salida es abierto y el material granular almacenado entra en movimiento.

Los experimentos de descarga de silos para el estudio de patranos de flujo de medios granulares a escala real son raros en la literatura y la instrumentación necesaria para realizar un experimento a esa escala es muy compleja [34, 35, 36]. Esta dificultad hace que las mediciones realizadas, cuyos resultados pueden ser encontrados en la literatura, generalmente no incluyan los valores para la distribución de presiones y su correlación con el patrón de flujo simultáneamente. En la bibliografia consultada se han encontrado muy pocos trabajos que presentan resultados completos de experimentos con silos a escala real [36, 43, 44].

Independientemente de la escala utilizada, los resultados de las mediciones son utilizados para el post-procesamiento de los datos y las variables de interés cuyos valores son medidos dependen de la metodología aplicada. Muchos investigadores suelen usar un conjunto mixto de técnicas para la etapa de post-procesamiento, por ejemplo, equipamiento

de rayos X, grabaciones de video o equipamiento de medición especializado (sensores de presión, temperatura, radiotransmisores, etc.) [36].

Es importante conocer los patrones de flujo durante la etapa de descarga dado que el comportamiento de las partículas influye directamente en la integridad física del silo [36, 43, 44]. Especialmente, en el caso de flujo afunilado excéntrico se observa movimiento relativo entre las partículas y las paredes del silo en una porción reducida de la estructura. Como consecuencia, diferentes secciones de las paredes del silo están sujetas a diferentes valores de presión y esta diferencia de presiones puede hacer que la estructura colapse [36, 44]. Los accidentes que ocurren durante la operación de un silo de almacenamiento pueden resultar fatales [45].

Independientemente de la escala elegida, el uso de simulaciones en computador ha demostrado ser una técnica válida que permite sustituir un experimento de laboratorio. Las dificultades encontradas en la instrumentación de experimentos a escala real en dinámica de medios granulares ha contribuido en transformar a las simulaciones computacionales en una herramienta valiosa en la investigación. Técnicas de Dinámica de Fluidos Computacional (*Computational fluid dynamics*, CFD) y metódos computacionales tales como DEM y el Método de los Elementos Finitos (*Finite element method*, FEM) han demostrado su robustez y confiabilidad a lo largo de las últimas décadas [10, 18, 19, 24, 25, 30, 37, 46, 47, 48, 49, 50].

Los técnicas de CFD y los métodos DEM y FEM son ampliamente difundidos y utilizados en la comunidad científica y el aumento constante de la capacidad de cómputo disponible en las plataformas de computación científica favorece su uso. Los métodos computaciones disponibles no suelen ser de carácter general y dado el problema que se quiere resolver, es necesario elegir un método adecuado. Para la dinámica de medios granulares el método más ampliamente utilizado es DEM [4] y es posible encontrar en la literatura un gran número de trabajos que comprueban la eficacia del método [4, 30, 37, 39]. En este proyecto de grado se utiliza el método DEM y se presenta sus bases en el capítulo 3.

El método DEM utiliza métodos computacionales de resolución de ecuaciones diferenciales que tienen grandes requisitos de cómputo cuando los problemas a resolver involucran un número considerable de partículas. Tales problemas son habituales cuando se quiere resolver problemas a escala real. Es necesario contar con sistemas computacionales que permitan resolver eficientemente problemas con estas características. Actualmente es muy común que universidades y empresas ofrezcan a sus miembros acceso a infraestructura computacional especializada para la ejecución eficiente de simulaciones de sistemas de dinámica granular. Es posible mejorar el desempeño de estos sistemas utilizando técnicas computacionales especiales. En la sección 2.3 se describe algunas de las técnicas computacionales utilizadas habitualmente en el contexto de dinámica de medios granulares.

2.3. Computación de alto desempeño aplicada al estudio del comportamiento de medios granulares en simulaciones de silos

La computación de alto desempeño (*High-Performance Computing*, HPC) es la disciplina que estudia el uso de técnicas avanzadas de computación para el abordaje de problemas de cómputo intensivo y de uso masivo de datos. Usualmente, los problemas

considerados en HPC no pueden ser resueltos dentro de un tiempo razonable por un computador de prestaciones reducidas debido a que muchas operaciones deben ser ejecutadas o una gran cantidad de datos deben ser procesados. Determinados problemas no pueden ser resueltos por los sistemas computacionales habituales debido a las limitaciones de los recursos disponibles (por ejemplo, problemas que involucran un gran número de elementos exigen gran cantidad de memoria RAM). La computación de alto desempeño busca superar estas limitaciones utilizando equipamiento computacional especializado o agregando recursos computacionales capaces de trabajar simultáneamente y considerando técnicas especiales de construcción de software para implentar la colaboración entre múltiples procesos de ejecución simultánea.

Los sistemas de software utilizados en HPC son implementados utilizandos los paradigmas de programación paralela y computación distribuida. Existen varias características comunes entre ambos paradigmas, y muchas veces no quedan claras las diferencias entre sus definiciones [51, 52]. Se debe observar que no son paradigmas disjuntos y es posible utilizar un enfoque que combina ambos paradigmas.

El paradigma de programación paralela permite implementar un programa de computador contemplando la ejecución simultánea de múltiples procesos que trabajan conjuntamente y coordinadamente en la resolución de un problema. El nivel de paralelismo que puede ser alcanzado depende de las características del problema y de los algoritmos usados para su resolución. Los sistemas programados para ejecutar en forma paralela son conocidos como sistemas paralelos.

El paradigma de computación distribuida permite utilizar múltiples recursos para la resolución de una instancia de un problema. Los recursos se interconectan a través de una red de comunicaciones y es posible que se encuentren en localizaciones físicas diferentes. Los sistemas programados para ejecutar en plataformas de computación distribuida son conocidos como sistemas distribuidos.

Para dar soporte a los paradigmas de programación paralela y computación distribuida, los recursos de hardware utilizados en sistemas computacionales para HPC son, habitualmente, organizados en: (1) computadores con memoria compartida y (2) agrupamientos (*clusters*) de computadores con memoria distribuida. Los computadores de memoria compartida permiten que los procesos accedan a espacios de memoria de uso común y que los utilicen para sincronizar su procesamiento. Los clusters de computadores están compuestos por múltiples unidades de procesamiento (nodos de cómputo o procesamiento) equipados con memoria de acceso exclusivo y que se comunican a través de una red LAN (Local Area Network). Es habitual encontrar infraestructuras híbridas formadas por nodos de cómputos con memoria compartida e interconectados a través de una red de comunicaciones. Ambas plataformas pueden estar compuestas por computadores especializados o por equipamiento de prestaciones reducidas de una misma institución (universidades, centros de investigaciones, etc.). Generalmente, para acceder a la infraestructura se encuentra disponible un gestor de recursos. Un ejemplo de plataforma híbrida para HPC que se utilizó en este proyecto de grado para la ejeución de simulaciones numéricas es el Cluster FING de la Facultad de Ingeniería de la Universidad de la República [53].

Es posible extender el nivel de agregación de los recursos de hardware disponibles y construir plataformas más complejas para dar soporte a HPC. Los sistemas de computación grid (*grid computing*) y de computación en la nube (*cloud computing*) son ejemplos de sistemas computacionales con alto nivel de heterogeneidad y control descen-

tralizado que pueden ser utilizados para construir plataformas para HPC. En un grid, múltiples instituciones contribuyen con diferentes tipos de recursos para la creación de un sistema que permite utilizarlos de forma coordinada. Nodos de cómputo, sistemas de almacenamiento de datos y aplicaciones de software son algunos ejemplos de los recursos que se encuentran en un grid. Usualmente, los recursos están conectados a través de una red WAN (Wide Area Network) y el usuario accede a ellos de forma transparente, sin importar su localización geográfica. Es posible encontrar varias plataformas grid activas que dan soporte a diferentes aplicaciones de HPC [54, 55].

Los sistemas de computación en la nube ofrecen acceso a recursos computacionales a través de un paradigma en el cuál los recursos son provistos bajo demanda. Los sistemas de computación en la nube se hicieron ampliamente conocidos a partir del lanzamiento del servicio Amazon Elastic Compute Cloud (EC2) [56] de Amazon en 2006. Generalmente, el usuario accede a los recursos según niveles de servicio y de acuerdo a un modelo de tarifas que permite pagar solamente por los recursos utilizado y por el tiempo durante el cuál fueron utilizados. La tabla 2.1 muestra los tres niveles de servicio ofrecidos habitualmente en sistemas de computación en la nube. Los sistemas de computación en la nube son utilizados mayoritariamente por instituciones y compañías que prefieren no construir y mantener infraestructuras computacionales propias. Es posible encontra en la literatura ejemplos de uso de sistemas de computación en la nube para HPC [57, 58, 59].

Nivel de servicio	**Descripcíon**
Software como servicio (*Software as a service*, SaaS)	Los recursos son las aplicaciones para usuario final y el acceso al recurso permite el uso de la aplicación
Plataforma como servicio (*Platform as a service*, PaaS)	Los recursos son los elementos intermedios de una infraestructura computacional (sistemas operativos, gestores de bases de datos, servidores de aplicaciones, etc.)
Infraestructura como servicio (*Infrastructure as a service*, IaaS)	Todos los elementos que permiten construir una infraestructura computacional pueden ser accedidos como recursos (servidores, equipamiento de red, sistemas de almacenamiento masivo de archivos, etc.)

Tabla 2.1: Niveles de servicio encontrados habitualmente en sistemas de computación en la nube

Los paradigmas mencionados previamente son utilizados a través de interfaces de programación (*Application Programming Interface*, API) que definen directivas y comandos que permiten sincronizar procesos mediante el intercambio de mensajes o áreas de memoria compartida. La API OpenMP [60] es un ejemplo de interface utilizada para la sincronización y la ejecución simultánea de procesos mediante memoria compartida. Para que un programador pueda utilizar un determinado lenguaje de programación para la implementación de sistemas distribuidos o paralelos usando OpenMP, los desarrolladores del lenguaje de programación deben implementar las directivas de esta API como parte del lenguaje de programación. OpenMP implementa un modelo de paralelismo de hilos de ejecución similar a las instrucciones fork-join de Unix. Las directivas de OpenMP son conocidas como pragmas y son utilizadas conjuntamente con las instrucciones del lenguaje de programación para determinar fragmentos de código que son ejecutados concurrentemente por diferentes hilos de ejecución.

La API MPI (*Message Passing Interface*) [61] es un ejemplo de interface utilizada para la sincronización y la ejecución simultánea de procesos a través del intercambio de mensajes. En el caso de MPI se tiene la definición de la sintaxis y de la semántica

de las operaciones que deben estar contenidas en las implementaciones de la API. MPI identifica los procesos en ejecución a través de un número entero j tal que $0 \leq j \leq E-1$, siendo E la cantidad de procesos usados en la ejecución paralela. En el ámbito de MPI, ese número intero es conocido como rango (*rank*) del proceso y un proceso de rango j es, usualmente, referenciado simplemente como el proceso j. No existen dos procesos cuyos rangos coinciden dentro de un mismo contexto de comunicación. El programador de sistemas distribuidos o paralelos elige una implementación en particular (generalmente disponibles como bibliotecas de software) y agrega a los sistemas las llamadas a los métodos que necesita utilizar. Dos ejemplos de implementaciones de la API MPI son OpenMPI [62] y MPICH [63]. En este proyecto de grado se utilizó un sistema de simulación numérica implementado utilizando MPI.

Investigaciones han demostrado que es posible obtener mejoras de desempeño en el estudio de dinámica molecular utilizando exclusivamente memoria compartida [12, 26] o intercambio de mensajes [11, 20]. En algunas situaciones, los investigadores utilizaron un enfoque híbrido y también obtuvieron resultados satisfactorios [19, 21]. En este proyecto de grado las técnicas de HPC son aplicadas para permitir la ejecución eficiente de un software de simulación numérica para dinámica de medios granulares. Las simulaciones ejecutadas requieren gran capacidad de cómputo para resolver en un tiempo razonable múltiples sistemas de ecuaciones diferenciales ordinarias. En particular, se utiliza la técnica de descomposición espacial de dominio [27] para permitir que varios procesos resuelvan simultáneamente múltiples sistemas de ecuaciones diferenciales. Existen dos enfoques para la técnica de descomposición espacial de dominio: (a) dinámica y (b) estática. La variante de LIGGGHTS utilizada por los investigadores del IMFIA (LIGGGHTS-PUBLIC), que es objeto de estudio para este proyecto de grado, tiene sorpote únicamente para descomposición estática de dominio. Otras variantes de LIGGGHTS ofrecen soporte a descomposición de dominio dinámica.

En el contexto de dinámica granular, una técnica o estrategia de descomposición espacial de dominio consiste en dividir el dominio del problema en subdominios segú criterios geométricos y asociar cada subdominio a un proceso responsable por la resolución de las ecuaciones físico-matemáticas que modelan el comportamiento de las partículas contenidas en el subdominio. Además, en este contexto, para la resolución de estas ecuaciones es necesario conocer la vecindad de cada partícula contenida en el sistema y para ello los procesos deben comunicarse entre sí para coordinar sus ejecuciones. Existe en la literatura material que muestra la aplicación de ambos enfoques de descomposición espacial de dominio para la implementación de versiones paralelas de DEM [10, 18, 19, 25, 30].

La descomposición del dominio ocurre al principio de la simulación y puede permanecer constante a lo largo de toda la simulación (se tiene entonces una descomposición estática de dominio) o puede sufrir cambios a lo largo del tiempo (se tiene entonces una descomposición dinámica de dominio). El software de simulación numérica debe definir criterios para la descomposición inicial del dominio y para las descomposiciones siguientes.

A las estrategias de descomposición de dominio está vinculado el concepto de carga de trabajo de un proceso. A través de este concepto se busca cuantificar la cantidad de trabajo que un proceso debe realizar para procesar los datos que tiene asignado. Existen en la literatura diferentes formas de definir la carga de trabajo de un proceso y la definición ocurre en base a las propiedades de una estrategia determinada de descomposición de dominio [19, 27, 30, 31, 64].

2.4. Mejoras de desempeño de simulaciones numéricas a través de técnicas de computación de alto desempeño

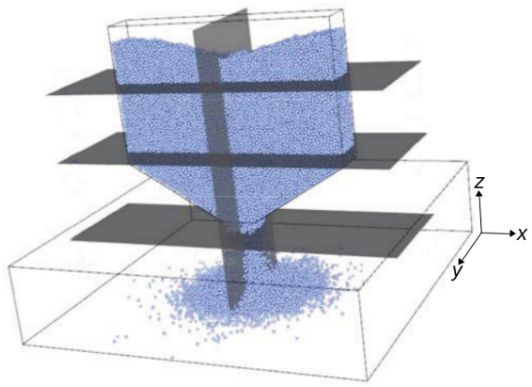
Este proyecto de grado está enfocado en el desarrollo e implementación de una estrategia de descomposición de dominio y balanceo de carga que permita mejorar el desempeño del método DEM implementado en el paquete de software de simulación numérica LIGGGHTS [22]. El método DEM se explica en el capítulo 3 y el paquete LIGGGHTS se describe en detalle en el capítulo 4. La sección 2.3 presentó los conceptos de descomposición de dominio y balance de carga en el contexto de dinámica de medios granulares. LIGGGHTS implementa una versión paralela de DEM y los problemas que resuelve involucran sistemas de medios granulares que interactúan entre si y también con el entorno en el cuál están inmersos. En esta clase de problemas es fundamental conocer el comportamieto dinámico de las partículas a lo largo del tiempo. LIGGGHTS es utilizado por investigadores del Instituto de Mecánica de los Fluidos e Ingeniería Ambiental (IMFIA) de la Facultad de Ingeniería (Universidad de la República) para la ejecución de simulaciones para el estudio de dinámica de medios granulares.

El desempeño de un sistema de software para simulación en computador puede ser mejorado de diferentes formas. Es posible obtener mejoras a partir de cambios de carácter computacional (por ejemplo, cambiando el paradigma de programación utilizado) o a través de cambios en las bases teóricas del problema (por ejemplo, sustituyendo las ecuaciones de un modelo físico-matemático). Este proyecto de grado propone abordar el problema con un enfoque puramente computacional que busca obtener mejoras de desempeño a través de modificaciones en determinados algoritmos ya implementados en LIGGGHTS.

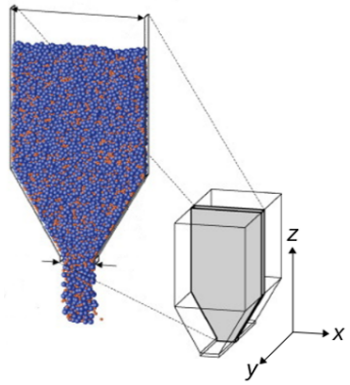
DEM es un método que favorece el desarrollo de sistemas de simulación numérica con alto nivel de paralelismo. Sin embargo, la propuesta inicial de Cundall y Strack [4] no contempla los aspectos computacionales de una posible implementación de los algoritmos usados en el método. Es posible encontrar en la literatura muchas referencias a trabajos de investigación que comprueban la factibilidad de implementar los algoritmos de DEM aplicando el paradigma de programación paralela [10, 18, 19, 24, 25]. Una de los enfoques habitualmente aplicados está relacionado a la forma en que el dominio espacial del problema es tratado [19, 24].

LIGGGHTS-PUBLIC utiliza una estrategia de descomposición estático de dominio basado en argumentos puramente geométricos al momento de crear la descomposición inicial: el espacio que delimita el problema es dividido en subregiones tales que todas tengan el mismo volumen. Para crear las subregiones, el sistema particiona los intervalos correspondientes a cada eje de coordenadas de modo que el total de subregiones debe ser igual al producto del número de subintervalos (particiones) creados en cada eje de coordenadas. La figura 2.3 presenta algunos ejemplos de descomposición de dominio generados por LIGGGHTS-PUBLIC utilizando diferentes números de particiones de los ejes de coordenadas.

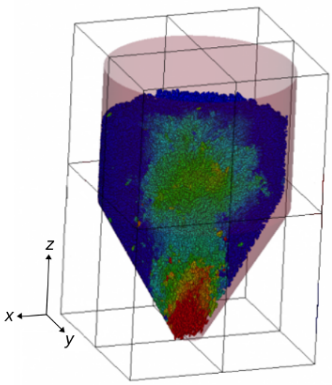
El enfoque de descomposición estática de dominio es adecuado para las simulaciones que involucran materia densa y homogénea [19]. Sin embargo, en los problemas resueltos con el método DEM es muy común realizar simulaciones en presencia de materia no homogénea y observar cambios importantes en la distribución de las partículas, que a lo largo de una simulacón pueden cambiar de subregión con mucha frecuencia [10, 25, 30]. Estos fenómenos pueden llevar a la ocurrencia de desbalanceo de carga entre los procesos,



(a) Descomposición de dominio a lo largo de los ejes x y z



(b) Descomposición de dominio a lo largo del eje y



(c) Descomposición de dominio a lo largo de los ejes x, y y z

Figura 2.3: Ejemplos de descomposición de dominio aplicados en LIGGGHTS
Fuente: DCS Computing - [65]

ocasionando que uno o más procesos deban procesar una cantidad de datos relativamente superior a la cantidad de datos asignada a los demás procesos. El desbalanceo de carga resulta, en general, en reducciones significativas del desempeño computacional del sistema [19].

Los beneficios del uso de estrategias de descomposición dinámica de dominio para simulaciones de medios granulares han sido ampliamente reconocidos en la literatura [19, 30, 31, 66, 67]. Por lo tanto, es posible concluir que la implementación de un algoritmo de descomposición dinámica de dominio para LIGGGHTS-PUBLIC potencia una herramienta de investigación de interés para los investigadores del IMFIA y del público académico en general, y permite la ejecución eficiente de simulaciones computacionales que contienen un gran número de partículas. En el capítulo 6 se describe detalladamente la solución propuesta en este proyecto de grado para la incorporación de una estrategia de descomposición dinámica de dominio en LIGGGHTS-PUBLIC y en el capítulo 7 se presenta la evaluación experimental y los resultados obtenidos sobre casos de interés.

Capítulo 3

El Método de los Elementos Discretos

El método DEM fue desarrollado por Cundall y Strack en el final de los años 1970 [4] y desde esos años ha capturado la atención de la comunidad científica dedicada al estudio de sistemas complejos de partículas. DEM permite simular el movimiento e interacción de partículas considerando un conjunto básico de propiedades de cada elemento (velocidad, aceleración, masa, posición en el espacio, etc.) sin que sea necesario simplificar demasiado la forma en que el sistema es modelado. Durante la última década DEM ha sido utilizado en una gran, y cada vez más amplia, variedad de aplicaciones industriales. En la actualidad, el crecimiento constante de la capacidad de cómputo disponible hace factible la ejecución de simulaciones de sistemas muy complejos y con gran número de partículas [39].

Este capítulo presenta una breve descripción de DEM e introduce los conceptos teóricos básicos utilizados por Cundall y Strack. La sección 3.1 introduce las ecuaciones matemáticas que rigen el desplazamiento de las partículas. La sección 3.2 presenta los modelos físico-matemáticos utilizados habitualmente para representar las partículas y para simular la interacción entre partículas. La sección 3.3 muestra la importancia fundamental del avance del tiempo en DEM y cómo el método simula la progresión temporal. Finalmente, la sección 3.4 presenta una técnica de programación usada habitualmente en los sistemas de dinámica granular (*conjuntos de vecinos*) para evitar la ejecución de operaciones innecesarias.

3.1. Ecuaciones de movimiento

El objetivo de DEM es servir de soporte para la investigación en el área de dinámica de medios granulares. A través de simulaciones realizadas en computadora se estudia el comportamiento de las partículas y la progresión de sus propiedades físico-químicas a lo largo del tiempo. Para una simulación DEM es fundamental conocer las fuerzas que actúan sobre las partículas contenidas en el sistema. Las principales fuerzas a considerar en las simulaciones son: (a) fuerzas que las partículas ejercen unas sobre otras, (b) fuerzas que las fronteras del sistema o superficies ejercen sobre las partículas, o (c) fuerzas externas que actúan sobre el sistema (por ejemplo, fuerza gravitacional o campos magnéticos). La sección 3.2 explica cómo se calculan las fuerzas resultantes en la propuesta original de DEM [4].

Conociendo la fuerza total aplicada sobre cada partícula del sistema y conociendo las propiedades físicas generales del sistema y las propiedades físicas específicas de cada partícula es posible concluir que el problema subyacente al método DEM es la resolución numérica de las ecuaciones de movimiento de Newton para la rotación y traslación de cada partícula del sistema (segunda ley de Newton). La forma matemática de expresar estas ecuaciones se presenta en 3.1 y 3.2, donde, para una partícula p_i, $m^{(i)}$ es su masa, $\mathbf{X}^{(i)}$ es su posición, $\mathbf{F}^{(i)}$ es la fuerza resultante aplicada sobre la partícula, $\mathbf{I}^{(i)}$ es su momento de inercia, $\boldsymbol{\omega}^{(i)}$ es su velocidad angular, $\mathbf{T}^{(i)}$ es el torque total y $\boldsymbol{g}$ representa a la fuerza de gravedad.

$$m^{(i)} \frac{\mathrm{d}^2}{\mathrm{dt}^2} \mathbf{X}^{(i)} = \mathbf{F}^{(i)} + m^{(i)} \boldsymbol{g} \tag{3.1}$$

$$\mathbf{I}^{(i)} \frac{\mathrm{d}}{\mathrm{dt}} \boldsymbol{\omega}^{(i)} = \mathbf{T}^{(i)} \tag{3.2}$$

Para cada partícula se formula un sistema de ecuaciones diferenciales ordinarias (EDO) compuesto por $\mathcal{D} + \dfrac{\mathcal{D}(\mathcal{D}-1)}{2}$ ecuaciones, siendo $\mathcal{D} \in \{2, 3\}$ la dimensión del sistema. Para cada partícula en la simulación es necesario resolver un sistema EDO y el número de partículas hace que la cantidad de sistemas a resolver crezca linealmente. Por lo tanto, se puede concluir que es necesario alta capacidad de cómputo para resolver en tiempos razonables las ecuaciones de movimiento de todas las partículas del sistema.

La propuesta de Cundall y Strack [4] considera el movimiento de las partículas cómo un fenómeno de propagación a través del medio en que están inmersas. Se trata de un proceso de propagación dinámico con perturbaciones que se originan en las fronteras del sistema. La manera en que ocurre la propagación depende de las propiedades físicas del medio.

Para describir numéricamente el comportamiento dinámico del proceso de propagación el tiempo es discretizado en unidades llamadas *pasos de tiempo* (*time-steps*). Cundall y Strack [4] definen dos propiedades fundamentales de DEM respecto de los pasos de tiempo: (1) la duración de un paso de tiempo debe ser lo suficientemente pequeña de modo que durante el tiempo transcurrido entre los pasos de tiempo t y $t+1$ las perturbaciones no se propaguen desde una partícula hacia otras partículas con las cuáles no tiene contacto y (2) durante un paso de tiempo las velocidades y aceleraciones se asumen constantes. Estas propiedades tienen un impacto directo en la forma en que se calculan las fuerzas resultantes. La sección 3.2 presenta cómo se llevan a cabo los cálculos de las fuerzas resultates. La sección 3.3 presenta detalladamente cómo ocurre la progresión del tiempo en DEM y la forma en que un tratamiento inadecuado del tiempo influencia en el método de resolución de los sistemas EDO y cómo este tratamiento del tiempo puede comprometer los resultados.

3.2. Modelos físico-matemáticos para DEM

El comportamiento observado en un experimento no puede ser reproducido con exactitud por una simulación en computadora. Sin embargo, los resultados obtenidos a partir de una simulación deben ser consistentes con el comportamiento observado en el experimento. La naturaleza física de los fenómenos que ocurren en un experimento es modelada a través de ecuaciones matemáticas que tratan de capturar sus principales propiedades. Es posible modelar un fenómeno físico utilizando diferentes modelos

matemáticos y contemplar, según la complejidad de cada modelo, un conjunto mayor o menor de propiedades.

Esta sección presenta una breve descripción del modelo físico-matemático propuesto por Cundall y Strack para DEM [4]. A los efectos de facilitar la presentación del modelo de Cundall y Strack, se divide su descripción en dos partes: la sección 3.2.1 presenta el modelo utilizado para modelar las partículas y la sección 3.2.2 presenta el modelo físico-matemático para la interacción de elementos de la simulacón. La separación previa se realiza únicamente con motivos de exponer claramente los conceptos; el modelo debe ser considerado como una herramienta compuesta por una única componente. Por último, se destaca que los modelos descritos en esta sección son herramientas que trabajan conjuntamente con las ecuaciones de movimiento presentadas en la sección 3.1. La sección 3.3 explica cómo se instrumenta el trabajo en conjunto.

3.2.1. Modelo de partícula de esfera blanda (*soft-sphere*)

Sean p_i y p_j dos partículas tales que r_i y r_j son, respectivamente, sus radios. Se dice que p_i y p_j están en contacto si, y solo si, la distancia D entre los centros de las partículas es menor que la suma de los radios. La condición de contacto se expresa mediante la ecuación 3.3.

$$D < r_i + r_j \tag{3.3}$$

Cundall y Strack proponen un modelo para partículas usando esferas blandas (*soft-sphere*) que considera insignificantes las deformaciones individuales de las partículas respecto de las deformaciones del aglomerado de partículas. Ese supuesto permite obtener buenas aproximaciones del comportamiento mecánico de las partículas sin la necesidad de modelar con exactitud la deformación de cada partícula [4].

En el modelo de esfera blanda la deformación sufrida por una partícula p_i en el punto de contacto con la partícula p_j es sustituida por el *solapamiento* (δ) entre p_i y p_j. En la propuesta original de DEM [4] los autores usan el término *desplazamiento* (*displacement*) en vez de solapamiento (*overlap*). Es importante observar que el valor de δ es pequeño respecto del diámetro de las partículas. Las fuerzas de contacto ejercidas mutuamente por las partículas son calculadas en función de δ y de las velocidades relativas según la descripción presentada en la sección 3.2.2.

Se sabe que cuando dos cuerpos coliden, parte de la energía cinética asociada a sus movimientos es disipada (por ejemplo, se transforma en calor). En el caso de DEM, los cambios de algunas propiedades físico-químicas de las partículas pueden no ser considerados. Los cambios de temperatura, por ejemplo, se consideran nulos en caso que el calor generado por las deformaciones es relativamente pequeño comparado con la energía térmica total del sistema. Las propiedades físico-químicas que son consideradas en una simulación dependen del problema estudiado y del modelo utilizado para representar los elementos contenidos en la simulación y la interacción entre ellos.

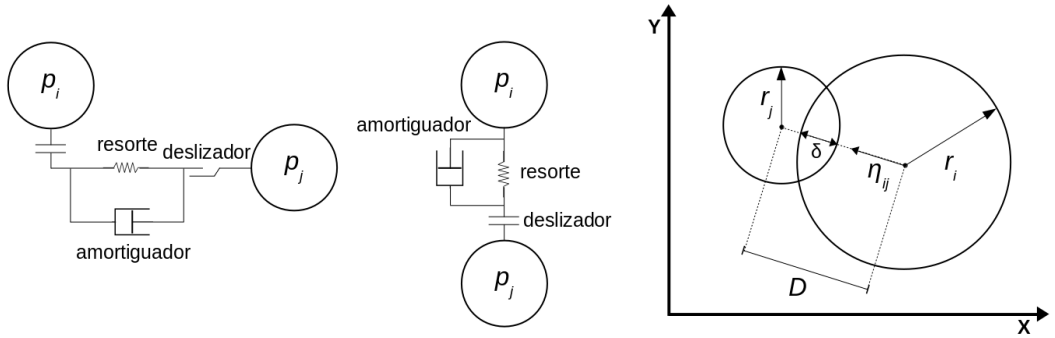
3.2.2. Modelo de contacto *linear spring-dashpot*

En una simulación de medios granulares, uno de los fenómenos más importantes es la interacción entre los elementos que componen el sistema. El comportamiento general del sistema es el resultado de la interacción entre los elementos que lo componen. En el caso del método DEM es necesario modelar la interacción entre partículas y entre partículas

y superficies. Las interacciones son modeladas a través de un *modelo de contacto*. El modelo de contacto es una pieza fundamental de DEM y a través de su aplicación se calculan las fuerzas resultantes actuando sobre las partículas del sistema.

Cundall y Strack [4] plantean el método DEM basándose en un modelo de contacto conocido como sistema resorte-amortiguador lineal (*linear spring-dashpot*, LSD). Este es el modelo que se utiliza en este proyecto de grado. Es posible encontrar en la literatura ejemplos de modelos de contacto más complejos que incorporan nuevos conceptos a DEM, incluyendo: (1) un modelo de contacto visco-elástico que se comporta adecuamente a cambios de temperatura del material granular [8], (2) un modelo de contacto para la adherencia de elementos discretos en fenómenos de erosión [13], (3) un modelo de contacto especializado en rocas frágiles o porosas (*bittle rocks*) [14] y (4) un modelo elasto-plástico para aprimoramiento de un principio conocido en la literatura como Ley de Fuerza-desplazamiento (*Force-displacement law*) [68]. También se encuentran en la literatura estudios comparativos de eficiencia entre diferentes modelos de contacto [15]. El campo de investigación de nuevos modelos físico-matemáticos es muy activo en el área de dinámica de medios granulares [13, 16]. Generalmente, los sistemas de simulación DEM permiten incorporar nuevos modelos sin exigir demasiado esfuerzo.

La idea fundamental del modelo LSD es asociar dos resortes imaginarios al solapamiento entre las partículas, tales que los resortes se suponen localizados en el punto de contacto entre las partículas y de modo que un resorte tenga orientación normal y el otro tenga orientación tangencial. El movimiento relativo de las partículas a lo largo de las direcciones normal y tangencial queda relacionado al movimiento armónico de los resortes. La figura 3.1 presenta un esquema del modelo LSD y algunos elementos geométricos que aparecen en el modelo (a los efectos de facilitar la descripción del modelo se incluyen en la figura las variables δ, $\boldsymbol{\eta}_{ij}$ y D).



(a) Esquema del modelo *linear spring-dashpot* (b) Solapamiento (δ) entre partículas

Figura 3.1: Modelo LSD y solapamiento entre partículas

El cálculo de las fuerzas de contacto entre partículas depende de δ y de las velocidades relativas entre las partículas. ean p_i y p_j dos partículas tales que $\mathbf{X}^{(i)}$ y $\mathbf{X}^{(j)}$ son, respectivamente, las coordenadas de sus centros, y tales que $\mathbf{V}^{(i)}$ y $\boldsymbol{\omega}^{(i)}$, y $\mathbf{V}^{(j)}$ y $\boldsymbol{\omega}^{(j)}$ son, respectivamente, sus velocidades lineal y angular. La velocidad relativa entre p_i y p_j está dada por la expresión de la ecuación 3.4, donde $\mathbf{R}^{(i)}$ y $\mathbf{R}^{(j)}$ son, respectivamente, vectores desde el centro de p_i y p_j hacia el punto de contacto.

$$\mathbf{V}_{ij} = \mathbf{V}^{(j)} - \mathbf{V}^{(i)} + \boldsymbol{\omega}^{(j)} \times \mathbf{R}^{(j)} - \boldsymbol{\omega}^{(i)} \times \mathbf{R}^{(i)} \tag{3.4}$$

La velocidad relativa $\mathbf{V}_{ij}$ se descompone en una componente normal ($\mathbf{V}_{n,ij}$) y una componente tangencial ($\mathbf{V}_{t,ij}$) a través de las ecuaciones 3.5 y 3.6, siendo $\boldsymbol{\eta}_{ij} = \frac{(\mathbf{X}^{(j)} - \mathbf{X}^{(i)})}{|\mathbf{X}^{(j)} - \mathbf{X}^{(i)}|}$ un vector unitario normal al punto de contacto y cuya dirección es desde el centro de p_i hacia el centro de pj. La figura 3.1 muestra la ocurrencia del vector $\boldsymbol{\eta}_{ij}$ en la definición del modelo LSD.

$$\mathbf{V}_{n,ij} = (\mathbf{V}_{ij} \cdot \boldsymbol{\eta}_{ij})\boldsymbol{\eta}_{ij} \tag{3.5}$$

$$\mathbf{V}_{t,ij} = \mathbf{V}_{ij} - \mathbf{V}_{n,ij} \tag{3.6}$$

Para calcular la fuerza de contacto resultante ($\mathbf{F}^{(i)}$) y el torque total ($\mathbf{T}^{(i)}$) sobre p_i es necesario considerar todas las partículas p_j que interactúan con p_i. Un principio básico de DEM es que a lo largo de un paso de tiempo la interacciones sólo ocurren entre las para partículas que efectivamente están en contacto (ver sección 3.1).

La fuerza ejercida por una partícula p_j sobre una partícula p_i puede ser descompuesta en una componente normal ($\mathbf{F}_{n,ij}$) y una componente tangencial ($\mathbf{F}_{t,ij}$). A partir de esa descomposición se definen las expresiones 3.7 y 3.8:

$$\mathbf{F}^{(i)} = \sum_{j=1, j\neq i}^{N} (\mathbf{F}_{n,ij} + \mathbf{F}_{t,ij}) \tag{3.7}$$

$$\mathbf{T}^{(i)} = \sum_{j=1, j\neq i}^{N} \left(\mathbf{R}^{(i)} \times \mathbf{F}_{t,ij}\right) \tag{3.8}$$

Dada la ecuación 3.7, la fuerza de contacto resultante se descompone en una componente conservativa $\left(\mathbf{F}^{C}\right)$ y una componente disipativa $\left(\mathbf{F}^{D}\right)$. A su vez, $\mathbf{F}^{C}$ y $\mathbf{F}^{D}$ son descompuestas en componentes normal y tangencial. La descomposición permite expresar $\mathbf{F}_{n,ij}$ y $\mathbf{F}_{t,ij}$ en sus componentes, de acuerdo a las ecuaciones 3.9 y 3.10:

$$\mathbf{F}_{n,ij} = \mathbf{F}^{C}_{n,ij} + \mathbf{F}^{D}_{n,ij} \tag{3.9}$$

$$\mathbf{F}_{t,ij} = \mathbf{F}^{C}_{t,ij} + \mathbf{F}^{D}_{t,ij} \tag{3.10}$$

Las componentes normal y tangencial de $\mathbf{F}^{C}$ son calculadas usando las expresiones de las ecuaciones 3.11 y 3.12, donde k_n es el coeficiente de rigidez del resorte imaginario de orientación normal, k_t es el coeficiente de rigidez del resorte imaginario de orientación tangencial, δ_n es el solapamiento entre las partículas en la dirección normal y δ_t es el vector de desplazamiento tangencial entre las partículas:

$$\mathbf{F}^{C}_{n,ij} = -k_n \delta_n \mathbf{n}_{ij} \tag{3.11}$$

$$\mathbf{F}^{C}_{t,ij} = -k_t \delta_t \tag{3.12}$$

Los coeficientes de rigidez k_n y k_t son considerados parámetros de entrada para el método DEM. Si los diámetros de las partículas p_i y p_j son, respectivamente, $D^{(i)}$ y $D^{(j)}$, δ_n puede ser calculado mediante la ecuación 3.13. Una descripción de la forma de calcular δ_t puede ser encontrada en [69].

$$\delta_n = \frac{\left(D^{(i)} + D^{(j)}\right)}{2} - \left|\mathbf{X}^{(i)} - \mathbf{X}^{(j)}\right| \tag{3.13}$$

El modelo de contacto LSD incorpora rozamiento dinámico y estático entre las partículas. Esta característica puede cambiar la forma de calcular la componente tangencial de la fuerza de contacto entre partículas que interactúan.

Sean μ el coeficiente de ficción estático entre p_i y p_j, y $\mathbf{t}_{ij} = \dfrac{\mathbf{V}_{t,ij}}{|\mathbf{V}_{t,ij}|}$ un vector unitario en la dirección tangencial. Si $|\mathbf{F}_{t,ij}| > \mu\,|\mathbf{F}_{n,ij}|$, ocurre movimiento relativo entre las partículas y se sustituye la ecuación 3.10 por la ecuación 3.14.

$$\mathbf{F}_{t,ij} = -\mu\,|\mathbf{F}_{n,ij}|\,\mathbf{t}_{ij} \tag{3.14}$$

Para el cálculo de las componentes normal y tangencial de $\mathbf{F}^D$ se utilizan las expresiones de las ecuacicones 3.15 y 3.16, siendo η_n y η_t los coeficientes de amortiguamiento normal y tangencial, respectivamente.

$$\mathbf{F}^D_{n,ij} = -\eta_n \mathbf{V}_{n,ij} \tag{3.15}$$

$$\mathbf{F}^D_{t,ij} = -\eta_t \mathbf{V}_{t,ij} \tag{3.16}$$

Por lo ya mencionado en la sección 3.1 respecto de la discretización del tiempo y su impacto en el cálculo de las fuerzas resultantes, en cualquier instante t_k la fuerza resultante ejercida sobre una partícula p_i depende solamente de la interacción de p_i con las partículas con las cuáles tiene contacto. Una consecuencia directa de esta propiedad intrínseca de DEM es la posibilidad de aplicar exitosamente técnicas de computación de alto desempeño basadas en descomposición espacial de dominio [27]. Las partículas pueden ser asignadas a diferentes procesos de cómputo tal que los sistemas EDO para partículas que no interactúan entre si son resueltos simultáneamente. La secciones 4.2 y 4.4 explican detalladamente cómo las técnicas de descomposición de dominio pueden ser aplicadas en DEM.

3.3. Ciclo de cómputo y progresión de una simulación en el tiempo

DEM es, por definición, un método numérico iterativo que simula el comportamiento de las partículas a lo largo del tiempo. En una ejecución del método el tiempo es discretizado en unidades llamadas *pasos de tiempo* y cada iteración corresponde a un instante t en el tiempo. El avance del tiempo a lo largo de la simulación es controlado a través de la ejecución de las iteraciones.

Según avanza el tiempo, el comportamiento del sistema cambia en respuesta a los resultados obtenidos, en cada paso de tiempo, por las ecuaciones del modelo de contacto y del movimiento. El ciclo de cómputo alterna entre la aplicación del modelo de contacto y de las ecuaciones de movimiento usando el principio Ley de Fuerza-desplazamiento. La segunda ley de Newton determina el movimiento (desplazamiento) de una partícula a partir de la resultante de todas las fuerzas ejercidas sobre ella y el modelo de contacto permite calcular las fuerzas de contacto a partir del desplazamiento de una partícula. Sobre las partículas pueden actuar no sólo fuerzas de contacto, sino también fuerzas externas (por ejemplo, fuerza gravitacional y fuerzas electromagnéticas).

La propuesta de Cundall y Strack [4] usa un método explícito de integración de las ecuaciones de movimiento conocido como *Central Difference Time Integration*. Un método de integración se define como explícito cuándo se utilizan los resultados de instantes

de tiempo previos $(t-k,\ldots,t-2,t-1)$ para calcular el resultado en el instante t, sin utilizar los valores ya calculados para el tiempo t. Una limitante del método usado por Cundall y Strack es la necesidad de utilizar pasos de tiempo pequeños para garantizar la estabilidad del método numérico de resolución de ecuaciones diferenciales [70].

La relación entre los movimientos de una partícula en instantes de tiempo consecutivos ($t-1$ y t) y el valor del paso de tiempo implica conocer el valor crítico del paso de tiempo que garantiza la estabilidad numérica de la simulación. La inestabilidad numérica puede ser observada como movimientos abruptos de las partículas debido a los resultados obtenidos para las ecuaciones de movimiento: ocurre una "explosión" y las partículas adquieren una aceleración mucho mayor que la esperada. En la etapa de posprocesamiento de los resultados de una simulación, la inestabilidad numérica puede, por ejemplo, hacer que las partículas atraviesen las superficies con las cuáles interactúan. Otra forma de observar el impacto en una simulación DEM de valores incorrectos de paso de tiempo es la violación de las leyes de la termodinámica [71].

Cundall y Strack [4] proponen un método de cálculo del paso de tiempo crítico que relaciona la masa uniforme m de las partículas y el coeficiente de rigidez K del resorte imaginario que conecta pares de partículas en la dirección normal (sección 3.2.2). Este método está basado en un sistema de un grado de libertad (*Single degree of freedom systems*, SDOF) compuesto por una partícula de masa m que oscila alrededor de su punto de equilibrio conectada a una pared fija a través de un resorte de coeficiente de rigidez K. La ecuación 3.17 muestra la fórmula de cálculo del paso de tiempo crítico propuesta por Cundall y Strack:

$$\Delta t_{crit} = 2\sqrt{\frac{m}{K}} \tag{3.17}$$

Es posible encontrar en la literatura métodos matemáticos que permiten calcular el valor crítico de paso de tiempo que garantiza la estabilidad en base a algunas características de la simulación. Sin embargo, no son métodos de carácter general [70, 17]. Estos métodos contemplan situaciones que no fueron consideradas en la propuesta original de DEM (por ejemplo, partículas con masas diferentes). Hay casos en que el paso de tiempo crítico es calculado de forma empírica [72].

La necesidad de considerar valores pequeños de paso de tiempo es una consecuencia directa del uso de un método expícito de integración numérica y del supuesto fundamental de DEM que la velocidad y la aceleración se mantienen constantes durante un paso de tiempo. Un método explícito de integración numérica conlleva innevitablemente a la ocurrencia de errores numéricos. En situaciones reales, la fuerza de contacto actuando sobre pares de partículas durante el periodo de tiempo t_c en que ocurre la interacción entre las partículas cambia dado que la velocidad de las partículas no es constante durante t_c. Por lo tanto, cuanto mayor el periodo de tiempo en el cual se consideran constantes la aceleraciones de las partículas, y como consecuencia las velocidades y fuerzas de contacto, mayor es el error numérico en los resultados.

Se debe observar que el valor de paso de tiempo también tiene relación directa con cuestiones prácticas de una simulación y el posprocesamiento de los resultados. Cuanto mayor sea el nivel de detalle con el cual la progresión del comportamiento del sistema debe ser representada, menor debe ser el valor de paso de tiempo. Por ejemplo, valores pequeños de paso de tiempo permiten construir animaciones en 3D con alto nivel de detalle.

Los valores de paso de tiempo están relacionados directamente con el tiempo real simulado. Una simulación con paso de tiempo t_{ps} construida para representar el comportamiento de un sistema durante un tiempo $\mathbf{T}$ requiere $i = \dfrac{\mathbf{T}}{t_{ps}}$ iteraciones. Los trabajos de Chen et al. [34, 36] relevados para este proyecto de grado mostraron que la descarga de un silo en un ambiente industrial requirió varias horas y el conjunto completo de experimentos requirió 4 meses. El tiempo de operación del silo estuvo directamente relacionado al volumen de material manejado. Además, en esta sección se mostró que la estabilidad numérica de una simulación DEM solo está garantizada si los valores de paso de tiempo son muy pequeños. Por lo tanto, estas observaciones muestran que es fundamental contar con algoritmos paralelos eficientes para simulaciones DEM o se corre el riesgo de no ser posible ejecutar tales simulaciones debido a la falta de recursos computacionales.

3.4. Construcción de conjuntos de vecinos

En el método DEM, dos partículas interactúan solo si hay contacto entre ellas. Dada una partícula p_i, las partículas p_j que están en contacto con p_i forman un conjunto de *vecinos inmediatos*. Para las partículas contenidas en un conjunto de vecinos inmediatos se verifica la ecuación 3.3. Conocer los vecinos inmediatos de una partícula evita desperdiciar tiempo de cómputo verificando si hay contacto entre pares de partículas cuyos centros están distantes.

La propuesta de Cundall y Strack [4] no incluye los conjuntos de vecinos y, por lo tanto, no establece cómo deben ser generados. El método DEM puede ser implementado sin incluir cualquier algoritmo para la generación de conjuntos de vecinos. Los conjuntos de vecinos son una herramienta de carácter exclusivamente computacional cuyo objetivo es disminuir el tiempo de cómputo necesario para la detección de contacto entre partículas. Los desarrolladores de un sistema de simulación de medios granulares tienen la libertad de elegir los algortimos a implementar para generar y manejar las listas de vecinos. Sin embargo, a los efectos de ejecutar eficientemente simulaciones DEM es necesario utilizar algoritmos que no afecten negativamente al desempeño del método y a los resultados numéricos de la simulación.

La creación de un conjunto de vecinos de una partícula p_i es un problema puramente geométrico y los algoritmos existentes se encuentran entre dos casos extremos. Un caso extremo consiste en seleccionar solamente las partículas p_j tales que p_i y p_j está en contacto, y el otro caso extremo consiste en considerar todas las partículas del sistema y transferir al modelo de contacto la responsabildad de determinar las partículas que efectivamente interactúan con p_i.

El algoritmo para el segundo caso extremo presentado es extremadamente ineficiente, aún para sistemas relativamente pequeños, porque su orden de complejidad es $\mathcal{O}(N^2)$ (siendo N la cantidad de partículas en el sistema). En este caso, para conocer los vecinos inmediatos de una partícula p_i el modelo de contacto necesita iterar sobre todas las partículas de la simulación y verificar cuáles cumplen la ecuación 3.3. Por otro lado, no es necesario implementar un algortimo para el primer caso extremo porque se verificaría más de una vez si hay interacción entre un par dado de partículas ya que el proceso de contrucción del conjunto de vecinos y el modelo de contacto repetirían la operación para verificar si hay interacción entre las partículas consideradas. Además, en este caso se desperdiciaría tiempo de cómputo en la detección de contactos.

Las listas de celdas encadenadas (*Linked-Cell Lists*) [6, 73, 74] y las listas de Verlet [7, 73, 75] son los algoritmos más ampliamente utilizados en dinámica de medios granulares para la creación de conjuntos de vecinos en métodos DEM [76, 77]. Para cada partícula p_i del sistema ambos algoritmos crean, a partir de ciertos parámetros conocidos en la literatura como *parámetros de búsqueda*, un conjunto $C_{i,v}$ de posibles vecinos inmediatos. Cuando es necesario conocer las partículas que están en contacto con p_i, el modelo de contacto necesita considerar solamente las partículas contenidas en $C_{i,v}$.

El algoritmo de listas de celdas encadenadas divide el espacio cartesiano donde se lleva a cabo la simulación en celdas de igual tamaño. Si la simulación se realiza en dos dimensiones el espacio es dividido en celdas con forma cuadrada, y si la simulación se lleva a cabo en tres dimensiones el espacio es dividido en celdas con forma cúbica. Cada partícula es asignada a una única celda según las coordenadas de su centro. El conjunto $C_{i,v}$ contiene las partículas que se encuentran en las celdas contiguas a la celda que contiene p_i. Para este algoritmo el largo del lado de celda L_c es un parámetro de búsqueda que influye directamente en el tamaño de los conjuntos de vecinos y, como consecuencia, influye directamente en el tiempo necesario para ejecutar una simulación. Si el radio de las partículas es uniforme ($r_i = r_j \forall i, j \in \{0, ..., N-1\}$) y el largo de celda no difiere demasiado del radio de las partículas ($L_c \approx r_i \forall i \in \{0, ..., N-1\}$) la simulación se realizará en una situación cercana a la óptima: cada celda albergará solamente una partícula y los conjuntos de vecinos contendrán partículas que muy probablemente estarán en contacto con la partícula para la cual se construye el conjunto. Si el sistema que está siendo simulado contiene partículas con radios diferentes o el largo de celda difiere considerablemente del radio de las partículas, un conjunto de vecinos puede resultar excesivamente grande y el tiempo necesario para la construcción de los conjuntos de vecinos puede afectar negativamente el desempeño de la simulación.

Para el algoritmo de listas de Verlet, un parámetro de búsqueda muy importante es el radio de corte R_c (*cut-off radius*). Dado una partícula p_i, el radio de corte determina la distancia máxima entre los centros de p_i y p_j para que p_j pertenezca al conjunto de vecinos de p_i. Las partículas p_k tales que la distancia entre el centro de p_i y p_k es superior a R_c no forman parte del conjunto $C_{i,v}$. En simulaciones DEM un valor típico para R_c es el diámetro de la partícula de mayor radio. Así como ocurre con el algoritmo listas de celdas encadenadas, diferentes valores de R_c y su relación con el radio de las partículas del sistema afectan directamente al tiempo necesario para ejecutar la simulación y al tamaño de los conjuntos de vecinos.

Es importante observar que dada la naturaleza dinámica de una simulación DEM, los conjuntos de posibles vecinos cambian a lo largo del tiempo cuando existe movimiento relativo entre las partículas. En tales situaciones, el conjunto de posibles vecinos $C_{i,v}$ de una partícula p_i debe ser actualizado periódicamente de modo que las partículas que se acercaron a p_i sean incluidas en $C_{i,v}$ y de modo las partículas que se alejaron de p_i sean eliminadas de $C_{i,v}$. Los sistemas de simulación DEM suelen permitir al usuario configurar el intervalo de tiempo (generalmente medido en *números de pasos de tiempo*) que debe transcurrir entre dos actualizaciones consecutivas de los conjuntos de posibles vecinos. Wan-Qing Li et al. [76] presentaron un estudio detallado sobre variaciones de los parámetros de búsqueda de las listas de Verlet y de listas de celdas encadenadas y el impacto sobre el desempeño de un sistema de simulación DEM. Welling y Germano [77] realizaron un estudio detallado de la eficiencia computacional del algortimo de listas de celdas encadenadas.

Los algoritmos de listas de Verlet y de listas de celdas encadenadas tienen orden de ejecución inferior a $\mathcal{O}(N^2)$. Sin embargo, el conjunto de posibles vecinos que construyen no es óptimo. El conjunto de posibles vecinos de una partícula p_i es óptimo si contiene solamente las partículas que están en contacto con p_i. La Triangulación de Delaunay (*Delaunay Triangulation*) es un ejemplo de algoritmo que permite construir conjuntos óptimos de posibles vecinos con orden de ejecución $\mathcal{O}(N)$ [78, 79]. Este algoritmo está basado en los trabajos de los matemáticos rusos Borís Delaunay y Georgy Voronoy.

Una triangulación es una manera de particionar adecuamente un polígono en triángulos. La partición es construida de modo que posee propiedades geométricas que facilitan los cálculos de atributos del polígono. En el contexto de dinámica de medios granulares, la trangulación de Delaunay hace que cada partícula esté completamente incluida en la celda que la contiene. Si las partículas p_i y p_j están incluidas, respectivamente, en las celdas K_i y K_j, y K_i intersecta a K_j, entonces p_i y p_j se solapan y p_j pertenece al conjunto $C_{i,v}$.

Usando la triangulación de Delaunay es posible obtener un conjunto óptimo de vecinos sin depender de parámetros de búsqueda. Sin embargo, aunque cuenta con propiedades geométricas importantes que facilitan la detección de contactos entre partículas, la triangulación de Delaunay no es usada habitualmente en sistemas de simulación para medios granulares por tratarse de un problema geométrico complejo que puede no tener solución única [80, 81] y que requiere algoritmos especializados y estructuras de datos avanzadas [82, 83].

En este proyecto de grado se utiliza el sistema de simulación numérica LIGGGHTS que implementa el algortimo de listas de Verlet y permite al usuario configurar el radio de corte R_c y la periodicidad con la cual los conjuntos de vecinos son regenerados. No fueron realizados cambios en la implementación del algortimo de generación de listas de vecinos. El capítulo 4 presenta los detalles de implementación de LIGGGHTS relevantes para este proyecto de grado.

Capítulo 4

El sistema de simulación numérica LIGGGHTS

Este capítulo describe el sistema de simulación numérica LIGGGHTS, incluyendo las principales características de las estrategias de descomposición de dominio aplicadas en LIGGGHTS. La sección 4.1 presenta la organización de LIGGGHTS desde el punto de vista de las jerarquías de clases de software. La sección 4.2 explica cómo funciona la estrategia original de descomposición de dominio implementada en LIGGGHTS. La sección 4.3 muestra las principales decisiones de diseño tomadas para utilizar las técnicas de computación paralela en LIGGGHTS. La sección 4.4 presenta un patrón de comunicación interprocesos utilizado habitualmente en las implementaciones paralelas de DEM. Finalmente, la sección 4.5 explica cómo LIGGGHTS implementa la comunicación interprocesos.

4.1. La anatomía de LIGGGHTS

Técnicamente, LIGGGHTS es una bifurcación (*fork*) del sistema de dinámica molecular LAMMPS, al cual expande agregando la capacidad de resolver problemas de dinámica de medios granulares aplicando el método DEM [84]. Las principales adaptaciones incorporadas por LIGGGHTS son: (a) el soporte al tratamiento geométrico de mallas (*mesh geometry*), (b) la inclusión de modelos de contacto para interacción partícula-partícula y partícula-superficie y (c) la capacidad de modelar la transferencia de calor entre partículas y superficies. LIGGGHTS es desarrollado por la empresa DCS Computing (Linz, Austria) [65] y se encuentra en actualización y ampliación constante. El equipo de desarrolladores está compuesto por integrantes de DCS Computing y algunos programadores independientes, y pone énfasis a la inclusión de funcionalidades que permiten abordar problemas a escala industrial [19].

LIGGGHTS es un sistema complejo para computación científica y está dirigido a usuarios avanzados del ámbito académico y de determinados sectores de la industria y de la ingeniería. El sistema posee un conjunto muy amplio de herramientas y opciones de configuración que permiten ejecutar simulaciones numéricas y obtener los resultados de manera conveniente para su posprocesamiento. Todo su código fuente está programado en lenguaje de programación C++ aplicando el paradigma de programación orientada a objetos. Las técnicas de programación paralela son implementadas a través de la biblioteca de intercambio de mensajes MPI.

Existen actualmente tres variantes de LIGGGHTS: (1) PUBLIC, (2) PREMIUM y (3) CONSORTIUM. La variante PUBLIC es distribuida gratuitamente a través de la licencia para código abierto GPL (*GNU Public License*). Las variantes PREMIUM y CONSORTIUM no son de uso gratuito. Una diferencia relevante para este proyecto de grado entre las variantes de acceso público y de acceso mediante pago es la disponibilidad de determinadas estrategias de descomposición de dominio. Una estrategia de descomposición dinámica de dominio solo está disponible en las variantes PREMIUM y CONSORTIUM. En este proyecto de grado se utiliza la variante PUBLIC dado que es la variante utilizada por los investigadores del IMFIA y por considerarse que se agrega una nueva funcionalidad a esta variante. Otro factor que fundamenta esta decisión es la facilidad de acceso al código fuente de la variante PUBLIC.

La versión de la variante PUBLIC utilizada en el proyecto es la 3.5.0, lanzada en setiembre de 2016. Las versiones lanzadas a partir de la versión 3.5.0 están enfocadas en correcciones de *bugs* que no afectan las estrategias de descomposición de dominio y que tampoco afectan la ejecución paralela de simulaciones. Las versiones más recientes incluyen funcionalidades que permiten resolver nuevos problemas o usar abordajes novedosos para resolver problemas ya conocidos. El método de descomposición dinámica de dominio implementado en este proyecto de grado puede ser incorporado desde las versiones 3.5.0 hasta la versión 3.8.0.

LIGGGHTS está construido en base a clases de software que implementan determinadas funcionalidades. Existen clases que implementan funcionalidades de carácter general (contadores de tiempo, comunicación entre procesos, interacción con el usuario, bucles de integración, etc) y clases que implementan los aspectos teóricos de la dinámica de medios granulares (detección de contacto entre objetos, cálculo de fuerzas y velocidades, etc.). Las principales clases están organizadas en forma jerárquica según se muestra en la figura 4.1.

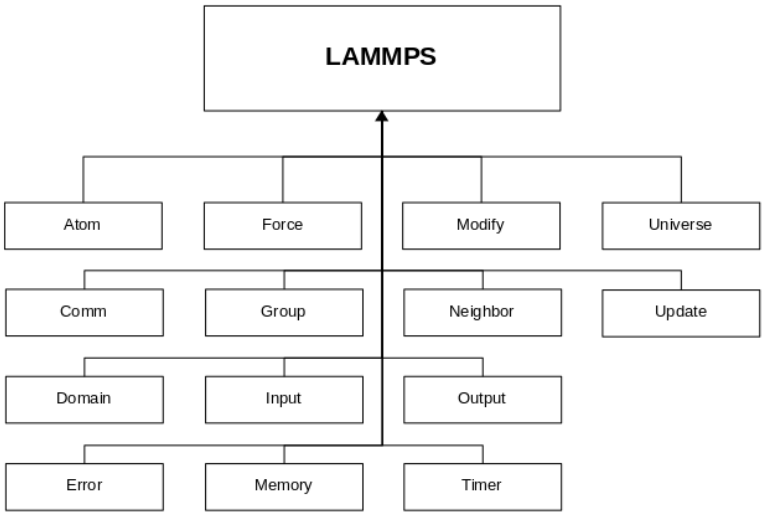


Figura 4.1: Jerarquía de las principales clases de software de LIGGGHTS

La clase `LAMMPS` representa la simulación que será ejecutada y tiene visibilidad sobre las demás clases de la jerarquía. El esquema de visibilidad general permite invocar los métodos especializados para ejecutar las tareas necesarias. La tabla 4.1 presenta una breve descripción de las principales clases de LIGGGHTS.

Clase	Descripcíon
`Atom`	Atributos y propiedades de las partículas
`Comm`	Gestión de comunicación interprocesos
`Domain`	Región en el espacio dónde se ejecuta la simulación
`Error`	Gestión de errores
`Force`	Métodos para cálculo de fuerzas
`Group`	Gestión de grupos de partículas
`Input`	Procesamiento del archivo de entrada
`Memory`	Gestión de memoria
`Modify`	Implementación de los comandos que realizan cambios en la simulación
`Neighbor`	Gestión de lista de vecinos
`Output`	Gestión de la salida de una simulación
`Timer`	Información sobre tiempos de cómputo
`Universe`	Universo MPI con los todos los procesadores
`Update`	Métodos para integración numérica

Tabla 4.1: Principales clases de software de LIGGGHTS

El ciclo de integración del método DEM está implementado cómo un bucle que es ejecutado una cantidad de veces igual al número de pasos de tiempo definido por el usuario. En cada iteración del bucle se realizan varias acciones invocando a los métodos que las implementan. La figura 4.2 muestra un esquema de las etapas del ciclo de integración. No todas las etapas del ciclo son ejecutadas en cada iteración del bucle. Las configuraciones definidas por el usuario y las características del problema a ser simulado determinan las etapas que serán ejecutadas y al inicio de cada etapa del ciclo de integración LIGGGHTS verifica el cumplimiento de determinadas condiciones para decidir si una etapa debe ser ejecutada.

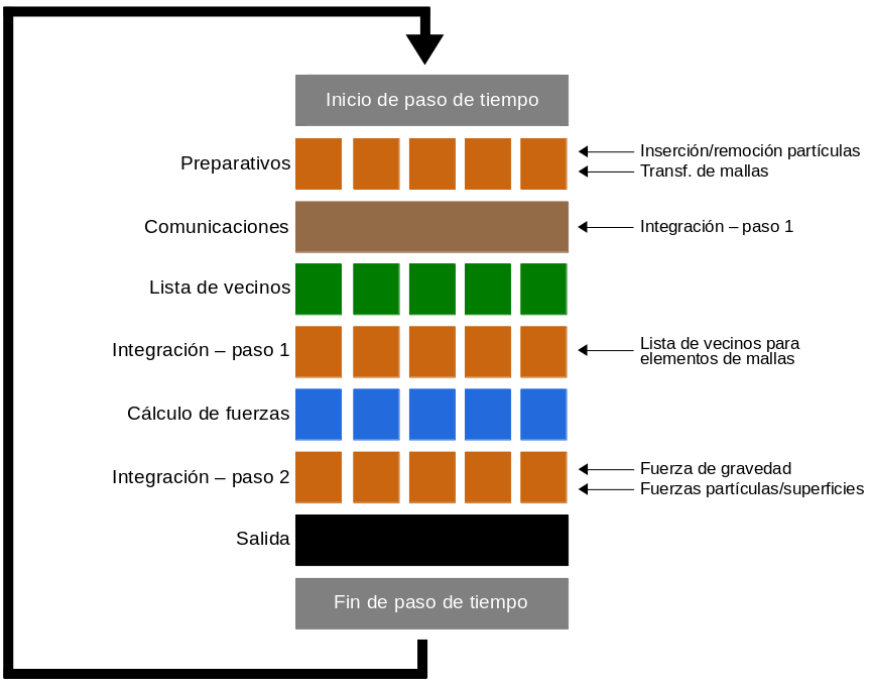


Figura 4.2: Esquema del ciclo de integración de LIGGGHTS

Para ejecutar una simulación el usuario configura el comportamiento de LIGGGHTS y las propiedades de la simulación a través de un archivo de entrada de carácter declarativo cuyo contenido indica los comandos que deberán ser ejecutados. El archivo de entrada es procesado secuencialmente por el método `Input::file()` y un bucle de integración inicia cuando un comando `run` es encontrado. Una simulación puede contener más de una invocación al comando `run`. La primera invocación de `run` debe ser realizada después que las propiedades físico-químicas del problema fueron definidas. La clase `Input` no contiene las implementaciones de los métodos invocados a partir de los comandos declarados en el archivo de entrada. `Input` implementa métodos cuyos objetivos son: (1) verificar la correctitud de los parámetros pasados a los comandos, (2) definir parámetros de control de LIGGGHTS y (3) invocar el método correspondiente en la clase correcta.

La estructura que debe seguir el archivo de entrada indica que una simulación se divide en dos fases. Existe una fase de configuración en la cuál se ejecutan comandos que determinan el comportamiento de LIGGGHTS y las propiedades físico-químicas de la simulación y una fase de cómputo dónde se ejecutan los comandos que modelan el comportamiento dinámico de las partículas. Estas fases son disjuntas y pueden repetirse varias veces en un mismo archivo de entrada.

Al comando `run` le corresponde el método abstracto `Integrate::run()`. Las clases que extienden a la clase `Integrate` deben proveer una implementación de este método. Están disponibles dos esquemas de integración de las ecuaciones de movimiento: (a) Velocity Verlet integrator [85] y (b) rRESPA multi-timescale integrator [86]. Las implementaciones de estos esquemas de integración se encuentran, respectivamente, en las clases `Verlet` y `Respa`. Dado que ambos esquemas de integración permiten calcular resultados numéricos con el mismo nivel de precisión [86, 87, 88], este proyecto de grado está enfocado en trabajar solamente con la clase `Verlet`. Según el esquema de la figura 4.2, cualquier implementación de la operación `Integrate::run()` es responsable de invocar los métodos que implementan las etapas del ciclo de integración. Los métodos invocados pueden ser locales o pertenecer a otras clases.

Una vez que un comando `run` es leído y procesado, LIGGGHTS está en condiciones de llevar a cabo la fase de cómputo numérico de la simulación. A lo largo de las iteraciones del bucle de integración, se aplican la estrategia de descomposición de dominio y el modelo de paralelismo adoptados.

4.2. Estrategia de descomposición de dominio

La descomposición de dominio se realiza declarando el comando `create_box` en el archivo de entrada. Este comando invoca a los métodos `Domain::set_initial_box()`, `Domain::set_global_box()`, `Comm::set_proc_grid()` y `Domain::set_local_box()`. Debe seguirse este orden para invocar los métodos para que la descomposición de dominio se realice adecuamente. Cambiar el orden de invocación genera errores en tiempo de ejecución. La estrategia de descomposición de dominio es estática porqué sólo es posible declarar el comando `create_box` una vez en el archivo de entrada y durante la ejecución del ciclo de integración el método `set_proc_grid()` no es invocado desde otros comandos o métodos que podrían cambiar la descomposición de dominio generada inicialmente. Además, una vez que un comando `run` es invocado y el ciclo de integración inicia, no se ejecutan otros comandos que fueron declarados en el archivo de entrada hasta que todas las iteraciones del ciclo sean finalizadas. Usualmente, el comando `run` es la forma de

iniciar una fase de cómputo de la simulación y de invocar al comando `set_proc_grid()`. Las tareas que el método `set_proc_grid()` realiza se presentan más adelante.

El comando `create_box` trabaja en conjunto con el comando `region`. Al declarar estos comandos en el archivo de entrada el usuario define la región en el espacio dónde se lleva a cabo la simulación. LIGGGHTS llama esta región *global box*. La región definida corresponde al dominio del problema. Los parámetros del comando `region` deben incluir seis números reales que definen los puntos máximo y mínimo de tres intervalos en los ejes de coordenadas x, y y z. Los puntos máximos y mínimos de los tres intervalos definen un ortoedro (paralelepípedo ortogonal) cuyas esquinas están localizadas en los puntos $\{(a, b, c) \mid a \in \{min_x, max_x\}, b \in \{min_y, max_y\}, c \in \{min_z, max_z\}\}$. La figura 4.3 muestra un ejemplo de región (dominio) y destaca las coordenadas de algunas de sus esquinas.

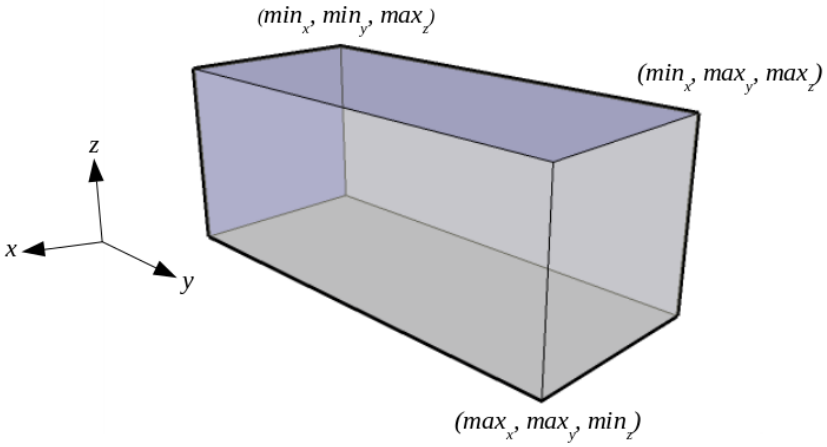


Figura 4.3: Ejemplo de región (dominio) de una simulación

LIGGGHTS permite iniciar una simulación a partir de un archivo binario especial (*restart file*) generado por otra simulación. En este caso el archivo de entrada no debe incluir a los comando `create_box` y `region`, y la simulación es configurada en base a los datos almacenados en el archivo binario generado previamente. Parte del contenido de este archivo binario son los parámetros de configuración de los comandos `create_box` y `region` usados previamente, y si estos comandos son incluidos junto con la declaración de inclusión del archivo binario se generara un error en tiempo de ejecución. Cuando un archivo *restart file* es declarado en el archivo de entrada, el método `set_proc_grid()` es invocado por otros comandos y se garantiza la no invocación de los comandos `create_box` y `region`.

El método `set_proc_grid()` es ejecutado por todos los procesos creados para ejecutar la simulación y tres de sus tareas fundamentales son: (1) determinar la asignación de procesos a subregiones, (2) determinar los vecinos directos de cada proceso y (3) calcular las coordenadas de los puntos máximo y mínimo en cada eje de coordenadas que determinan la subregión a la cuál un proceso es asignado.

La asignación de procesos a subregiones se hace de forma indirecta usando el método `ProcMap::cart_map()` cuya implementación está basada en las instrucciones `MPI_Cart_create` y `MPI_Cart_rank` para la creación de grillas cartesianas. Los procesos son asignados a los elementos de una estructura auxiliar en forma de grilla y estos elementos son asignados a las subregiones. De esta forma se define la asignación de procesos

a subregiones y la estructura auxiliar es descartada. La clase `Comm` posee un atributo que corresponde a una instancia de la clase `Procmap` y a través de este atributo se realizan las invocaciones a los métodos de esa clase.

La instrucción `MPI_Cart_create` crea un nuevo comunicador MPI a partir de un comunicador existente. Al nuevo comunicador se le adjunta la información correspondiente para la creación de la topología cartesiana. El nuevo comunicador se crea de modo que para todo proceso j su rango es igual en ambos comunicadores (este comportamiento puede ser alterado, pero no se modifica en el caso de LIGGGHTS). `MPI_Cart_create` requiere que se definida el número de procesos $PROCS_1$, $PROCS_2$ y $PROCS_3$ que serán usados, respectivamente, en las dimensiones x, y y z del espacio cartesiano que da propiedades topológicas al nuevo comunicador. Estos valores son definidos por el usuario y para ello se debe declarar el comando `processors` en el archivo de entrada junto con tres parámetros de configuración. La cantitad total de procesos usados en la simulación y los parámetros del comando `processors` debe verificar $E = PROCS_1 \times PROCS_2 \times PROCS_1$. Los valores $PROCS_1$, $PROCS_2$ y $PROCS_3$ son transferidos a la instrucción `MPI_Cart_create` para la construcción de una grilla G de tres dimensiones que contiene E elementos. Los elementos de G son identificados usando coordenadas cartesianas (x, y, z) tales que $x, y, z \in \mathbb{N}$.

La grilla G no contiene información geométrica del dominio del problema. Su objetivo es permitir la asignación de procesos a subregiones de forma conveniente, aprovechando las potencialidades de la biblioteca MPI. La construcción de G se hace de forma que un elemento puede tener a lo sumo seis vecinos directos. Los vecinos directos del elemento cuyas coordenadas son (x, y, z) son los elementos cuyas coordenadas pertenecen al conjunto $\{(x-1, y, z), (x+1, y, z), (x, y-1, z), (x, y+1, z), (x, y, z-1), (x, y, z+1)\}$. Para asignar los procesos a los elementos de G, la instrucción `MPI_Cart_create` se comporta como una secuencia de tres bucles anidados que iteran sobre los ejes de coordenadas x, y y z, respectivamente, y tales que los procesos son asignados según el orden creciente de rangos, empezando por el proceso de menor rango. El algoritmo 1 presenta una manera de representar la asignación biunívoca de procesos a los elementos de G, siendo r_0, r_1, ..., r_{E-1} los rangos de los procesos en ejecución y tales que $r_0 < r_1 < \ldots < r_{E-1}$.

Algoritmo 1 Asignación de procesos a elementos de G

```
1: for i := 0 to PROCS_1 − 1 do
2:   for j := 0 to PROCS_2 − 1 do
3:     for k := 0 to PROCS_3 − 1 do
4:       m := m + 1;
5:       (i,j,k) := r_m;
6:     end for
7:   end for
8: end for
```

Para conocer el rango del proceso asignado a un elemento, el método `cart_map()` invoca a la instrucción `MPI_Cart_rank`. La invocación de esta instrucción se hace usando bucles anidados que iteran sobre los tres ejes de coordenadas y producen, al final de todas las iteraciones, una matriz llamada `grid2proc` que almacena en `grid2proc[i][j][k]` el rango del proceso asignado al elemento (i, j, k). La figura 4.4 muestra un ejemplo de asignación de procesos a elementos de la grilla, tal que $PROCS_1 = PROCS_2 = 4$ y $PROCS_3 = 3$.

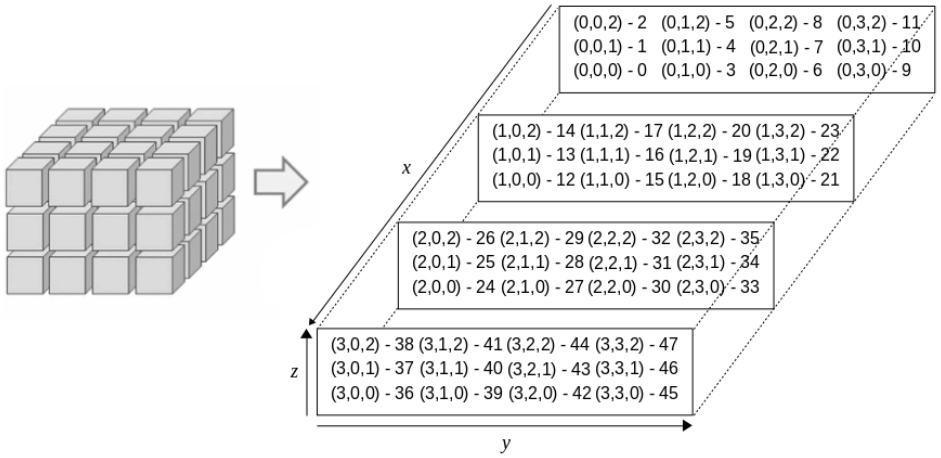


Figura 4.4: Ejemplo de grilla cartesiana para $PROCS_1 = PROCS_2 = 4$ y $PROCS_3 = 3$

Más adenlante, en la presente sección, se muestra la metodología según la cual se calculan las coordenadas de los puntos máximo y mínimo en cada eje de coordenadas que definen la subregión a la cual un proceso es asignado y se explica cómo LIGGGHTS usa la grilla G para concluir la tarea de asignación de procesos a subregiones. A los efectos de explicar la descomposición de dominio utilizanda en LIGGGHTS, es suficiente observar que cada elemento de la grilla se relaciona biunívocamente con una subregión del dominio del problema. Además, por la exposición presentada en los párrafos anteriores, ya se conoce la forma en que LIGGGHTS asigna los procesos a los elementos de G. Por lo tanto, si la subregión r está relacionada al elemento g, es posible identificar r a través de las coordenadas (x_g, y_g, z_g) que identifican a g y el proceso asignado a r es, de hecho, el proceso asignado a g.

Para calcular las coordenadas que definen una subregión, LIGGGHTS utiliza un procedimiento de dos etapas. En la primera etapa se descompone un dominio que corresponde al cubo $U = [0, 1]^3$ y en la segunda etapa se utiliza esta descomposición para obtener las subregiones del dominio. La primera etapa empieza calculando puntos de corte sobre los intervalos $[0, 1]$ correspondientes a cada eje de coordenadas según la expresión 4.1, donde $dim \in \{1, 2, 3\}$, $PARTS_{dim}$ es la cantidad de particiones que deben ser generadas para el intervalo $[0, 1]$ en la dimensión dim y P^k_{dim} es el k-ésimo punto de corte en la dimensión dim.

$$P^k_{dim} = \begin{cases} k \times \frac{1}{PARTS_{dim}}, & \text{si } 0 \leq k < PARTS_{dim} - 1 \\ 1, & \text{si } k = PARTS_{dim} \end{cases} \tag{4.1}$$

Las dimensiones 1, 2 y 3 corresponden, respectivamente, a los ejes de coordenadas x, y y z, y los valores $PARTS_1$, $PARTS_2$ y $PARTS_3$ son definidos por el usuario a través del comando `processors`. En realidad, los valores $PARTS_1$ y $PROCS_1$ se corresponden al mismo parámetro declarado en el comando `processors` y se verifica la igualdad $PARTS_1 = PROCS_1$. Lo mismo ocurre con $PARTS_2$ y $PROCS_2$, y $PARTS_3$ y $PROCS_3$. Por lo tanto, a través del comando `processors` el usuario define cómo se crea la grilla G y cómo se particionan los ejes de coordenadas x, y y z tal que ambas estructuras son compatibles y de modo que sobre ellas se puede realizar la descomposición de dominio. Este proyecto de grado utiliza dos nomenclaturas diferentes para enfatizar los roles que cumplen los parámetros del comando `processors` en cada tarea del método `set_proc_grid()`.

Según la expresión 4.1, cada intervalo $[0,1]$ que define el cubo U tiene asociado un conjunto de puntos de corte $K_{dim} = \left\{0, \frac{1}{PARTS_{dim}}, \frac{2}{PARTS_{dim}}, \ldots, 1\right\}$. A partir de los conjuntos K_1, K_2 y K_3 se construye un total de $S = \prod_{dim=1}^{3} PARTS_{dim}$ subregiones, tal que cada subregión es un ortoedro cuyos lados miden, respectivamente, $\frac{1}{PARTS_1}$, $\frac{1}{PARTS_2}$ y $\frac{1}{PARTS_3}$, y las coordenadas de sus esquinas son los puntos que pertenecen al conjunto $\{(a,b,c) \mid a \in K_1, b \in K_2, c \in K_3\}$. Como las subregiones son estructuras ortogonales y considerando e_1 el eje de coordenadas para el cuál se generan las particiones, las fronteras entre subregiones son paralelas entre si, perpendiculares al eje de coordenadas e_1 y paralelas a los ejes de coordenadas e_2 y e_3, siendo $e_j \in \{x, y, z\}$ con $j \in \{1, 2, 3\}$ y tal que $e_1 \neq e_2$, $e_1 \neq e_3$ y $e_2 \neq e_3$.

La figura 4.5 muestra un ejemplo de descomposición del cubo U utilizando dos particiones por intervalo ($PARTS_1 = PARTS_2 = PARTS_3 = 2$). Se observa que los puntos de corte sobre el eje x determinan fronteras que son paralelas entre si, perpendiculares al eje x y paralelas a los ejes y y z. Una situación simular ocurre con los ejes y y z.

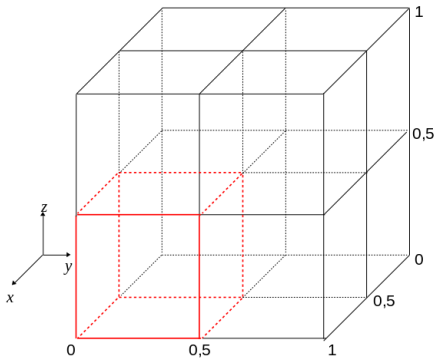


Figura 4.5: Descomposición del cubo $I = [0,1]^3$ usando dos particiones por intervalo

Dado que $PARTS_{dim} = PROCS_{dim}$, $\forall dim \in \{1,2,3\}$, se verifica que $E = S$ y es posible asociar biunívocamente los elementos de la grilla G a las subregiones del cubo U. Esta asociación es implícita, no está implementada en método alguno. Se asume que las subregiones u de U son identificadas con coordenadas enteras (x, y, z) tal como ocurre con los elementos de g de G. De esta forma, la subregión u es asociada al elemento g si, y solo si, las coordenadas de u y g coinciden.

Según la metodología presentada en esta sección para la construcción de las subregiones del dominio, se puede concluir que LIGGGHTS determina las dimensiones de cada subregión basándose en: (1) los puntos de corte definidos para el cubo U y (2) en las dimensiones de la región del problema. Por lo tanto, hay una relación isomorfa entre las subregiones de U y las subregiones del dominio del problema y se determina la asignación de procesos a cada subregión del dominio. La información necesaria para generar los puntos extremos de una subregión está almacenada en las clases `Comm` y `Domain`. Los métodos de estas clases trabajan en forma conjunta para garantizar que los valores almacenados sean coherentes y para permitir generar correctamente la descomposición de dominio.

Para finalizar la primera etapa del cálculo de las coordenadas que definen una subregión, el método `cart_map()` invoca la instrucción `MPI_Cart_get` para obtener las coordenadas de un proceso en el espacio cartesiano asociado al nuevo comunicador creado previamente con la instrucción `MPI_Cart_create` (las coordenadas de un proceso son las coordenadas que identifican al elemento de G al cual el proceso fue asignado). El resultado se almacena en el atributo `myloc` de la clase `Comm`. Este atributo es un vector de números enteros con tres posiciones y, dada la forma en que es construido, se cumple $0 \leq \texttt{myloc[dim]} \leq PROCS_{dim} - 1, \forall dim \in \{1, 2, 3\}$.

La instrucción `MPI_Cart_get` retorna diferentes resultados según el proceso que la ejecuta y esta diferencia en los resultados permite que un proceso calcule correctamente las dimensiones de la subregión a la cual fue asignado. Las instrucciones invocadas antes de `MPI_Cart_get` devuelven resultados idénticos para todos los procesos. Por lo tanto, todos los procesos conocen las dimensiones de la región (dominio) del problema y cada proceso conoce las dimensiones de la subregión (subdominio) al cuál fue asignado. Independientemente de la forma en que opera la instrucción `MPI_Cart_get`, la información de las propiedades geométricas del dominio a nivel global y local que cada proceso conoce es una decisión de implementación.

La segunda etapa del cálculo de las coordenadas que definen una subregión consiste en usar la descomposición construida para el cubo U para obtener la descomposición del dominio. Esta etapa está implementada en el método `set_local_box()` y usa información generada previamente por `set_proc_grid()`. Se considera el intervalo $I_{dim} = [min_{dim}, max_{dim}]$ y los puntos de corte del conjunto K_{dim}; se define $L_{dim} = max_{dim} - min_{dim}$ y se calculan los valores $minLocal_{dim}$ y $maxLocal_{dim}$ mediante las ecuaciones 4.2 y 4.3.

$$minLocal_{dim} = min_{dim} + L_{dim} \times P^k_{dim} \tag{4.2}$$

$$maxLocal_{dim} = \begin{cases} min_{dim} + L_{dim} \times P^{k+1}_{dim}, & \text{si } P^k_{dim} \neq 1 \\ max_{dim}, & \text{si } P^k_{dim} = 1 \end{cases} \tag{4.3}$$

El intervalo I_{dim} está determinado por los puntos extremos de la región del problema para la dimensión dim. Para determinar los puntos de corte P^k_{dim} y P^{k+1}_{dim} que intervienen en las ecuaciones 4.2 y 4.3, el método `set_local_box()` utiliza los valores almacenados en el vector `myloc` aplicando las ecuaciones 4.4 y 4.5.

$$P^k_{dim} = \frac{\texttt{myloc}[dim - 1]}{PARTS_{dim}} \tag{4.4}$$

$$P^{k+1}_{dim} = \frac{\texttt{myloc}[dim - 1] + 1}{PARTS_{dim}} \tag{4.5}$$

Considerando, por ejemplo, una simulación tal que $PARTS_1 = PARTS_2 = 4$ y $PARTS_3 = 3$, y siendo j uno de los procesos que la ejecutan tal que `myloc[0]` $= 2$, `myloc[1]` $= 0$ y `myloc[2]` $= 3$, se obtienen los siguientes puntos de corte para el subdominio al cual j fue asignado $P^2_1 = \frac{2}{4}$ y $P^3_1 = \frac{3}{4}$, $P^0_2 = 0$ y $P^1_2 = \frac{1}{4}$, y $P^2_3 = \frac{2}{3}$ y $P^3_3 = 1$.

Las coordenadas de los puntos extremos que definen la subregión a la cual el proceso es asignado pertenecen al conjunto $\{(a, b, c) \mid a \in \{minLocal_x, maxLocal_x\}, b \in \{minLocal_y, maxLocal_y\}, c \in \{minLocal_z, maxLocal_z\}\}$ y son almacenadas en atributos de la clase `Domain`. Los valores retornados por `MPI_Cart_get` a j y l son diferentes. Esto significa que al menos una coordenada de los vectores $\texttt{myloc}_j$ y $\texttt{myloc}_l$ difieren entre

sí y se generan, al menos, un par de puntos de corte diferentes. Por lo tanto, es posible corroborar que la relación entre subregión y procesos es efectivamente biunívoca.

Los métodos `set_initial_box()` y `set_global_box()` pueden cambiar las dimensiones de la región donde el problema está definido y afectar indirectamente la descomposición de dominio. Estos métodos pueden ser invocados como consecuencia del uso de los comandos `change_box`, `box/relax`, `deform` y `minimize`. Sin embargo, estos comandos deben ser declarados deliberadamente por el usuario y las declaraciones leídas desde el archivo de entrada puede realizar cambios en el dominio a lo sumo una vez. Además, se sabe que mientras las iteraciones de un ciclo de integración no hayan terminado, ningún comando declarado por el usuario es ejecutado.

La descomposición de dominio es generada al principio de la simulación, durante la fase de configuración del sistema. Una vez finalizada, LIGGGHTS pasa a la fase de cómputo numérico mediante la invocación de un comando `run` o sigue con otras tareas de configuración.

4.3. Modelo de paralelismo utilizado en LIGGGHTS

LIGGGHTS es un sistema que implementa el modelo de paralelismo *Single Program, Multiple Data* (SPMD). En general, los comandos declarados en el archivo de entrada son ejecutados por todos los procesos, pero existen determinados comandos (por ejemplo, comandos para manejo de entrada y salida) que son ejecutados solamente por el proceso de rango 0. Esta decisión de diseño no significa que LIGGGHTS también implemente un modelo de paralelismo *maestro-esclavo*. En LIGGGHTS no se crea proceso alguno que controle y coordine a los demás procesos. El objetivo de ejecutar de forma centralizada determinados comandos es mantener sencilla la implementación de sus algoritmos. Este comportamiento ocurre, por ejemplo durante la lectura del archivo de entrada.

La sincronización de los procesos se da a través de instrucciones MPI y de la organización y estructura de los algoritmos que implementan los métodos y comandos de LIGGGHTS. El algoritmo 2 muestra un ejemplo de coordinación de procesos que ocurre durante la lectura del archivo de entrada.

Algoritmo 2 Ejemplo de sincronización de procesos – lectura del archivo de entrada

```
1:  while true do
2:    // leer 1 línea desde el archivo de entrada
3:    if rank = 0 then
4:      read_line()
5:    end if
6:    // enviar la línea leída a todos los procesos
7:    MPI_Bcast()
8:    // procesar la línea de entrada
9:    parse()
10:   // se termina de leer un comando, ejecutar el 0método correspondiente
11:   if read_command = 1 then
12:     execute_command()
13:     break()
14:   end if
15: end while
```

El objetivo de la instrucción `MPI_Bcast()` (línea 7 del algoritmo 2) es difundir a todos los procesos en ejecución una línea leída por el proceso de rango 0 desde el archivo de entrada. La difusión se realiza solamente a los procesos que pertenecen a un mismo comunicador MPI. En el algoritmo 2, la sincronización de procesos ocurre como consecuencia del modo en que MPI implementa la instrucción `MPI_Bcast()`. Cuando un proceso de rango j, con $j \neq 0$, invoca `MPI_Bcast()` su ejecución se detiene hasta que el proceso de rango 0 ejecute `MPI_Bcast()` y envíe al proceso de rango j los datos que éste requiere para seguir su ejecución. Cuando termina la transferencia de datos los procesos de rango 0 y de rango j siguen su ejecución.

En la fase de configuración de la simulación, el proceso de rango 0 obtiene los parámetros desde el archivo de entrada y los envía a los demás procesos. Todos los procesos reciben los mismos parámetros y configuran el sistema adecuamente según los parámetros recibidos. Por ejemplo, todos los procesos reciben los parámetros que definen las dimensiones del dominio del problema, pero cada proceso calcula solamente las dimensiones de la subregión a la cuál fue asignado. En esta fase puede ser necesario escribir datos en el disco y leer datos del disco.

En las fases de cómputo todos los procesos son utilizados (inclusive el proceso de rango 0). En este caso todos los procesos ejecutarán los mismos métodos y los resultados dependen de la subregión del dominio a la cuál el proceso fue asignado. Si dos procesos requieren intercambiar mensajes para sincronización o envío de información durante la fase de cómputo, lo hacen directamente entre ellos, sin intermediarios. En esta fase puede ser necesario escribir datos en el disco.

Las tareas de entrada y salida son ejecutadas solamente por el proceso de rango 0. Este proceso recibe de los demás procesos los datos que deben ser escritos y se encarga de persistir la información en los archivos correspondientes.

4.4. Comunicación interprocesos en el método DEM

Como se comentó en el capítulo 3, DEM es un método numérico de cómputo intensivo. El comportamiento de cada partícula del sistema debe ser estudiado y múltiples instancias de las ecuaciones de movimiento deben ser resueltas. La propuesta de Cundall y Strack [4] no define las características computacionales que deben tener las implementaciones de DEM. Los desarrolladores de una determinada implementación del método tienen la libertad de elegir las metodologías y paradigmas de desarrollo que consideren convenientes. Investigadores han observado que DEM posee características intrínsecas que favorecen la aplicación de técnicas de computación distribuida y, especialmente, la aplicación del paradigma de programación paralela [10, 19, 18, 24, 25].

La técnica de descomposición espacial de dominio [27] es una de las más utilizadas para una implementación paralela de DEM. Los subdominios generados por una estrategia de descomposición espacial de dominio estarán dispuestos sobre una grilla de dos o tres dimensiones y tendrán, como consecuencia, una topología cartesiana que define las posiciones relativas entre los subdominos. Para un subdominio d_i, los subdominios adyacentes a d_i respecto de cada eje de coordenadas son definidos como *vecinos directos*. La topología de los subdominios define: (1) fronteras entre los subdominios adyacentes y (2) la pertenencia de cada partícula a un subdomino determinado. Las partículas que se encuentran sobre la frontera deben ser consideradas como si estuvieran contenidas en ambos dominios.

La pertenencia de una partícula a la frontera entre subdominios se define paramétricamente según el valor máximo del radio de las partículas que pertenecen al sistema (variable `maxrad`). En LIGGGHTS el ancho de la frontera es, por defecto, igual a $H = 2,2 \times$ `maxrad` (el usuario puede ajustar el valor de H a través de comandos de usuario). De esta forma, las partículas contenidas en un subdominio d_i cuyos centros están a una distancia menor o igual que H de alguno de sus límites se encuentran en la frontera entre d_i y d_j, siendo d_j un vecino directo de d_i. Así se garantiza que en cada subdominio la frontera tiene espacio suficiente para contener partículas cuyos radios son menores o iguales al radio máximo de partículas contenidas en la simulación. Esta definición paramétrica de pertenencia a una frontera entre subdominios es una decisión de diseño que debe ser tomada por los desarrolladores de una implementación paralela de DEM.

En la práctica, una partícula P que se encuentra en la frontera entre los subdominios d_i y d_j pertenece al contexto del proceso p_i (proceso relacionado con d_i) y es considerada como una *partícula fantasma* por el proceso p_j (proceso relacionado con d_j). Otra posibilidad es considerar que P está en el contexto del proceso p_j y que es una partícula fantasma para p_i. Para ambas posibilidades los resultados numérico deben ser idénticos. Dado un subdominio d_i y el proceso p_i con el cuál está relacionado, las partículas fantasma de p_i son aquellas que: (a) se encuentran en el contexto de los procesos relacionados con los vecinos directos de d_i y (b) como se encuentran sobre las fronteras entre d_i y d_j, siendo d_j un vecino directo de d_i, interactúan con las partículas que están completamente dentro de d_i y que se encuentran cercanas a la frontera. La necesidad de mantener actualizada la información sobre la frontera entre subdominios exige que durante una simulación DEM, los procesos asignados a dos subdominios adyacentes intercambien información constantemente.

La figura 4.6 muestra un ejemplo en dos dimensiones de una descomposición de dominio y destaca las fronteras entre subdominios. Las partículas están coloreadas para destacar a cuál subdominio pertenecen y se indican con flechas de colores algunos ejemplos de partículas fantasma. Las partículas señaladas con la flecha de color rojo pertenecen al subdominio (0,1) y deben ser consideradas partículas fantasma para el proceso relacionado con el dominio (0,0). Las partículas señaladas con la flecha de color azul pertenecen al subdominio (2,2) y deben ser consideradas partículas fantasma para el proceso relacionado con el dominio (1,2).

Los procesos que se encuentran ejecutando una simulación DEM también deben intercambiar mensajes cuando sea necesario transferir partículas entre subdominios. Como se comentó en la sección 3.3, se espera un desplazamiento muy pequeño de las partículas entre dos pasos de tiempo consecutivos. El desplazamiento resultante al final de un paso de tiempo puede realizarse hacia cualquier dirección y de modo tal que si (x, y, z) son las coordenadas del subdominio de origen, las coordenadas del subdominio de destino deben pertenecer al conjunto $\{(a, b, c) \mid a \in \{x-1, x, x+1\}, b \in \{y-1, y, y+1\}, c \in \{z-1, z, z+1\}\}$. La forma de implementar la transferencia de las partículas es una decisión de implementación. La sección 4.5.2 muestra cómo implementa LIGGGHTS la transferencia de partículas según el comportamiento descrito en este párrafo. Considerando el ejemplo de descomposición de dominio de la figura 4.6, una partícula que sale del subdominio (1,1) y entra al subdominio (1,2) es transferida directamente, y una partícula que sale del subdominio (1,1) y entra al subdominio (2,0) es transferida primero desde (1,1) a (1,0), y después desde (1,0) a (2,0).

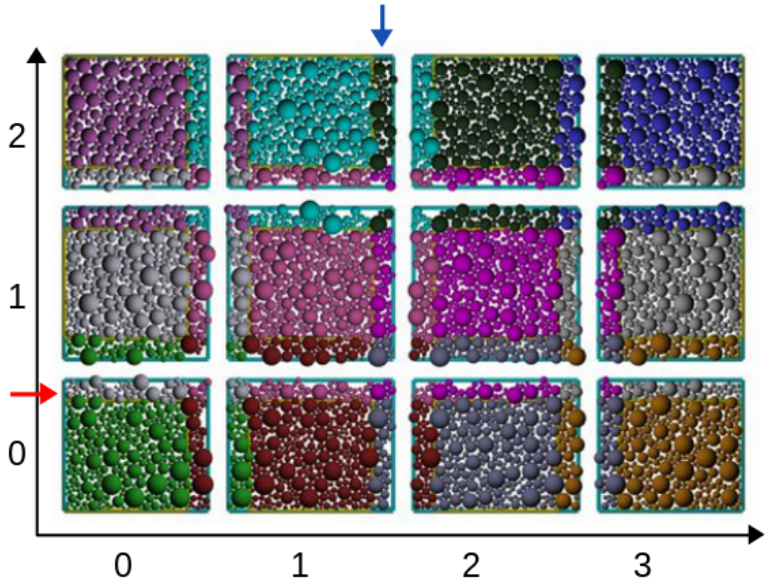


Figura 4.6: Descomposición de un dominio de 2 dimensiones

La actualización de información sobre partículas fantasma y la transferencia de partículas entre subdominios son las únicas comunicaciones interprocesos que exige una implementación paralela de DEM basada en descomposición espacial de dominio. Otras comunicaciones interprocesos pueden ocurrir debido a decisiones de diseño de los desarrolladores. La figura 4.6 permite observar gráficamente el comportamiento de ambos patrones de comunicación descritos en esta subsección. Se puede observar que queda justificado el hecho que es habitual encontrar en implementaciones paralelas de DEM algún mecanismo de comunicación ágil entre vecinos directos. En LIGGGHTS, esos patrones de comunicación están implementados en la classe `Comm`.

4.5. Patrones de comunicación interprocesos

Esta sección presenta cómo LIGGGHTS implementa la comunicación entre procesos. La sección 4.5.1 contiene una descripción general de los patrones de comunicación que LIGGGHTS implementa y las secciones 4.5.2 y 4.5.3 presentan los detalles de dos patrones de comunicación que LIGGGHTS utiliza para la comunicación interprocesos.

4.5.1. Descripción general

La comunicación interprocesos se lleva a cabo a través de instrucciones MPI. Un proceso j puede, a priori, intercambiar mensajes MPI con cualquier proceso i (es posible que $i = j$). Sin embargo, la revisión del código fuente de LIGGGHTS muestra que los procesos se comunican siguiendo patrones de comunicación basados en: (1) las características del método DEM, (2) la cercanía de los procesos y (3) las configuraciones de la simulación (por ejemplo, las condiciones de periodicidad de frontera —*periodic boundary conditions*, PBC—influyen en la comunicación interprocesos). Esta sección presenta los patrones de comunicación *regular* e *irregular*. Se observa en la figura 4.2 que en cada iteración del ciclo de integración puede resultar necesario intercambiar mensajes entre los procesos.

En este caso la comunicación ocurre durante la fase de cómputo de la simulación.

La ejecución de comandos invocados durante la fase de configuracón de la simulación o de comandos invocados durante la fase de cómputo cuyo objetivo es manejar la entrada y salida, hace que los procesos se comuniquen sin seguir un patrón de comunicación determinado. En tales casos, las implementaciones de los comandos contienen las instrucciones MPI necesarias y no invocan a los métodos de las clases que implementan los patrones de comunicación soportados. Las implementaciones de los métodos y las definiciones de las estructuras de datos que dan soporte a los patrones de comunicación regular e irregular se encuentran, respectivamente, en las clases `Comm` e `Irregular`.

En este proyecto de grado no fueron modificados métodos que intercambian mensajes entre procesos sin seguir un patrón de comunicación determinado. Esta decisión está basada en el hecho que esas comunicaciones no interfieren en la descomposición de dominio. Las modificaciones realizadas se concentran, por lo tanto, en las clases `Comm` e `Irregular`, especialmente, en los métodos que implementan los patrones de comunicación. Las clases `Comm` e `Irregular` son componentes que integran una posible implementación de una versión paralela de DEM. Es posible encontrar en la literatura otras implementaciones paralelas de DEM que utilizan patrones de comunicación diferentes [18, 10, 30].

Según los desarrolladores de LIGGGHTS, es posible utilizar solamente el patrón de comunicación irregular. Sin embargo, por cuestiones de desempeño ellos dan preferencia al patrón de comunicación regular. En la seccón 6.4 se muestra que la estrategia de descomposición de dominio propuesta e implementada en este proyecto de grado utiliza el patrón irregular siempre que se genera una nueva descomposición de dominio y los límites de las subregiones deben ser ajustados.

4.5.2. La clase `Comm`

La clase `Comm` implementa el modelo regular de comunicación interprocesos. Su principal característica es permitir que en la fase de cómputo un proceso se comunique de forma sencilla con sus vecinos directos. Se dice que un proceso p_j es un vecino directo del proceso p_i si la subregión r es adyacente a la subregión s, siendo p_i y p_j los procesos asignados, respectivamente, a r y s. En la sección 4.2 se muestra que un elemento g de la grilla G está relacionado biunívocamente a una subregión r del dominio del problema y que puede tener un máximo de seis vecinos directos. Si el elemento g está relacionado a la región r y (x_g, y_g, z_g) y (x_r, y_r, z_r) son las coordenadas que identifican, respectivamente, a g y a r, se verifican las igualdades $x_g = x_r$, $y_g = y_r$ y $z_g = z_r$. Por lo tanto, un vecino directo de g es también un elemento de G que está asociado a una subregión adyacente a r y se define al conjunto de vecinos directos de r cómo las subregiones adyacentes a r.

Los vecinos directos pueden estar localizados delante o detrás, a derecha o a izquierda, y arriba o abajo a lo largo, respectivamente, de los ejes de coordenadas x, y, y z. Una vez descompuesto el dominio, los vecinos de una subregión se mantienen constantes a lo largo de la simulación. Como consecuencia, los vecinos directos de un proceso también se mantienen constantes una vez que se realiza la descomposición del dominio del problema. Las subregiones que se encuentran en direcciones diagonales no son consideradas vecinos directos dado que no hay comunicación entre estas subregiones a través del patrón regular.

Como se comentó en la sección 3.3, se espera un desplazamiento muy pequeño de las partículas durante dos pasos de tiempo consecutivos de una simulación basada en el método DEM y de modo que puede realizarse hacia cualquier dirección. Por lo tanto, es

razonable adoptar un mecanismo de comunicación interprocesos que asume que entre los instantes $t-1$ y t una partícula puede, a lo sumo, pasar de una subregión r a una subregión s, independientemente de si r y s son adyacentes o no. En la práctica, esta suposición implica que durante la fase de cómputo un proceso j no intercambiar información con procesos l que no son sus vecinos directos. Si el desplazamiento de la partícula es hacia un vecino directo, entonces la comunicación se da directamente. Si el desplazamiento de la partícula es hacia alguna subregión en la diagonal, entonces la partícula es enviada a un vecino directo de la región de origen y ese vecino envia la partícula a la subregión destino. Si el desplazamiento de una partícula no es suficiente para transponer la frontera entre subregiones, no es necesario intercambiar información entre procesos.

Para implementar el patrón de comunicación regular, la clase `Comm` utiliza el atributo `procneigh` para almacenar una matriz de tres filas y dos columnas que contiene los rangos de los vecinos directos de un proceso j en ejecución. Se almacenan los datos de modo que en la posición $(i, 0)$ se encuentra el rango del proceso que está detrás $(i = 0)$, a la izquerda $(i = 1)$ o abajo $(i = 2)$ del proceso j, y en la posición $(i, 1)$ se encuentra el rango del proceso que está delante $(i = 0)$, a la derecha $(i = 1)$ o arriba $(i = 2)$ del proceso j. Dado que LIGGGHTS es un software implementado utilizando la biblioteca MPI, cada proceso j en ejecución tiene acceso a una instancia de la clase `Comm` y, como consecuencia, tiene asociada una instancia de la matriz `procneigh`. Si j y l son procesos tales que $j \neq l$, entonces existe un valor $k \in \{0, 1, 2\}$ que cumple $\texttt{procneigh}_j[k][0] \neq \texttt{procneigh}_l[k][0] \vee \texttt{procneigh}_j[k][1] \neq \texttt{procneigh}_l[k][1]$, siendo $\texttt{procneigh}_j$ y $\texttt{procneigh}_l$ las instancias de la matriz `procneigh` asociadas a los procesos j y l, respectivamente. Por lo tanto, si j y l son procesos diferentes, sus respectivas matrices `procneigh` difieren.

Para cargar los datos de la matriz `procneigh` el método `cart_map()` invoca tres veces a la instrucción `MPI_Cart_shift`. Esta instrucción MPI permite encontrar los vecinos de un proceso j que se encuenran a una distancia d a lo largo de un eje de coordenadas. De este modo, considerando $d = 1$ se puede utilizar la instrucción `MPI_Cart_shift` para encontrar los vecinos directos de un proceso j a lo largo de los ejes de coordenadas x, y y z. El método `cart_map()` es invocado por `set_proc_grid()` cuándo se genera la descomposición de dominio.

De acuerdo a la estructura del ciclo de integración de la figura 4.2, los métodos que realizan el intercambio de mensajes entre procesos deben ser invocados por las implementaciones del método abstracto `Integrate::run()`. Dado que en este proyecto de grado se trabaja solamente con el esquema Velocity Verlet integrator para integración de las ecuaciones de movimiento, la sección presente introduce el patrón de comunicación regular implementado en el método `Verlet::run()`. Los métodos de la classe `Comm` que implementan el intercambio de mensajes entre los procesos son `borders()`, `exchange()`, `forward_comm()` y `reverse_comm()`. Existen dos casos en que estos métodos pueden ser ejecutados: (1) el sistema está en una iteración del ciclo de integración que no requiere recalcular la lista de vecinos de las partículas y (2) el ciclo de integración requiere recalcular la lista de vecinos de las partículas. Una iteración del ciclo no puede estar en ambos casos, simultáneamente. Los métodos `borders()`, `exchange()`, `forward_comm()`, y `reverse_comm()` son parte integral de LIGGGHTS, no fueron implementados como parte de la estrategia de descomposición de dominio propuesta en este proyecto. El algoritmo 3 muestra el esquema del método `Verlet::run()`. Las líneas que contienen puntos suspensivos indican fragmentos de códigos contenidos en el método `Verlet::run()` que fueron omitidos dado que no participan en las tareas de comunicación.

Algoritmo 3 Esquema del método `Verlet::run()`

```
1: for i := 0 to (n-1) do
2:    (...)
3:    nflag := neighbor→decide();
4:    if nflag == 0 then
5:       (...)
6:       comm→forward_comm();
7:       (...)
8:    else
9:       comm→exchange();
10:      (...)
11:      comm→borders();
12:      (...)
13:   end if
14: end for
15: (...)
16: if force→newton then
17:   comm→reverse_comm();
18:   (...)
19: end if
```

El caso (1) corresponde a una situación en la que la lista (conjunto) de vecinos no necesita ser recreada y un proceso j envía a un proceso vecino i solamente las coordenadas de las partículas que están en el subdominio asignado a j y que actúan como partículas fantasma para el proceso i. El comando `forward_comm()` realiza el envío de coordenadas de partículas fantasmas a los procesos correspondientes. Una lista de vecinos es reconstruida para evitar que el desempeño de los algortimos de detección de contacto disminuya debido a que se consideran partículas que se alejaron demasiado. El método `Neighbor::decide()` determina si es necesario actualizar, en un paso de tiempo t, la lista de vecinos de las partículas. La decisión se basa en atributos de la clase `Neighbor` y en los valores de los parámetros de los comandos declarados en el archivo de entrada de la simulación. Por defecto, las listas de vecinos son reconstruidas en intervalos de 10 pasos de tiempo.

Los comandos que pueden alterar el comportamiento por defecto de la recreación de listas de vecinos son `fix insert/*`, `fix mesh/*`, `neigh_modify` y `restart`. Estos comandos permiten inserir partículas, procesar las mallas geométricas para modelado de superficies, configurar los algoritmos de creación de listas de vecinos y habilitar la creación de archivos binarios *restart file*, respectivamente. En este proyecto de grado no se realizan cambios en el comportamiento del método `decide()` y los valores por defecto son utilizados en la configuración de las simulaciones ejecutadas. La tabla 4.2 presenta una breve descripción de los comandos que permiten alterar el comportamiento por defecto de la recreación de listas de vecinos.

El caso (2) corresponde a una situación en que el contenido de la lista de vecinos es aún válido y un proceso j envía a un proceso vecino i los datos completos de las partículas que son transferidas del subdominio asignado a j al subdominio asignado a i. Los datos de las partículas salientes son eliminados de las estructuras de datos pertenecientes a j y pasan a estar almacenados solamente en las estructuras de datos del proceso i.

Comando	Descripcíon
`fix insert/*`	Comando para insertar partículas
`fix mesh/*`	Comando para procesar las mallas geométricas que modelan las superficies utilizadas en la simulación
`neighbor_modify`	Comando para configurar las propiedades de las listas de vecinos y el comportamiento de los algoritmos usados para construirlas
`restart`	Comando para la creación de los archivos *restart file*

Tabla 4.2: Comandos que permiten alterar la actualización de listas de vecinos

Los métodos `exchange()` y `borders()` son invocados para realizar las transferencias de partículas. Para transferir las partículas es necesario diferenciar entre las partículas que se encuentran sobre la frontera entre las subregiones y las que se encuentran en el interior de una subregión. La sección 4.4 explica cómo se define la partenencia de partículas a las fronteras entre subdominios. El método `exchange()` transfiere las partículas interiores y el método `borders()` transfiere las partículas que se encuentran sobre las fronteras. El procesamiento de las partículas que se encuentran sobre las fronteras incluye a las partículas fantasmas.

El algoritmo 4 muestra un esquema del bucle de iteración ejecutado en el método `exchange()`. Cada iteración corresponde a un eje de coordenadas y el orden de ejecución es x, y y z (estos ejes de coordenadas son identificados a través de los números 1, 2 y 3, respectivamente). Los métodos `borders()`, `forward_comm()` y `reverse_comm()` tienen una estructura muy similar. La estructura de control `while` en la línea 6 itera sobre todas las partículas del subdominio y la estructura de control `if-then-else` en la línea 7 determina si la partícula se desplazó hacia fuera del subdominio respecto del eje `dim`. Si la partícula debe ser enviada a otro proceso, su información es empaquetada convenientemente y agregada al *buffer* `buf_send`. Toda la información respecto de la partícula debe estar almacenada en las estructuras correspondientes en otro proceso. Esto significa que no es suficiente enviar solamente las coordenadas de la partícula. La variables `x`, `checkhi` y `checklo` almacenan, respectivamente, las coordenadas de las partículas y los extremos máximo y mínimo del subdominio respecto del eje `dim`.

La instrucción `MPI_Sendrecv` envia al proceso que está *detrás* el valor `nsend` y recibe del proceso que se encuentra *delante* el valor `nrecv1`. La estructura de control `if-then-else` en la línea 16 verifica si existen más de dos subdominios a lo largo del eje `dim`, y si este es el caso, la instrucción `MPI_Sendrecv` es invocada nuevamente para enviar el valor `nsend` al proceso que está *delante* y para recibir el valor `nrecv2` del proceso que se encuentra *detrás*. De esta forma, los vecinos directos del proceso que ejecuta `exchange()`, respecto del eje de coordenadas `dim`, conocen el total de partículas que recibirán y el proceso que ejecuta `exchange()` conoce el total de partículas que recibirá desde sus vecinos directos.

Las instrucciones `MPI_Irecv` y `MPI_Send` entre las líneas 20 y 27 envían la información completa de cada partícula saliente a los posibles vecinos directos respecto del eje `dim` (por construcción, puede haber un máximo de dos vecinos por eje de coordenadas). Dado que se utiliza una operación no bloqueante de recepción (instrucción `MPI_Irecv`), es necesario usar la instrucción `MPI_Wait` para no completar la ejecución del método `exchange()` sin haber recibido todas las partículas entrantes. Finalmente, la estructura de control `while` en la línea 29 extrae del *buffer* de recepción la información completa de cada partícula recibida.

Algoritmo 4 Esquema del método `Verlet::exchange` para eje `dim`

```
checkhi := subhi(dim);
checklo := sublo(dim);
nsend := nrecv := 0;
nrecv1 := nrecv2 := 0;
i := 0;
while i < nlocal do
   if x(i,dim) < checklo || x(i,dim) ⩾ checkhi then
      nsend + = pack_exchange(i,buf_send);
      nlocal −−;
   else
      i++;
   end if
end while
MPI_Sendrecv(nsend,1,MPI_INT,procneigh(dim,0),0,
nrecv1,1,MPI_INT,procneigh(dim,1),0,world,sts);
nrecv := nrecv1;
if procdim(dim) > 2 then
   MPI_Sendrecv(nsend,1,MPI_INT,procneigh(dim,1),0,
   nrecv2,1,MPI_INT,procneigh(dim,0),0,world,sts);
   nrecv + = nrecv2;
end if
MPI_Irecv(buf_recv,nrecv1,MPI_DOUBLE,procneigh(dim,1),0,world,rqt);
MPI_Send(buf_send,nsend,MPI_DOUBLE,procneigh(dim,0),0,world);
MPI_Wait(rqt,sts);
if procdim(dim) > 2 then
   MPI_Irecv(buf_recv,nrecv2,MPI_DOUBLE,procneigh(dim,0),0,world,rqt);
   MPI_Send(buf_send,nsend,MPI_DOUBLE,procneigh(dim,0),0,world);
   MPI_Wait(rqt,sts);
end if
m := 0;
while i < nlocal do
   m + = unpack_exchange(buf_recv);
   nlocal ++;
end while
```

Las iteraciones del algoritmo 4 siguen el orden de ejes de coordenadas x, y y z, e implementan un mecanismo de transferencia de partículas basado en el patrón de comunicación regular y tal que las partículas son enviadas unicamente a los vecinos directos, independientemente de la posición del proceso destino en relación al proceso origen. Por lo tanto, si una partícula se desplaza de un subdominio d_i a un subdominio d_k tal que d_k se encuentra en una posición diagonal respecto de d_i, la partícula será enviada primero a un dominio d_j y después (en otra iteración del método `exchange()`) desde d_j a d_k. El subdominio d_j es vecino directo de d_i respecto de una dimensión m y es vecino directo de d_k respecto de una dimensión n, siendo $m \neq n$.

Para ambos casos de invocación de métodos de comunicación mostrados en el algoritmo 3 puede ser necesario ejecutar el método `reverse_comm()`, cuyo objetivo es permitir

a un proceso j enviar a sus vecinos directos i los valores de fuerzas calculados para las partículas que están en el subdominio asignado a j. El comando `newton` permite determinar si el método `reverse_comm()` es ejecutado o no. El valor por defecto de este comando es `on` y significa que el método debe ser ejecutado. El método `reverse_comm()` implementa el cálculo de fuerzas en el choque entre partículas y entre partículas y superficies según la tercera ley de Newton.

4.5.3. La clase `Irregular`

La clase `Irregular` implementa el patrón irregular de comunicación interprocesos. Un proceso j sigue este patrón de comunicación cuando requiere intercambiar mensajes con procesos que no son sus vecinos directos. La variante LIGGGHTS-PUBLIC no usa este patrón de comunicación durante el ciclo de integración. El patrón irregular es utilizado solamente en tres situaciones: (1) los comandos `change_box` o `deform` son declarados en el archivo de entrada de la simulación, (2) el comando `fix insert/*` es declarado en el archivo de entrada de la simulación o (3) una simulación lee un archivo *restart file* y toma como punto de partida el estado en el cual se encontraba una simulación anterior.

En el caso (1) los extremos de la región donde se lleva a cabo la simulación pueden ser modificados y es necesario descomponer el dominio nuevamente. En el caso (2) las partículas fueron insertadas en una subregión e inmediatamente después deben ser ubicadas en la subregión correspondiente y, finalmente, en el caso (3) puede ser necesario utilizar el patrón irregular dado que la descomposición de dominio inicial puede no ser adecuada para una simulación que es configurada en base parámetros utilizados en otra simulación. Las nuevas dimensiones de la región son aplicadas en una fase de configuración que se encuentra entre dos fases de cómputo consecutivas. En la prática, lo que ocurre es que al término de todas las iteraciones de un ciclo de integración, y antes que empiece la primera iteración de un nuevo ciclo de integración, las dimensiones de la región son modificadas de tal modo que puede resultar necesario transferir partículas entre subregiones que no son vecinos directos.

La implementación del patrón de comunicación irregular se encuentra en el método `Irregular::migrate_atoms`. Este método itera sobre el conjunto de las partículas contenidas en la subregión r a la cuál fue asignado el proceso que ejecuta el método. La implementación está basada en un plan de comunicación y utiliza los métodos auxiliares `Irregular::coord2proc`, `Irregular::create_atoms`, `Irregular::destroy_atoms` e `Irregular::exchange_atoms`. El algortimo 5 muestra un esquema de la implementación del método `migrate_atoms`.

La estructura de control `while` de la línea 3 recorre todas las partículas contenidas en r y la estructura de control `if-then-else` de la línea 4 verifica si una partícula transpone alguna de las fronteras de r. Si la partícula debe ser transferida a otro proceso, el método `coord2proc` es invocado para determinar a cuál proceso debe ser enviada. Su información es empaquetada convenientemente y agregada al *buffer* de envío `buff_send`. Así como ocurre con la clase `Comm`, toda la información sobre la partícula debe estar almacenada en las estructuras correspondientes en el proceso destino. Por lo tanto, no es suficiente enviar solamente las coordenadas de la partícula. Las variables `x`, `subhi`, `sublo` y `nlocal` almacenan, respectivamente, las coordenadas de las partıculas, los extremos máximo y mınimo de r, y el total de partículas en r. El vector `proclist` almacena en la posición j el proceso al cuál debe ser enviada la j-ésima partícula del *buffer* `buff_send`. La estructura de control `while` de la línea 16 extrae las partículas recibidas que fueron almacenadas

Algoritmo 5 Esquema del método `migrate_atoms`

```
1: nsendatom := 0;
2: i := 0;
3: while i < nLocal do
4:    if (x(i,0) < sublo(0) || x(i,0) ⩾ subhi(0)) || (x(i,1) < sublo(1) || x(i,1) ⩾ subhi(1))
          || (x(i,2) < sublo(2) || x(i,2) ⩾ subhi(2)) then
5:       proclist(nsendatom) = coord2proc(x(i));
6:       pack_exchange(i,buf_send);
7:       nsendatom++;
8:       nlocal−−;
9:    end if
10:   i++;
11: end while
12: nrecv := create_data(nsendatom, proclist);
13: exchange_data(buf_send, buf_recv);
14: destroy_data();
15: i := 0;
16: while i < nrecv do
17:   pack_exchange(i,buf_recv);
18:   nlocal++;
19:   i++;
20: end while
```

en el *buffer* de recepción `buf_recv`. Estas partículas pasan a estar contenidas en r.

Se define el plan de comunicación como una estructura de datos auxiliar generada por el método `create_data` (línea 12) cuyo objetivo es facilitar el intercambio de mensajes. Los parámetros del método `create_data` indican cuántas partículas serán enviadas y hacia cuáles procesos deben ser enviadas, y el método retorna cuántas partículas serán recibidas. El total de partículas a recibir queda almacenado en la variable `nrecv` de la clase `Irregular` y no es parte del plan de comunicación. La generación del plan consiste en definir variables de control que son utilizadas por las instrucciones MPI invocadas por el método `exchange_data`. Las variables son configuradas con valores generados por el mismo proceso que ejecuta `migrate_atoms` o por otros procesos que ejecutan simultáneamente otras instancias de `migrate_atoms` (por ejemplo, el total de partículas que serán recibidas es enviado desde el proceso que las enviará).

Para enviar y recibir partículas, el método `exchange_data` se basa en el plan de comunicación generado. Se tiene un método que itera sobre los *buffers* de envío y recepción y que está basado en las instrucciones `MPI_Send` y `MPI_Irecv`. Se utiliza una instrucción no-bloqueante (`MPI_Irecv`) para la recepción de partículas por cuestiones de desempeño. De esta forma, un proceso puede empezar a enviar partículas apenas haya terminado de recibir partículas. Sin embargo, es necesario utilizar la instrucción `MPI_Waitall` para evitar que `exchange_data` termine su ejecución sin haber recibido todas las partículas entrantes. El algoritmo 6 muestra un esquema del método `exchange_data`.

Los atributos de control para el envío de mensajes (`length_send` y `proc_send`) y los atributos de control para la recepción de mensajes (`length_recv`, `nrecv`, `proc_recv` y `request`) almacenan valores calculados durante la creación del plan de comunicación. La tabla 4.3 presenta una breve descripcción de la función de esos atributos.

Algoritmo 6 Esquema del método `exchange_data`

```
1: offset := 0;
2: for irecv := 0 to nrecv do
3:    MPI_Irecv(buf_send(offset), length_recv(irecv), MPI_DOUBLE, proc_recv(irecv),
          0, world, request(irecv));
4:    offset + = length_recv(irecv);
5: end for
6: offset := 0;
7: for isend := 0 to nsend do
8:    MPI_Send(buf_recv(offset), length_send(isend), MPI_DOUBLE, proc_send(isend),
          0,world);
9:    offset + = length_recv(isend);
10: end for
11: if nrecv then
12:    MPI_Waitall(nrecv, request, status);
13: end if
```

Atributo	Descripcíon
`length_send`	Vector que almacena en la posición j el largo del j-ésimo mensaje a enviar
`proc_send`	Vector que almacena en la posición j el rango del proceso hacia dónde se envia el j-ésimo mensaje
`length_recv`	Vector que almacena en la posición j el largo del j-ésimo mensaje a recibir
`nrecv`	Cantidad de elementos a recibir
`proc_nrecv`	Vector que almacena en la posición j el rango del proceso desde dónde se recibe el j-ésimo mensaje
`request`	Variable de control para verificar que todos los mensajes fueron recibidos

Tabla 4.3: Atributos calculados en la generación del plan de comunicación

Otro método auxiliar invocado por `migrate_atoms` es `destroy_atoms`. Su objetivo es liberar la memoria asignada para la contrucción del plan de comunicación. El método `destroy_atoms` está implementado utilizando invocaciones sucesivas de la instrucción `free` de C++.

Capítulo 5

Trabajos relacionados

Este capítulo presenta un estudio del estado del arte respecto del método DEM y de su aplicabilidad al estudio de dinámica de medios granulares. Se reseñan trabajos de relevancia internacional que fueron publicados en la última década. Cada sección contiene un resumen de los principales resultados obtenidos en los diferentes trabajos resenñados y la seccíon final presenta conclusiones generales sobre esos trabajos y se establecen relaciones con este proyecto de grado.

5.1. Introduccón a DEM y a las bases para una implemetación paralela

Ferrez y Liebling [10] presentaron una introducción sucinta y precisa de DEM y de las adaptaciones necesarias para implementar eficientemente una versión paralela del método adecuada para ejecutar simulaciones de dinámica de medios granulares. Este artículo discute los desafíos computacionales fundamentales del método DEM: (1) detección de puntos contacto entre partículas, y entre partículas y superficies y (2) el uso de un modelo adecuado para la representación de los contactos detectados. Se presentan claramente las razones por las cuáles los desafíos computacionales de DEM demandan gran capacidad de cómputo.

Los experimentos que los autores realizaron permitieron: (1) realizar estudios de eficiencia computacional con el objetivo de mostrar la factibilidad del uso de técnicas de computación paralela para implementar el algoritmo del método DEM y (2) ejemplificar el uso de DEM en simulaciones de medios granulares utilizando problemas encontrados en Ingeniería Civil.

Para los estudios de eficiencia computacional los autores utilizaron dos versiones diferentes de DEM: (1) una versión serial implementada por Müller [89] y (2) una versión paralela implementada por los autores aplicando técnicas de paralelismo utilizando memoria compartida. Las ejecuciones de ambos algoritmos fueron realizadas en un PC equipado con cuatro procesadores Pentium III Xeon con arquitectura UMA (*Uniform Memory Access*) e incluyeron ejecuciones utilizando dos y cuatro procesadores. Los resultados obtenidos mostraron un speedup de hasta 2,3 correspondiente al uso de cuatro procesadores. El valor de speedup fue calculado utilizando la ecuación 5.1, siendo t_S y t_P

los tiempos de ejecución de los algoritmos DEM secuencial y paralelo, respectivamente:

$$S = \frac{t_S}{t_P} \tag{5.1}$$

Los autores destacaron que es posible obtener mejoras de desempeño independientemente del uso de técnicas de computación paralela. Por ejemplo, un algoritmo que use la técnica *Dynamic Weighted Delaunay Triangulation* (DWDT) [78] requiere del orden de $O(N)$ operaciones para detectar los puntos de contactos entre partículas, y entre partículas y superficies, siendo N la cantidad de partículas en el sistema. Por otra parte, una técnica de fuerza bruta requiere del orden de $O(N^2)$ operaciones. La implementación paralela de DEM propuesta por Ferrez y Liebling utilizó la técnica DWDT para la detección de contactos.

Para ejemplificar el uso de DEM en simulaciones de dinámica granular los autores consideraron dos problemas: (1) colisión de rocas contra un terraplén y (2) empaquetado (*packing*) de partículas de polvo en recipientes cilíndricos. Según los autores, estos problemas corresponden a líneas de investigación en las cuáles trabajan investigadores del Departamento de Ingeniería Civil del École Polytechnic Fédérale de Lausanne (EPFL) de Suiza y fueron propuestos por miembros de ese departamento. Las simulaciones en computadora correspondientes a estos problemas fueron ejecutadas en un PC equipado con cuatro procesadores Pentium III Xeon con arquitectura UMA utilizando la versión paralela del algoritmo DEM. Sin embargo, los autores no reportaron el número de procesadores utilizados y los tiempos de ejecución obtenidos.

Los autores no realizaron estudios más avanzados sobre el comportamiento del algoritmo paralelo que implementaron. No se informó, por ejemplo, la distribución de cargas de trabajo entre los procesadores para simulaciones con múltiples procesadores o diferencias de desempeño del algoritmo paralelo según las técnicas de programación paralela utilizadas.

El trabajo de Ferrez y Liebling presentó las virtudes del método DEM para la simulación de dinámica de medios granulares. Estre trabajo logró exponer claramente formas de explorar las ventajas de la computación paralela para potenciar las virtudes de DEM y posibilitar la aplicación de este método en simulaciones de escala real.

5.2. Algoritmo de descomposición de dominio dinámica basado en planos móviles

Markauskas, Kačeniauskas y Maknickas [30] introdujeron un algoritmo de descomposición dinámica de dominio basado en planos móviles (*moving planes*). Los investigadores plantearon las bases teóricas del algoritmo y explicaron su funcionamiento. Una implementación del método de planos móviles fue incorporada a un simulador DEM desarrollado por los autores y utilizado en la etapa de análisis experimental.

El objetivo del algoritmo de planos móviles es distribuir de forma equitativa la carga de trabajo total entre varios procesadores. Los autores propusieron dos maneras de definir la carga de trabajo de un procesador p: (1) el total de partículas asignadas a p divido por el tiempo necesario para ejecutar una iteración del método DEM y (2) el número total de partículas asignadas a p. Los autores utilizaron la primera definición de carga de trabajo para evitar que procesadores relativamente más lentos fueran sobrecargados con un número elevado de partículas.

Un nivel de tolerancia es utilizado para determinar el momento en que el algoritmo de distribución de carga de trabajo debe ser ejecutado. Al final de un ciclo de DEM se verifican las cargas de trabajo de cada procesador y en caso que algunomde ellos esté sobrecargado (la carga de trabajo relativa es superior al nivel de tolerancia definido), el algoritmo de planos móviles es invocado. El algoritmo de planos móviles permite ajustar el volumen de cada subdominio de modo que la carga de trabajo de cada procesador no sea superior al nivel de tolerancia definido. El modelo de descomposición de dominio que los autores utilizaron asigna cada subdominio a un procesador y el ajuste de volúmenes se hace de forma iterativa.

Los experimentos numéricos fueron realizados en el cluster VILKAS de la Universidad Técnica Gediminas (Vilnius, Lituania) estudiando dos instancias de problemas de descarga de silos. El cluster está compuesto por 20 PCs de escritorio conectadas a través de una red Ethernet Gigabit, y es parte de la iniciativa BalticGrid-II, un proyecto para fomentar el uso de tecnología de cómputo distribuido en los Países Bálticos. Los PCs están equipados con procesador Intel Core2Quad modelo Q6600 de cuatro núcleos de 2,40GHz y 4GB de memoria RAM.

Una instancia contiene 100000 partículas y la otra contiene 300000 partículas, y en ambas instancias los investigadores utilizaron un silo afunilado de base rectangular cuyos lados miden 200mm y 40mm. El orificio de salida es un cuadrado de dimensiones 40×40mm que se encuentra en el centro de la base de la estructura y que está conectado a las paredes a través de una plano inclinado de 45°. Respecto al método de descomposición de dominio utilizado y la cantidad de procesos por nodo, fueron propuestos cuatro escenarios: (a) descomposición estática y dos procesos por nodo, (b) descomposición estática y cuatro procesos por nodo, (c) descomposición dinámica y dos procesos por nodo, y (d) descomposición dinámica y cuatro procesos por nodo.

Para una instancia y un escenario determinados fueron realizadas 10 simulaciones y el tiempo de procesamiento resportado es el promedio de la suma del tiempo de procesamiento de las 10 simulaciones realizadas. Los resultados obtenidos para ambas instancias y los cuatro escenarios permitieron concluir que el método de planos móviles es adecuado para reducir el tiempo de procesamiento necesario para simulaciones DEM aplicadas a la descarga de silos. Un speedup de apróximadamente $35\times$ fue obtenido para la instancia de 300000 partículas, descomposición dinámica de dominio, cuatro procesos por nodo y 48 procesadores (eficiencia computacional de 72 %). Un speedup de apróximadamente $12\times$ fue obtenido para la instancia de 100000 partículas, descomposición dinámica de dominio, dos procesos por nodo y 16 procesadores (eficiencia computacional de 75 %).

Los autores concluyeron que el procesador Intel Core2Quad tiene peor desempeño en aplicaciones de uso intensivo de memoria RAM cuando los cuatro núcleos de un nodo son utilizados. Los resultados reportados mostraron que las simulaciones que utilizaron 2 núcleos por nodo tuvieron un speed aproximadamente 10 % superior al speedup observado en la simulaciones que utilizaron 4 núcleos por nodo. Fueron reportados resultados similares respecto de este fenómeno para ambas estrategias de descomposición de dominio y los autores afirmaron que aunque se observara este comportamiento en los resultados de la simulaciones debido la plataforma de hardware utilizada, los resultados que reportaron son similares a los que ellos encontraron en la literatura.

Los autores concluyeron su artículo presentando un análisis de la distribución de carga de trabajo entre los procesadores según avanzaban las simulaciones. Fueron reportados

resultados de distribución de carga de trabajo para las dos definiciones propuestas y se observó que al inicio de una simulación existía un desbalanceo de carga muy acentuado. Según la simulación avanzaba en el tiempo, el algoritmo de descomposición dinánima de dominio implementado era invocado y las distribuciones de carga de trabajo alcanzaban un punto de equilibrio a partir del cuál no había diferencias relativas de cargas de trabajo superiores al nivel de tolerancia definido.

Markauskas, Kačeniauskas y Maknickas corroboraron que el método de planos móviles es adecuado para mejorar el desempeño de simulaciones DEM.

5.3. Implementación de un modelo híbrido de paralelismo para el software LIGGGHTS

Berger et al. [19] presentaron una implementación de un modelo híbrido de paralelismo para el paquete de software LIGGGHTS [22] utilizando MPI (memoria distribuida) y OpenMP (memoria compartida). La idea consistió en agregar un segundo nivel de paralelismo utilizando OpenMP cuyos objetivos son: (1) aumentar el nivel de paralelismo de LIGGGHTS y (2) proporicionar un mecanismo de balanceo dinámico de carga. El modelo híbrido de paralelismo propuesto posee dos componentes principales: (1) un componente utilizando la biblioteca MPI responsable por la descomposición de dominio y (2) un componente utilizando la biblioteca MPI y las directivas de OpenMP responsable por los hilos de ejecución creados para el procesamiento paralelo de las operaciones sobre las partículas contenidas en un subdominio. El componente (1) mantuvo el código original de LIGGGHTS y no sufrió modificaciones.

Según los autores, OpenMP permite paralelizar operaciones ejecutadas independientemente sin demasiado esfuerzo. Berger et al. relevaron trabajos de investigadores que lograron paralelizar algunas secciones de código de versiones seriales del algoritmo de DEM aplicando únicamente las directivas de compilación de OpenMP [12, 21]. En el caso de DEM algunos cálculos pueden ser interdependientes y es necesario tener especial cuidado al momento de utilizar las directivas que OpenMP ofrece. Una situación de este tipo ocurre, por ejemplo, en el cálculo de la fuerza resultante ejercida mutuamente por pares de partículas. Los autores implementaron las ecuaciones físico-matemáticas de DEM considerando aspectos que permiten optimizar las operaciones. Por ejemplo, para reducir la cantidad de operaciones realizadas en la etapa de cálculo de fuerzas resultantes y aumentar el nivel de paralelismo se implementó la componente (2) basándose en la tercera ley de Newton. Para mantener la correctitud de los resultados numéricos, los autores adoptaron un enfoque en el cuál cada hilo de ejecución almacena los valores de fuerza resultante calculados en áreas de memoria exclusiva (*array forces*) y una operación de sincronización (*array reduction*) consolida los datos.

Los autores describieron el funcionamiento de la estrategia de descomposición dinámica de dominio disponible en LIGGGHTS y el modo cómo adaptaron los algoritmos de cálculo numérico utilizando OpenMP. Sin embargo, no reportaron la variante de LIGGGHTS que utilizaron en su trabajo. LIGGGHTS está disponible en tres variantes que se diferencian según las funcionalidades ofrecidas al usuario: (1) PUBLIC, (2) PREMIUM y (3) CONSORTIUM. Las variantes PREMIUM y CONSORTIUM ofrecen descomposición dinámica de dominio. Independientemente de la estrategia de descomposición de dominio utilizada, LIGGGHTS asigna biunívocamente un proceso MPI a cada subdominio generado. La principal modificación realizada por los autores consistió

en cambiar la forma en que opera el proceso MPI asignado a un subdominio. La versión modificada generó n particiones del subdominio correspondiente, creó n hilos de ejecución de OpenMP y los asignó biunívocamente a cada partición. La función de los hilos de OpenMP fue procesar el movimiento e interacción de las partículas que se encuentran en las partición del subdominio. Los autores eligieron el algoritmo *Recursive Coordinate Bisection* (RCB) de Berger y Bhokhari [31] para la generación de las particiones. La estrategia que LIGGGHTS adopta para la asignación de procesos MPI a subdominios y la estrategia de descomposición dinámica de dominio reportadas no fueron modificadas.

Berger et al. no modificaron el algoritmo iterativo con el cuál LIGGGHTS implementa el método DEM. Al principio de cada iteración de este algoritmo se ejecuta el primer componente de la implementación que los autores propusieron y las dimensiones de los subdominios son ajustadas tomando en consideración el número de partículas en cada subdominio. Posteriormente, se ejecuta el segundo componente de la solución propuesta y ocurre la generación de las particiones de los subdominios y la creación de los hilos OpenMP. Finalmente, ocurre la asignación de hilos de ejecución a particiones. La estrategia de descomposición dinámica de dominio reportada por los autores es similar a la estrategia de planos móviles propuesta por Markauskas et al. [30].

Los cálculos de detección de contacto e interacción partícula-partícula y partícula-superficie fueron implementados utilizando hilos de procesamiento de OpenMP. Los autores estudiaron cómo ciertas características intrínsecas de una simulación DEM favorecen el uso de técnicas de HPC basadas en memoria compartida. La técnica *Partitioned Spatial Sorting* (PSS) fue utilizada para posibilitar que los datos de partículas que se encontraban en la misma partición ocuparan regiones contiguas de memoria. De esta manera los autores garantizaron que el principio de localidad espacial de datos en memoria se cumpliera y el desempeño global del modelo híbrido no fuera perjudicado debido a un posible bajo desempeño de memoria caché (por ejemplo, tasas altas de *caché miss* pueden tener un impacto negativo en los resultados).

Dos casos de estudio fueron utilizados para comparar el desempeño del modelo híbrido y del modelo de paralelismo original de LIGGGHTS. Los autores informaron que los resultados numéricos obtenidos no cambiaron al modificar el modelo de paralelismo utilizado y reportaron los valores de eficiencia computacional obtenidos cuando compararon los tiempos de procesamiento para ambos modelos. Las simulaciones fueron ejecutadas en computadores equipados con dos procesadores AMD Opteron modelo 6272 de 16 núcleos por procesador y utilizaron hasta un máximo de cuatro computadores (128 núcleos). Los computadores fueron conectados a través de una red Infiniband QDR de 40Gb/s.

Berger et al. observaron que en la etapa de experimentación es necesario realizar pruebas que cubran un conjunto amplio de puntos de corte a lo largo de los ejes de coordenadas cartesianas. Los puntos de corte son definidos por el usuario de LIGGGHTS y son utilizados como entrada para la asignacíon de procesos MPI a los subdominios. Si P_x, P_y y P_z son las cantidades de puntos de corte a lo largo de los ejes x, y y z, respectivamente, la cantidad total de procesos debe ser igual a $P_x \times P_y \times P_z$. Fueron considerados, por ejemplo, casos de estudio en los cuáles $P_x = P_y = 4$ y $P_z = 2$. Esta configuración de puntos de corte dividió el dominio en 32 subdominios, utilizó 32 procesos MPI e determinó cuatro divisiones del eje x, cuatro divisiones del eje y y dos divisiones del eje z. Los autores reportaron que dados un caso de estudio y un número de procesos, al variar los valores de P_x, P_y y P_z fue posible verificar cambios considerables en el desempeño de LIGGGHTS.

El primer caso de estudio consistió en utilizar 50000 partículas esféricas de 5,0 mm de diámetro para llenar un recipiente cúbico de dimensiones 30×30×30 cm que se encontraba dentro de un dominio cúbico de dimensiones 50×50×50 cm. Las partículas son insertadas en la región superior del dominio en un movimiento de caída libre y la cantidad de partículas utilizadas fue suficiente para que el recipiente se llenara completamente y que las partículas excedentes cayeran por las laterales del recipiente. El segundo caso de prueba fue basado en un experimento a escala real e involucró un silo contieniendo $1{,}5 \times 10^6$ partículas esféricas de 1,4mm de diámetro. Se utilizó un modelo de silo afunilado con 40cm de altura, 25cm de radio en la mitad superior y 4cm de radio en la extremidad interior. Al final de la etapa de llenado el orificio de salida fue abierto y se procedió a la descarga del silo. Según los autores, las configuraciones de los casos de pruebas tenían el objetivo de generar desbalanceo de carga entre los subdominios.

Los resultados de desempeño reportados para el primer caso de estudio mostraron que el modelo híbrido logró reducir el tiempo de ejecución en un 58 % y para el segundo caso de estudio mostraron que el modelo híbrido logró reducir el tiempo de ejecución en un 44 %. Los resultados del estudio de balanceo de carga mostraron que el método híbrido logró mantener los niveles de desbalanceo de carga inferiores al 22 % para simulaciones que utilizaron 128 núcleos. El modelo original de LIGGGHTS logró mantener los niveles de desbalanceo de carga inferiores al 60 % para simulaciones que utilizaron 128 núcleos. Estos fueron los niveles máximos de desbalanceo de carga que fueron reportados. Simulaciones que utilizaron menos que 128 núcleos presentaron niveles menores.

Los valores speedup de ambos modelos de paralelismo mostraron que para hasta 32 núcleos el modelo original de LIGGGHTS tuvo mayor valor de speedup y cuando se utilizaron más que 32 núcleos, el modelo híbrido tuvo mayor valor de speedup. Los autores informaron un valor de speedup de 50 (eficiencia computacional de 39 %) para simulaciones que usaron 128 núcleos y el modelo híbrido de paralelismo. El valor de speedup reportado para simulaciones que usaron 128 núcleos y el modelo original de paralelismo fue de 30 (eficiencia computacional de 23 %).

Berget et al. informaron resultados de estudios de escabilidad y balanceo de carga solamente para el el primero caso de estudio. Los autores concluyeron que el método híbrido de paralelismo fue adecuado para manejar situaciones de desbalanceo de carga pronunciado y que el método propuesto pudo aumentar el desempeño de LIGGGHTS.

5.4. Algoritmo para simulaciones DEM de medios granulares polidispersos

Kačianauskas et al. [24] presentaron un algoritmo paralelo para simulaciones DEM de medios granulares polidispersos. En estos entornos la razón entre el mayor y menor diámetro de partículas es mayor que 1. Esta relación se verifica cuando el sistema contiene partículas de diferentes radios y puede ser expresada según la ecuación 5.2. En la literatura esta relación es conocida como fator de heterogeneidad.

$$\alpha = \frac{d_{max}}{d_{min}} > 1 \tag{5.2}$$

El algoritmo que los autores propusieron utilizó la biblioteca MPI para la comunicación entre procesos y aplicó una estrategia de descomposición estática de dominio. La

implementación se realizó en lenguaje FORTRAN 90 y correspondió a una extensión de un paquete de software DEM secuencial desarrollado previamente por los autores [23].

Kačianauskas et al. platearon una división del dominio en subdominios de igual volumen. Cada subdominio fue asignado a un único procesador para que las fuerzas actuantes sobre las partículas correspondientes fueran calculadas. El procesador que maneja un subdominio también fue responsable de mantener actualizada la información sobre el posicionamiento de las partículas. Dado un subdominio s, puede resultar necesario disponer de información sobre partículas que se encuentran en subdominios contiguos. Esta información es requerida para operar con las partículas contenidas en s.

Dos características importantes del algoritmo propuesto son: (1) utiliza el algoritmo de listas de celdas encadenadas (*Linked-Cell Lists*) para la detección eficiente de contactos entre partículas y (2) utiliza el esquema predictor-corrector Gear de 5^{to} Orden para integración de ecuaciones diferenciales con mayor nivel de precisión. Los autores eligieron este esquema de integración basándose en los resultados obtenidos por otros investigadores en un estudio comparativo sobre métodos numéricos utilizados habitual en DEM [90].

Tres problemas de prueba fueron considerados para estudiar mejoras de desempeño y determinar valores de speedup. Las simulaciones correspondientes fueron ejecutadas en el cluster VILKAS de la Universidad Técnica Gediminas (Vilnius, Lituania). Los tres problemas abordados consistieron en la compactación de un volumen en forma de paralelepípedo de dimensiones 160×40×40 mm rellenado de partículas. El primer problema de prueba tiene factor de heterogeneidad $\alpha_1 = 4{,}3$ y está compuesto por dos subcasos: (a) un subcaso que contiene 19890 partículas cuyos radios varían entre 1,031 mm y 4,466 mm y (b) otro subcaso que contiene 100037 cuyos radios varían entre 0,603 mm y 2,613 mm. El segundo problema de prueba tiene factor de heterogeneidad $\alpha_2 = 9{,}0$ y contiene 19890 partículas cuyos radios varían entre 0,55 mm y 4,75 mm. El tercer problema de prueba estudiado está formado por dos subcasos de partículas monodispersas (factor de heterogeneidad $\alpha_3 = 1{,}0$) que contienen 19890 partículas de radio 2,22 mm y 100037 partículas de radio 1,33 mm, respectivamente.

Los resultados reportados mostraron valores de speedup entre 7 y 9 al utilizar 10 procesadores. Los resultados de desempeño computacional mostraron que los casos de prueban que contenían partículas polidispersas necesitaron una capacidad de cómputo considerablemente mayor que los casos de prueba que contenían partículas monodispersas (los autores reportaron un aumento de capacidad de cómputo requeridad de alrededor de 345 %). Según los autores, el aumento de capacidad de cómputo requerida ocurrió debido a la forma en que es necesario dividir el dominio del problema. Los subdominios deben ser de forma cúbica y tales que tengan un valor de largo de lado ligeramente superior al diámetro de la partícula más grande. De esta manera el procesamiento realizado por el algoritmo de detección de contacto aumentó de forma pronunciada.

Las simulaciones realizadas utilizaron un máximo de 10 procesadores y los autores afirmaron que este valor fue elegido en base a las características del dominio del problema y de modo a minimizar la cantidad de comunicaciones entre procesos y mantener niveles adecuados de balanceo de carga. La descomposición de dominio fue generada anticipadamente, antes de iniciar las simulaciones y el número de procesadores corresponde al total de subdominios creados. Los autores no realizaron un estudio de balanceo de carga porqué, según ellos, en los problemas de prueba considerados las cantidades de partículas en cada subdominio se mantuvieron casi constantes a lo largo del tiempo.

Kačianauskas et al. lograron confirmar la eficacia del método DEM para simulaciones con medios granulares. Los autores mostaron que este método posee característica intrínsecas que pueden ser exploradas en el desarrollo de un algoritmo paralelo y presentaron resultados importantes que comprobaron la influencia que tiene el nivel de heterogeneidad de las partículas en la capacidad de cómputo requerida.

5.5. Algoritmo paralelo para DEM basado en OpenMP – caso de estudio con GPUs y CPUs

Shigeto y Sakai [25] propusieron un algoritmo paralelo para simulaciones DEM utilizando OpenMP adecuado para procesadores multinúcleos. Para corroborar las cualidades del algoritmo propuesto se realizaron simulaciones de gran escala de un sistema de transporte y mezcla de partículas de polvo. Los autores compararon los resultados obtenidos en experimentos ejecutados en GPUs y ejecutados en CPUs, y mostraron la aplicabilidad de ambos tipos de unidades de procesamiento para simulaciones DEM.

El algoritmo porpuesto por los autores incorporó la técnica de listas de celdas encadenadas (*Linked-Cell Lists*) para representar adecuadamente las colisiones entre partículas y entre partículas y superficies con el objetivo de ejecutar simulaciones con gran número de partículas sin que haya un uso poco eficiente de memoria. El uso de esta técnica logró reducir la cantidad de memoria requerida en hasta un 92 % respecto de la cantidad de memoria requerida para almacenar las estructuras de datos utilizadas en un método de detección de colisiones utilizando divisiones del espacio [91]. Para reducir la cantidad de cálculos que son realizados al momento de calcular la fuerza resultante que actúa sobre una partícula, los autores utilizaron el concepto *Reaction Calculation Method*. De esta forma se incorporan al algoritmo las ecuaciones de la Tercera Ley de Newton.

Fueron implementadas dos versiones del algoritmo paralelo, una optimizada para ejecuciones en GPUs y otra optimizada para CPUs. Cada versión posee dos subversiones que se diferencian en la representación del tipo de dato *número real*. Las representaciones utilizadas son: (1) número de punto flotante de simple precisión (*float*) y (2) número de punto flotante de doble precisión (*double*). Al momento de realizar los experimentos cada una de las versiones fue ejecutada en el hardware apropiado. Las simulaciones fueron ejecutadas en un computador equipado con una GPU modelo Tesla C1060 y dos procesadores Intel Xeon W5590 de 3,33GHz, y 24GB de memoria RAM.

Los experimentos de prueba que los autores prepararon consistieron en simular un un sistema de transporte y mezcla de partículas de polvo compuesto por un cilindro de 4m de largo y 1m de diámetro conteniendo en su interior un mecanismo de hélices dobles que giran simultáneamente. El cilindro fue llenado con partículas de 2 mm de diámetro y los experimentos generados se diferenciaban según la cantidad de partículas utilizadas. Los valores varían entre 10000 y 1280000 partículas. Cada experimento fue ejecutado utilizando ambas versiones del algoritmo implementado y en ambos hardwares disponibles.

Los resultados de los experimentos mostraron que para la representación float y uso de GPUs los tiempos de ejecución fueron hasta 3,4 veces menores respecto a los tiempos de ejecución obtenidos para el mismo tipo de representación y uso de CPUs. Para la representación double, los autores mostraron que para un número relativamente pequeño de partículas (hasta 640000 partículas) las simulaciones ejecutadas en GPUs lograron ser hasta 3 veces más rápidas. Simulaciones con representación doble y una cantidad mayor

de partículas tuvieron menor tiempo de ejecución cuando se utilizan CPUs. Hubo una reducción del tiempo de ejecución de hasta un 13 %. Los autores no reportaron resultados respecto del balanceo de carga.

Según los autores, la diferencia en los resultados de acuerdo a la representación de números reales utilizada fue debido a la capacidad que posee una GPU para procesamiento de datos de tipo double. Esta propiedad de las GPUs favoreció el uso de este tipo de unidades de procesamiento para simulaciones con un total de partículas relativamente inferior. Cuando fue necesario considerar un número superior de partículas, los autores mostraron que es conveniente utilizar CPUs.

Shiget y Sakai concluyeron que su algoritmo es apropiado para la ejecución eficiente de simulaciones DEM y mostraron que los resultados obtenidos en diferentes plataformas de hardware fueron similares, aunque según la cantidad de partículas consideradas, determinado tipo de hardware puede ofrecer mejor desempeño. La diferencia en los tiempos de ejecución favorable al uso de CPUs en simulaciones que contiene gran número de partículas permitió concluir que este tipo de procesador puede ser usado satisfactoriamente en simulaciones DEM de gran escala.

5.6. Algoritmo paralelo basado en MPI para acoplar el método DEM y técnicas de CFD

Gopalakrishnan y Tafti [18] presentaron un algoritmo paralelo para DEM utilizando MPI que pudo ser acoplado a técnicas de CFD con el objetivo de sustituir el algoritmo secuencial para DEM de la aplicación MFIX (*Multiphase Flow with Interphase eXchanges*) [29]. MFIX es un sistema de código abierto con soporte al enfoque de métodos acoplados CFD/DEM utilizado ampliamente para simulaciones de hidrodinámica, transferencia de calor y reacciones químicas que ocurren en lechos fluidizados. Este software es desarrollado por el National Energy Technology Laboratory (NETL) de Estados Unidos.

Gopalakrishnan y Tafti propusieron un enfoque basado en computación paralela que permitió mantener el método acoplado y soportar simulaciones con un gran número de partículas. El algoritmo desarrollado utilizó descomposición estática de dominio y siguió la propuesta de Cundall y Strack [4] para el método DEM, utilizando el modelo de esfera blanda para partículas y el modelo de contacto LSD para interacciones partícula-partícula y partícula-superficies.

Los investigadores probaron su algoritmo utilizando dos sistemas de lecho fluidizado. El primer sistema estaba compuesto por un recipiente de dimensiones 44×10 mm, 9240 partículas de 1,2 mm de diámetro y un lecho fluidizado de 36 mm de altura. El segundo sistema estaba compuesto por un recipiente de dimensiones 640×640 mm, partículas de 4 mm de diámetro y un lecho fluidizado de 400 mm de altura. El segundo sistema dio origen a tres problemas de prueba que se diferenciaban según el número de partículas: (a) 2,5 millones de partículas, (b) 5 millones de partículas, y (c) 10 millones de partículas. Los recipientes en ambos sistemas fueron rellenados con un fluido de densidad 1,2 kg/m^3 y viscosidad $1{,}8\times10^{-5}$. Los autores no informaron la altura de los recipientes, pero según la descripción de la metodología de experimentación utilizada, se observó que la altura de cada recipiente fue suficiente para contener todo el volumen del fluido.

Los principales objetivos de los experimentos propuestos por los autores fueron: (1) corroborar la escalabilidad del nuevo algoritmo para DEM respecto de la cantidad de procesadores y del tamaño del sistema simulado y (2) verificar la correctitud numérica

de los resultados obtenidos cuando se utilizó el nuevo algoritmo. El primer sistema fue construido a partir de especificaciones que los autores encontraron en trabajos de Müller et al. [92, 93]. Las configuraciones del segundo sistema y de los problemas derivados fueron definidas por los mismos autores.

Para verificar la correctitud numérica de los resultados obtenidos con el nuevo algoritmo para DEM, los autores compararon sus resultados con los resultados obtenidos por Müller et al. [92, 93] para el problema correspondiente al primer sistema. Los resultados que los autores obtuvieron mostraron que al paralelizar el algoritmo secuencial, MFIX siguió calculando resultados numéricamente correctos acordes a la teoría en la cuál se basan las técnicas de CFD y el método DEM.

Para realizar estudios de escalabilidad Gopalakrishnan y Tafti utilizaron las herramientas Tuning and Analysis Utilities (TAU) [94] y Program Database Toolkit (PDT) [95] para el *profiling* y la instrumentation del código implementado. Se dio énfasis al algoritmo que resuelve la etapa DEM del problema ya que el algoritmo para la etapa CFD ya se encontraba paralelizado. Las simulaciones en computador ejecutadas para las pruebas de escalibilidad utilizaron los tres problemas creados a partir del segundo sistema de lecho fluidizado.

Los resultados de escalabilidad mostraron que el speedup es casi lineal para hasta 64 procesadores, y para 256 procesadores el speedup fue de 208 y una eficiencia computacional del 81 % para simulaciones con 2,5 millones de partículas. Los autores mostraron que el aumento del número de procesadores hizo que la comunicación entre procesos ocupara una parcela mayor del tiempo total de simulación. Cuando se utilizaron 16 procesadores, 15 % del tiempo total de cómputo correspondió a comunicación entre procesos, mientras que para 256 procesadores su valor subó a 35 % para simulaciones con 2,5 millones de partículas. Los resultados reportados para los problemas con 5 y 10 millones de partículas incluyeron la comparación entre los tiempos de ejecución necesarios en ambos casos según la cantidad de procesadores utilizados. Los autores mostraron que el crecimiento del tiempo de ejecución es casi lineal respecto del número de partículas y de la cantidad de procesadores utilizados.

Gopalakrishnan y Tafti observaron que la descomposición estática de dominio generó desbalanceo de cargas entre los procesadores. No fueron reportados resultados numéricos respecto de los niveles de balanceo de carga, pero según los autores, se pudo inferir el desbalanceo de carga entre procesadores observando el incremento del tiempo de comunicación entre procesos necesario según el número de partículas aumentaba.

Los autores lograron mostrar las fortalezas del método DEM respecto de la escalabilidad del número de procesadores y del tamaño del sistema simulado, y su versatilidad respecto de la conjunción con otras herramientas computacionales. El enfoque adoptado por los autores para la implementación del algoritmo permitió utilizar DEM como una herramienta de trabajo capaz de abordar eficientemente problemas de dinámica de medios granulares a escala real.

5.7. Estudio de patrones de flujo de medios granulares en la etapa de descarga de un silo

Chen et al. [34] presentaron un estudio experimental para cuantificar de forma confiable los patrones de flujo de partículas durante la descarga de un silo a escala real. Según los autores, el procedimiento de descarga tiene impacto directo en decisiones de

diseño y en aspectos funcionales y modo de operación del silo. Existe fuerte evidencia en la literatura que los patrones de flujo están sujetos a las diferencias de escala y los resultados obtenidos a escala reducida no pueden ser extrapolados con la intención de considerar sistemas más grandes [5, 38]. Los autores destacaron las diferencias existentes al momento de trabajar con silos a escala real y con modelos a escala reducida y observaron que las técnicas aplicadas en experimentos con silos a escala reducida son de poca utilidad en los experimentos con silos a escala real.

Los autores comentaron trabajos realizados por otros investigadores que ofrecieron resultados parciales respecto de los patrones de flujo de partículas. En los trabajos comentados por Chen et al. no se realizaron mediciones rigurosas de evidencia que mostrara el comportamiento de las partículas dentro del silo. Algunos métodos de verificación empleados por otros investigadores consistían, por ejemplo, en observar el desgaste de las caras interiores de las paredes del silo [96]. Otros trabajos relevados por Chen et al. dedujeron el patrón de flujo a partir de mediciones de presión en puntos determinados del silo [97] o posicionando varillas en puntos específicos de las paredes del silo y observando su movimiento durante la descarga del material granular almacenado [98]. Según los autores, los métodos que fueron aplicados en los trabajos comentados indicaron solamente la presencia de movimiento de partículas en regiones específicas del silo, sin proporcionar información sobre el canal de flujo que se formó cuando las partículas salieron de la estructura de almacenamiento.

Para realizar los experimentos de forma rigurosa, los autores construyeron un silo cilíndrico a escala real y especificaron claramente las características físicas de esta estructura y del material granular utilizado (mineral de hierro). El silo considerado tenía 4,2 m de diámetro y la sección de almacenamiento tenía 9,5 m de altura. La altura total era de aproximadamente 12 m y el silo contaba con tres orificios de descarga ubicados al fondo de la estructura y alineados sobre una misma recta. Un orificio se encontraba en el centro de la circunferencia que sirve de base para la estructura (orificio de salida (1)), otro orificio (orificio de salida (2)) se encuentra en el extremo radial opuesto y el tercer orificio (orificio de salida (3)) se encuentra en una posición intermedia entre los orificios (1) y (2). Todos los orificios tenían 0,48 m de diámetro.

Los experimentos pusieron especial énfasis en los métodos de carga y descarga, y en la especificación de un método que permitiera conocer el volumen de material almacenado dentro del silo en cualquier momento. Los autores se basaron en el método propuesto por Chen [35] para realizar las lecturas de volumen en tiempo real. No fueron realizadas simulaciones en computador y todos los experimentos utilizaron el silo contruido.

La obtención de datos durante la operación del silo se realizó utilizando 288 radiotransmisores insertados junto con el material granular. Los radiotransmisores tenían dimensiones similares a las partículas almacenadas (forma elipsoidal, diámetro medio 12,9 mm) y fueron distribuidos en ocho niveles de diferentes alturas tal que cada nivel tenía aproximadamente la misma cantidad de radiotransmisores. Cada radiotransmisor era identificado unívocamente y se registraba el momento de salida cuando pasaba por un orificio de descarga. La disposición de los radiotransmisores mostró la intención deliberada de los autores de simular un procedimiento de carga de uso común en plantas industriales y de reducir la interferencia externa. La ubicación de los radiotransmisores no fue afectada por el efecto de caída libre de las partículas durante la carga.

Según Chen et al., fue fundamental conocer la posición inicial de cada radiotransmisor antes de realizar un experimento. Durante la carga del silo, una vez que la altura

correspondiente a un nivel era alcanzada, se ubicaban los dispositivos correspondientes a ese nivel y una camada de medio granular era esparcida utilizando un rastrillo. Esta etapa del experimento requerió que un investigador entrara dentro del silo y utilizara el rastrillo para espaciar los granos de mineral de hierro. El tiempo que un radiotransmisor permanece dentro del silo fue definido como *tiempo de permanencia*.

Los experimentos fueron realizados a lo largo de cuatro meses y combinaron diferentes configuraciones de volúmenes de carga y procedimientos de descarga. Cada uno de los experimentos utilizó un único orificio de descarga y los tres orificios fueron utilizados en el conjunto de experimentos realizados. Los resultados reportados mostraron detalladamente el comportamiento del medio granular durante la descarga y fueron obtenidos con el silo completamente lleno y siendo vaciado a través del orificio localizado en el extremo radial de la circunferencia que sirve de base para el silo (orificio de salida (2)). El tiempo de descarga fue de aproximadamente 7:45 horas.

El artículo presentó diagramas en dos y tres dimensiones que permiten visualizar nítidamente el canal de flujo formado y su progresión durante los experimentos. Los resultados permitieron a los autores reconstruir la dinámica del aglomerado de partículas durante la descarga. Fue posible, por ejemplo, determinar el comportamiento de la superficie superior del aglomerado de partículas durante la ejecución de los experimentos y describir el comportamiento de la porción de partículas correspondiente a cada nivel de altura del silo.

Los principales aportes de este trabajo incluyeron: (1) la especificación minuciosa de un procedimiento de experimentación que sirve de punto de partida para otros investigadores, (2) la ejecución de experimentos que exigen un entorno de trabajo muy especializado y con poca disponibilidad, (3) la ejecución de experimentos en escala real que generalmente son realizados en laboratorio utilizando modelos a escala reducida, y (4) la determinación de resultados que deben ser obtenidos por las herramientas computacionales utilizadas en el estudio de dinámica de medios granulares para la descarga de silos a través de orificios de saida excéntricos.

5.8. Correlación de patrones de flujo de medios granulares y presiones ejercicidas en las paredes de un silo en la etapa de descarga

Chen et al. [36] basaron este trabajo en un trabajo previo de su autoria [34] y que ha sido presentado en la sección 5.7. Los investigadores aplicaron y extiendieron el método de experimentación utilizado anteriormente con el objetivo de investigar la correlación entre las presiones observadas en las paredes del silo y los patrones de flujo formados durante la etapa de descarga. Se construyó un silo de forma cilíndrica con 4,2 m de diámetro, sección de almacenamiento de 9,5 m de altura y altura total de aproximadamente 12 m para el almacenamiento de mineral de hierro. Los autores observaron que aún siendo un silo construido para estudios experimentales, las dimensiones utilizadas y el total de carga almacenada resultaron compatibles con los valores utilizados habitualmente en la industria.

Los autores presentaron los resultados de experimentos a escala real que lograron establecer la correlación entre el patrón de flujo y la distribución de presión ejercida en las paredes del silo durante la descarga del material almacenado. Existen en la literatura trabajos de otros autores que afirman que ambos fenómenos están directamente rela-

cionados [38, 99]. Sin embargo, según los autores, existe fuerte evidencia que los patrones de flujo están sujetos a las diferencias de escala. Chen et al. relevaron trabajos de otros investigadores mostrando que los resultados obtenidos a escala reducida no pueden ser extrapolados con la intención de considerar sistemas de mayor escala [5, 38]. Fueron incluidos en este artículo los detalles de la construcción del silo y de la preparación de los materiales de prueba.

Así como en el trabajo previo de Chen et al. [34], el silo fue rellenado con 250 toneladas de mineral de hierro y se utilizaron radiotransmisores ubicados estratégicamente en el interior de la estructura para determinar el patrón de flujo formado durante la etapa de descarga. Los experimentos que los autores realizaron en este trabajo fueron similares a los realizados en su trabajo anterior [34]. El método de experimentación fue ampliado de modo a incluir medidores de presión ubicados en puntos específicos de la faz interna de las paredes del silo. El silo fue descargado usando el orificio de descarga localizado en el extremo radial externo del fondo de la estructura. Cuando un radiotransmisor pasó a través de este orificio se marcó el momento de salida y se determinó cuánto tiempo el radiotransmisor estuvo dentro del silo (este periodo de tiempo fue definido por los autores como *tiempo de permanencia*).

Chen et al. lograron determinar la formación y la evolución de un patrón de flujo a partir de los tiempos de permanencia de cada radiotransmisor y de su ubicación inicial. Estos datos también permitieron establecer cualítativamente las propiedades del canal de flujo por donde son drenadas las partículas. Los medidores de presión operaron a lo largo de todo el experimento ofreciendo mediciones instantáneas de la presión ejercida en las paredes del silo.

Los resultados experimentales permitieron a los autores confirmar la fuerte correlación existente entre los patrones de flujo formados durante la etapa de descarga y las mediciones de presión observadas. Se mostró que existieron importantes diferencias de presión en las paredes del silo según se formara o no un canal de flujo contiguo a la pared. No fueron realizadas simulaciones en computador y todos los experimentos utilizaron el silo contruido.

Los principales aportes de este trabajo incluyeron: (1) ofrecer resultados confiables en los cuáles otros miembros de la comunidad científica pueden basar sus experimentos, (2) determinar las propiedades de un marco de trabajo que simula fidedignamente las características encontradas en un ambiente industrial, (3) determinar resultados que deben ser obtenidos por las herramientas computacionales utilizadas en el estudio de dinámica de medios granulares para la descarga de silos a través de orificios de saida excéntricos, y (4) corroborar de manera confiable los resultados que se encontraraban en la literatura en forma de conjeturas [100, 101, 102].

5.9. Resumen y conclusiones

Los trabajos relevados en este capítulo muestran dos herramientas utilizadas exitosamente en la investigación de dinámica granular: (1) simulaciones en computador aplicando el método DEM y técnicas de computación de alto desempño y (2) experimentos de laboratorio utilizando modelos a escala real y equipamiento de medición especializado. La tabla 5.1 presenta las líneas de investigación de los trabajos relevados.

Ferrez y Liebling [10], Markauskas, Kačeniauskas y Maknickas [30], Berger et al. [19], Shigeto y Sakai [25], Kačianauskas et al. [24] y Gopalakrishnan y Tafti [18] aplicaron el

Autores	Año	Línea de investigacíon
Ferrez y Liebling	2001	Aspectos básicos de DEM e implementaciones de algoritmos paralelos para dinámica granular incorporando técnicas avazadas de detección de contacto
Markauskas, Kačeniauskas y Maknickas	2011	Estrategias de descomposición dinámica de dominio para DEM
Berger et al.	2015	Métodos híbridos de paralelismo para sistemas de simulación DEM
Kačianauskas et al.	2010	Algoritmos paralelos para DEM con soporte a sistemas de partículas polidispersas
Shigeto y Sakai	2011	Algoritmos para DEM implementados para CPU y GPU
Gopalakrishnan y Tafti	2013	Algoritmos paralelos para DEM y su acoplamiento con otras técinas de dinámica molecular
Chen et al.	2005	Experimentos a escala real para el estudio de patrones de flujo de medios granulares
Chen et al.	2007	Experimentos a escala real para el estudio de la correlación entre patrones de flujo de medios granulares y la distribución de presión en las paredes de un silo

Tabla 5.1: Resumen de trabajos relevados

método DEM para realizar simulaciones en computador y confirmaron que este método es una herramienta computacional adecuada para el estudio de aspectos generales de dinámica de medios granulares. Las áreas de aplicación mostraron ser muy amplias y el método sirvió de soporte a la investigacíon de diferentes aspectos de los sistemas de partículas estudiados. Para el caso concreto de simulaciones de flujo de medios granulares durante la etapa de descarga de un silo, se observa que DEM fue el principal método computacional elegido por los investigadores [30, 19]. Además, los trabajos de estos investigadores mostraron que las técnicas de HPC lograron mejorar considerablemente el desempeño de la versión secuencial de DEM. Por ejemplo, Gopalakrishnan y Tafti [18] lograron una eficiencia computacional del 82 % utilizando una versión paralela de DEM para el estudio de sistemas de lechos fluidizados, Markauskas, Kačeniauskas y Maknickas [30] lograron una eficiencia del 75 % utilizando una estrategia de descomposición dinámica de dominio para DEM.

Chen et al. [34, 36] aplicaron técnicas de experimentación no computacionacioles para realizar experimentos de laboratorios utilizando modelos a escala real y mostraron las dificultades encontradas cuando se quiere reproducir en laboratorio, con alto grado de fidelidad, el mismo comportamiento encontrado en entornos industriales. Los investigadores mostraron detalladamente cómo se pueden construir y realizar experimentos con tales características y los resultados reportados ofrecen a la comunidad ciéntifica un marco de referencia para el estudio de la correlación entre el flujo de medios granulares y la distribución de presiones sobre las paredes de un silo durante la etapa de descarga a través de un orificio de salida excéntrico.

El relevamiento realizado muestra que no es frecuente encontrar en la literatura trabajos que modelan entornos industriales a escala real. Sin embargo, de manera general, los resultados reportados en los trabajos relevados mostraron la eficiencia y la robustez de DEM en simulaciones con un número relativamente grande de partículas (por ejemplo, Berger et al. [19] simularon un sistema con $1{,}5 \times 10^6$ partículas, y Gopalakrishnan y Tafti [18] realizaron simulaciones de un sistema con $1{,}0 \times 10^7$ partículas). Los resultados

reportados indican que es factible utilizar una versión paralela de DEM en modelos a escala real basados en situaciones encontradas en la industria.

Los investigadores reconocieron que los recursos de hardware disponibles pueden imponer un límite a las dimensiones de los sistemas que se pretende simular. Shigeto y Sakai [25] mostaron, por ejemplo, que las estructuras de datos utilizadas en las implementaciones paralelas de DEM deben poner énfasis en el uso eficiente de memoria RAM o puede resultar imposible representar en computador el sistema estudiado. Esta característica presenta desafíos importantes a los investigadores y se observa en la literatura relacionada que el desarrollo de algoritmos paralelos eficientes para DEM es un campo de investigación muy activo en dinámica de medios granulares. Los trabajos relevados mostraron que las mejores de desempeño de un algoritmo para dinámica granular pueden ser obtenidas a través de estrategias no computacionales. Por ejemplo, Markauskas, Kačeniauskas y Maknickas [30] implementaron un algoritmo paralelo para DEM que utiliza un esquema de integración de EDO que requiere menos pasos de tiempo).

Los trabajos de Markauskas, Kačeniauskas y Maknickas [30] y Berger et al. [19] incluyeron estudios de balanceo de carga entre procesadores. Los investigadores desarrollaron versiones paralelas de DEM que incorporaron estrategias de descomposición dinámica de dominio. La estrategias de descomposición de dominio utilizadas en ambos trabajos son muy similares y lograron mantener niveles bajos de desbalanceo de carga entre procesadores. Los resultados reportados mostraron que las versiones paralelas de DEM que incorporan estrategias de descomposición estática de dominio tuvieron desempeño inferior respecto de las versiones paralelas que utilizaron estrategias de descomposición dinámica. Según los autores, esta diferencia de desempeño ocurrió debido a la estrategia de descomposición aplicada.

Los trabajos relevados mostraron que DEM es ampliamente utilizado tanto en la industria y como en el ambiente académico. Algunos investigadores utilizaron sistemas comerciales con acceso mediante pago (por ejemplo, Berger et al. [19] utilizaron el sistema LIGGGHTS), otros investigadores utilizaron sistemas de código abierto construidos en laboratorios de investigación (por ejemplo, Gopalakrishnan y Tafti [18] utilizaron el sistema MFIX) y hubo investigadores que utilizaron implementaciones propias del método (por ejemplo, Markauskas, Kačeniauskas y Maknickas [30] y Shigeto y Sakai [25]). En todos los casos, el método pudo ser aplicado y uno de sus puntos fuertes —la versatilidad—se corrobora observando la amplia variedad de problemas estudiados.

Por lo tanto, dado los resultados reportados en los trabajos relevados, es posible concluir que el método DEM es una de las principales herramientas computacionales utilizadas en las investigaciones realizadas en el área de dinámica de medios granulares. Esta área de investigación es muy activa y los sistemas computacionales utilizados se encuentran en constante desarrollo. Las técnicas de HPC son ampliamente utilizadas en los sistemas que incorporan implementaciones paralelas de DEM con el objetivo de potenciar aún más esta herramienta. Las mejoras de desempeño de los sistemas de simulación DEM son un contribución muy importante para la comunidad científica en general.

Capítulo 6

Estrategia de descomposición dinámica de dominio basada en planos móviles

Este capítulo presenta los detalles de la estrategia de descomposición de dominio implementada en este proyecto de grado. La sección 6.1 introduce los conceptos teóricos básicos en los cuáles se basa la estrategia propuesta. La secciones 6.2, 6.3 y 6.4 explican detalladamente cómo funcionan las etapas de la estrategia de descomposición de dominio propuesta y describen detalles de su implementación. La sección 6.5 presenta los algoritmos que fueron agregados para el tratamiento de mallas triangulares en el contexto de la estrategia de descomposición dinámica de dominio propuesta en este proyecto de grado. Finalmente, la sección 6.6 describe la forma en que el usuario puede utilizar la estrategia de descomposición dinámica de dominio en una simulación numérica.

6.1. Introducción

La estrategia de descomposición dinámica de dominio propuesta está basada en el concepto de planos móviles (*moving planes*) presentada por Markauskas et al. [30]. La principal idea de esta estrategia es mover las fronteras entre subdominios de modo que la carga de trabajo asignada a cada proceso se encuentre dentro de un límite de tolerancia preestablecido. La sección 5.2 muestra detalladamente los fundamentos de esa estrategia. La carga de trabajo de un proceso puede ser caracterizada de varias formas. Según Markauskas et al., una diferencia significativa en el número de partículas en un subdominio es un indicador de desbalanceo de carga. Otra forma de caracterizar la carga de trabajo de un proceso que según Markauskas et al. es más precisa que la anterior, es considerar el tiempo de cómputo que un proceso requiere para ejecutar completamente un ciclo de integración. En este proyecto de grado la carga de trabajo de un proceso está caracterizada según la primera definición mencionada.

El algoritmo de descomposición dinámica que este proyecto de grado propone es ejecutado en intervalos regulares de pasos de tiempo definidos por el usuario. El algoritmo propuesto por Markauskas et al. [30] es ejecutado siempre que exista un proceso cuya carga de trabajo difiere más de un 20 % respecto de la carga de trabajo de otro proceso. Es posible expresar matemáticamente esta condición según la expresión 6.1, donde $l(p_i)$ es la carga de trabajo del proceso p_i y E es la cantidad de procesos en ejecución. Los

procesos son numerados a partir de 0 para adoptar una nomenclatura similiar a la que utiliza MPI al momento de asignar los rangos a los procesos.

$$\{p_i \mid \exists p_j, i, j \in \{0, \ldots, E-1\}, i \neq j, |(l(p_i) - l(p_j)| > 0{,}2 \times l(p_j)\} \tag{6.1}$$

Las siguientes secciones muestran detalladamente los cambios realizados en el código fuente de LIGGGHTS. La organización de las subsecciones es tal que los métodos y algoritmos descritos en una subsección utilizan las herramientas presentadas en subsecciones previas. Debe observarse que el ciclo de integración de la figura 4.2 fue sustituido por el que se representa en el esquema de la figura 6.1.

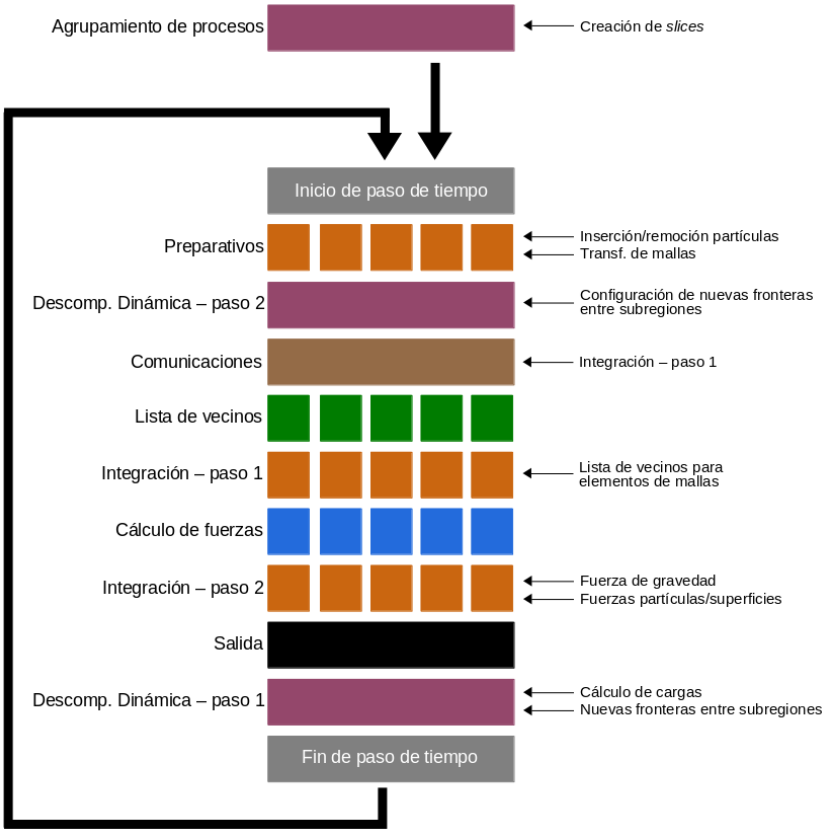


Figura 6.1: Esquema del ciclo de integración modificado de LIGGGHTS

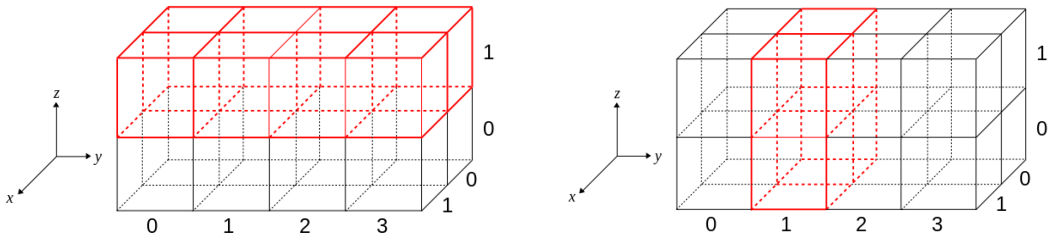
Las principales modificaciones son: (1) la incorporación de una etapa de generación de agrupamientos de procesos previa al inicio de un ciclo de cómputo, (2) la incorporación de una etapa para el cálculo de cargas de trabajo y generación de nuevas fronteras para subdominios, y (3) la incorporación de una etapa para la aplicación de las nuevas fronteras calculadas.

6.2. Generación de agrupamientos de procesos

Un *agrupamiento de procesos respecto del eje e* se define como un conjunto de procesos tales que son iguales las coordenadas en el eje e de los puntos extremos que definen los subdominios a los cuáles los procesos fueron asignados. La sección 4.2 muestra que

el usuario puede defirnir la cantidad de particiones que serán creadas para el eje de coordenadas e. Las particiones de los ejes de coordenadas cartesianas generan subdominios tales que sus fronteras respecto de un eje de coordenadas e son paralelas entre si y perpendiculares a e.

Por lo tanto, para un eje de coordenadas e se generan $PARTS_e$ agrupamientos y los procesos p_i y p_j pertenecen al mismo agrupamiento si, y solo si, $e_i^{min} = e_j^{min}$ y $e_i^{max} = e_j^{max}$, siendo $PARTS_e$ el parámetro del comando `processors` que define la cantidad de particiones del eje e, y e_i^{min} y e_i^{max} son, respectivamente, los valores máximo y mínimo de las coordenadas en e de los puntos que definen el subdominio al cual p_i fue asignado. La figura 6.2 muestra dos ejemplos de agrupamientos creados respecto del eje y (figura 6.2a) y del eje z (figura 6.2b) para los cuáles $PARTS_1 = PARTS_3 = 2$ y $PARTS_2 = 4$. Se destacan de color rojo los miembros de los agrupamientos con el fin de facilitar la visualización.



(a) Agrupamientos respecto del eje y

(b) Agrupamientos respecto del eje z

Figura 6.2: Ejemplos de agrupamientos de procesos

En el caso de la figura 6.2a los procesos p_i y p_j pertenecen al mismo agrupamiento si, y solo si, $y_i^{min} = y_j^{min}$ y $y_i^{max} = y_j^{max}$. El caso de la figura 6.2b los procesos p_i y p_j pertenecen al mismo agrupamiento si, y solo si, $z_i^{min} = z_j^{min}$ y $z_i^{max} = z_j^{max}$. Cada proceso pertenece a tres agrupamientos, uno respecto de cada eje de coordenadas. Si e y f son ejes de coordenadas tales que $PARTS_e = PARTS_f = 1$, siendo $e \neq f$, existe solamente un agrupamiento respecto de e y solamente un agrupamiento respecto de f. Además, estos agrupamientos coinciden.

El método `Verlet::run()` ejecuta el código que agrupa los procesos. Esta tarea, según se muestra en la figura 6.1, es realizada antes que empiece el bucle de integración. La sección 6.4 explica que las fronteras determinadas por nuevo puntos de corte en el eje e siguen siendo paralelas a este eje y que los subdominios no dejan de ser ortoedros. Por lo tanto, los procesos pertenecientes a un agrupamiento no cambian a lo largo de la simulación y no es necesario generar un agrupamiento mútiples veces.

El algoritmo 7 muestra cómo se generan los agrupamientos respecto del eje de coordenadas x. La idea es que cada iteración del bucle de la línea 1 corresponda a un agrupamiento respecto del eje de coordenadas para el cual se construyen los agrupamientos, y que los bucles de las líneas 3 y 4 permitan recorrer los subdominios cambiando de forma adecuada las coordenadas para los ejes restantes. Para generar los agrupamientos respecto de los ejes y y z es necesario:

- Elegir una permutación adecuada de las líneas 1, 3 y 4.
- Sustituir el témino `rankMatrix(0,i,t)` en la línea 5 por `rankMatrix(0,j,t)` y `rankMatrix(0,k,t)`, para las coordenadas y y z, respectivamente.

- Sustituir el témino `mySlice(0) := i` en la línea 7 por `mySlice(1) := j` y `mySlice(2) := k`, para las coordenadas y y z, respectivamente.

El término `grid2proc(i,j,k)` es una matriz de 3 dimensiones con capacidad para almacenar $PARTS_1 \times PARTS_2 \times PARTS_3$ nú meros enteros. Esta matriz hace referencia a un atributo de mismo nombre que pertenece a la clase `Comm` cuyo objetivo es almacenar en la posición (i, j, k) el rango del proceso asignado al subdominio identificado por las coordenadas (i, j, k). El término `me` es una variable entera y hace referencia a un atributo de mismo nombre que pertenece a la clase `Comm` cuyo objetivo es almacenar el rango del proceso en ejecución.

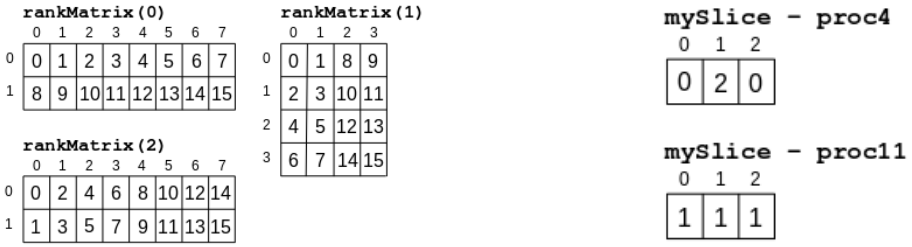
Algoritmo 7 Algoritmo para generación de agrupamientos respecto del eje x

```
 1: for i := 0 to PARTS_1 − 1 do
 2:    t := 0;
 3:    for j := 0 to PARTS_2 − 1 do
 4:       for k := 0 to PARTS_3 − 1 do
 5:          rankMatrix(0,i,t) := grid2proc(i,j,k);
 6:          if me = grid2proc(i,j,k) then
 7:             mySlice(0) := i;
 8:          end if
 9:          t := t + 1;
10:       end for
11:    end for
12: end for
```

Las estructuras de datos fundamentales utilizadas en el algoritmo 7 que permiten agrupar los procesos adecuadamente son `mySlice` (vector de 3 posiciones de números enteros) y `rankMatrix` (vector de 3 posiciones de matrices de números enteros). Estas estructuras hacen referencia, respectivamente, a dos atributos privados que fueron agregados a la clase `Verlet`. Los nombres de las estructuras en al algoritmo 7 coinciden con los nombres dados a los atributos.

Una vez que el algortimo 7 termina su ejecución los agrupamientos respecto de cada eje de coordenadas han sido generados, el vector `mySlice` almacena en cada una de sus posiciones el agrupamiento al cual pertenece el proceso en ejecución respecto de cada uno de los ejes de coordenadas y el vector `rankMatrix` almacena en cada una de sus posiciones una matriz que relaciona todos los procesos a sus respectivos agrupamientos. Estos atributos son usados por otros métodos que participan en la estrategia de descomposición dinámica de dominio. Dado que LIGGGHTS está en MPI y que el atributo `grid2proc` coincide para todos los procesos. Por lo tanto, el vector `rankMatrix` es igual para todos los procesos. Por otro lado, el vector `mySlice` puede ser diferente.

La figura 6.3a presenta un ejemplo para el vector `rankMatrix` y la figura 6.3b presenta dos ejemplos para el vector `mySlice`. El vector de la figura 6.3a corresponde a los agrupamientos creados para los ejemplos de agrupamientos de la figura 6.2, y los vectores de la figura 6.3b corresponden, respectivamente, a los procesos de rango 4 y rango 11.



(a) Agrupamientos respecto del eje y (b) Agrupamientos respecto del eje z

Figura 6.3: Ejemplos de estructuras `mySlice` y `rankMatrix`

6.3. Cálculo de cargas de trabajo

La carga de trabajo de un proceso p se define como la proporción de partículas en la simulación que están contenidas en el subdominio al cual p fue asignado. Ésta es una de las formas de caracterizar la carga de trabajo de un proceso según Markauskas et al. [30] y es el enfoque adoptado en este proyecto de grado. Esta definición puede ser expresada matemáticamente según la ecuación 6.2.

$$l(p) = \frac{\#particulas\_subdominio}{\#particulas\_total} \tag{6.2}$$

Dado que la variante LIGGGHTS-PUBLIC no soporta estrategias de descomposición dinámica de dominio, no dispone de estructura de datos que almacene informaciones sobre las cargas de trabajo de los procesos. Analizando el código fuente de LIGGGHTS se puede observar que la clase `Atom` almacena en el atributo `natoms` la cantidad total de partículas en la simulación y en los atributos `nlocal` y `nghost` almacena, respectivamente, la cantidad total de partículas contenidas en el subdominio al cual el proceso fue asignado y la cantidad total de partículas fantasma que deben ser consideradas por el proceso en ejecución al momento de resolver las ecuaciones de movimiento. Los atributos `natoms`, `ghost` y `nlocal` son utilizados para tareas de control durante una simulación (por ejemplo, para saber si ha sido insertada la cantidad correcta de partículas) y no fueron incluidos por los desarrolladores de LIGGGHTS para dar soporte a alguna estrategia de descomposición. Sin embargo, considerando la forma en que se caracteriza la carga de trabajo de un proceso, estos atributos pueden ser utilizados con el objetivo de ofrecer soporte a una estrategia de descomposición dinámica de dominio.

La figura 6.1 muestra que al inicio de un paso de tiempo, en la etapa *Preparativos*, pueden ser insertadas o removidas partículas del sistema. Además, en la etapa *Comunicaciones*, las partículas que siguen en el sistema pueden ser transferidas de un subdominio a otro. Finalmente, en la etapa *Salida* los resultados del paso de tiempo actual son redirigidos a los métodos de manejo de entrada y salida elegidos por el usuario (archivos escritos en disco o información impresa en pantalla). A partir de ese momento y hasta que no empiece una nueva iteración del ciclo de integración, el estado del sistema no cambia. Por lo tanto, es adecuado invocar los métodos que calculan la carga de trabajo de un proceso una vez que la salida del sistema ha sido procesada y antes que empiece una nueva iteración del ciclo de integración.

El método de planos móviles [30] se basa en mover las fronteras entre subdominios respecto de los ejes de coordenadas para redistribuir la carga de trabajo entre los procesos.

La sección 6.4 explica detalladamente cómo funciona el movimiento de las fronteras. Por el momento es suficiente observar que es un procedimiento iterativo sobre los ejes de coordenadas y que en vez de calcular la carga de trabajo individual de cada proceso, calcula la carga de trabajo de un agrupamiento de procesos. La sección 6.2 describe como los agrupamientos de procesos son creados. La carga de trabajo de un agrupamiento A es definida como la suma de carga de trabajo de los procesos que pertenecen a A. Esta definición puede ser expresada matemáticamente según la ecuación 6.3:

$$l(A) = \sum_{p \in A} l(p) \tag{6.3}$$

La carga de trabajo de un agrupamiento de procesos es calculada utilizando un procedimiento iterativo de dos etapas basado en MPI. Las iteraciones ocurren primero sobre el eje x, a continuación sobre el eje y y, finalmente, sobre el eje z. La implementación del algoritmo de cálculo de carga de trabajo de un agrupamiento se encuentra en el método `Verlet::compute_load()`. Este método no pertenece a la versión original de LIGGGHTS y es parte integrante de la solución propuesta. La figura 6.1 muestra que el método `compute_load()` es invocado después que la salida fue procesada y antes que ocurra una posible descomposición de dominio. El usuario puede configurar un parámetro que determina intervalos regulares en los cuales este método es invocado. Los resultados obtenidos son utilizados para descomponer el dominio y para ofrecer retroalimentación respecto del balanceo de carga. El algoritmo 8 muestra un esquema del método `compute_load()`.

Algoritmo 8 Algoritmo para cálculo de $l(A)$

```
for dim := 1 to 3 do
  MPI_Comm_split(MPI_COMM_WORLD, mySlice(dim), myRank,
      subWorldSlices);
  MPI_Allreduce(nlocal, subWorldSlicesPart, 1, MPI_INT, MPI_SUM,
      subWorldSlices);

  localLoad := subWorldSlicesPart / natoms;

  myColor := 0;
  if rankMatrix(dim,mySlice(dim),0) = me then
    myColor := 1;
  end if

  MPI_Comm_split(MPI_COMM_WORLD, myColor, myRank,
      subWorldHeadSlices);
  MPI_Gather(localLoadRatio, 1, MPI_DOUBLE, sendRecvSliceLoadBuf, 1,
      MPI_DOUBLE, 0, subWorldHeadSlices);

  if myRank = 0 then
    for i := 0 to PARTS_dim − 1 do
      sliceLoadArray(dim,i) := sendRecvSliceLoadBuf(i);
    end for
  end if
end for
```

La primera etapa del algoritmo de cálculo de cargas de trabajo para agrupamientos de procesos empieza en la línea 2. La instrucción `MPI_Comm_Split` crea nuevos comunicadores a partir del comunicador `MPI_COMM_WORLD` y asigna los procesos existentes a uno de los nuevos comunicadores según el parámetro `mySlice(dim)` y tal que el proceso de rango 0 en el nuevo comunicador sea aquel de menor rango entre todos los procesos cuyo valor de `mySlice(dim)` coincide. Los nuevos comunicadores se llaman `subWorldSlices`. Sin embargo, no existe riesgo de que haya comunicadores diferentes con el mismo nombre porque la biblioteca MPI se encarga de asegurar que exista solamente un comunicador llamado `subWorldSlices` en el contexto de cada proceso. El vector `mySlice` se calcula cuando se generan los agrupamientos de procesos.

La instrucción `MPI_Allreduce` hace que todos los procesos que pertenecen al mismo comunicador `subWorldSlices` envien el parámetro `nlocal` al proceso 0 de ese comunicador. Una vez recibidos los mensajes, la instrucción suma todos los valores recibidos, almacena el resultado en la variable entera `subWorldSlicesPart` y los distribuye a todos los procesos de `subWorldSlices`. El parámetro `MPI_SUM` indica que se realiza la operación de suma. Así, al terminar la ejecución de la instrucción `MPI_Allreduce` todos los procesos de un agrupamiento almacenan en `subWorldSlicesPart` la cantidad total de partículas que se encuentran en los subdominios a los cuáles los procesos del comunicador `subWorldSlices` fueron asignados. En la línea 5 se calcula la carga de trabajo del agrupamiento.

La segunda etapa empieza en la línea 12. La instrucción `MPI_Comm_Split` crea nuevos comunicadores, llamados `subWorldHeadSlices`, de forma similar a la explicada anteriormente. En este caso, los procesos son clasificados en base al parámetro `myColor`. Este parámetro es calculado en las líneas 7–10 de modo que se generan solamente dos instancias del comunicador `subWorldHeadSlices`. Las instancias generadas son tales que en una estarán todos los procesos con rango 0 de cada comunicador `subWorldSlices` creado previamente y en la otra estarán todos los demás procesos. El vector `rankMatrix` se calcula cuando se generan los agrupamientos de procesos.

La instrucción `MPI_Gather` se invoca para enviar desde cada proceso del comunicador `subWorldHeadSlices` el valor del parámetro `localLoad` al proceso 0 de ese comunicador. Dado que el proceso 0 de `subWorldHeadSlices` y el proceso 0 del comunicador `MPI_COMM_WORLD` son el mismo proceso, al terminar de ejecutar a la instrucción `MPI_Gather` la carga de trabajo de cada agrupamiento estará almacenada en una de las posiciones del vector `sendRecvSliceLoadBuf` del proceso 0 del comunicador `MPI_COMM_WORLD`. El cálculo de la carga de trabajo de todos los agrupamientos respecto del eje considerado termina con la ejecución del bucle en la línea 16 que solo es ejecutado por el proceso 0 del comunicador `MPI_COMM_WORLD`. Las cargas de trabajo de todos los agrupamientos respecto de cada eje de coordenadas son almacenadas en la matriz `sliceLoadArray`.

La figura 6.4 muestra un esquema gráfico de las comunicaciones interprocesos que ocurren durante el cálculo de cargas de trabajo de agrupamientos para el ejemplo de la figura 6.3a. Para este caso se tiene $dim = 2$, cuatro agrupamientos respecto del eje y y cuatro comunicadores `subWorldSlices`. La figura muestra el contenido del vector `rankMatrix` y se destacan los procesos de rango 0 en cada instancia del comunicador `subWorldSlices`. Por la forma en que el vector `rankMatrix` es construido, los valores de rango de los procesos de rango 0 de cada comunicador `subWorldSlices` se encuentran almacenados en las posiciones $(1, k, 0)$ del vector `rankMatrix`. Los procesos 0, 2,

4 y 6 están en círculos mayores que los demás procesos para destacar sus roles en los comunicadores `subWorldSlices`.

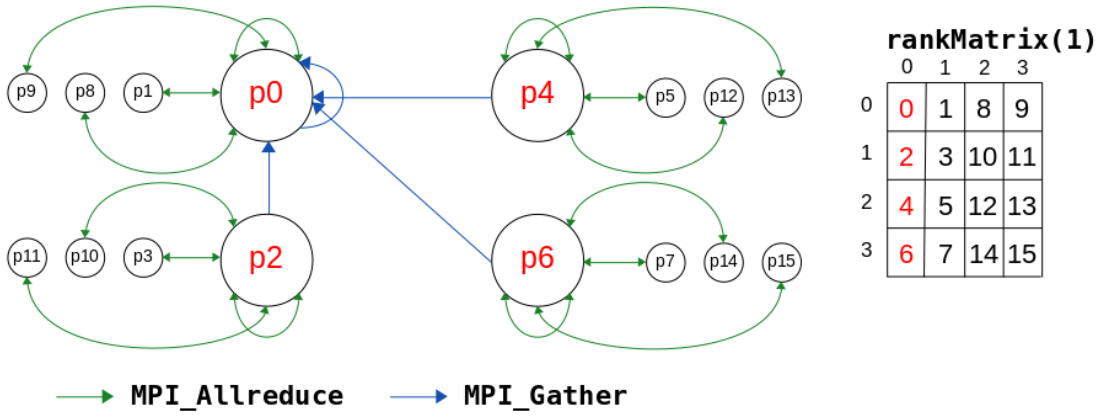


Figura 6.4: Esquema gráfico del cálculo de cargas trabajo de agrupamientos

6.4. Nuevos límites para fronteras de subdominios

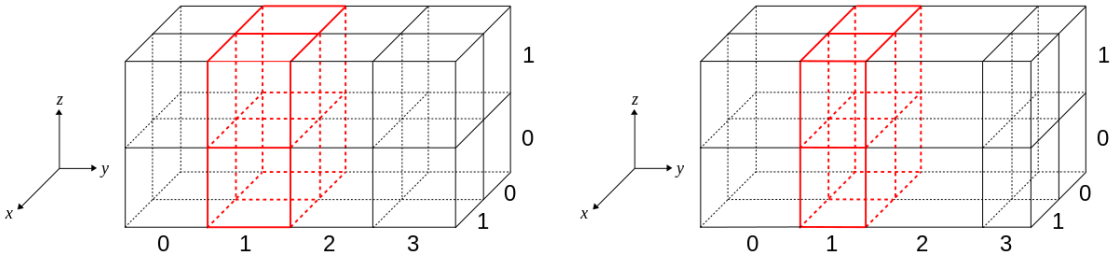
Para generar una nueva descomposición de dominio es necesario conocer las cargas de trabajo de los procesos. La estrategia de planos móviles [30] requiere esta información para ajustar las fronteras de los subdominios. Los ajustes ocurren de modo que para un agrupamiento de procesos A y un eje de coordenadas e, los puntos extremos en e de las dimensiones de los subdominios a los cuales fueron asignados los procesos de A son iguales antes y después de los movimientos. Esto significa que si p_i y p_j son procesos de A asignados, respectivamente, a los subdominios r y s, y si e_r^{min}, e_r^{max}, e_s^{min} y e_s^{max} son, respectivamente, los puntos extremos de las dimensiones de los subdominios r y s, entonces las igualdades $e_r^{min} = e_s^{min}$ y $e_r^{max} = e_s^{max}$ son válidas antes y después del movimiento de las fronteras. Por lo tanto, las fronteras de los subdominios asociados a los procesos de A son ajustadas simultáneamente y en igual proporción. Esta conclusión tiene un impacto importante en la forma de calcular las cargas de trabajo. En vez de calcular la carga de trabajo individual de cada proceso, deben calcularse las cargas de trabajo de cada agrupamiento de procesos. Las cargas de trabajo de los agrupamientos son calculadas según el procedimiento presentado en la sección 6.3.

Esta sección está divida en dos partes para facilitar la presentación y destacar que, según muestra la figura 6.1, el cálculo de las nuevas fronteras de los subdominios ocurre en dos momentos diferentes del ciclo de integración.

6.4.1. Nuevas fronteras para subdominios

El cálculo de de las nuevas fronteras para subdominios, así como ocurre con el cálculo de cargas de trabajo, trabaja en forma iterativa ajustando primero las fronteras respecto del eje x, luego respecto del eje y y, finalmente, respecto del eje z. La forma en que las fronteras son ajustadas justifica el método de cálculo de cargas de trabajo presentado en la sección 6.3. La figura 6.5 muestra un ejemplo de movimiento de fronteras respecto del

eje y donde las dimensiones de los agrupamientos 0 y 2 crecieron, y las dimensiones de los agrupamientos 1 y 3 disminuyeron.



(a) Antes del ajuste (b) Después del ajuste

Figura 6.5: Ajuste de frontreras de subdominios respecto del eje y

La figura 6.1 muestra que las nuevas fronteras para subdominios son calculadas inmediatamente antes de finalizar una iteración del bucle de integración y los resultados calculados son usados en el próximo paso de tiempo, inmediatamente después de la etapa *Preparativos*. Es necesario seguir esta secuencia de etapas porque al principio de una iteración LIGGGHTS puede ejecutar algunas acciones que dependen de la descomposición de dominio anterior (por ejemplo, la inserción de nuevas partículas). El algoritmo de cálculo de nuevos límites para fronteras está implementado en el método `Verlet::generate_split()`. Este método no pertenece a la versión original de LIGGGHTS y es parte integrante de la solución propuesta en este proyecto de grado.

El algoritmo 9 muestra un esquema del método `generate_split()`. Desde el punto de vista del intercambio de mensajes entre procesos, el método `generate_split()` funciona de forma similar al método `Verlet::compute_load()`. Sin embargo, la instrucción `MPI_Allreduce` se sustituye por `MPI_Reduce` debido a que no es necesario devolver los valores calculados a los procesos. La figura 6.6 muestra un esquema gráfico de las comunicaciones interprocesos que ocurren durante el cálculo de las nuevas fronteras entre subdominios. Así como en el caso de la figura 6.4, se muestra un ejemplo basado en la figura 6.3a.

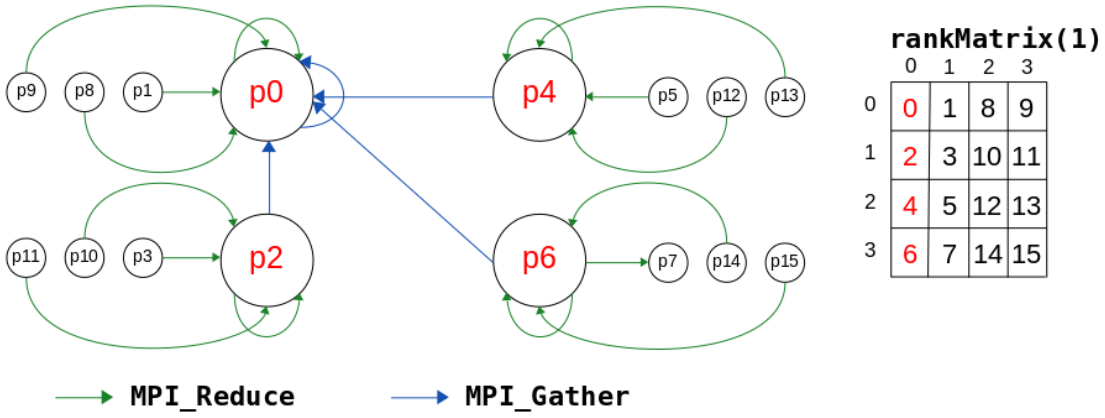


Figura 6.6: Esquema gráfico del cálculo de cargas trabajo de agrupamientos

El algoritmo 9 está basado en el método *Dynamic strip decomposition* (DSD) de Hanxleden y Scott [64]. Este método recorre el intervalo $Z_e = [min_e, max_e]$ de modo que

al haber recorrido un volumen que contiene más de $\frac{natoms \times f}{PARTS_e}$ partículas, se determina un punto de Z_e que corresponde a la coordenada de la f-ésima frontera en el eje de coordenadas e, siendo $PARTS_e$ el total de particiones generadas para el eje de coordenadas e, $natoms$ el total de partículas en la simulación y tal que se cumple $1 \leqslant f \leqslant PARTS_e + 1$. Por lo tanto, si para el eje e se generan $PARTS_e$ particiones, se determinan $PARTS_e$ puntos en el intervalo Z_e y se crean $PARTS_e + 1$ fronteras. El punto min_e no determina la coordenada de una frontera y el punto max_e determina la frontera de la última partición generada.

Algoritmo 9 Algoritmo para cálculo de límites de subdominios en el eje `dim`

```
1:  for part := 0 to nlocal − 1 do
2:    subSliceIndex := coord2SubSlice(x(part), dim);
3:    subSliceCantPart(subSliceIndex)++;
4:  end for
5:
6:  MPI_Comm_split(MPI_COMM_WORLD, mySlice(dim), myRank, subWorldSlices);
7:  MPI_Reduce(subSliceCantPart, totalPartInSubslice, numSubSlices, MPI_INT,
        MPI_SUM, 0, subWorldSlices);
8:
9:  myColor := 0;
10: if rankMatrix(dim,mySlice(dim),0) = me then
11:   myColor := 1;
12: end if
13:
14: MPI_Comm_split(MPI_COMM_WORLD, myColor, myRank, subWorldHeadSlices);
15: MPI_Gather(totalPartInSubslice, numSubSlices, MPI_INT,
        totalPartInSubsliceGlobal, numSubSlices, MPI_INT, 0, subWorldHeadSlices);
16:
17: if myRank = 0 then
18:   splitTmp(0) := 0.0;
19:   splitTmp(PARTS_dim) := 1.0;
20:   subSliceIndex := 0;
21:   numSlice := 1;
22:   partialPartSum := 0;
23:
24:   repeat
25:     newSplitPointCriterium := (natoms × numSlice) / PARTS_dim;
26:     partialPartSum += totalPartInSubsliceGlobal(subSliceIndex);
27:     while partialPartSum < newSplitPointCriterium do
28:       subSliceIndex++;
29:       partialPartSum += totalPartInSubsliceGlobal(subSliceIndex);
30:     end while
31:     splitTmp(numSlice) := newSplitPointBoxCoord(subSliceIndex, dim);
32:     numSlice++;
33:   until numSlice < PARTS_dim
34: end if
```

Para aplicar DSD, los subintervalos en e que definen cada subdominio son discretizados. La sección 4.2 muestra que un subdominio r está definido por los valores x_r^{min} y x_r^{max}, y_r^{min} y y_r^{max}, y z_r^{min} y z_r^{max} que definen los extremos de subintervalos sobre, respectivamente, los ejes de coordenadas x, y y z. El objetivo de la discretización del subintervalo $Y_r = [e_r^{min}, e_r^{max}]$ es permitir recorrer adecuadamente un subintervalo de números reales definido en el eje de coordenadas e para el conteo de partículas pertenecientes al área total ya recorrida. La discretización se hace dividiendo cada subintervalo Y_r en D subintervalos de longitud $L_{e,r} = \dfrac{e_r^{max} - e_r^{min}}{D}$ y generando un conjunto de puntos $P_{e,r} = \{e_r^{min} + L_{e,r}, e_r^{min} + 2L_{e,r}, \ldots, e_r^{max} - L_{e,r}\}$. Si r y s son subdominios tales que los procesos p_i y p_j pertenecen al mismo agrupamiento y fueron, respectivamente, asignados a r y a s, entonces se cumple $e_r^{min} = e_s^{min}$ y $e_r^{max} = e_s^{max}$ y los resultados del procedimiento de discretización de Y_r y Y_s coinciden (esto significa que $P_{e,r} = P_{e,s}$).

Dado que se tienen $PARTS_e$ particiones a lo largo del eje e, se contruyen un total de $D \times PARTS_e$ subintervalos. Si W_r^i, con $1 \leqslant i \leqslant D$, son los subintervalos creados para el subdominio r, se verifica $Y_r = \bigcup_{i=1}^{D} W_r^i$. Además, es válida la igualdad $Z_e = \bigcup Y_r$. Los subintervalos W_r^i son definidos como *fracciones* del subdominio r y, por construcción, se generan D fracciones para cada subdominio. Siempre que es necesario recorrer el intervalo Z_e las discretizaciones de cada subdominio respecto del eje de coordenadas e son utilizadas. El valor utilizado para D es definido por el usuario como un parámetro de configuración de la simulación. El valor elegido para D determina la precisión con la cual son calculadas las nuevas fronteras de los subdominios. Los puntos del conjunto $P_{e,r}$ definen rectas perpendiculares al eje e que tienen las mismas propiedades geométricas de las fronteras de los subdominios. Por lo tanto, las fracciones son ortoedros paralelos entre sí. La figura 6.7 muestra un ejemplo de fracciones creadas para un subdominio respecto del eje y.

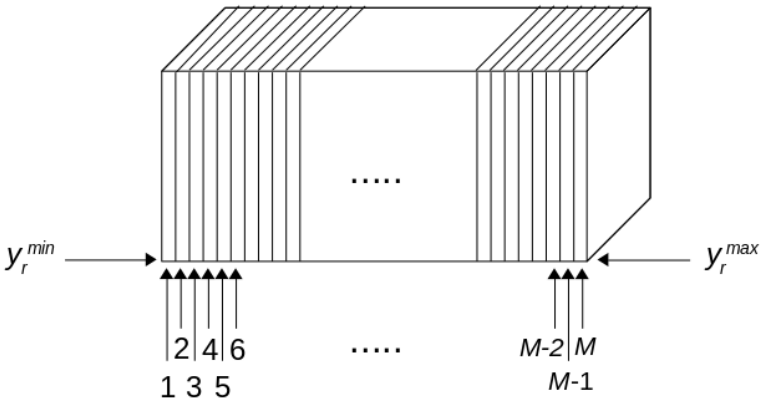


Figura 6.7: Fracciones para un subdominio respecto del eje y

Las líneas 1-4 del algoritmo 9 almacenan en las posiciones del vector `subSliceCantPart` los totales de partículas en cada fracción W_r^i del subdominio r al cuál el proceso en ejecución fue asignado. El método `Verlet::coord2SubSlice()` no pertenece a la versión original de LIGGGHTS y es parte integrante de la solución propuesta. Su objetivo es devolver el índice i de la fracción W_r^i al cuál pertenece la partícula

`x(part)` respecto del eje de coordenadas *dim*. El vector `x` almacena las coordenadas de las partículas que pertenecen al subdominio r. La classe `Atom` almacena en un atributo de igual nombre las coordenadas de las partículas contenidas en r. El resultado obtenido es por proceso y, nuevamente, se crean comunicadores MPI en base al agrupamiento al cual un proceso pertenece para consolidar en el proceso de rango 0 de los comunicadores generados la cantidad total de partículas en las fracciones W_r^i. Los nuevos comunicadores se llaman `subWorldSlices`. La instrucción `MPI_Reduce` (línea 7) permite la consolidación de los resultados en el vector `totalPartInSubslice`. La biblioteca MPI es responsable por asignar cada proceso a una única instancia del comunicador `subWorldSlices`.

Las líneas 9-15 del algoritmo 9 son responsables de enviar los valores almacenados en el vector `totalPartInSubslice` de cada proceso 0 de los comunicadores `subWorldSlices` al proceso 0 del comunicador `subWorldHeadSlices`. Dado que el proceso 0 del comunicador `subWorldHeadSlices` y el proceso 0 del comunicador `MPI_COMM_WORLD` son el mismo proceso, luego de ejecutar la instrucción `MPI_Gather` (línea 15) el proceso 0 del comunicador `MPI_COMM_WORLD` almacena en el vector `totalPartInSubsliceGlobal` la distribución de las partículas en las fracciones W_r^i de cada subintervalo Y_r definido sobre el intervalo Z_e.

El código que se encuentra entre las líneas 17 y 24 es ejecutado solamente por el proceso de rango 0 del comunicador `MPI_COMM_WORLD`. Por lo tanto, las nuevas fronteras son calculadas de forma centralizada. La implementación para el método DSD es iterativa y recorre el intervalo Z_e en base a las discretizaciones para los subintervalos Y_r. Cada iteración de la instrucción `repeat` de la línea 24 permite calcular una nueva frontera según la cantidad de partículas que se encuentran en el área ya recorrida del intervalo Z_e. El método `Verlet::newSplitPointBoxCoord()` no pertenece a la versión original de LIGGGHTS y es parte integrante de la solución propuesta. Su objetivo es obtener el punto de corte que determina la nueva frontera en base al índice que identifica a cada una de las fracciones W_r^i.

Una vez concluido el cálculo de los puntos de corte para las nuevas fronteras para todos los ejes de coordenadas, una bandera de control es habilitada para indicar que en el próximo paso de tiempo es necesario aplicarlas a la simulación.

6.4.2. Aplicando nuevas fronteras

Cuando un nuevo paso de tiempo inicia debe verificarse si una nueva descomposición de dominio fue generada. Si este es el caso, las fronteras entre los subdominios deben ser ajustadas según los resultados calculados en el paso de tiempo inmediatamente anterior. El ajuste se hace iterativamente aprovechando las instrucciones que ejecuta el bucle correspondiente al ciclo de integración. El código implementado fue incorpado al método `Verlet::run()` y el algoritmo 10 muestra un esquema de la implementación desarrollada. El algoritmo 10 fue construido a partir del algoritmo 3 y una de las principales diferencias es la adición de una estructura de control `if-then-else` (línea 9–15) para verificar si es necesario cambiar la descomposición de dominio. Otra diferencia importante es el uso del patrón de comunicación Irregular. Cuando es necesario ajustar las dimensiones de los subdominios es fundamental usar este patrón de comunicación porque al cambiar las fronteras puede ser necesario mover una partícula a un proceso cuyo subdominio asociado no es contiguo al subdominio del proceso origen.

Hacer efectiva la nueva descomposición de dominio significa actualizar las estructuras de datos que almacenan los puntos de corte para los ejes de coordenadas y reconfigurar las dimensiones de los subdominios. La classe `Comm` almacena en los atributos `xsplit`,

`ysplit` y `zsplit` los valores de los puntos de corte en los ejes x, y y z, respectivamente. Estos atributos son vectores de números reales que contienen, respectivamente, $PARTS_1 + 1$, $PARTS_2 + 1$ y $PARTS_3 + 1$ elementos. El método `Verlet::applyDomain()` escribe los nuevos puntos de corte en los respectivos atributos e invoca los métodos necesarios para reconfigurar las dimensiones de los subdominios. La asignación de procesos a subdominios (subregiones) no cambia. El algoritmo 11 presenta un esquema del método `applyDomain()`.

Algoritmo 10 Esquema del método `Verlet::run()` modificado

```
 1: void Verlet::run(int n) {
 2: (...)
 3: for i := 0 to (n − 1) do
 4:    (...)
 5:    nflag := neighbor→decide();
 6:    if nflag = 0 then
 7:       (...)
 8:    else
 9:       if applyNewDomain then
10:          applyDomain();
11:          irregular→migrate_atoms();
12:       else
13:          comm→exchange();
14:          (...)
15:       end if
16:       comm→borders();
17:       (...)
18:    end if
19: end for
20: (...)
21: }
```

Una nueva descomposición de dominio no interfiere en la posición de las partículas en el espacio. Si, previo al ajuste de las fronteras, las coordenadas de una partícula m son (x_m, y_m, z_m) y representan un punto en el espacio que está contenido en un subdominio r, después que las nuevas fronteras han sido aplicadas las coordenadas de m seguirán siendo las mismas. Sin embargo, la posición en el espacio que corresponde a las coordenadas de m puede estar contenida en otro subdominio s. En este caso la partícula m deja de estar en el contexto del proceso que fue asignado a r y debe ser movida al contexto del proceso que fue asignado a s. Además, puede ocurrir que r y s no son subdominios adyacentes. En tal caso, es necesario intercambiar mensajes entre procesos que no son vecinos directos.

El esquema del algoritmo 10 permite observar una decisión de diseño tomada en este proyecto de grado respecto del momento en que una nueva descomposición de dominio se hace efectiva. El método `Verlet::generate_split()` es ejecutado en intervalos regulares definidos por el usuario, inmediatamente antes de finalizar una iteración del ciclo de integración. LIGGGHTS vuelve a crear la lista de vecinos de cada partícula en intervalos regulares de 10 pasos de tiempo y se verifica si una nueva descomposición de dominio fue generada solo si las listas de vecinos deben creadas nuevamente (la estructura de control

`if-then-else` en la línea 6 verifica si deben las listas de vecinos deben ser recreadas). Por lo tanto, el paso de tiempo en que una nueva descomposición de dominio debe ser aplicada puede no coincidir con el paso de tiempo en que se recrean las listas de vecinos. En este caso la aplicación de la nueva descomposición es retrasada hasta que ambos intervalos de pasos de tiempo estén sincronizados.

Algoritmo 11 Esquema del método `Verlet::applyDomain()`

```
1: for i := 0 to PARTS_1 + 1 do
2:    xsplit(i) := newXSplit(i);
3: end for
4: for i := 0 to PARTS_2 + 1 do
5:    ysplit(i) := newYSplit(i);
6: end for
7: for i := 0 to PARTS_3 + 1 do
8:    zsplit(i) := newZSplit(i);
9: end for
10: domain→reset_box();
11: comm→setup();
```

No se espera que la decisión tomada sobre el momento de hacer efectiva una descomposición de dominio tenga impactos negativos en las simulaciones. En general, el total de pasos de tiempo utilizados en una simulación es muy superior a la amplitud del intervalo para la recreación de listas de vecinos (por lo menos cuatro órdenes de magnitud). Además, como ha sido expuesto en el capítulo 3, de modo de mantener la estabilidad numérica de una simulación DEM el desplazamiento de las partículas al final de un paso de tiempo es muy pequeño. Como consecuencia, durante el periodo de tiempo comprendido entre dos recreaciones consecutivas de las listas de vecinos, la distancia relativa entre pares de partículas p_i y p_j que pertenecen a la simulación no se incrementa demasiado. Por lo tanto, cuando ambos intervalos se sincronizan no se ha incrementado significativamente el desbalanceo de carga y la nueva descomposición de dominio aún es vigente. El capítulo 7 presenta los resultados experimentales obtenidos para el esquema de descomposición dinámica de dominio implementada en este proyecto de grado. Los resultados numéricos presentados corroboran que tal decisión de diseño no impacta en los mismos.

6.5. Tratamiento de mallas triangulares

Esta sección presenta la forma en que LIGGGHTS representa las superficies que son incorporadas en una simulación. Habitualmente los sistemas de simulación numérica utilizan un formato de representación de superficies que posee propiedades geométricas que pueden ser explotadas. La sección 6.5.1 explica como LIGGGHTS representa las superficies incorporadas en una simulación. La sección 6.5.2 muestra como el formato de representación de superficies utilizado en LIGGGHTS influye en la estrategia de descomposición de dominio implementada en este proyecto de grado.

6.5.1. Incorporando mallas triangulares en LIGGGHTS

Uno de los principales objetivos de LIGGGHTS, además de servir como un sistema de simulación numérica que ofrece soporte completo al método DEM, es poder ser utilizado como una herramienta de simulación a nivel industrial. Para ello, LIGGGHTS incluye funcionalidades que permiten incorporar a las simulaciones modelos de superficies complejas [19, 22]. Las superficies incorporadas son modeladas utilizando mallas geométricas, en particular mallas triangulares (*triangle meshes*): las superficies son representadas como un conjunto de triángulos que pueden compartir aristas y vértices.

Las mallas triangulares son habituales en sistemas de simulación numérica debido a que los contactos entre partículas y superficies pueden ser detectados más rápidamente usando polígonos. Un triángulo de una malla es representado a través de las coordenadas de sus vértices. Para incluir los modelos de superficie en una simulación el usuario de LIGGGHTS debe usar el comando `fix mesh/surface` para indicar un archivo de entrada que contiene la especificación de la malla. Los formatos de archivo soportados son STL [103] y VTK [104].

Las superficies modeladas pueden ser estáticas o móviles, y aunque el modelo de malla utilizado en ambos casos es el mismo, la forma de procesarlas es diferente. Es necesario contemplar el procesamiento paralelo de las mallas de forma similar al procesamiento de otros componentes de la simulación (por ejemplo, partículas y cálculo de fuerzas). Esto significa que la estrategia de descomposición de dominio tiene impacto directo en el funcionamiento de los algoritmos de tratamiento de mallas triangulares.

La variante LIGGGHTS-PUBLIC puede procesar mallas estáticas y móviles sin inconvenientes usando la estrategia de descomposición estática de dominio. Así como ocurre con las partículas, no se esperan movimientos abruptos de los vértices de los triángulos de las mallas móviles y es suficiente implementar el intercambio de mensajes interprocesos con un patrón de comunicación similar al patrón regular. Sin embargo, en el caso de mallas triangulares la clase `Comm` no es utilizada para el procesamiento paralelo de las mallas y el código que implementa las funcionalidades de comunicación interprocesos se encuentra en una clase separada. Cualquier clase que implemente los métodos de comunicación para mallas triangulares solo necesita contemplar las mallas móviles porque una vez que los elementos de una malla estática son asignados a un proceso, esta asignación no cambia durante la simulación (los componentes de los triángulos de la malla no cambian de proceso) y no ocurren comunicaciones interprocesos debido al procesamiento de mallas triangulares.

La estategia de descomposición de dominio propuesta en este proyecto de grado requiere que los procesos intercambien mensajes de forma diferente a la propuesta originalmente por LIGGGHTS. Por lo tanto, fue necesario implementar nuevos algoritmos de comunicación interprocesos para el tratamiento de mallas. Los conceptos aplicados en el patrón Irregular de comunicación son reutilizados. Sin embargo, no se puede reutilizar completamente la clase `Irregular`.

Las clases que implementan las funcionales para tratamiento de mallas triangulares están organizadas en una jerarquía basada en el paradigma de programación orientada a objetos. Las clases más especializadas definen concretamente las superficies que pueden ser usadas. Los volúmenes son modelados por la clase `TetMesh`, las superficies planas son modeladas por la clase `TriMeshPlanar` y las superficies no planas son modeladas por la clase `TriMesh`. La figura 6.8 muestra cómo está organiza la jerarquía de clases y se destacan en color rojo las clases más especializadas.

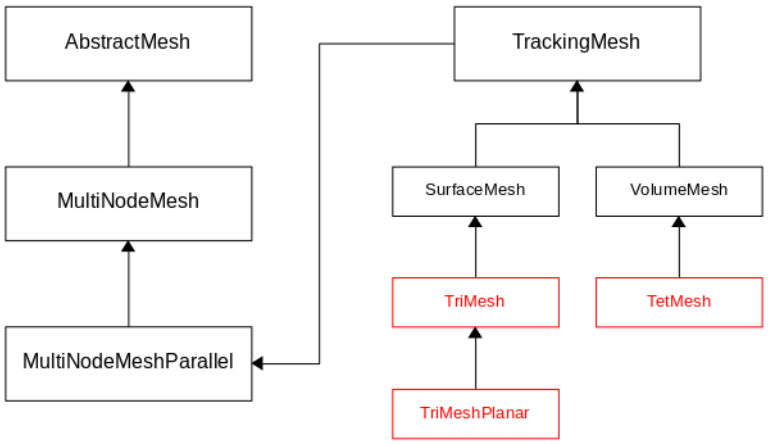


Figura 6.8: Jerarquía de clases para tratamiento de mallas triangulares en LIGGGHTS

Para implementar un mecanismo de comunicación interprocesos que contemplara las mallas en un contexto de descomposición dinámica de dominio fue necesario cambiar el método `MultiNodeParallelMesh::pbcExchangeBorders()`. Para las partículas, la comunicación se hace a través de métodos de diferentes clases que implementan separadamente diferentes etapas del intercambio de mensajes. Para las mallas geométricas la decisión tomada por los desarrolladores de LIGGGHTS fue concentrar todas las etapas de comunicación en el método `pbcExchangeBorders()`. Por aspectos de organización del código, algunas tareas puntuales están implementadas en métodos auxiliares de la clase `MultiNodeParallelMesh`.

Los métodos agregados en la versión desarrollada en este proyecto de grado para el intercambio de mensajes para el procesamiento de mallas triangulares tratan de no cambiar las decisiones de diseño tomadas originalmente. Para ello, solo se realizan cambios en la clase `MultiNodeParallelMesh`. El principal método agregado para el procesamiento de mallas triangulares en el contexto de una estrategia de descomposición dinámica de dominio se llama `migrate_elements()`, que se describe en la sección 6.5.2. Este nuevo método no sustituye al algortimo original implementado y solo es necesario utilizarlo cuando una nueva descomposición de dominio fue generada. Por lo tanto, el código debe verificar cuál método debe ser invocado. El algoritmo 12 muestra un esquema del método `pbcExchangeBorders()` luego de que los ajustes necesarios fueron implementados. Los métodos `setup()` y `borders()` son originales de LIGGGHTS y se explica más adelante cuáles son sus funciones.

Algoritmo 12 Esquema del método `pbcExchangeBorders`

```
1: (...)
2: setup();
3: if applyNewDomain then
4:     migrate_elements();
5: else
6:     exchange();
7: end if
8: borders();
9: (...)
```

La revisión del código fuente de LIGGGHTS permite corroborar que algunas funcionalidades con objetivo similar (por ejemplo, la intercomunicación entre procesos durante la etapa de cómputo) son implementadas en métodos separados según el elemento sobre el cuál operan (partículas o elementos de mallas triangulares). Se puede concluir que esta decisión ha sido tomada por los desarrolladores del sistema para mantener desacoplado el código para el método DEM y el código que expande las funcionalidades de LIGGGHTS. Así es posible, por ejemplo, sustituir el código de manejo de mallas triangulares sin afectar la correctitud del código para DEM.

6.5.2. Intercambio de elementos de mallas triangulares

En el contexto de una estrategia de descomposición dinámica de dominio, el intercambio de elementos de mallas triangulares se realiza usando el método `migrate_elements`. Su implementación está basada en tres métodos auxiliares: (1) `create_elements`, (2) `exchange_elements` y (3) `destroy_elements`. Estos métodos no pertenecen a la versión original de LIGGGHTS y son parte integrante de la solución propuesta. Sus implementaciones están basadas, respectivamente, en los métodos `migrate_atoms`, `create_data`, `exchange_data` y `destroy_data` implementados en la clase `Irregular`. Así como ocurre con las partículas, cuando se aplica una nueva descomposición de dominio puede ocurrir que un elemento de una malla que está en el contexto de un proceso p_i pasa a estar en el contexto de un proceso p_j que no es vecino directo de p_i. La solución propuesta en este proyecto de grado es aplicar un mecanismo iterativo de intercambio de mensajes. Las iteraciones ocurren sobre los ejes de coordenadas siguiendo el orden x, y y z.

Para cada eje de coordenadas se verifica si un elemento deja de estar en el contexto del proceso que ejecuta el método `migrate_elements` y, si este es el caso, el elemento es agregado a un *buffer*. Después que los elementos correspondientes fueron agregados a ese *buffer*, se genera un *plan de comunicación* y ocurre el intercambio de mensajes. Los elementos recibidos son agregados al contexto del proceso que invoca a `migrate_elements` y, finalmente, el *buffer* es eliminado. El algoritmo 13 muestra un esquema del método `migrate_elements` ejecutado para el eje de coordenadas `dim`.

La estructura de control `while` de la línea 5 recorre todos los elementos que están en el contexto del proceso que invoca a `migrate_elements` y la estructura de control `if-then-else` de la línea 6 verifica si el centro de cada elemento está fuera de las dimensiones del subdominio respecto del eje `dim`. Si la condición de verificación resulta verdadera, el método `coord2proc` es invocado para determinar a cuál proceso debe ser enviado el elemento para el cual la posición de su centro está siendo verificada y el método `pushElemToBuffer` lo agrega al *buffer* `buf_send`. El vector `proclist` almacena en la posición j el proceso al cuál debe ser enviado el j-ésimo elemento del *buffer* `buff_send`. Las variables `checklo` y `checkhi` almacenan, respectivamente, los extremos inferior y superior del subdominio correspondiente respecto del eje de coordenadas `dim`, y la variable `nLocal` almacena el total de elementos de mallas que pertenecen al contexto del proceso que invoca `migrate_elements`.

La estructura de control `while` de la línea 17 extrae los elementos recibidos, que fueron almacenados en el *buffer* `buf_recv`. Estos elementos pasan a estar en el contexto del proceso que ejecuta `migrate_elements`. Se destaca el uso de métodos los auxiliares `pushElemToBuffer` y `popElemFromBuffer` para manejar los *buffers* de elementos de mallas. Estos métodos son parte de LIGGGHTS y están dedicados especialmente para operar con elementos de mallas triangulares.

Algoritmo 13 Esquema del método `migrate_elements` para el eje `dim`

```
 1: checklo := sublo(dim);
 2: checkhi := subhi(dim);
 3: nsendelem := 0;
 4: i := 0;
 5: while i < nLocal do
 6:    if center(i,dim) ⩾ checklo && center(i,dim) ⩽ checkhi then
 7:       proclist(nsendelem) = coord2proc(center(i));
 8:       pushElemToBuffer(i,buf_send));
 9:       nsendelem++;
10:    end if
11:    i++;
12: end while
13: create_elements(nsendelem, proclist);
14: exchange_elements(buf_send, buf_recv);
15: destroy_elements();
16: i := 0;
17: while i < nrecv do
18:    popElemFromBuffer(buf_recv);
19:    nLocal++;
20:    i++;
21: end while
```

El plan de comunicación utilizado para el intercambio de elementos de mallas tiene la misma estructura y cumple la misma función que los planes de comunicación utilizados en la clase `Irregular`. El método `create_elements` construye el plan de comunicación. Las variables de control que dan soporte a las instrucciones MPI invocadas por el método `exchange_elements` son configuradas con valores generados por el mismo proceso que ejecuta `migrate_elements` o por otros procesos que ejecutan simultáneamente otras instancias de `migrate_elements`. Para enviar y recibir los elementos de mallas, el método `exchange_elements` se basa en el plan de comunicación generado. Así como ocurre con el método `exchange_data`, el método `exchange_elements` itera sobre los *buffers* de envío y recepción, y también está basado en las instrucciones `MPI_Send` y `MPI_Irecv`.

El método `exchange_elements` utiliza una instrucción no bloqueante para la recepción de mensajes por cuestiones de desempeño. De esta forma un proceso puede empezar a enviar los elementos correspondientes apenas haya terminado de recibir los elementos que pasan a estar en su contexto. Sin embargo, es necesario utilizar la instrucción `MPI_Waitall` para evitar que `exchange_elements` termine su ejecución sin haber recibido todos los elementos necesarios. El algoritmo 14 muestra un esquema del método `exchange_elements`.

Los atributos de control para el envío de mensajes (`length_send` y `proc_send`) y los atributos de control para la recepcíon de mensajes (`length_recv`, `nrecv`, `proc_recv` y `request`) almacenan valores calculados durante la creación del plan de comunicación. La tabla 6.1 presenta una breve descripcción de la función de esos atributos.

El último método invocado por `migrate_elements` es `destroy_elements`. El objetivo de `destroy_elements` es liberar la memoria asignada para la contrucción del plan de comunicación y está basado en invocaciones sucesivas de la instrucción `free` de C++.

Algoritmo 14 Esquema del método `exchange_elements`

```
1: offset := 0;
2: for irecv := 0 to nrecv do
3:    MPI_Irecv(buf_send(offset), length_recv(irecv), MPI_DOUBLE, proc_recv(irecv),
          0, MPI_COMM_WORLD, request(irecv));
4:    offset + = length_recv(irecv);
5: end for
6: offset := 0;
7: for isend := 0 to nsend do
8:    MPI_Send(buf_recv(offset), length_send(isend), MPI_DOUBLE, proc_send(isend),
          0, MPI_COMM_WORLD);
9:    offset + = length_recv(isend);
10: end for
11: if nrecv then
12:    MPI_Waitall(nrecv, request, status);
13: end if
```

Atributo	Descripcíon
`length_send`	Vector que almacena en la posición j el largo del j-ésimo mensaje a enviar
`proc_send`	Vector que almacena en la posición j el rango del proceso hacia dónde se envia el j-ésimo mensaje
`length_recv`	Vector que almacena en la posición j el largo del j-ésimo mensaje a recibir
`nrecv`	Cantidad de elementos a recibir
`proc_nrecv`	Vector que almacena en la posición j el rango del proceso desde dónde se recibe el j-ésimo mensaje
`request`	Variable de control para verificar que todos los mensajes fueron recibidos

Tabla 6.1: Comandos que permiten alterar la actualización de listas de vecinos

6.6. Comandos de interfaz de usuario

Esta sección introduce los comandos agregados a LIGGGHTS para permitir al usuario configurar y utilizar la estrategia de descomposición de dominio implementada en este proyecto de grado. Dado que la variante LIGGGHTS-PUBLIC no posee soporte a descomposición dinámica de dominio, esta variante no ofrece al usuario ningún método de modificar la descomposición de dominio durante la ejecución de una simulación. Por lo tanto, cualquier estrategia de descomposición de dominio incorporada a LIGGGHTS-PUBLIC debe proveer al usuario los comandos necesarios para su configuración.

6.6.1. Introducción

El usuario de LIGGGHTS configura el comportamiento del sistema y las características de la simulación a través de un archivo de texto que es procesado por el método `Input::file()`. Las funciones de este método son: (1) corroborar la correctitud de los comandos declarados por el usuario y leer los parámetros incluidos en las respectivas declaraciones, (2) definir parámetros de control de LIGGGHTS y (3) invocar otros métodos de configuración del sistema y de la simulación.

La descomposición del dominio se genera en base a los parámetros definidos para los comandos `processors` y `region`. No existen comandos originales de LIGGGHTS que permiten cambiar dinámicamente la descomposición de dominio. Para que el usuario pueda utilizar la estrategia de descomposición dinámica de dominio propuesta en este proyecto de grado fueron incluidos los comandos `cload` y `dddecomp`.

Cuando el método `file()` encuentra el comando `cload` en el archivo de entrada el método `Input::cload()` es invocado y los parámetros correspondientes son configurados. Un comportamiento similar se observa cuando el comando `dddecomp` es encontrado en el archivo de entrada. En este caso el método `Input::dddecomp()` es invocado. Las instrucciones de uso de los comandos y los respectivos parámetros de configuración son presentados, respectivamente, en las subsecciones 6.6.2 y 6.6.3.

6.6.2. El comando `cload`

El comando `cload` permite al usuario configurar los parámetros del algortimo de cálculo de carga de trabajo. La sintaxis del comando `cload` se muestra en el listado 6.6.2.

```
cload no | yes call_freq <VALOR>
```

Listado 6.1: Sintaxis del comando `cload`

El algoritmo de descomposición dinámica de dominio depende de este comando. Si el cálculo de cargas de trabajo no está habilitado y se trata de configurar el comando `dddecomp` con el parámetro `yes` se producirá un error en tiempo de ejecución. La sección 6.6.3 describe el funcionamiento del comando `dddecomp`.

El valor del parámetro `call_freq` del comando `cload` debe ser un divisor del parámetro `call_freq` del comando `dddecomp`. Si esta condición no se cumple se producirá un error en tiempo de ejecución.

6.6.3. El comando `dddecomp`

El comando `dddecomp` permite al usuario habilitar y configurar el algoritmo de descomposición dinámica de dominio. Su valor por defecto es `no` y significa que LIGGGHTS usará el algoritmo estático de descomposición de dominio. La sintaxis del comando `dddecomp` se presenta en el listado 6.6.3.

```
dddecomp no | yes call_freq <VALOR> [stats_file <NOMBRE_ARCHIVO>]
```

Listado 6.2: Sintaxis del comando `dddecomp`

El valor del parámetro `call_freq` del comando `dddecomp` debe ser un múltiplo del valor del parámetro `call_freq` del comando `cload`. Si esta condición no se cumple se producirá un error en tiempo de ejecución.

El parámetro `stats_file` es opcional y permite definir un archivo donde se almacena información sobre la descomposición de dominio. Cada vez que una nueva descomposición de dominio es generada, la información correspondiente es almacenada en el archivo definido por el usuario.

Capítulo 7

Análisis experimental

Este capítulo presenta los resultados obtenidos en los experimentos realizados para la validación de la estrategia de descomposición dinámica de dominio propuesta en este proyecto de grado. Se consideran tres problemas de dinámica granular: (1) dos problemas estándares de prueba (*benchmarks*) que son distribuidos junto con el código fuente de LIGGGHTS y (2) un problema de dinámica granular propuesto por los investigadores del IMFIA. La sección 7.1 presenta la metodología aplicada para validación de los resultados y las secciones 7.2 y 7.3 describen detalladamente los problemas considerados y los resultados obtenidos.

7.1. Metodología para la validación de resultados

Para cada problema considerado se define una configuración para las simulaciones ejecutadas y se crean dos escenarios que se diferencian por la estrategia de descomposición de dominio utilizada: (1) el escenario estático corresponde a simulaciones que utilizan descomposición estática de dominio y (2) el escenario dinámico corresponde a simulaciones que utilizan descomposición dinámica de dominio. Para cada escenario de un problema se ejecutan múltiples instancias utilizando siempre la configuración que ha sido definida para el problema considerado. El número de ejecuciones realizadas depende de las dimensiones del problema, el tiempo de ejecución promedio necesario para ejecutar una simulación y la disponibilidad de recursos de cómputo.

Uno de los parámetros de configuración de una simulación es la distribución de procesos en cada eje de coordenadas. Este parámetro se configura a través del comando `processors` y definiendo los valores $PROCS_1$, $PROCS_2$ y $PROCS_3$ tales que $E = PROCS_1 \times PROCS_2 \times PROCS_3$, donde E es la cantidad de procesos a utilizar en la simulación y $PROCS_j$ define la cantidad de procesos asignados al eje de coordenadas j. Los valores $PROCS_1$, $PROCS_2$ y $PROCS_3$ se leen desde los parámetros del comando `processors` y deben ser declarados en el archivo de entrada para la simulación. La sección 4.2 explica cómo el comando `processors` permite definir la descomposición inicial de dominio.

Berger et al. [19] mostraron que diferentes permutaciones de los parámetros del comando `processors` influyen en el tiempo de cómputo de una simulación. La propuesta de proyecto de grado no incluye una evaluación del comportamiento de la estrategia de descomposición dinámica de dominio implementada según los parámetros del comando `processors`. Todas las instancias de cada escenario de un problema son evaluadas

usando siempre la configuración del comando `processors` que ha sido definida para ese problema.

Un segundo parámetro muy importante que debe ser configurado por el usuario es el valor D aplicado en la discretización de subdominios al momento de definir nuevas fronteras de subdominios. La selección 6.4.1 explicó cómo este parámetro es utilizado por la estrategia de descomposición dinámica de dominio. En la etapa de validación se definió $D = 160$. Este valor fue definido empíricamente en base a los resultados obtenidos durante las pruebas realizadas en la etapa de implementación de la estrategia propuesta.

Las métricas evaluadas son: (a) el tiempo de ejecución, (b) la distribución de la carga de trabajo durante la simulación y (c) la escalabilidad respecto del número de procesos. Las ejecuciones se realizan en el Cluster FING [53] y debido a la heterogeneidad de los nodos de cómputo disponibles, las ejecuciones de las instancias de un problema determinado se llevan a cabo siempre en un mismo conjunto de nodos. Los nodos contenidos en un conjunto dependen del problema considerado para validación de la estrategia de descomposición de dominio implementada. El análisis reporta los resultados obtenidos y una descripción de los nodos de cómputo utilizados. Las herramientas de usuario disponibles en el Cluster FING permiten garantizar que el conjunto de nodos de cómputo para un problema en particular no cambie.

Una característica común de los nodos de cómputo disponibles en el Cluster FING es la presencia de procesadores multi-núcleos que varían entre 8 y 64 núcleos por procesador. En este contexto, cada proceso MPI es ejecutado en un núcleo y no se ejecuta más de un proceso por núcleo. Por lo tanto, en la etapa de validación y análisis experimental una simulación que ejecuta n procesos requiere un conjunto de nodos que pueda ofrecer n núcleos. La asignación de recursos de cómputo es gestionada por el gestor de recursos del Cluster FING y todas las simulaciones para validación del código desarrollado en este proyecto de grado son ejecutadas a través del gestor de recursos.

Las métricas evaluadas permiten corroborar la eficacia del algoritmo de descomposición dinámica de dominio. Sin embargo, no son suficientes para corroborar la correctitud de los resultados numéricos de la implementación. El principal objetivo de este proyecto de grado es mejorar el desempeño de una herramienta computacional utilizada en simulaciones numéricas sin afectar negativamente los resultados numéricos obtenidos. Para validar los resultados numéricos se aplican técnicas de posprocesamiento que permiten analisar los datos de salida y visualizar los resultados obtenidos. LIGGGHTS permite escribir los resultados en disco usando el formato VTK [104] de modo que un amplio conjunto de herramientas pueda ser utilizado en la etapa de posprocesamiento. En este proyecto de grado se validan los datos de salida usando la aplicación ParaView [105].

Las simulaciones del método DEM son de cómputo intensivo y el manejo de entrada y salida no es un cuello de botella. El volumen de datos almacenados al final de una simulación puede ser muy grande (aproximadamente 500GB), pero la cantidad de información que debe ser escrita en disco entre dos pasos de tiempo consecutivos es en general pequeña (aproximadamente 50KB). De este modo, las características importantes de los nodos de cómputo resultaron ser el modelo de procesador, la cantidad de núcleos disponibles y la cantidad de memoria RAM disponible (la cantidad de memoria condiciona las dimensiones de una simulación). Estas características tuvieron un rol fundamental al momento de seleccionar los nodos de cómputo adecuados para las ejecuciones.

La etapa de validación no estudia la influencia de las comunicaciones de red en los tiempos de ejecución de las simulaciones. El tráfico de red entre nodos de cómputo

es habitual en ambientes de computación de alto desempeño y permite que un alto número de nodos puedan ser utilizados simultáneamente en la resolución de un problema. Además, los nodos de cómputo utilizados en este proyecto de grado poseen 8 o 24 núcleos. Por lo tanto, no se ejecutaron simulaciones que necesitaran más de 24 núcleos, de modo a garantizar que no hubiera comunicación entre nodos de cómputo durante la ejecución de una simulación. Otro factor que determinó el número máximo de núcleos utilizados para una simulación es el hecho que los nodos del Cluster FING con más de 24 núcleos son muy demandados y la ejecución de todas las instancias de una versión requeriria demasiado tiempo de espera en el gestor del cluster.

7.2. Problemas de validación (*benchmarks*)

El código fuente de LIGGGHTS incluye un conjunto de problemas para validación y análisis que pueden ser usados con diferentes objetivos. Cada problema viene acompañado de un archivo de configuración que contiene explicaciones detalladas sobre el funcionamiento de los comandos que fueron declarados en ese archivo. Esta sección describe los problemas utilizados para corroborar la eficiencia computacional de la estrategia de descomposición de dominio implementada en este proyecto de grado.

7.2.1. Descripción de los problemas de validación (*benchmark*)

El problema de validación elegido se llama *binflow* y pertenece al conjunto de problemas de validación disponibilizados junto con el código fuente de LIGGGHTS. El problema *binflow* simula la carga y descarga de un silo. Esta sección presenta las características físico-químicas de los materiales utilizados en las simulaciones ejecutadas. Estos materiales son utilizados en las diferentes versiones del problema *binflown* generadas. Las secciones 7.2.2 y 7.2.3 presentan los detalles específicos de cada versión y reportan los respectivos resultados obtenidos.

La figura 7.1 muestra las dimensiones del silo utilizado en este problema. Se trata de un silo con dos secciones y un orificio de salida concéntrico. La primera sección (superior) tiene forma cilíndrica y la segunda sección (inferior) tiene forma afunilada.

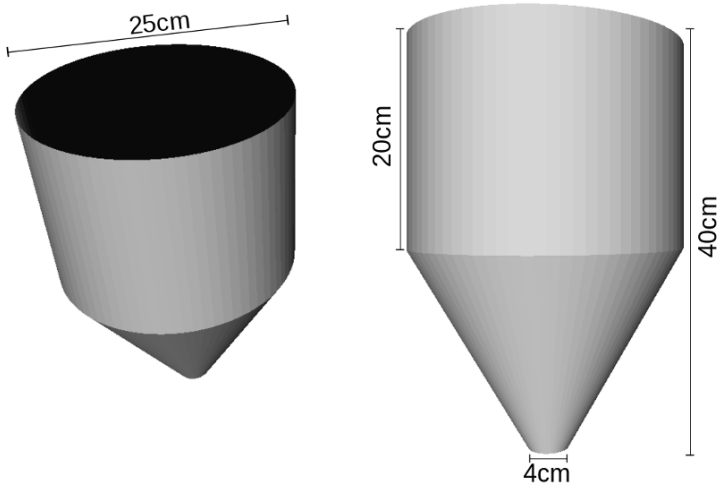


Figura 7.1: Dimensiones del silo para el problema *binflow*

Las partículas son insertadas en la parte superior del silo con velocidad inicial $\vec{v_0} = (0, 0, -1)$ y se desplazan hacia el fondo del silo en un movimiento de caída libre. Al configurar la inserción de partículas asignándoles una velocidad inicial con dirección hacia el fondo del silo, se simula un procedimiento de carga habitualmente encontrado en la industria, donde las partículas llegan a la parte superior del silo a través de una cinta transportadora y son introducidas por un orificio. La etapa de descarga empieza una vez que la etapa de carga ha finalizado y las partículas dejan el silo a través del orificio que se encuentra en el fondo de la estructura.

El principal objetivo del análisis experimental es estudiar el desempeño computacional de la estrategia de descomposición dinámica de dominio implementada, utilizando el *benchmark* elegido. Las propiedades físico-químicas de los materiales presentes en las simulaciones no corresponden, necesariamente, a materiales utilizados en situaciones reales. Los parámetros de configuracíon del problema *binflow*, tales como el módulo de Young o los coeficientes de fricción, pueden ser ajustados caso se necesario. Para este proyecto no se realizaron cambios en los valores estándares que ya se encuentran definidos. La tabla 7.1 presenta los valores utilizados para las simulaciones realizadas.

Propiedad	**Valor**
Módulo de Young	$25MPa$
Coef. de fricción – partícula-partícula	0,175
Coef. de fricción – partícula-superficie	0,2
Densidad de partícula	$1000kg/m^3$

Tabla 7.1: Propiedades físico-químicas del problema *binflow*

Los únicos parámetros modificados para ejecutar las simulaciones fueron: (1) diámetro de partícula y (2) número de partículas contenidas en el sistema. Las modificaciones realizadas tienen como objetivo generar deliberadamente un aumento en la carga de cómputo que permita estudiar la robustez de la estrategia de descomposición dinámica de dominio implementada respecto de cambios en estas dos variables. Las secciones 7.2.2 y 7.2.3 presentan los resultados obtenidos para experimentos con partículas de igual radio y para partículas con diferentes radios, respectivamente.

7.2.2. Partículas con igual radio

Inicialmente, el problema *binflow* está configurado para insertar 150000 partículas de radio 1,5 mm. Una segunda configuración fue definida para insertar un total de 450000 partículas con radio 1,5 mm. Como se explicó en la sección 7.1, se definen dos escenarios para cada problema estudiado: (1) escenario estático y (2) escenario dinámico. Los escenarios se diferencian por la estrategia de descomposición de dominio utilizada.

A partir de las configuraciones generadas se definieron cuatro versiones del problema *binflow* para el escenario estático y cuatro versiones para el dinámico. Las tablas 7.2 y 7.3 reportan las configuraciones de las versiones del problema *binflow* según la estrategia de descomposición de dominio utilizada. Las columnas `cload` y `dddecomp` reportan, cuando corresponde, los parámetros de configuración utilizados para los comandos `cload` y `dddecomp`, respectivamente. Las columnas `#part.` y `Cfg.` reportan el total de partícu-

las insertadas y la configuración de procesos definidad en el archivo de entrada de una versión del problema *binflow*, respectivamente. Las columnas `Total` y `Dur.(s)` reportan el número de pasos de tiempo considerados en una simulación y la duración, en segundos, de cada paso de tiempo, respectivamente.

Versión	**#part.**	**Pasos de tiempo**		**Procesos**		**Frecuencias**	
		Total	Dur.(s)	#	Cfg.	`cload`	`dddecomp`
bf_serial	$1{,}5\times10^5$	$3{,}5\times10^5$	$5{,}0\times10^{-6}$	1	–	–	–
bf_8procs	$1{,}5\times10^5$	$3{,}5\times10^5$	$5{,}0\times10^{-6}$	8	$2\times2\times2$	20000	–
bf_16procs	$1{,}5\times10^5$	$3{,}5\times10^5$	$5{,}0\times10^{-6}$	16	$2\times2\times4$	25000	–
bf_24procs	$1{,}5\times10^5$	$3{,}5\times10^5$	$5{,}0\times10^{-6}$	24	$2\times2\times8$	25000	–
bf_450000part	$4{,}5\times10^5$	$1{,}35\times10^7$	$5{,}0\times10^{-6}$	16	$2\times2\times4$	25000	–

Tabla 7.2: Configuración del escenario estático del problema *binflow*

Versión	**#part.**	**Pasos de tiempo**		**Procesos**		**Frecuencias**	
		Total	Dur.(s)	#	Cfg.	`cload`	`dddecomp`
bf_mp_8procs	$1{,}5\times10^5$	$3{,}5\times10^5$	$5{,}0\times10^{-6}$	8	$2\times2\times2$	20000	100000
bf_mp_16procs	$1{,}5\times10^5$	$3{,}5\times10^5$	$5{,}0\times10^{-6}$	16	$2\times2\times4$	25000	50000
bf_mp_24procs	$1{,}5\times10^5$	$3{,}5\times10^5$	$5{,}0\times10^{-6}$	24	$2\times2\times8$	25000	50000
bf_mp_450000part	$4{,}5\times10^5$	$1{,}35\times10^7$	$5{,}0\times10^{-6}$	16	$2\times2\times4$	25000	100000

Tabla 7.3: Configuración del escenario dinámico del problema *binflow*

Fue creada, además, una versión especial del problema *binflow* con 150000 partículas. Esta versión se llama bf_serial y corresponde a ejecuciones seriales de la simulación. Su objetivo es servir de base de comparación para verificar la eficiencia computacional del algoritmo implementado. Berger et al. [19] reportan los tiempos de ejecución de una versión serial de un problema similar. Sin embargo, no es posible utilizar los resultados reportados porque el número de partículas y la duración de cada paso de tiempo son diferentes de los valores utilizados en este proyecto de grado. Dado que bf_serial es una versión serial del problema, las simulaciones correspondientes a este problema pueden ser ejecutadas aplicando ambas estrategias de descomposición de dominios. El tiempo de ejecución es siempre el mismo porque se utiliza un solo proceso. Por lo tanto, las configuraciones de la versión bf_serial es reportada únicamente en la tabla 7.2.

Para las versiones con 150000 partículas se realizaron ejecuciones para 30 instancias y para las versiones con 450000 partículas se realizaron ejecuciones para 20 instancias. Las instancias de las versiones bf_serial, bf_8procs y bf_mp_8procs fueron ejecutadas en nodos del Cluster FING equipados con procesador Intel Xeon modelo E5430 de 8 núcleos de 2,66GHz y un total 8GB de memoria RAM. Las instancias de las vesiones bf_16procs, bf_mp_16procs, bf_24procs, bf_mp_24procs, bf_450000part y bf_mp_450000part fueron ejecutadas en nodos equipados con procesador AMD Opteron modelo 6172 de 24 núcleos de 2,10GHz y un total de 24GB de memoria RAM. Las variables evaluadas para la selección

de un nodo de cómputo para la ejecución de una simulación son: (1) la disponibilidad de los recursos de cómputo del Cluster FING, (2) la cantidad de núcleos y de memoria RAM disponibles en los nodos de cómputo y (3) la cantidad de procesos que ejecutan la simulación. Los criterios de selección de los nodos de cómputo son: (a) la cantidad de núcleos disponibles debe ser mayor o igual a la cantidad de procesos necesarios para ejecutar la simulación y (b) no deben existir núcleos ociosos. En los casos en que el criterio (b) no pudo ser satisfecho, se utilizó algún nodo cuya cantidad de núcleos disponibles fuera la menor posible que cumpliera el criterio (a).

La tabla 7.4 reporta los tiempos de procesamiento de las instancias de simulación ejecutadas para las versiones bf_8procs, bf_mp_8procs, bf_16procs, bf_mp_16procs, bf_24procs y bf_mp_24procs. La columna `Inst.` reporta los identificadores de las instancias.

Los resultados reportados en la tabla 7.4 indican que los tiempos de ejecución de las simulaciones de las versiones bf_16procs y bf_mp_16procs tienen mayor desviación estándar que las simulaciones de las versiones bf_8procs, bf_mp_8procs, bf_24procs y bf_mp_24procs. Este fenómeno es corroborado por las diferencias entre los tiempos de ejecución mínimo y máximo que se reportan en la tabla 7.8. Los valores de desviación estándar para las versiones bf_16procs y bf_mp_16procs se explican por el hecho que las instancias de estas versiones requieren 16 procesos y son ejecutadas en nodos de cómputo equipados con procesadores de 24 núcleos. De este modo, simultáneamente a la ejecución de una instancia, pueden encontrarse en ejecución procesos de otros usuarios que no están relacionados con las simulaciones. Los procesos ejecutados en un mismo nodo de cómputo no comparten CPU, pero sí pueden compartir otros recursos (acceso a memoria RAM, acceso a discos de almacenamiento, comunicación a través de la red, etc.), causando desviaciones considerables en los tiempos de ejecución de las simulaciones de las versiones bf_16procs y bf_mp_16procs. Un comportamiento similar ocurre con las versiones bf_serial, bf_450000part y bf_mp_450000part. Los tiempos de procedimiento de estas versiones están reportados en las tablas 7.5, 7.6 y 7.7, respectivamente.

En el caso de las versiones bf_8procs y bf_mp_8procs no quedan núcleos libres en los nodos de cómputo dado que se ejecutan simulaciones de 8 procesos en nodos equipados con procesadores de 8 núcleos. Por lo tanto, una vez que un nodo empieza la ejecución de una simulación de las versiones bf_8procs y bf_mp_8procs, no ejecutará procesos de otros usuarios hasta que la simulación no haya finalizado. Lo mismo ocurre con las simulaciones ejecutadas para las versiones bf_24procs y bf_mp_24procs. En este caso los 24 núcleos están dedicados a ejecutar una simulación y el nodo de cómputo no ejecuta procesos de otros usuarios hasta que la simulación no haya finalizado. Por los motivos explicados, no existe competencia por otros recursos que ocasionen en variaciones considerables en los tiempos de ejecución de las simulaciones de las versiones bf_8procs, bf_mp_8procs, bf_24procs y bf_mp_24procs.

La tabla 7.4 también muestra que el modo de uso de los nodos de cómputo puede influir en el desempeño relativo de simulaciones que aplican la estrategia de descomposición dinámica de dominio y que difieren únicamente en la cantidad de procesos en ejecución. Por ejemplo, se observan dos situaciones (instancias 15 y 18) donde el tiempo requerido por una simulación que utiliza 16 procesos (versión bf_mp_16procs) es superior al tiempo requerido por una simulación que utiliza 8 procesos (versión bf_mp_8procs). Cómo se explicó anteriormente en esta sección, este fenómeno puede ocurrir debido a que las simulaciones de 16 procesos no utilizan todos los núcleos disponibles en los nodos de

Inst.	Versión del problema *binflow*					
	bf_8procs	bf_mp_8procs	bf_16procs	bf_mp_16procs	bf_24procs	bf_mp_24procs
1	9:12	6:20	7:58	4:40	6:19	3:23
2	9:17	6:22	7:46	4:44	6:04	3:41
3	9:22	6:06	8:31	4:31	5:59	3:22
4	9:28	6:11	7:54	3:50	6:09	3:56
5	9:18	6:05	8:40	4:37	5:58	3:44
6	9:14	6:04	8:12	4:26	6:01	3:42
7	9:41	6:06	7:55	5:02	6:03	3:28
8	9:25	6:05	8:00	4:48	6:10	3:27
9	9:28	6:05	9:07	4:45	5:59	3:24
10	9:32	6:04	9:12	4:33	6:02	3:34
11	9:31	6:04	9:42	5:00	6:13	3:27
12	9:30	6:08	8:57	5:24	5:59	3:36
13	9:15	6:05	10:06	5:58	6:00	3:38
14	9:20	6:04	8:40	5:08	5:59	3:29
15	9:23	6:05	10:05	6:44	5:58	3:33
16	9:35	6:06	8:51	4:39	5:58	3:27
17	9:37	6:11	9:40	5:23	6:08	3:30
18	9:17	6:15	7:38	7:36	5:58	3:24
19	9:16	6:05	8:31	4:51	6:01	3:25
20	9:21	6:13	7:44	4:38	6:04	3:27
21	9:32	6:21	7:34	4:27	6:02	3:25
22	9:18	6:04	7:44	4:29	5:55	3:27
23	9:32	6:05	7:38	4:33	6:01	3:25
24	9:23	6:08	7:28	4:25	5:54	3:28
25	9:16	6:14	7:39	4:28	6:05	3:25
26	9:13	6:08	7:50	4:05	6:01	3:24
27	9:20	6:14	7:27	4:44	6:03	3:26
28	9:17	6:23	7:32	4:31	6:23	3:30
29	9:16	6:23	8:07	4:43	6:15	3:24
30	9:17	6:07	7:59	5:08	6:10	3:25

Tabla 7.4: Tiempos de ejecución para el problema *binflow* con 150000 partículas

cómputos y, por lo tanto, es posible que ocurra la ejecución simultánea de LIGGGHTS y de procesos de otros usuarios que compiten por diferentes recursos (acceso a memoria RAM, acceso a disco, comunicación a través de la red, etc.).

La tabla 7.5 reporta los tiempos de procesamiento de las instancias de simulación ejecutadas para las versión bf_serial. Las tablas 7.6 y 7.7 reportan los tiempos de proce-

samiento de las instancias de simulación ejecutadas para las versiones bf_450000part y bf_mp_450000part, respectivamente. Las columnas `Inst.` y `Tiempo de ejecución` reportan los identificadores de las instancias y los tiempos de ejecución de cada instacia, respectivamente.

Inst.	Tiempo de ejecucón	Inst.	Tiempo de ejecucón	Inst.	Tiempo de ejecucón
1	37:39	11	39:06	21	42:05
2	38:07	12	39:07	22	41:32
3	41:12	13	40:35	23	41:17
4	39:33	14	37:39	24	43:53
5	39:26	15	42:06	25	42:38
6	39:00	16	41:59	26	41:23
7	39:44	17	44:02	27	42:07
8	39:03	18	42:51	28	41:41
9	38:55	19	45:52	29	42:16
10	38:29	20	42:39	30	41:23

Tabla 7.5: Tiempos de ejecución para la versión bf_serial del problema *binflow*

Inst.	Tiempo de ejecucón	Inst.	Tiempo de ejecucón
1	87:10	11	87:04
2	82:26	12	86:16
3	82:38	13	83:31
4	80:50	14	87:17
5	83:27	15	80:24
6	80:28	16	89:42
7	84:13	17	87:04
8	88:30	18	83:43
9	86:33	19	84:53
10	81:30	20	83:02

Tabla 7.6: Tiempos de ejecución para la versión bf_450000part del problema *binflow*

Inst.	Tiempo de ejecucón	Inst.	Tiempo de ejecucón
1	54:30	11	55:56
2	52:45	12	52:40
3	51:03	13	52:23
4	51:15	14	50:00
5	53:59	15	50:02
6	53:56	16	57:34
7	50:41	17	58:17
8	53:55	18	55:40
9	52:28	19	52:17
10	55:22	20	55:22

Tabla 7.7: Tiempos de ejecución para la versión bf_mp_450000part del problema *binflow*

Las figuras 7.2 y 7.3 reportan gráficamente los resultados de tiempo de ejecución para las nueve versiones del problema *binflow*. La figura 7.2 reporta los tiempos de ejecución para las versiones bf_8procs, bf_mp_8procs, bf_16procs, bf_mp_16procs, bf_24procs y bf_mp_24procs. La figura 7.3a reporta los tiempos de ejecución para la versión bf_serial. La figura 7.3b reporta los tiempos de ejecución para las versiones bf_450000part y bf_mp_450000part. Todos los tiempos de ejecución reportados en las figuras 7.2 y 7.3 están expresados en horas.

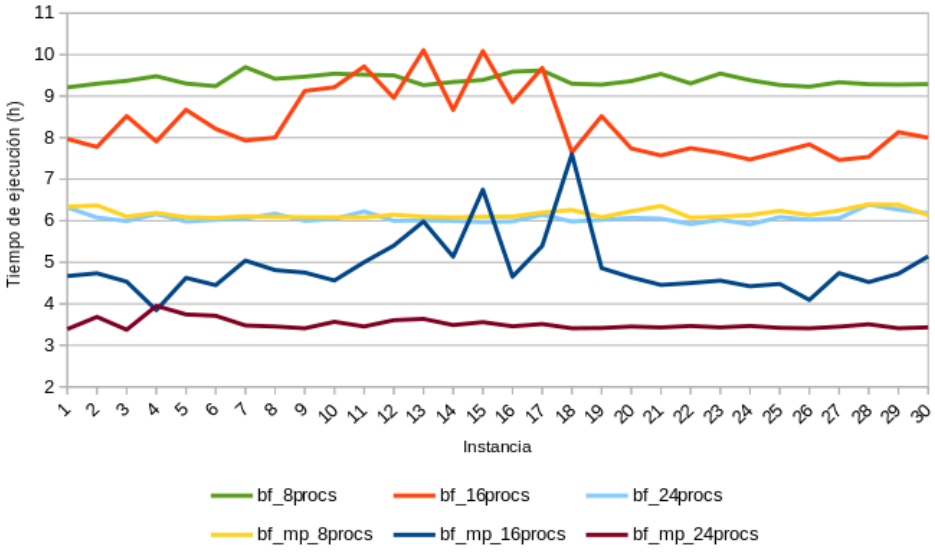
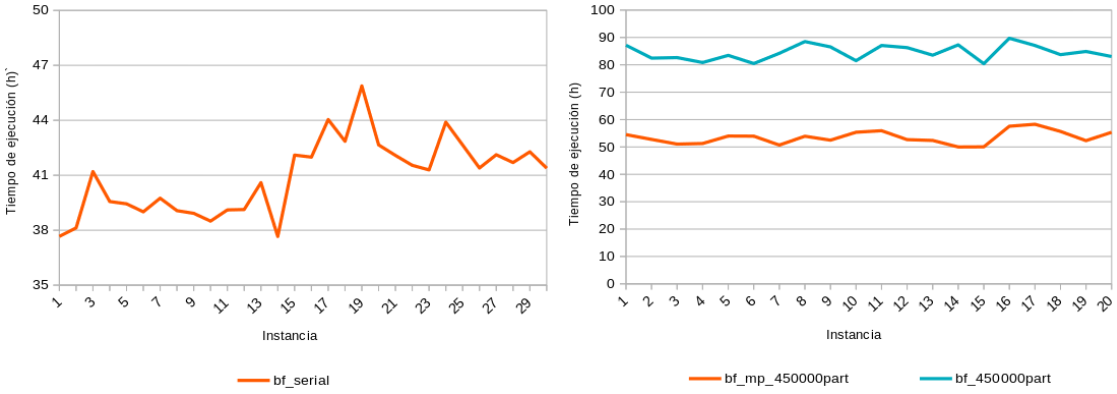


Figura 7.2: Versiones paralelas con 150000 partículas



(a) Versión serial con 150000 partículas (b) Versiones con 450000 partículas

Figura 7.3: Tiempos de ejecución para el problema *binflow*

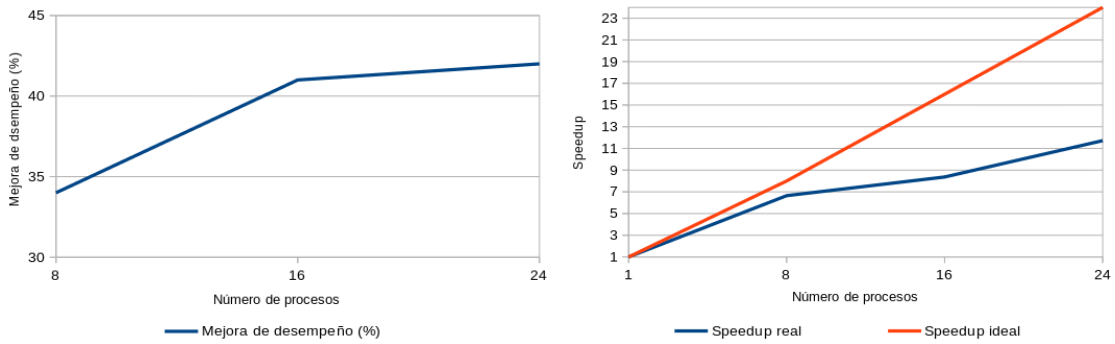
La tabla 7.8 reporta los valores mínimo, máximo y promedio de tiempo de ejecución, la mejora promedio (M) de tiempo de ejecución y speedup (S) cuando se utiliza la estrategia de descomposición dinámica de dominio, y la desviación estándar de los tiempos de ejecución observados. El valor de M se obtiene considerando versiones con configuraciones similares que corresponden a escenarios diferentes y calculando la división de los respectivos tiempos promedio de ejecución. Por ejemplo, considerando versiones de 16 procesos y 150000 partículas se obtiene una mejora del 41 % correspondiente a $M = 1 - \frac{4:54}{8:20} \approx 0,41$. El valor de S para una versión v del problema *binflow* se obtiene calculando la división de los tiempos promedio de ejecución de las versiones bf_serial y v. Por ejemplo, para la versión bf_mp_8procs se obtiene un speedup de $S = \frac{41:02}{6:10} \approx 6,65$.

La figura 7.4 reporta: (a) la progresión de la mejora de desempeño según se incrementa el número de procesos y (b) los resultados del estudio de escalabilidad realizado para las versiones de 150000 partículas del problema *binflow*. Esta figura está basada en

Versión	Tiempo de ejecución			Mejora	Speedup	Desviación
	Min.	Máx.	Prom.	(%)		estándar
bf_serial	37:39	45:52	41:02	–	–	1:56
bf_8procs	9:12	9:41	9:23	–	–	0:07
bf_mp_8procs	6:04	6:23	6:10	34 %	6,65X	0:06
bf_16procs	7:27	10:06	8:20	–	–	0:47
bf_mp_16procs	3:50	7:36	4:54	41 %	8,37X	0:44
bf_24procs	5:55	6:23	6:04	–	–	0:07
bf_mp_24procs	3:23	3:57	3:30	42 %	11,72X	0:06
bf_450000part	80:28	88:30	84:32	–	–	2:46
bf_mp_450000part	50:02	58:17	53:30	37 %	–	2:22

Tabla 7.8: Tiempos de ejecución mínimo, máximo y promedio, mejora de desempeño, speedup relativo y desviación estándar del problema *binflow*

los resultados reportados en la tabla 7.8 y muestra que la estrategia de descomposición de dominio implementada en este proyecto de grado ofrece mejores resultados cuando se incrementa el número de procesos que ejecutan la simulación. Una estrategia de descomposición de dominio basada en el método de planos móviles se beneficia de las propiedades geométricas de este método dado que permite: (1) crear más subdominios y (2) ajustar las dimensiones de los subdominios con mejor precisión de modo que la carga de trabajo se distruya más equitativamente. El estudio de escalabilidad respecto del número de procesos fue realizado solamente para las versiones de 150000 partículas porque los tiempos de ejecución para la versión de 450000 partículas es considerablmente más elevado y un estudio completo de escabilidad para estas versiones exigiría los recursos del Cluster FING por un periodo de tiempo excesivamente prolongado.



(a) Mejora de desempeño (b) Speedup

Figura 7.4: Mejora de desempeño y speedup para diferente número de procesos para las versions de 150000 partículas del problema *binflow*

La figura 7.5 presenta el promedio de la distribución de carga de trabajo entre los procesos a lo largo del tiempo para la versión bf_mp_8procs del problema *binflow*. Esta versión fue elegida con el objetivo de mostrar un ejemplo del comportamiento observado

en la distribución de cargas de trabajo durante una simulación. La versión bf_mp_8procs es completamente representativa del comportamiento de la estrategia de descomposición de dominio en todas las instancias del problema ya que no posee ninguna característica especial respecto de las otras versiones. El apéndice A presenta las gráficas con los resultados de distribución de cargas de trabajo para las demás versiones.

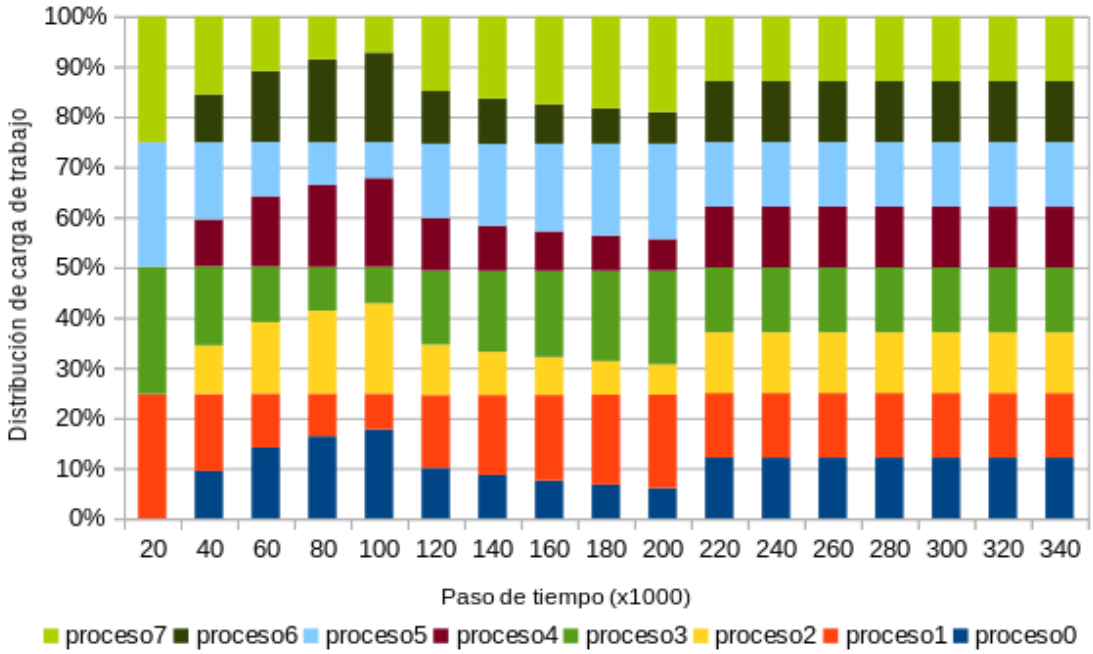


Figura 7.5: Distribución de cargas de trabajo para la versión bf_mp_8procs del problema *binflow*

Cada columna de la figura 7.5 muestra, en un paso de tiempo dado, la proporción del total de partículas que pertenece al subdominio al cual fue asignado cada proceso. Por ejemplo, la columna para el paso de tiempo 20000 indica que los procesos 2, 4, 6 y 8 están ociosos y que las partículas contenidas en la simulación se distribuyen entre los procesos 1, 3, 5 y 7 (alrededor de 25 % de las partículas son asignadas a cada proceso). La distribución de carga de trabajo mostrada para el paso de tiempo 20000 se debe a que el algoritmo de descomposición dinámica de dominio todavía no ha sido invocado y las partículas no se desplazaron hasta un subdominio al cuál los procesos 2, 4, 6 y 8 haya sido asignados.

Según avanza el tiempo, más partículas son insertadas en la simulación, las partículas que ya se encuentran en la simulación se desplazan a otros subdominios, el algoritmo de descomposición dinámica de dominio es invocado y la distribución de carga de trabajo alcanza un punto de equilibrio que se mantiene hasta el final de la simulación. Los resultados observados son: (1) un mejor aprovechamiento de recursos, (2) una distribución adecuada de la carga de trabajo y (3) una reducción del tiempo de ejecución.

Los resultados reportados en el apéndice A indican, por ejemplo, que para la versión bf_8procs (figura A.1) la distribución de carga de trabajo no es adecuada y lo que realmente ocurre es una transferencia total de la carga de trabajo de los procesos 1, 3, 5 y 7 a los procesos 2, 4, 6 y 8. De esta forma hay recursos ociosos y es inevitable obtener tiempos de ejecución elevados. Un comportamiento similar se observa para las versiones bf_16procs, bf_24procs y bf_450000part.

Las tablas 7.9 y B.1 reportan los promedios del número de partículas por proceso según avanza el tiempo para las versiones bf_mp_8procs y bf_8procs del problema *binflow*, respectivamente. Estas tablas corroboran que la estrategia de descomposición de dominio implementada logra distribuir las cargas de trabajos de forma más homogénea entre los procesos que ejecutan una simulación. Los valores de desviación estándar del promedio de partículas reportados en la tabla 7.9 indican que a partir del paso de tiempo 240000 el número de partículas asignado a cada proceso varia muy poco respecto del número de partículas contenidas en el sistema. Por otro lado, los resultados reportados en la tabla B.1 indican que a partir del paso de tiempo 240000 los procesos 1, 3, 5 y 7 no son más utilizados y las cargas de trabajo no están bien distribuidas. El apéndice B contiene las tablas de promedio de partículas por proceso para las versiones bf_8procs, bf_16procs, bf_mp_16procs, bf_24procs, bf_mp_24procs, bf_450000part y bf_mp_450000part.

Paso de	**Proceso**								**Desv.**
tiempo	0	1	2	3	4	5	6	7	**estándar**
20000	0	3716	0	3807	0	3712	0	3765	2005
40000	2828	4616	2893	4722	2804	4609	2854	4674	969
60000	6342	4853	6391	4989	6300	4859	6349	4918	772
80000	9816	5107	9904	5227	9795	5099	9888	5164	2513
100000	13302	5362	13442	5476	13263	5357	13380	5418	4246
120000	9057	13039	9053	13274	9434	13293	9443	13407	2149
140000	9062	16720	9055	16917	9439	17119	9448	17240	4147
160000	9066	20421	9058	20612	9441	20929	9450	21023	6147
180000	9068	24157	9058	24306	9442	24705	9451	24813	8150
200000	9070	27858	9062	28007	9443	28500	9452	28608	10153
220000	18168	19274	18139	19390	18159	19277	18194	19398	627
240000	18160	19283	18136	19405	18150	19282	18187	19397	634
260000	18160	19287	18136	19404	18151	19285	18187	19391	634
280000	18160	19285	18136	19406	18151	19284	18187	19391	634
300000	18160	19285	18136	19406	18151	19284	18187	19391	634
320000	18150	19278	18126	19399	18161	19291	18197	19398	634
340000	18150	19278	18126	19399	18161	19291	18197	19398	634

Tabla 7.9: Promedio de partículas por proceso según el paso de tiempo para la versión bf_mp_8procs del problema *binflow*

La sección 4.2 explicó cómo LIGGGHTS asigna los procesos a los subdominios. En el caso de las versiones bf_8procs y bf_mp_8procs, LIGGGHTS asigna los procesos de rango impar a los subdominios identificados con coordenadas $(x, y, 1)$ y asigna los procesos de rango par a los subdominios identificados con coordenadas $(x, y, 0)$, siendo $(x, y) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$. Por lo tanto, dado que las partículas se insertan por la parte superior del silo y que salen del silo por la parte inferior, la etapa de cómputo de las simulaciones de la versión bf_8procs tiene 3 fases: (1) solamente los procesos impares realizan procesamiento de datos, (2) los procesos pares e impares realizan procesamiento de datos y (3) solamente los procesos pares realizan procesamiento de datos. Este argumento puede ser generalizado para explicar la presencia de procesos ociosos en la simulaciones que son ejecutadas por 16 y 24 procesos.

La figura 7.6 muestra una sección transversal del silo generada utilizando un plano cartesiano paralelo al plano cartesiano $y \times z$. La sección transversal representa un momento intermedio de la etapa de descarga y corresponde a una simulación de la versión bf_mp_8procs. Además, la figura 7.6 presenta las velocidades (en m/s) adquiridas por las partículas durante la descarga. Las velocidades se reportan en la figura mediante el eje `v Magnitude` y la escala de colores de este eje es utilizada para colorear las partículas durante la simulación. Por lo tanto, el color de cada partícula en la figura 7.6 indica la velocidad que tiene. Esta imagen muestra como la salida de una simulación puede ser generada convenientemente para la etapa de posprocesamiento.

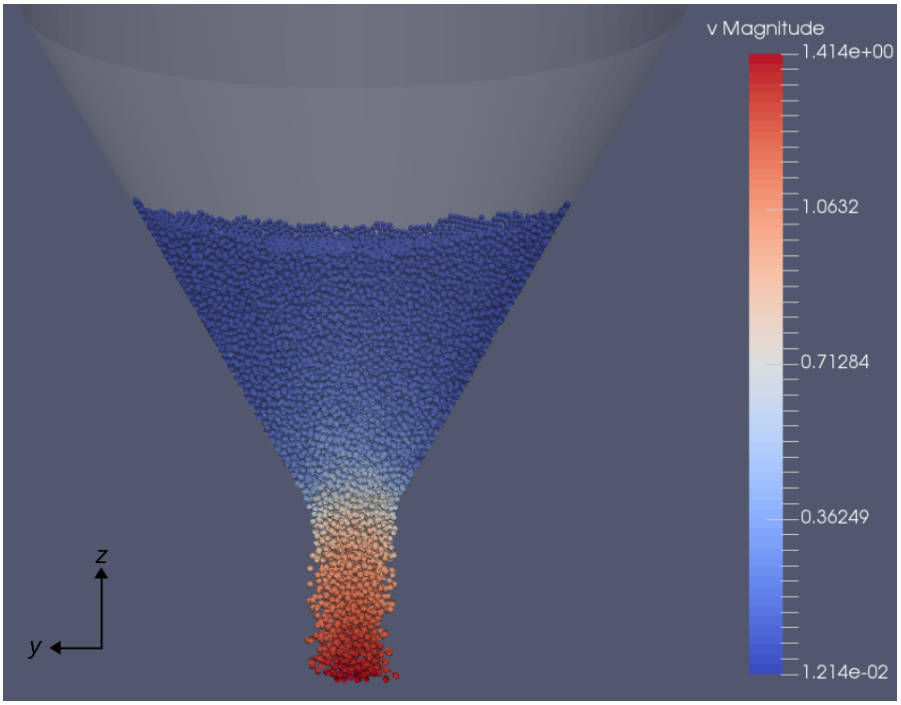


Figura 7.6: Velocidades de las partículas para la versión bf_mp_8procs

La relación entre el diámetro de las partículas y el diámetro del orificio de salida hace que, durante la descarga, se observen los siguientes fenómenos: (1) no atascamiento de partículas en el orificio de salida, (2) formación de un canal de flujo con características de flujo de masa y (3) formación de zonas de estancamiento compuestas por las partículas ubicadas en las camadas superiores de material granular y que están muy próximas a las paredes del silo.

Una característica muy importante de la estrategia de descomposición dinámica de dominio implementada en este proyecto de grado es que permite reducir el número de recursos de cómputo sin causar una pérdida de desempeño. Los tiempos de ejecución reportados en la tabla 7.4 y representados gráficamente en la figura 7.2 para las versiones bf_mp_8procs, bf_16procs y bf_24procs muestran que cambiar la estrategia de descomposición de dominio hace posible que: (1) una simulación que requiere 16 procesos pueda ser ejecutada más rápidamente utilizando solamente 8 procesos y (2) una simulación que requiere 24 procesos pueda ser ejecutada en el mismo tiempo utilizando solamente 8 procesos. Los ejemplos mencionados tratan de simulaciones de un mismo problema que se diferencian únicamente en la cantidad de procesos utilizados en su ejecución.

Los resultados reportados en esta sección permiten concluir que la estrategia de descomposición dinámica de dominio basada en planos móviles logra reducir los tiempos de ejecución para las simulaciones numéricas para el estudio de dinámica de medios granulares. Las mejoras de rendimiento son consecuencia directa del uso más eficiente de los recursos disponibles y se ven acentuadas cuando se encuentra disponible un número mayor de recursos de cómputo. La tabla 7.8 muestra que el porcentaje de mejora aumenta según se incrementa el número de procesos utilizados. A partir de los resultados reportados se verifica que la carga de trabajo se distribuye de forma más homogénea entre los procesos y que no existen procesos ociosos.

7.2.3. Partículas con distribución uniforme de radio

Los problemas considerados en la sección 7.2.2 utilizan partículas con radio constante. Esta suposición es adecuada cuando los radios de las partículas a simular difieren muy poco entre sí. Una situación de este tipo puede ser encontrada, por ejemplo, en el manejo de granos de amaranto (*Amaranthus cruentus* L.) en aplicaciones para agricultura [106]. Sin embargo, existen otras situaciones en las cuáles el radio de las partículas tiene variaciones superiores al 100 % [24]. Por lo tanto, es de especial interés estudiar el comportamiento de la estrategia de descomposición dinámica de dominio implementada en este proyecto de grado cuando se consideran sistemas compuestos por partículas de diferentes radios.

LIGGGHTS tiene funcionalidades que permiten considerar partículas cuyos radios están distribuidos uniformemente sobre un intervalo de números reales (a, b). Por lo tanto, el conjunto de problemas de validación fue ampliado agregando un nuevo problema de dinámica granular definido a partir del problema *binflow* original. Este nuevo problema simula la carga y descarga de un silo en el cual se insertan partículas cuyos radios están distribuidos uniformemente entre 1,0 mm y 2,5 mm ($r \sim U(1{,}0,\ 2{,}5)$). Al igual que en la sección 7.2.2, se consideran dos escenarios que se diferencian según la estrategia de descomposición de domino aplicada. Para cada escenario se crea una versión del problema de partículas con distribución uniforme de radio y para cada versión se ejecutan 15 instancias de simulación. El número de instancias ejecutadas está relacionado al tiempo promedio de ejecución y a la disponibilidad de recursos de cómputo. Todas las instancias de las vesiones creadas fueron ejecutadas en nodos equipados con procesador AMD Opteron modelo 6172 de 24 núcleos de 2,10GHz, y 24GB de memoria RAM. Los criterios de selección de nodos de cómputo para la ejecución de simulaciones son los mismos que fueron descritos en la sección 7.2.2.

Las dimensiones del silo son idénticas a las utilizadas en los problemas de la sección 7.2.2. La etapa de descarga del silo comienza una vez que haya finalizado la etapa de carga y debido a la relación entre el diámetro de las partículas y el diámetro del orificio de salida se observan los siguientes fenómenos: (1) no atascamiento de partículas en el orificio de salida, (2) formación de un canal de flujo con características de flujo de masa y (3) formación de zonas de estancamiento compuestas por las partículas ubicadas en las camadas superiores de material granular y que están muy próximas a las paredes del silo.

Las tablas 7.10 y 7.11 muestran las configuraciones de las versiones creadas para los experimentos. Las columnas `cload` y `dddecomp` contienen, cuando corresponde, los parámetros de configuración utilizados en los comandos `cload` y `dddecomp`, respectivamente. Las columnas `#part.` y `Cfg.` reportan el total de partículas insertadas y la

configuración de procesos definida en el archivo de entrada de una versión del problema *binflow*, respectivamente. Las columnas `Total` y `Dur.(s)` reportan el número de pasos de tiempo considerados en una simulación y la duración, en segundos, de cada paso de tiempo, respectivamente.

#part.	Radio	Pasos de tiempo		Procesos		Frecuencias	
	(mm)	Total	Dur.(s)	#	Cfg.	`cload`	`dddecomp`
$2{,}0 \times 10^5$	$U(1{,}0,\ 2{,}5)$	$4{,}0 \times 10^5$	$5{,}0 \times 10^{-6}$	16	$2 \times 2 \times 4$	25000	–

Tabla 7.10: Configuraciones del escenario estático del problema *binflow* con distribución uniforme de radio de partículas

#part.	Radio	Pasos de tiempo		Procesos		Frecuencias	
	(mm)	Total	Dur.(s)	#	Cfg.	`cload`	`dddecomp`
$2{,}0 \times 10^5$	$U(1{,}0,\ 2{,}5)$	$4{,}0 \times 10^5$	$5{,}0 \times 10^{-6}$	16	$2 \times 2 \times 4$	25000	75000

Tabla 7.11: Configuraciones del escenario dinámico del problema *binflow* con distribución uniforme de radio de partículas

Las tablas 7.12 y 7.13 reportan los resultados numéricos de las instancias de simulación ejecutadas para las versiones bf_16procs_mpart y bf_mp_16procs_mpart, respectivamente. Las columnas `Inst.` y `Tiempo de ejecución` reportan los identificadores de las instancias y los tiempos de ejecución de cada instacia, respectivamente.

Inst.	Tiempo de ejecucón	Inst.	Tiempo de ejecucón
1	10:13	9	10:09
2	11:11	10	10:12
3	10:01	11	10:03
4	11:10	12	10:46
5	10:18	13	10:17
6	10:48	14	11:52
7	10:06	15	10:21
8	10:15		

Tabla 7.12: Versión bf_16procs_mpart

Inst.	Tiempo de ejecucón	Inst.	Tiempo de ejecucón
1	7:23	9	7:24
2	8:22	10	7:12
3	7:10	11	7:57
4	7:39	12	7:59
5	7:30	13	7:26
6	7:51	14	7:32
7	7:28	15	7:50
8	7:58		

Tabla 7.13: Versión bf_mp_16procs_mpart

La figura 7.7 reporta gráficamente los resultados de tiempo de ejecución de las instancias de simulación correspondientes a las dos versiones creadas para el problema de partículas con distribución uniforme de radio. Todos los tiempos de ejecución están expresados en horas.

La tabla 7.14 muestra los valores mínimo, máximo y promedio de tiempo de ejecución, la mejora promedio (M) de tiempo de ejecución cuando se utiliza la estrategia de

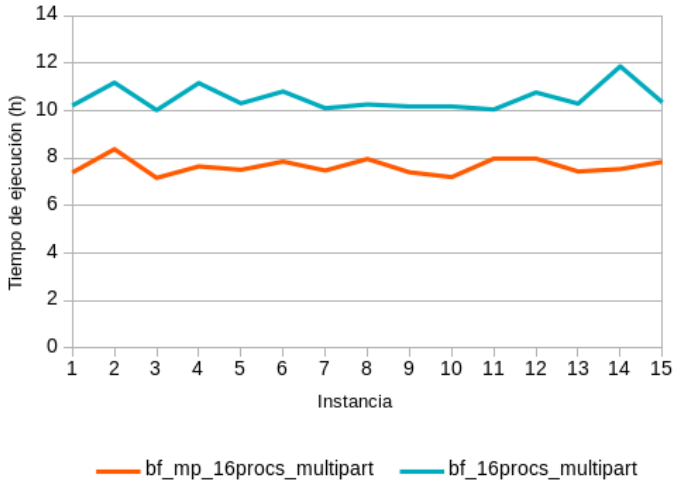


Figura 7.7: Tiempos de ejecución para el problema *binflow* con distribución uniforme de radios de partículas

descomposición dinámica de dominio y la desviación estándar de los tiempos de ejecución observados. El valor de M se calcula de forma idéntica a la que se presentó en la sección 7.2.2. En el caso del problema de partículas con distribución uniforme de radio se calcula la mejora de desempeño utilizando únicamente los tiempos promedio de ejecución de las versiones bf_16procs_mpart y bf_mp_16procs_mpart. Para este problema se obtuvo una mejora de desempeño del 38 % correspondiente a $M = 1 - \frac{7:39}{10:31} \approx 0,38$.

Versión	Tiempo de ejecución			Mejora	Desviación
	Min.	Máx.	Prom.	(%)	estándar
bf_16procs_mpart	10:02	11:52	10:31	–	0:31
bf_mp_16procs_mpart	7:12	8:23	7:39	38 %	0:02

Tabla 7.14: Tiempos de ejecución mínimo, máximo y promedio, y desviación estándar del problema *binflow* con distribución uniforme de radios de partículas

La presencia de partículas con diferentes valores de radio implica una mayor necesidad de capacidad de cómputo, debido al incremento de la complejidad del problema de detección de contacto entre partículas [24]. Comparando el tiempo promedio de ejecución de la versión bf_mp_16procs presentado en la tabla 7.8 con el tiempo promedio de ejecución de la versión bf_mp_16procs_mpart presentado en la tabla 7.14 se observa un incremento del 36 % $\left(1 - \frac{4:54}{7:39} \approx 0,36\right)$ del tiempo promedio de ejecución. Una comparación similar de los tiempos promedio de ejecución de las versiones bf_16procs y bf_16procs_mpart muestra un incremento del 21 % $\left(1 - \frac{8:20}{10:31} \approx 0,21\right)$. Este incremento de la capacidad de cómputo exigida influye en el número de pasos de tiempo que deben ser ejecutados en las simulaciones. Para que una instancia de simulación del problema de partículas con distribución uniforme de radios pueda contener completamente las etapas de carga y descarga es necesario utilizar al menos 400000 pasos de tiempo. Además, es necesario considerar más pasos de tiempo porque las propiedades físico-químicas de las partículas

cambian, mientras que las dimensiones del silo permanecen constantes. De esta forma, la razón de flujo de partículas para el problema de partículas con distribución uniforme es menor respecto de la razón de flujo de partículas para el problema tratado en la sección 7.2.2. La razón de flujo de partículas se define como el número de partículas que salen del silo por unidad de tiempo.

La figura 7.8 muestra el promedio de la distribución de carga de trabajo entre los procesos a lo largo del tiempo para la versión bf_mp_16procs_mpart. Esta versión fue elegida con el objetivo de mostrar un ejemplo del comportamiento observado en la distribución de cargas de trabajo durante una simulación. La versión bf_mp_16procs_mpart es completamente representativa del comportamiento de la estragia de descomposición dinámica de dominio en todas las instancias del problema ya que esta versión no posee ninguna característica especial respecto de la versión bf_16procs_mpart. El apéndice A presenta la gráfica con los resultados de distribución de cargas de trabajo para la versión bf_16procs_mpart.

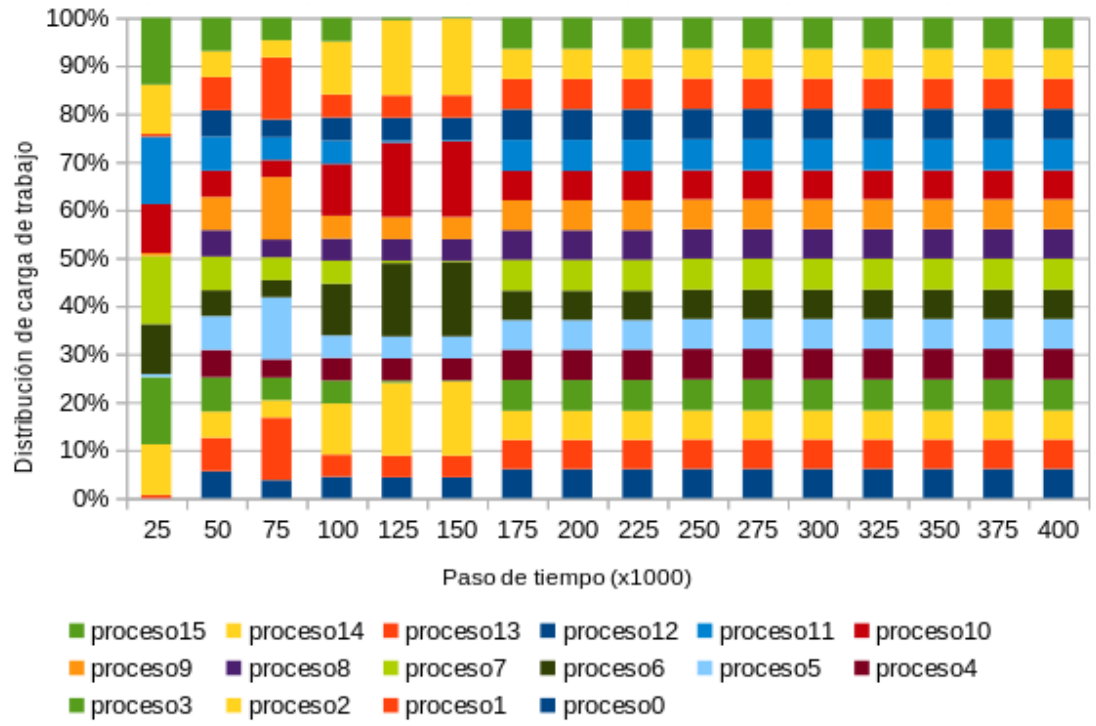


Figura 7.8: Distribución de cargas de trabajo para la versión bf_mp_16_mpart del problema *binflow*

Cada columna de la figura 7.8 reporta, para un paso de tiempo específico, la proporción del total de partículas contenidas en el sistema que pertenece al subdominio al cuál un proceso fue asignado. Por ejemplo, la columna para el paso de tiempo 25000 indica que los procesos 0, 4, 8 y 12 están ociosos y que las partículas contenidas en la simulación se distribuyen entre los procesos restantes. La distribución de carga de trabajo mostrada para el paso de tiempo 25000 se debe a que el algoritmo de descomposición dinámica de dominio todavía no ha sido invocado y las partículas no se han desplazado hasta un subdominio asignado a los procesos 0, 4, 8 y 12. Los resultados reportados en la columna 25000 también muestran que la cargas de trabajo se distribuen de forma no equitativa entre los procesos 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14 y 15.

Según avanza el tiempo, más partículas se insertan en la simulación, las partículas que ya se encuentran en la simulación se desplazan a otros subdominios, el algoritmo de descomposición dinámica de dominio es invocado y la distribución de carga de trabajo alcanza un punto de equilibrio que se mantiene hasta el final de la simulación. Los resultados observados son: (1) un mejor aprovechamiento de los recursos de cómputo, (2) una distribución adecuada de la carga de trabajo y (3) una reducción del tiempo de ejecución. El apéndice A contiene las gráfica con los resultados de distribución de carga de trabajo para la versión bf_16procs_mpart.

La figura 7.9 muestra una sección transversal del silo generada utilizando un plano cartesiano paralelo al plano cartesiano $y \times z$. La sección transversal representa un momento intermedio de la etapa de descarga y corresponde a una simulación de la versión bf_mp_16procs_mpart. Además, la figura 7.9 presenta las velocidades (en m/s) adquiridas por las partículas durante la descarga. Las velocidades se reportan en la figura mediante el eje `v Magnitude`. Las velocidades se reportan en la figura mediante el eje `v Magnitude` y la escala de colores de este eje es utilizada para colorear las partículas durante la simulación. Por lo tanto, el color de cada partícula en la figura 7.6 indica la velocidad que tiene. Esta imagen muestra como la salida de una simulación puede ser generada convenientemente para la etapa de posprocesamiento.

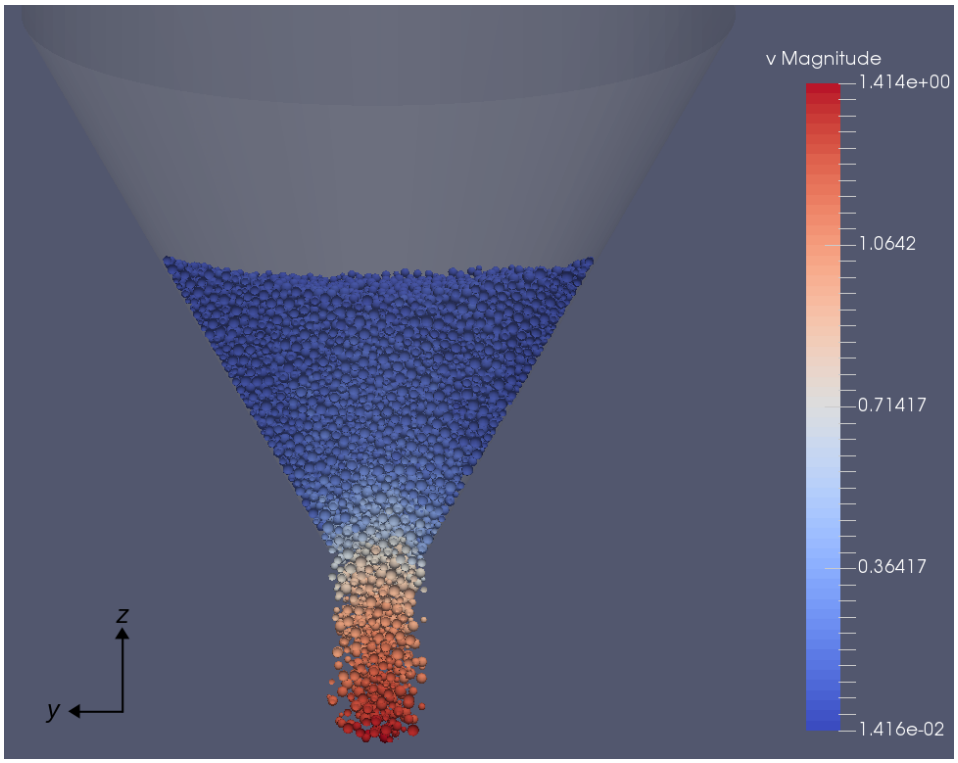


Figura 7.9: Velocidades de las partículas para la versión bf_mp_16procs_mpart

Las tablas 7.15 y B.8 reportan los promedios del número de partículas por proceso según avanza el tiempo para las versiones bf_mp_16procs_mpart y bf_16procs_mpart del problema *binflow*, respectivamente. Estas tablas permiten corroborar que la estrategia de descomposición de dominio implementada efectivamente distribuye las cargas de trabajos de forma más homogénea entre los procesos que ejecutan una simulación. Los valores de desviación estándar del promedio de partículas reportados en las tablas 7.15 y B.8 indican

que a partir del paso de tiempo 250000 el número de partículas asignado a cada proceso varia muy poco respecto del número de partículas contenidas en el sistema.

Paso de tiempo	Proceso							
	0	1	2	3	4	5	6	7
25000	0	323	5209	6930	0	316	5138	7013
50000	5487	6846	5347	7020	5587	6928	5254	6902
75000	5521	19073	5295	6926	5611	18908	5260	6978
100000	8672	8978	20772	9326	9192	9132	21016	9336
125000	8663	8974	30363	962	9183	9123	30588	989
150000	8665	8977	30910	424	9185	9131	31137	418
175000	12058	12175	12092	12833	12675	12293	12231	12862
200000	12058	12175	12092	12833	12675	12293	12231	12862
225000	12058	12174	12092	12833	12675	12293	12230	12863
250000	12141	12295	12062	12885	12760	12413	12207	12911
275000	12141	12295	12062	12885	12760	12413	12207	12911
300000	12141	12295	12062	12885	12760	12413	12207	12911
325000	12141	12295	12062	12885	12760	12413	12207	12911
350000	12141	12295	12062	12885	12760	12413	12207	12911
375000	12141	12295	12062	12885	12760	12413	12207	12911
400000	12141	12295	12062	12885	12760	12413	12207	12911

(a) Promedio de partículas de los procesos 0 a 7

Paso de tiempo	Proceso								Desviación estándar
	8	9	10	11	12	13	14	15	
25000	0	311	5031	6938	0	285	5072	6896	3200
50000	5418	6793	5323	6946	5412	6793	5294	6846	795
75000	5430	19065	5225	6917	5440	18959	5238	6910	6128
100000	8960	9276	21057	9399	9434	9336	21553	9656	5542
125000	8954	9276	30891	931	9423	9343	31311	1025	11951
150000	8954	9280	31418	415	9425	9347	31876	438	12346
175000	12352	12397	12190	12950	12761	12616	12511	13006	291
200000	12352	12398	12189	12950	12760	12616	12511	13006	291
225000	12352	12398	12189	12950	12760	12616	12511	13006	291
250000	12269	12432	12064	12898	12676	12642	12388	12957	308
275000	12269	12432	12064	12898	12676	12642	12388	12957	308
300000	12269	12432	12064	12898	12676	12642	12388	12957	308
325000	12269	12433	12064	12898	12676	12642	12388	12957	308
350000	12269	12433	12064	12898	12676	12642	12388	12957	308
375000	12269	12432	12064	12898	12676	12642	12388	12957	308
400000	12269	12432	12064	12898	12676	12642	12388	12957	308

(b) Promedio de partículas de los procesos 8 a 15

Tabla 7.15: Promedio de partículas por proceso según el paso de tiempo para la versión bf_mp_16procs_mpart del problema *binflow*

Los resultados presentados en esta sección permiten concluir que la estrategia de descomposición dinámica de dominio basada en planos móviles es robusta respecto de la variación del radio de las partículas contenidas en el sistema. La estrategia de descomposición de dominio logra manejar adecuadamente sistemas de partículas cuya variación de radios es superior al 100 %. La reducción del tiempo promedio de ejecución de las simulaciones es similar a la reportada en la sección 7.2.2. Se verifica que la carga de trabajo se distribuye de forma más homogénea entre los procesos y que no existen procesos ociosos.

7.3. Simulación de descarga de un silo a través de un orificio de salida excéntrico

En la sección 7.2 se consideraron problemas de dinámica de medios granulares cuyo objetivo principal era corroborar la eficiencia computacional de la estrategia de descomposición de dominio propuesta en este proyecto de grado. Los problemas estudiados en esa sección son simulaciones a escala reducida de problemas inspirados en situaciones encontradas en la industria. Sin embargo, las propiedades físico-químicas utilizadas en esos problemas se definieron de forma sintética, sin considerar un tipo de medio granular concreto y sin tener en cuenta el material utilizado en la construcción del silo.

En esta sección se considera otra clase de problemas de dinámica de medios granulares que también representa situaciones encontradas habitualmente en la industria y que despierta interés especial en la comunidad académica [36, 43, 44]. El problema abordado fue sugerido por investigadores del IMFIA y su objetivo es estudiar la distribución de las presiones ejercidas sobre las paredes del silo durante la etapa de descarga a través de un orifico de descarga excéntrico. Una forma de obtener información sobre la distribución de presiones es simular la descarga del silo durante un periodo de tiempo corto respecto del tiempo total necesario para vaciar completamente la estructura y observar la formación de un canal de flujo conocido como flujo afunilado excéntrico. En la sección 2.2 se explicó los posibles canales de flujo observados durante la descarga de un silo según la ubicación del orificio de salida utilizado. La información sobre el canal de flujo formado puede ser complementada considerando también las velocidades de cada partícula durante la descarga. LIGGGHTS permite incluir en los archivos de salida el identificador de cada partícula contenida en la simulación y su respectiva velocidad. Esta funcionalidad hace posible registrar la posición y la velocidad de cada partícula a lo largo de una simulación.

Existen tres diferencias fundamentales entre los problemas estudiados en la sección 7.2 y el problema estudiado en esta sección: (1) la descarga del silo se realiza a través de un orificio de salida excéntrico, (2) las propiedades físico-químicas de los materiales involucrados corresponden a las propiedades de materiales concretos y (3) las dimensiones del modelo de silo utilizado son proporcionales a las dimensiones de silos encontrados en la industria. Estudiar el comportamiento de la estrategia de descomposición de dominio propuesta en este proyecto de grado respecto de un problema con estas propiedades muestra la robustez de la solución implementada y la valida como herramienta computacional que puede sustituir no sólo experimentos de laboratorio con propiedades sintéticas, sino también experimentos cuyas propiedades se encuentran en aplicaciones reales.

La figura 7.10 muestra las dimensiones del silo considerado en los experimentos. Este tipo de estructura cuenta, en general, con dos o tres orificios de salida, de los cuales al menos uno es concéntrico y uno es excéntrico. En el caso que exista un tercer orificio

de salida, suele ser ubicado entre los dos orificios extremos. La figura 2.1c muestra un ejemplo real del tipo de silo considerado en esta sección.

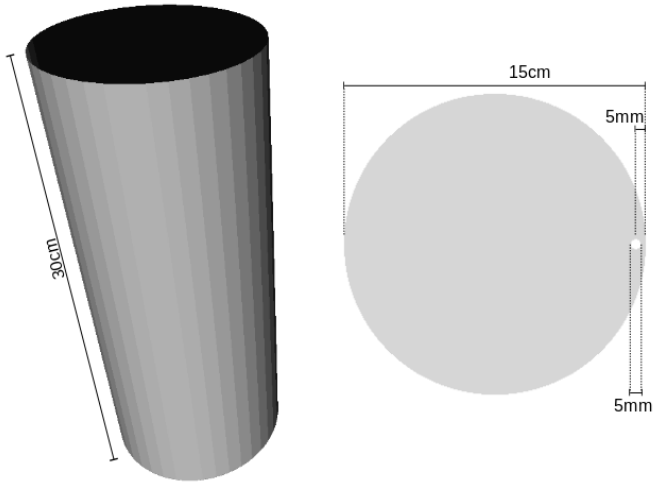


Figura 7.10: Dimensiones del silo para el problema de descarga de amaranto a través de un orificio de salida excéntrico

Las partículas usadas en este problema tienen radio constante de 1,0mm y se usan como un modelo para granos de amaranto (*Amaranthus cruentus* L.). Los datos encontrados en la literatura muestran que los granos de amaranto tienen forma casi esférica cuyo diámetro es apróximadamente de 1,0mm [106]. La figura 7.11 muestra ejemplos de granos reales de amaranto que permiten verificar la uniformidad del diámetro de las muestras.

(a) Ejemplo 1 (b) Ejemplo 2

Figura 7.11: Ejemplos de granos de amaranto
Fuente: Internet [107, 108]

La tabla 7.16 reporta las propiedades físico-químicas de las partículas de amaranto y de láminas de acero utilizadas en la construcción de silos. Los valores fueron informados por los investigadores del IMFIA y complementados con datos obtenidos en referencias bibliográficas [109].

El análisis experimental realizado para el problema abordado tiene dos objetivos: (1) corroborar que la estrategia de descomposición de dominio implementada hace que LIGGGHTS mantenga un funcionamiento correcto respecto de la teoría en la cual se basa el método DEM y (2) mostrar cómo una herramienta de simulación numérica puede ser utilizada para el estudio de un problema complejo de dinámica de medios granulares.

Propiedad	Valor
Módulo de Young	$15MPa$
Coef. de fricción – partícula-partícula	0,532
Coef. de fricción – partícula-superficie	0,275
Densidad de partícula	$1350kg/m^3$
Diámetro promedio	$1mm$

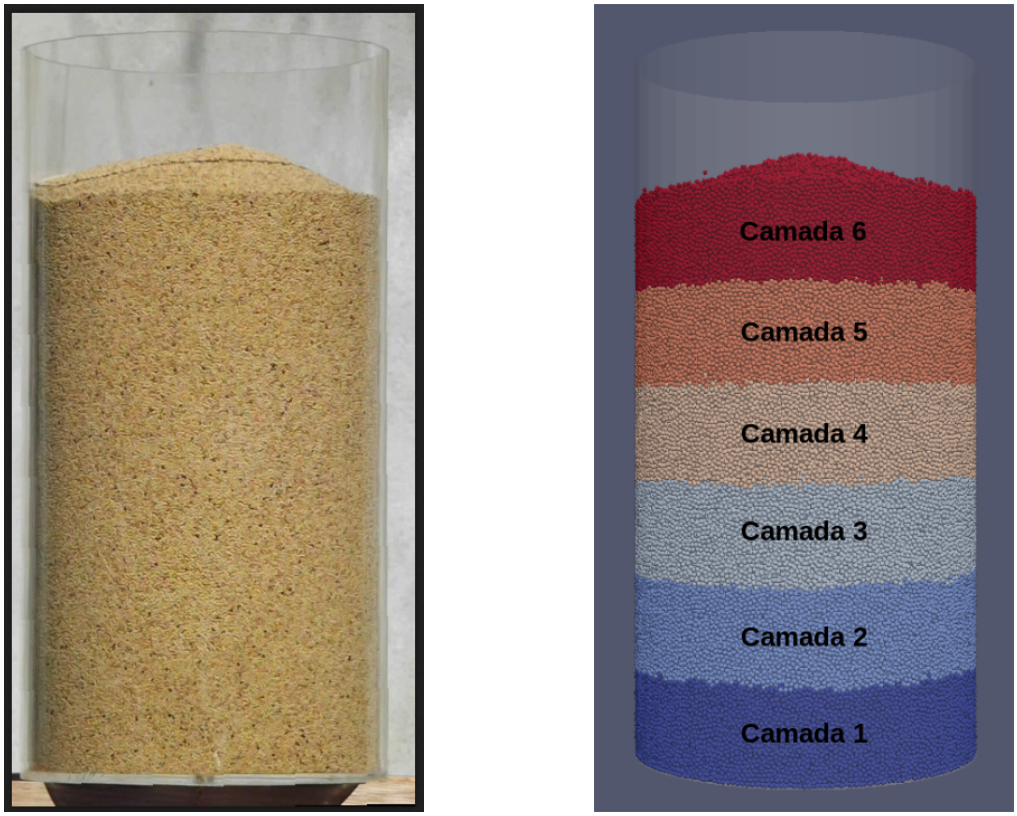
Tabla 7.16: Propiedades físico-químicas de los granos de amaranto

Kloss et al. [22] reportaron los resultados numéricos que muestran que la estrategia de descomposición de dominio estática no afecta negativamente al comportamiento de LIGGGHTS y los resultados reportados fueron coherentes con la teoría en la cual se basa el método DEM. Por lo tanto, las simulaciones realizadas utilizaron solamente la estrategia de descomposición dinámica de dominio. Todas las simulaciones fueron ejecutadas en nodos del Cluster FING equipados con procesador AMD Opteron modelo 6172 de 24 núcleos de 2,10GHz y 24GB de memoria RAM. No se realizaron estudios de eficiencia computacional y la ejecución de varias simulaciones se debió a la necesidad de calibrar correctamente las configuraciones del sistema.

Para obtener los resultados en un formato conveniente para el posprocesamiento se utilizó un procedimiento especial durante la etapa de carga. Fueron insertadas 600000 partículas en el silo con el objetivo de ocupar aproximadamente 80 % de su capacidad y tal que se formaran seis camadas conteniendo 100000 partículas cada una. Para que las partículas fueran dispuestas adecuadamente en camadas, las mismas fueron insertadas mediante ciclos de inserción intercalados por breves intervalos de tiempo. Agregar un intervalo de tiempo entre dos ciclos de inserción consecutivos permitió: (1) asegurar que las partículas de una camada no se mezclaran con las partículas de otra camada durante el movimiento de caída libre que ocurre durante la carga y (2) asegurar que las últimas partículas insertadas en un ciclo se desplazaran lo suficiente para acomodarse adecuadamente en la cima de la camada correspondiente.

Cada camada fue identificada unívocamente con un valor perteneciente al conjunto $\{1, 2, 3, 4, 5, 6\}$ y todas las partículas de una misma camada recibieron una etiqueta de tipo (variable de salida `type`) cuyo valor era igual al identificador de la camada a la cual pertenecía. Por lo tanto, el valor i fue asignado a la etiqueta de tipo de todas las partículas que pertenecían a la camada i. La etiqueta de tipo fue agregada a la salida de la simulación mediante una funcionalidad del formato de archivo VTK [104] que permite agregar metadatos definidos por el usuario. Para agregar los metadatos necesarios a la salida, se utilizó se utilizó el comando `dump` de LIGGGHTS. Utilizando las etiquetas de tipo es posible saber a cual camada una partícula pertenece. Además, estas etiquetas permiten asignar un mismo color a todas las partículas de una misma camada y, por lo tanto, hace posible visualizar adecuadamente el comportamiento de las partículas en la etapa de posprocesamiento.

Las figuras 7.12a y 7.12b muestran el silo utilizado en el IMFIA para la experimentos y un modelo del silo real utilizado en las simulaciones, respectivamente. Ambas figuras corresponden a instantes intermedios entre el final de las etapas de carga y el comienzo de las estapas de descarga correspondientes.



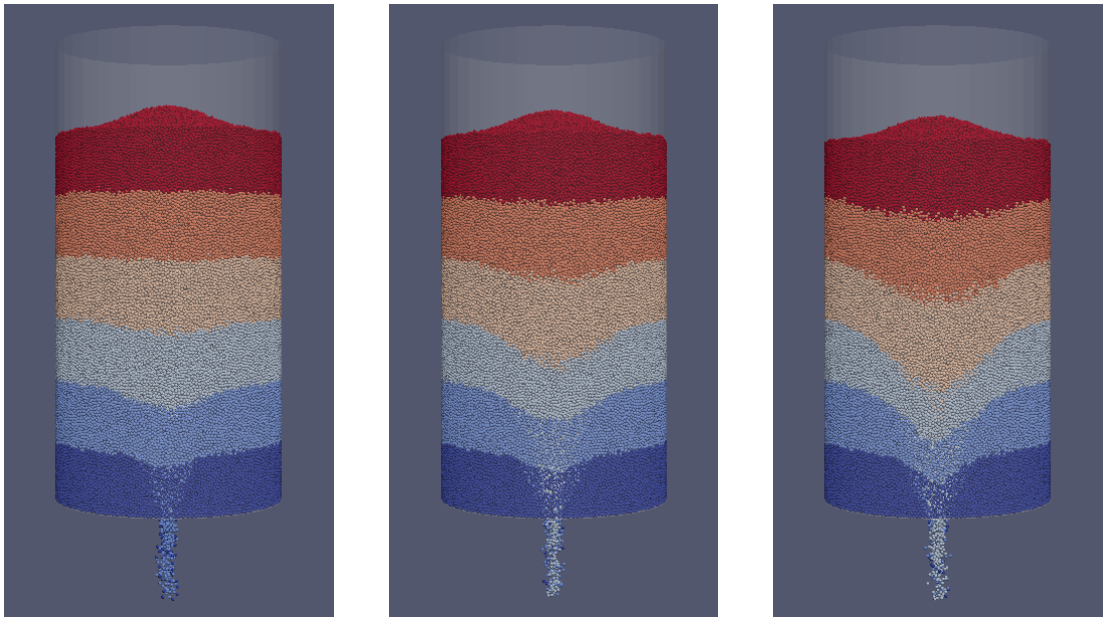
(a) Silo real utilizado para experimentos en el IMFIA

(b) Modelo del silo real utilizado en simulaciones numéricas

Figura 7.12: Silo real utilizando en experimentos y modelo del silo para simulaciones numéricas

La etapa de descarga no simula el procedimiento completo de vaciado de la estructura. Se ejecuta una simulación un periodo de tiempo suficiente para observar la formación de un canal de flujo afunilado excéntrico. El total de pasos de tiempo ejecutados y la duración de cada paso de tiempo fueron $T_{pt} = 2{,}0 \times 10^6$ y $D_{pt} = 5{,}0 \times 10^{-6}$ segundos, respectivamente. Estos valores fueron definidos empíricamente durante la ejecución de las simulaciones de calibración de las configuraciones del sistema. Dados los valores de T_{pt} y D_{pt}, el tiempo total simulado fue igual a $T_{pt} \times D_{pt} = 10$ segundos. Las figuras 7.13a, 7.13b y 7.13c muestran tres instantes de tiempo durante la descarga de las partículas y permiten observar la formación y evolución de un canal de flujo afunilado excéntrico. La figura 7.13a corresponde al instante $t = 5s$ (paso de tiempo $1{,}0 \times 10^6$), la figura 7.13b corresponde al instante $t = 7,5s$ (paso de tiempo $1{,}5 \times 10^6$) y la figura 7.13c corresponde al instante $t = 10s$ (paso de tiempo $2{,}0 \times 10^6$).

Cuando se utiliza LIGGGHTS para el estudio del comportamiento de las partículas en el interior del silo durante la operación de vaciado y para corroborar la ocurrencia de algún patrón de flujo, es suficiente ejecutar una simulación hasta que se observe la formación de un canal de flujo, dado que al interrumpir la simulación es posible guarda el estado del sistema. De este modo, no se realizaron simulaciones con un total de pasos de tiempo mayor a T_{pt}. Si resultara necesario ejecutar una simulación hasta que el silo estuviera vacío, la salida de una simulación previa podría ser utilizada como entrada. Los trabajos relacionados relevados en el capítulo 5 que reportaron resultados



(a) Evolución de la formación de un canal de flujo afunilado excéntrico con $t = 5s$

(b) Evolución de la formación de un canal de flujo afunilado excéntrico con $t = 7,5s$

(c) Evolución de la formación de un canal de flujo afunilado excéntrico con $t = 10s$

Figura 7.13: Silo real utilizando en experimentos y modelo del silo para simulaciones numéricas

de experimentos realizados en laboratorio o utilizando silos a escala real, realizaron la descarga completa de las partículas ya que fueron observados cambios en el canal de flujo formado una vez que se interrumpía el funcionamiento del silo [34, 36]. Por lo tanto, los resultados experimentales obtenidos no fueron invalidados por el hecho de no ejecutar por completo una simulación de descarga.

Para validar los resultados obtenidos en las simulaciones en computador, se comparó el comportamiento de las partículas observado en la etapa de posprocesamiento con el comportamiento de las partículas observado en los experimentos realizados en los laboratorios del IMFIA. Algunos de los experimentos realizados en el IMFIA fueron registrados en video y la comparación de comportamiento no presentó dificultades. Además, para verificar la calidad de los resultados, se consultó a los investigadores responsables de estos experimentos. Los resultados mostraron la calidad esperada y fueron validados como correctos, mostrando que la estrategia de descomposición dinámica de dominio implentada en este proyecto de grado no afecta negativamente al comportamiento de LIGGGHTS.

Capítulo 8

Conclusiones y trabajo futuro

Este capítulo resume las principales conclusiones del proyecto de grado y las posibles líneas de trabajo a seguir. La sección 8.1 contiene las conclusiones obtenidas del desarrollo del proyecto. La sección 8.2 presenta algunas líneas de investigación para el desarollo de trabajo futuro relacionado con los temas tratados en este proyecto.

8.1. Conclusiones

Este proyecto de grado tuvo el objetivo de mejorar el desempeño computacional del sistema de simulación numérica LIGGGHTS. Este sistema implementa el método DEM de Cundall y Strack [4] y es utilizado por los investigadores del IMFIA para la ejecución de simulaciones de sistemas de medios granulares. Para lograr el objetivo planteado, el proyecto propuso una implementación de una estrategia de descomposición dinámica de dominio basada en el concepto de planos móviles de Markauskas et al. [30]. Esta estrategia busca distribuir homogeneamente entre los procesos en ejecución la carga de trabajo total. Para ello ajusta las ejes de coordenadas del espacio cartesiano $\mathbb{R}^3$ que definen las dimensiones de cada subdominio que compone el dominio del problema.

Todas las tareas de programación realizadas para implementar las modificaciones necesarias siguieron el paradigma de programación orientada a objetos. Este paradigma es aplicado fuertemente por los desarrolladores de LIGGGHTS. Con el objetivo de integrar cambios que no afectaran negativamente a LIGGGHTS, todas las modificaciones respetaron la arquitectura propuesta por los desarrolladores. Se utilizó el lenguaje de programación C++ (el mismo lenguaje con el cual LIGGGHTS está implementado) y las configuraciones necesarias para la compilación automática del código fuente fueron incluidas en un conjunto de archivos de entrada para el comando `make`. La compilación automática del código fuente permitió agilizar la puesta en producción de una versión del sistema y se mostró un método importante para hacer disponible a investigadores no especializados en computación cualquier nueva versión de LIGGGHTS generada. La gestión del código fuente se realizó mediante el sistema de versionado GIT.

La estrategia implementada fue incorporada a LIGGGHTS y un conjunto amplio de simulaciones numéricas para validación fueron ejecutadas. Las simulaciones consistieron en cargar y descargar silos de almacenamiento utilizando partículas esféricas que, dependiendo del problema, fueron utilizadas como modelo para granos de amaranto (*Amaranthus cruentus* L.). Estas simulaciones fueron configuradas a partir: (1) de un problema de validación (*benchmark*) distribuido junto con el código fuente de LIGGGHTS y

(2) de un problema de dinámica granular propuesto por los investigadores del IMFIA. En ambos casos se utilizaron modelos de silos que corresponden a estructuras encontradas habitualmente en la industria.

Los resultados obtenidos en la etapa de análisis experimental para la eficiencia computacional de la estrategia implementada mostraron que el desempeño de LIGGGHTS pudo ser mejorado en hasta un 42 % y se obtuvo un speedup de hasta 12X para simulaciones de un problema de validación compuesto por 150000 partículas y cuyas ejecuciones se realizaron utilizando 24 procesos. Las simulaciones fueron ejecutadas en nodos de cómputo del Cluster FING. La estrategia implementada se mostró robusta respecto: (1) al incremento del número de partículas contenidas en la simulación y (2) a la presencia de partículas de diferentes radios. En el caso de una simulación con mayor número de partículas, se requirió mayor capacidad de cómputo debido a la ocurrencia de un mayor número de interacciones entre partículas. En el caso de una simulación con partículas de diferentes radios, se requirió mayor capacidad de cómputo debido a que la detección de contactos se torna un problema más complejo. En ambos casos la estrategia de descomposición de dominio pudo manejar adecuadamente las características de cada simulación.

La mejora de desempeño fue una consencuencia directa del uso más eficiente de los recursos de cómputo disponibles. La estrategia de descomposición de dominio implementada en este proyecto de grado logró distribuir homogeneamente la carga de trabajo entre los recursos de cómputo y evitó la ocurrencia de recursos ociosos. En promedio no se registraron desbalanceos de carga entre procesos superior al 10 % y en el peor caso el desbalanceo de carga llegó al 20 % aproximadamente. Los resultados obtenidos mostraron que la mejora de eficiencia computacional y el speedup obtenido se incrementan según aumenta la cantidad de procesos utilizados. El aumento de procesos permitió que las dimensiones de los subdominios fueran ajustadas más precisamente y, por lo tanto, la carga de trabajo total fuera distribuida más homogeneamente.

Además, se obtuvieron resultados numéricos coherentes con la teoría que da soporte al método DEM. Estos resultados fueron validados utilizando resultados de experimentos realizados en los laboratorios del IMFIA y comparando el comportamiento de las partículas en ambos casos.

Por lo tanto, dado los resultados obtenidos en la etapa de validación, se concluye que: (1) la estrategia de descomposición dinámica de dominio implementada en este proyecto de grado es adecuada para problemas de dinámica granular de descarga de silo y (2) la implementación de esta estrategia que ha sido incorporada a LIGGGHTS permite utilizar este software de forma más eficiente.

8.2. Trabajo futuro

El trabajo realizado en este proyecto de grado estuvo enfocado en modificaciones de algunos aspectos computacionales de LIGGGHTS que permitieran mejorar su desempeño sin afectar negativamente los resultados numéricos. Como se explicó en la sección 8.1, este objetivo fue alcanzado mediante la implementación de una estrategia de descomposición dinámica de dominio. A continuación se introduce brevemente algunas líneas de investigación muy activas en el área de dinámica de medios granulares y a partir de las cuales se define el trabajo futuro a desarrollarse.

Una primera línea de investigación es el estudio de otras estrategias de carácter geométrico para la descomposición de dominio. Plimpton [27], Berger y Bokhari [31]

y Hanxleden y Scott [64] mostraron que los algoritmos de descomposición de dominio en base a la geometría del escenario simulado son adecuados para balanceo de carga de simulaciones que aplican el método DEM. En este contexto, la implementación de una estrategia de descomposición de dominio basada en el algoritmo RCB de Berger y Bokhari [31] puede ser una forma de obtener incrementos de desempeño aún mayores a los obtenidos por el método implementado en este proyecto de grado.

Una segunda línea de investigación enfocada en el desarrollo de las herramientas de simulación que aplican DEM es el estudio del modelo de contacto utilizado para la interacción entre los componentes de una simulación y sus comportamientos. Navarro y de Souza Braun [110], Di Renzo y Di Maio [15] y Horabik y Molenda [111] reportaron estudios comparativos de diferentes modelos de contacto utilizados habitualmente en DEM. La sustitución del modelo de contacto puede mejorar el desempeño de un sistema de simulación numérica dado que modela de forma más precisa la interacción entre los componentes y sus comportamientos. Sin embargo, una implementación eficiente de un nuevo modelo de contacto y su validación experimental requieren conocimiento avanzados sobre los aspectos físico-matemáticos del modelo.

Existe una línea de investigación que tiene mucha relevancia en el área de dinámica granular y que resultó ser de mucho interés para los integrantes del Grupo de Mecánica de los Fluidos Computacional del IMFIA. Esta línea de investigación se relaciona con los trabajos de Adam Sadowski [112] para la caracterización de fallas estructurales de silos durante la descarga a través de orificios de salida excéntrico. Sadowski es integrante del Departamento de Ingeniería Civil y Ambiental del Imperial College de Londres y visitó la Facultad de Ingeniería de la Universidad de la República en el año 2017 y brindó una serie de seminarios sobre sus líneas de trabajo. Desde hace algunos años existe un contacto muy fluido entre integrantes del Grupo de Mecánica de los Fluidos Computacional y el grupo de investigación dirigido por Sadowski. En este contexto, una herramienta eficiente de simulación numérica es fundamental para abordar los problemas considerados.

Finalmente, para la implementación de un sistema computacional para ejecución paralela es necesario utilizar herramientas y bibliotecas de software que permitan la ejecución concurrente de varios procesos. LIGGGHTS está implementado utilizando la biblioteca MPI y enfocado en paralelismo de memoria distribuida. Berger et al. [19] reportaron los resultados obtenidos al utilizar una estrategia híbrida de paralelismo que conjuga MPI y OpenMP aprovechando también sistemas computacionales de memoria distribuida. Los resultados reportados mostraron que los investigadores lograron obtener mejoras de desempeño debido a la estrategia de paralelismo utilizada. A nivel de hardware, una estrategia híbrida de paralelismo puede obtener buenos resultados computacionales utilizando diferentes arquitecturas. Shigeto y Sakai [25] propusieron técnicas de paralelismo específicas para GPUs. Blaze-DEMGPU [113] es un ejemplo de sistema de simulación numérica para DEM que utiliza fuertemente nodos de cómputo equipados con GPUs.

Como línea de trabajo principal a desarrollarse, se propone investigar la factibilidad de implementar una versión del algoritmo RCB que pueda ser incorporada a LIGGGHTS. Al mismo tiempo, como línea de trabajo secundaria, se propone participar en las tareas del Grupo de Mecánica de los Fluidos Computacional enfocadas en dinámica granular con el objetivo de aprofundar las relaciones entre este grupo de investigación y el grupo dirigido por Adam Sadowski.

Apéndices

Apéndice A

Gráficas de distribución promedio de carga de trabajo por proceso

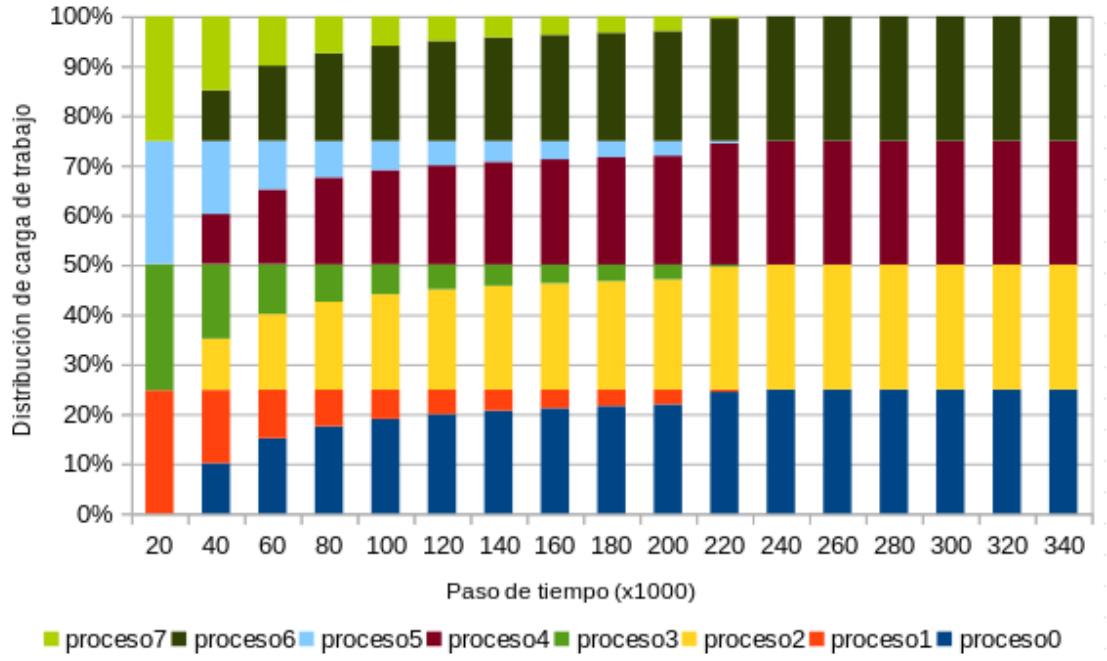


Figura A.1: Distribución de cargas de trabajo para la versión bf_8procs del problema *binflow*

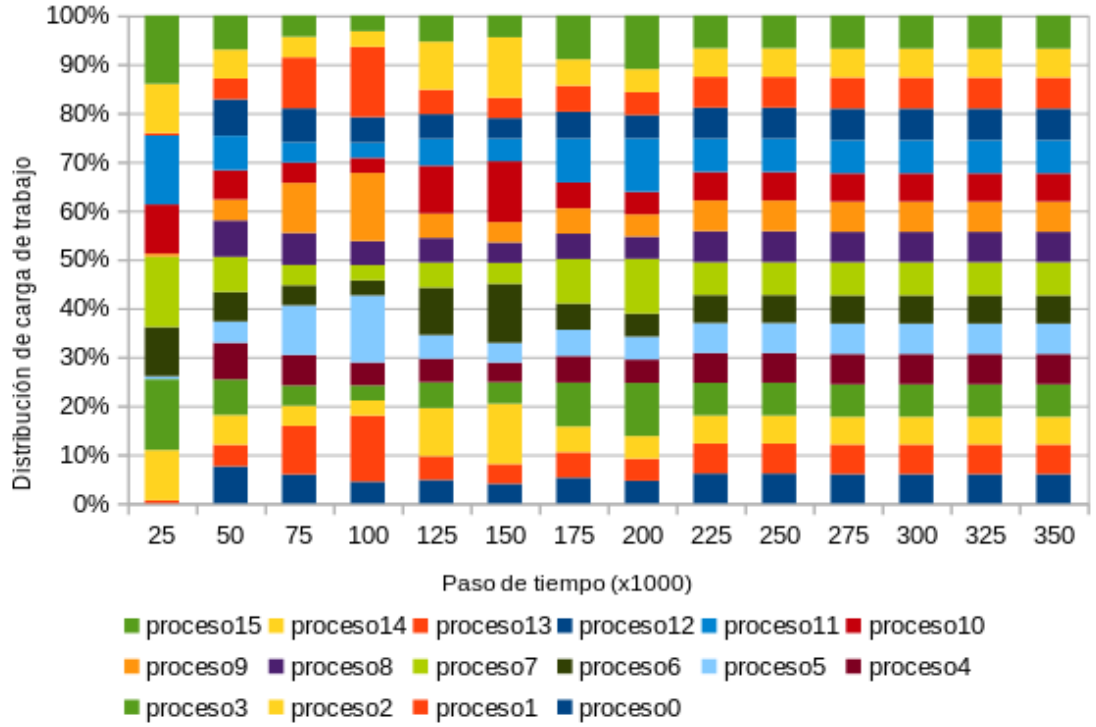


Figura A.2: Distribución de cargas de trabajo para la versión bf_mp_16procs del problema *binflow*

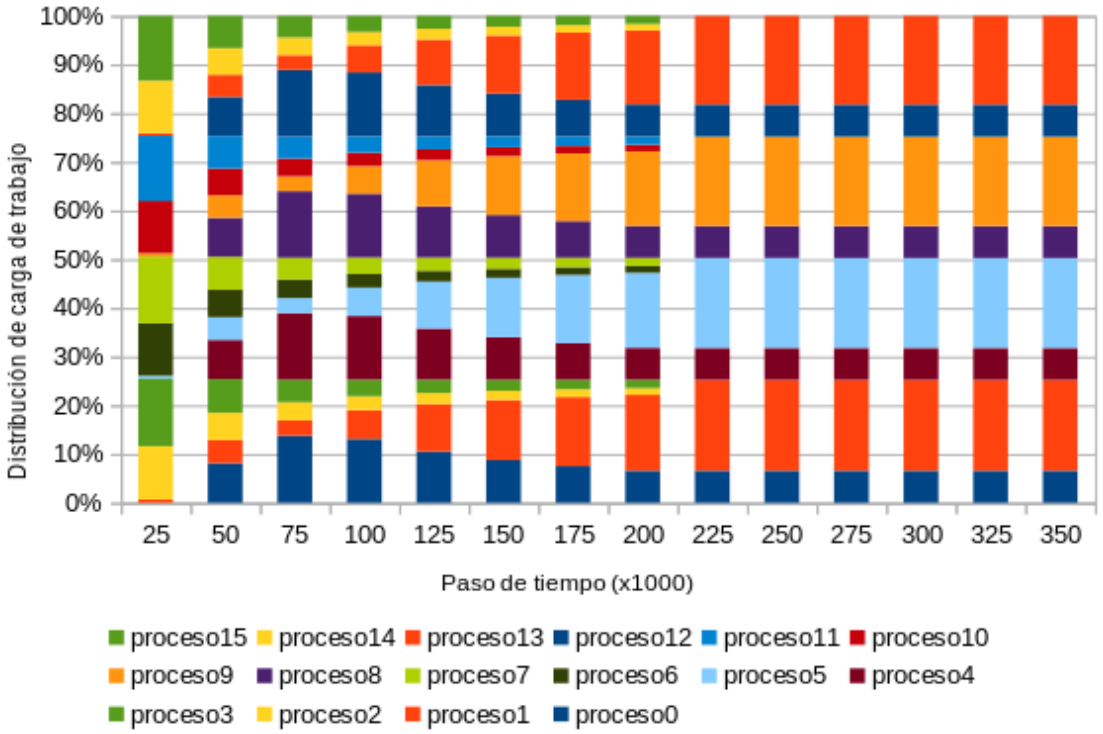


Figura A.3: Distribución de cargas de trabajo para la versión bf_16procs del problema *binflow*

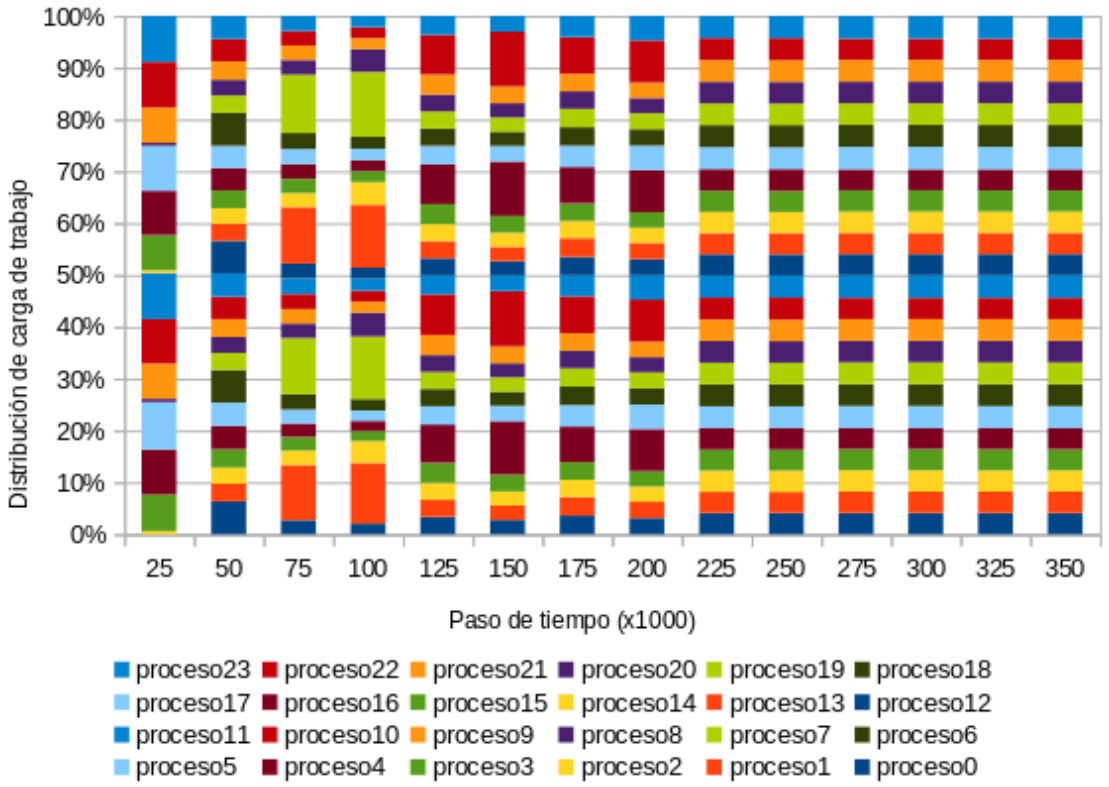


Figura A.4: Distribución de cargas de trabajo para la versión bf_mp_24procs del problema *binflow*

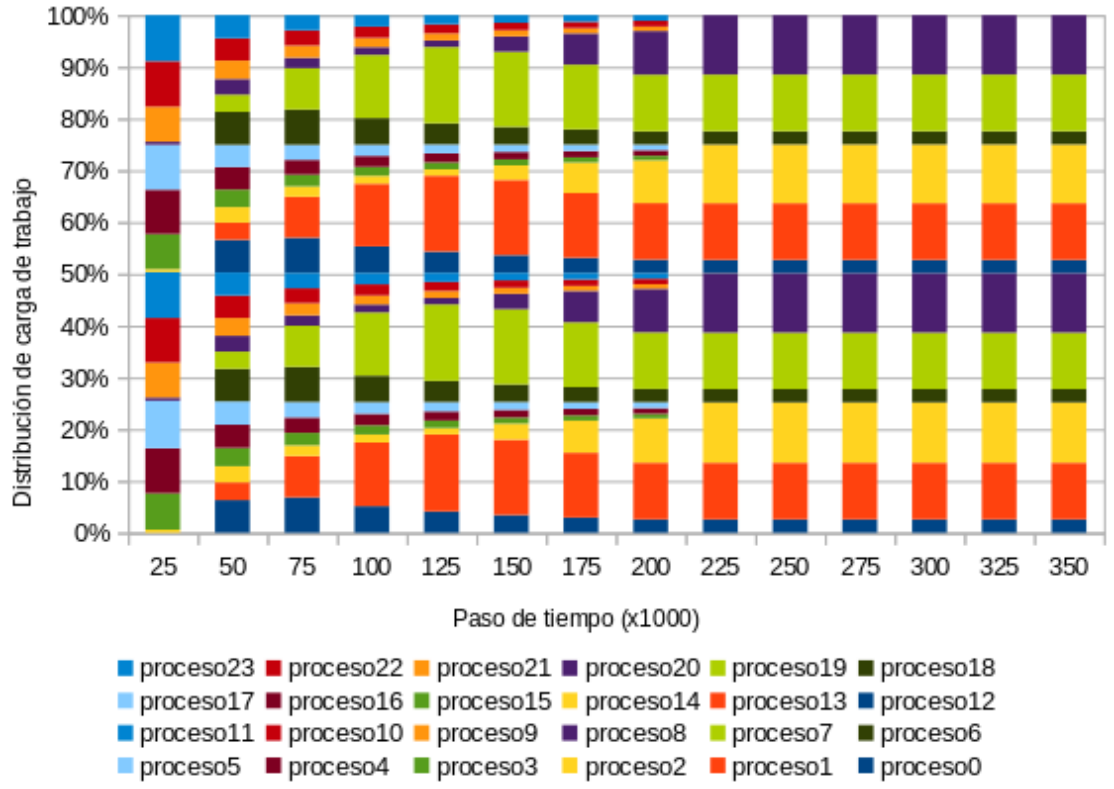


Figura A.5: Distribución de cargas de trabajo para la versión bf_24procs del problema *binflow*

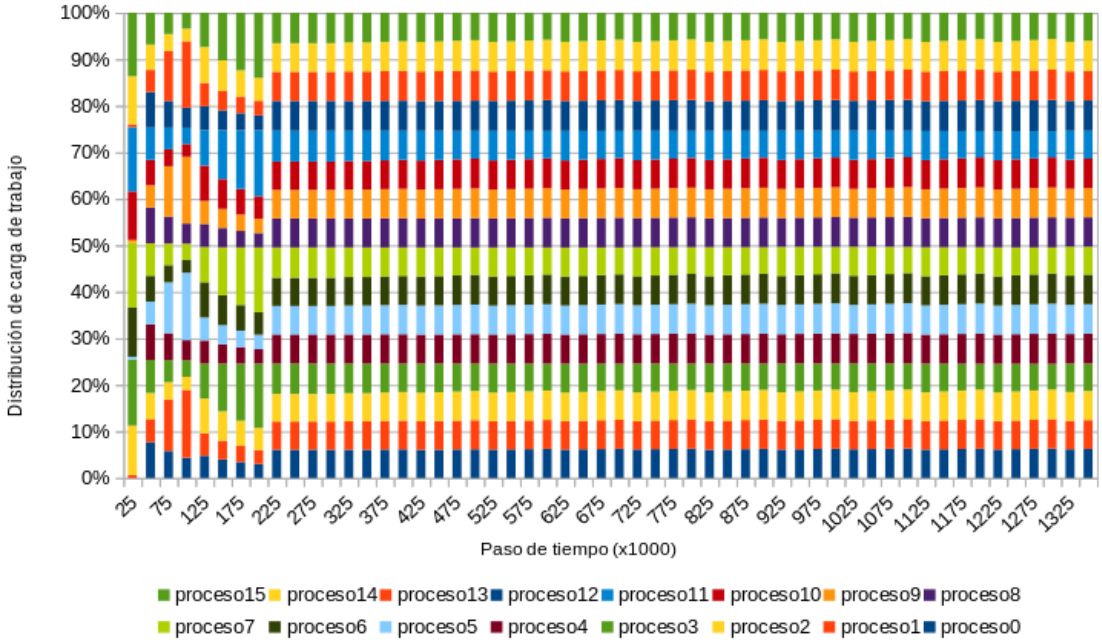


Figura A.6: Distribución de cargas de trabajo para la versión bf_mp_450000part del problema *binflow*

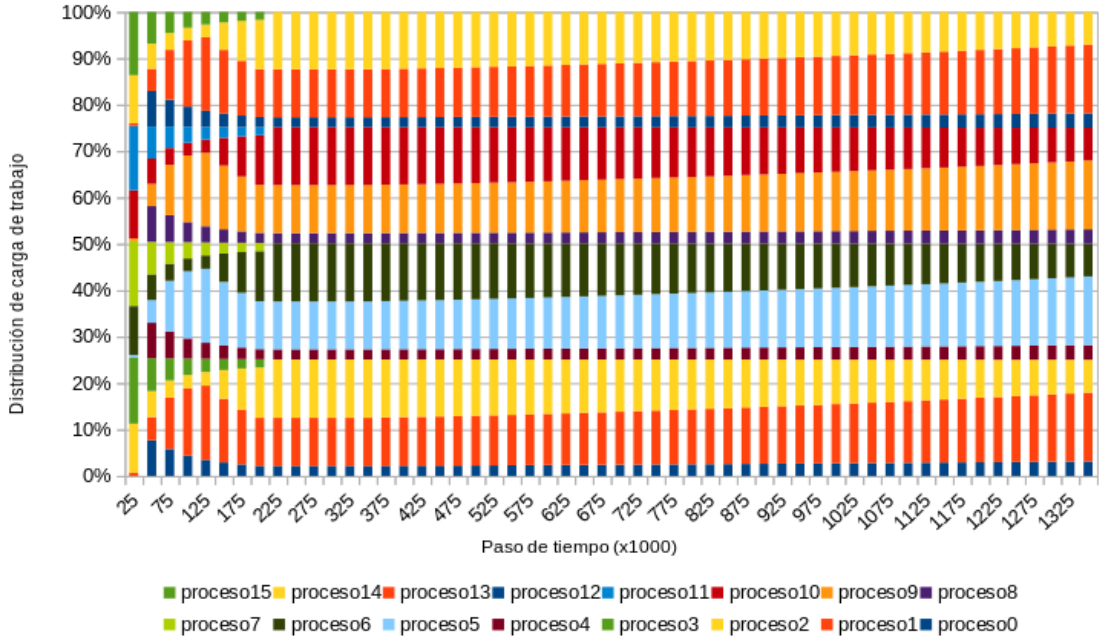


Figura A.7: Distribución de cargas de trabajo para la versión bf_450000part del problema *binflow*

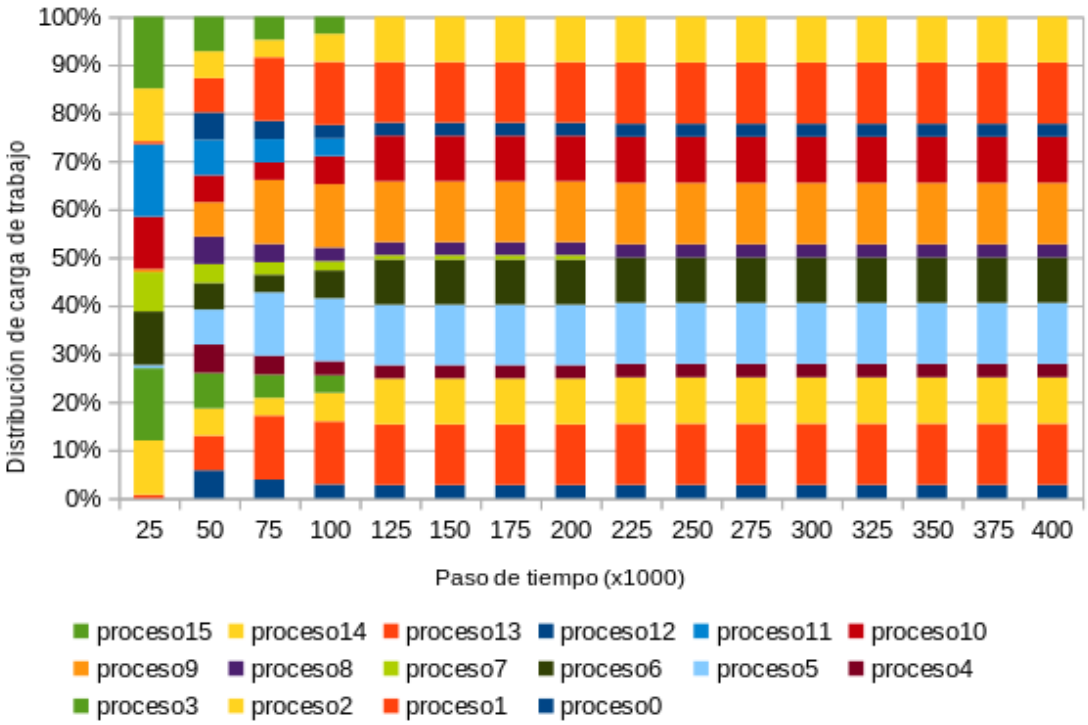


Figura A.8: Distribución de cargas de trabajo para la versión bf_16_mpart del problema *binflow*

Apéndice B

Tablas de promedio de partículas por proceso según avanza el tiempo

Paso de tiempo	**Proceso**								**Desv. estándar**
	0	1	2	3	4	5	6	7	
20000	0	3716	0	3807	0	3712	0	3765	2005
40000	3030	4414	3100	4515	3004	4409	3058	4470	752
60000	6795	4402	6847	4533	6750	4408	6802	4463	1255
80000	10517	4406	10611	4520	10495	4399	10594	4458	3266
100000	14252	4410	14402	4518	14210	4410	14336	4462	5265
120000	17981	4406	18170	4524	17931	4415	18110	4463	7268
140000	21710	4413	21940	4519	21666	4417	21873	4462	9271
160000	25418	4416	25713	4508	25427	4394	25655	4469	11282
180000	29166	4415	29471	4518	29137	4406	29421	4466	13282
200000	32911	4409	33253	4517	32860	4414	33174	4462	15287
220000	36677	691	36994	716	36597	690	36942	693	19299
240000	37371	0	37693	0	37320	0	37616	0	20045
260000	37369	0	37699	0	37319	0	37613	0	20045
280000	37369	0	37700	0	37319	0	37612	0	20045
300000	37370	0	37700	0	37319	0	37611	0	20045
320000	37370	0	37700	0	37319	0	37611	0	20045
340000	37370	0	37700	0	37319	0	37611	0	20045

Tabla B.1: Promedio de partículas por proceso según el paso de tiempo para la versión bf_8procs del problema *binflow*

Paso de tiempo	Proceso							
	0	1	2	3	4	5	6	7
25000	0	119	2062	2586	0	122	2017	2575
50000	3044	1779	2103	2593	3019	1753	2100	2554
75000	7745	1766	2103	2599	7696	1749	2115	2532
100000	9825	4436	2134	2575	9804	4365	2078	2561
125000	9865	9135	2116	2597	9837	9031	2078	2556
150000	9870	13868	2110	2580	9838	13712	2074	2566
175000	9872	18604	2117	2575	9839	18386	2092	2551
200000	9873	23325	2113	2589	9841	23089	2079	2546
225000	9876	27935	27	0	9843	27699	22	0
250000	9873	27957	0	0	9841	27729	0	0
275000	9873	27959	0	0	9841	27726	0	0
300000	9873	27959	0	0	9841	27726	0	0
325000	9873	27959	0	0	9841	27726	0	0
350000	9873	27959	0	0	9841	27726	0	0

(a) Promedio de partículas de los procesos 0 a 7

Paso de tiempo	Proceso								Desviación estándar
	8	9	10	11	12	13	14	15	
25000	0	125	2014	2517	0	103	2012	2498	1166
50000	3003	1725	2069	2512	2999	1714	2047	2486	496
75000	7683	1728	2057	2534	7697	1705	2036	2505	2516
100000	9832	4250	2061	2519	9841	4192	2041	2486	3183
125000	9854	8895	2077	2502	9872	8774	2071	2490	3684
150000	9861	13608	2056	2517	9883	13421	2033	2503	5077
175000	9866	18260	2059	2517	9884	18092	2040	2496	6834
200000	9868	22937	2051	2524	9887	22750	2019	2509	8744
225000	9869	27499	19	0	9890	27300	21	0	11640
250000	9867	27511	0	0	9886	27336	0	0	11656
275000	9867	27511	0	0	9886	27337	0	0	11656
300000	9867	27511	0	0	9886	27337	0	0	11656
325000	9867	27511	0	0	9886	27337	0	0	11656
350000	9867	27511	0	0	9886	27337	0	0	11656

(b) Promedio de partículas de los procesos 8 a 15

Tabla B.2: Promedio de partículas por proceso según el paso de tiempo para la versión bf_16procs del problema *binflow*

Paso de tiempo	Proceso							
	0	1	2	3	4	5	6	7
25000	0	111	1935	2721	0	114	1893	2708
50000	2841	1660	2286	2731	2818	1636	2281	2691
75000	3328	5591	2330	2332	3521	5716	2322	2338
100000	3335	10143	2316	2323	3528	10308	2323	2336
125000	4500	4543	9249	4975	4505	4552	9125	4831
150000	4501	4545	13921	4960	4506	4553	13638	4816
175000	6890	6835	6935	11791	7177	7041	7087	11948
200000	6890	6836	6939	16347	7178	7043	7093	16836
225000	9162	9218	8613	10048	9170	9217	8604	10112
250000	9159	9212	8611	10066	9167	9213	8602	10098
275000	8976	9096	8523	9965	9350	9324	8695	10200
300000	8976	9096	8523	9964	9350	9324	8695	10200
325000	8976	9098	8521	9964	9350	9325	8694	10200
350000	8976	9098	8521	9964	9350	9325	8694	10200

(a) Promedio de partículas de los procesos 0 a 7

Paso de tiempo	Proceso								Desviación estándar
	8	9	10	11	12	13	14	15	
25000	0	117	1890	2649	0	96	1887	2630	1189
50000	2803	1610	2250	2647	2799	1600	2228	2620	477
75000	3701	5776	2343	2339	3903	5882	2381	2448	1435
100000	3708	10477	2316	2350	3910	10800	2402	2424	3471
125000	4720	4693	9202	5170	4707	4705	9244	5031	2004
150000	4719	4694	14039	5216	4709	4706	13953	5024	4094
175000	6843	6831	6939	11827	7184	7028	7096	11798	2172
200000	6844	6833	6942	16400	7186	7029	7102	16504	4264
225000	9530	9443	8783	10246	9518	9464	8777	10096	539
250000	9527	9439	8781	10259	9516	9460	8777	10114	543
275000	9319	9326	8694	10155	9723	9568	8869	10218	557
300000	9320	9326	8694	10155	9723	9568	8869	10218	557
325000	9319	9327	8692	10155	9723	9570	8868	10218	558
350000	9319	9327	8693	10155	9723	9570	8868	10218	558

(b) Promedio de partículas de los procesos 8 a 15

Tabla B.3: Promedio de partículas por procesos según el paso de tiempo para la versión bf_mp_16procs del problema *binflow*

Paso de tiempo	Proceso							
	0	1	2	3	4	5	6	7
25000	0	0	113	1319	1630	1718	0	0
50000	2365	1284	1169	1336	1670	1693	2365	1234
75000	3846	4489	1162	1365	1654	1693	3824	4457
100000	3849	9209	1177	1347	1661	1688	3827	9166
125000	3850	13946	1162	1364	1648	1698	3829	13855
150000	3851	16375	3490	1326	1669	1703	3831	16349
175000	3851	16432	8127	1345	1645	1705	3831	16400
200000	3852	16436	12835	1332	1667	1702	3830	16405
225000	3852	16438	17460	21	0	0	3832	16409
250000	3851	16433	17487	0	0	0	3831	16401
275000	3851	16433	17488	0	0	0	3831	16401
300000	3851	16433	17488	0	0	0	3831	16401
325000	3851	16433	17488	0	0	0	3831	16401
350000	3851	16433	17488	0	0	0	3831	16401

(a) Promedio de partículas de los procesos 0 a 7

Paso de tiempo	Proceso							
	8	9	10	11	12	13	14	15
25000	115	1276	1608	1666	0	0	104	1274
50000	1150	1298	1619	1674	2365	1237	1127	1277
75000	1142	1320	1620	1663	3818	4452	1126	1282
100000	1135	1325	1599	1673	3828	9081	1153	1294
125000	1153	1297	1633	1661	3829	13760	1128	1275
150000	3325	1320	1620	1658	3829	16328	3184	1282
175000	7982	1289	1613	1670	3830	16393	7793	1297
200000	12646	1306	1619	1655	3830	16400	12428	1304
225000	17201	25	0	0	3830	16408	17025	17
250000	17245	0	0	0	3830	16401	17051	0
275000	17244	0	0	0	3830	16401	17050	0
300000	17244	0	0	0	3830	16401	17052	0
325000	17244	0	0	0	3830	16401	17052	0
350000	17244	0	0	0	3830	16401	17052	0

(b) Promedio de partículas de los procesos 8 a 15

Paso de tiempo	Proceso								Desviación estándar
	16	17	18	19	20	21	22	23	
25000	1603	1626	0	0	111	1274	1648	1665	771
50000	1623	1617	2388	1250	1138	1332	1615	1674	418
75000	1600	1646	3825	4509	1139	1314	1648	1656	1333
100000	1596	1633	3831	9164	1141	1321	1633	1669	2901
125000	1617	1631	3834	13803	1166	1312	1633	1666	4629
150000	1630	1624	3834	16325	3303	1348	1624	1672	5410
175000	1595	1647	3835	16394	7953	1317	1631	1675	5516
200000	1606	1639	3835	16399	12657	1319	1614	1684	6138
225000	0	0	3835	16406	17222	19	0	0	7764
250000	0	0	3835	16397	17238	0	0	0	7772
275000	0	0	3835	16397	17239	0	0	0	7772
300000	0	0	3835	16397	17237	0	0	0	7772
325000	0	0	3835	16397	17237	0	0	0	7772
350000	0	0	3835	16397	17237	0	0	0	7772

(c) Promedio de partículas de los procesos 16 a 23

Tabla B.4: Promedio de partículas por procesos según el paso de tiempo para la versión bf_24procs del problema *binflow*

Paso de tiempo	Proceso							
	0	1	2	3	4	5	6	7
25000	0	0	113	1319	1630	1718	0	0
50000	2365	1284	1169	1336	1670	1693	2365	1234
75000	1529	5974	1568	1494	1444	1529	1642	6120
100000	1530	8791	3151	1453	1470	1516	1643	9069
125000	3109	3110	3072	3634	6894	3330	3094	3103
150000	3110	3110	3076	3641	11592	3328	3094	3105
175000	4691	4689	4383	4474	9012	5439	4672	4664
200000	4691	4691	4383	4476	12200	7049	4673	4666
225000	6116	6131	6128	6125	6238	6272	6350	6288
250000	6116	6128	6124	6121	6264	6244	6349	6286
275000	6149	6147	6172	6162	6067	6419	6334	6278
300000	6149	6147	6172	6162	6067	6419	6334	6278
325000	6145	6145	6170	6160	6063	6416	6339	6280
350000	6145	6145	6170	6160	6063	6416	6339	6280

(a) Promedio de partículas de los procesos 0 a 7

Paso de tiempo	Proceso							
	8	9	10	11	12	13	14	15
25000	115	1276	1608	1666	0	0	104	1274
50000	1150	1298	1619	1674	2365	1237	1127	1277
75000	1549	1586	1641	1667	1627	6131	1515	1565
100000	3443	1588	1615	1677	1629	9084	3311	1576
125000	3047	3655	7340	3427	3088	3110	3061	3645
150000	3048	3659	12009	3395	3088	3112	3062	3652
175000	4400	4465	9359	5313	4665	4682	4405	4477
200000	4400	4467	12225	7027	4665	4685	4407	4477
225000	6244	6245	6350	6390	6108	6164	6120	6125
250000	6241	6240	6377	6373	6107	6161	6118	6123
275000	6264	6260	6153	6529	6121	6171	6157	6152
300000	6264	6260	6153	6529	6121	6171	6157	6152
325000	6268	6262	6154	6532	6116	6168	6156	6149
350000	6268	6262	6154	6532	6116	6168	6156	6149

(b) Promedio de partículas de los procesos 8 a 15

Paso de tiempo	Proceso								Desviación estándar
	16	17	18	19	20	21	22	23	
25000	1603	1626	0	0	111	1274	1648	1665	771
50000	1623	1617	2388	1250	1138	1332	1615	1674	418
75000	1587	1637	1733	6342	1570	1580	1590	1630	1737
100000	1590	1632	1734	9396	3309	1572	1584	1639	2799
125000	7163	3366	3099	3102	3067	3647	7162	3425	1494
150000	11811	3376	3101	3102	3069	3651	11863	3445	3265
175000	9192	5393	4682	4672	4397	4467	9292	5364	1742
200000	12205	7085	4684	4673	4397	4469	12219	7086	2881
225000	6255	6337	6375	6311	6261	6245	6363	6458	104
250000	6278	6323	6374	6309	6257	6243	6387	6456	106
275000	6069	6490	6341	6289	6270	6253	6153	6602	143
300000	6069	6489	6341	6289	6270	6253	6152	6602	142
325000	6066	6487	6345	6292	6274	6255	6153	6605	144
350000	6066	6487	6345	6292	6274	6255	6153	6605	144

(c) Promedio de partículas de los procesos 16 a 23

Tabla B.5: Promedio de partículas por procesos según el paso de tiempo para la versión bf_mp_24procs del problema *binflow*

Paso de tiempo	Proceso																Desv. estándar
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
25000	0	367	5959	7986	0	337	5943	7864	0	337	5850	7775	0	326	5823	7683	3611
50000	8662	5592	6292	7989	8586	5506	6242	7886	8641	5398	6134	7798	8583	5393	6101	7697	1364
75000	9761	18690	6263	8002	9747	18498	6217	7877	9748	18288	6131	7810	9747	18161	6119	7691	4984
100000	9759	32815	6313	7987	9746	32611	6202	7895	9748	32286	6187	7769	9747	32170	6082	7683	11359
125000	9760	44939	8255	8008	9747	44825	8065	7890	9750	44594	7941	7768	9747	44527	7748	7686	16743
150000	9763	46364	20888	7990	9749	46366	20587	7870	9748	46360	20276	7780	9750	46342	19966	7701	16439
175000	9764	46478	34913	7985	9751	46492	34518	7877	9749	46537	34103	7783	9753	46507	33832	7708	17508
200000	9765	46513	48876	7988	9753	46529	48589	7890	9750	46566	48172	7775	9759	46539	47839	7697	20619
225000	9764	46491	56691	0	9753	46512	56433	0	9749	46549	56054	0	9755	46537	55712	0	25289
250000	9765	46497	56671	0	9753	46513	56423	0	9750	46552	56059	0	9756	46540	55721	0	25291
275000	9765	46500	56669	0	9753	46515	56422	0	9750	46552	56058	0	9756	46540	55720	0	25291
300000	9764	46498	56672	0	9753	46513	56423	0	9750	46552	56057	0	9756	46540	55722	0	25291
325000	9765	46500	56669	0	9753	46514	56423	0	9750	46552	56057	0	9756	46540	55721	0	25291
350000	9765	46499	56670	0	9753	46513	56424	0	9750	46552	56057	0	9756	46540	55721	0	25291
375000	9764	46444	55910	0	9768	46493	55618	0	9785	46504	55303	0	9757	46483	54978	0	25042
400000	9723	46355	55146	0	9722	46382	54877	0	9732	46393	54560	0	9718	46371	54252	0	24800
425000	9698	46241	54413	0	9691	46236	54133	0	9722	46256	53847	0	9667	46295	53509	0	24559
450000	9684	46120	53670	0	9675	46098	53395	0	9691	46138	53125	0	9657	46141	52793	0	24315
475000	9669	46029	52880	0	9662	46020	52608	0	9675	46070	52327	0	9676	46059	51983	0	24056
500000	9660	45949	52054	0	9671	45929	51797	0	9679	45971	51524	0	9697	45969	51180	0	23793
525000	9653	45895	51229	0	9678	45892	50952	0	9672	45926	50693	0	9646	45931	50352	0	23546
550000	9667	45824	50373	0	9657	45843	50108	0	9679	45800	49899	0	9676	45836	49602	0	23293
575000	9667	45791	49490	0	9680	45804	49220	0	9678	45835	49007	0	9665	45859	48723	0	23048

Paso de tiempo	Proceso																Desv. estándar
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
600000	9680	45769	48626	0	9651	45785	48369	0	9677	45749	48220	0	9653	45765	47925	0	22806
625000	9685	45733	47749	0	9659	45757	47501	0	9674	45772	47335	0	9652	45741	47047	0	22563
650000	9687	45735	46817	0	9688	45812	46553	0	9673	45802	46391	0	9658	45738	46154	0	22318
675000	9685	45707	46010	0	9694	45721	45746	0	9657	45749	45561	0	9662	45720	45270	0	22084
700000	9656	45738	45098	0	9671	45780	44820	0	9671	45754	44621	0	9677	45726	44402	0	21848
725000	9665	45704	44255	0	9653	45700	43985	0	9652	45683	43820	0	9649	45689	43599	0	21636
750000	9683	45740	43327	0	9657	45761	43005	0	9654	45750	42866	0	9671	45716	42672	0	21414
775000	9678	45691	42470	0	9682	45706	42143	0	9660	45692	42040	0	9654	45705	41829	0	21205
800000	9653	45704	41553	0	9665	45733	41240	0	9655	45694	41183	0	9661	45704	40992	0	21007
825000	9652	45696	40653	0	9667	45738	40311	0	9663	45705	40291	0	9689	45719	40072	0	20801
850000	9655	45687	39781	0	9641	45747	39426	0	9683	45687	39442	0	9645	45692	39247	0	20612
875000	9645	45723	38878	0	9668	45792	38477	0	9651	45747	38501	0	9646	45723	38322	0	20430
900000	9669	45708	37981	0	9660	45771	37600	0	9656	45733	37637	0	9630	45740	37439	0	20252
925000	9675	45678	37099	0	9662	45746	36721	0	9668	45703	36762	0	9651	45719	36579	0	20069
950000	9688	45682	36167	0	9667	45746	35842	0	9675	45729	35867	0	9655	45726	35698	0	19904
975000	9661	45692	35297	0	9666	45788	34953	0	9662	45706	35012	0	9670	45724	34826	0	19742
1000000	9683	45705	34388	0	9660	45740	34084	0	9684	45688	34128	0	9667	45697	33979	0	19581
1025000	9659	45679	33513	0	9661	45774	33193	0	9671	45709	33225	0	9680	45703	33120	0	19439
1050000	9672	45749	32582	0	9666	45756	32327	0	9666	45732	32299	0	9648	45689	32248	0	19302
1075000	9640	45720	31715	0	9652	45732	31447	0	9675	45671	31470	0	9660	45660	31410	0	19160
1100000	9639	45740	30820	0	9641	45698	30543	0	9663	45660	30611	0	9665	45656	30579	0	19039
1125000	9667	45756	29905	0	9667	45674	29662	0	9661	45668	29715	0	9659	45675	29703	0	18927
1150000	9669	45753	28964	0	9649	45688	28725	0	9675	45713	28822	0	9683	45698	28831	0	18828

Paso de tiempo	Proceso																Desv. estándar
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1175000	9672	45757	28065	0	9658	45687	27788	0	9673	45713	27937	0	9692	45735	27914	0	18733
1200000	9680	45782	27144	0	9667	45701	26880	0	9653	45739	27020	0	9690	45773	27020	0	18653
1225000	9651	45745	26263	0	9673	45677	26058	0	9685	45710	26159	0	9663	45759	26149	0	18564
1250000	9662	45741	25357	0	9667	45698	25172	0	9666	45726	25262	0	9671	45751	25259	0	18491
1275000	9630	45766	24495	0	9683	45671	24337	0	9652	45718	24374	0	9646	45751	24354	0	18427
1300000	9663	45749	23630	0	9657	45675	23448	0	9658	45702	23500	0	9652	45742	23470	0	18365
1325000	9672	45718	22744	0	9641	45650	22571	0	9669	45687	22618	0	9651	45750	22606	0	18316
1350000	9651	45735	21826	0	9656	45712	21642	0	9679	45748	21690	0	9638	45752	21710	0	18290

Tabla B.6: Promedio de partículas por proceso según el paso de tiempo para la versión bf_450000part del problema *binflow*

Paso de tiempo	Proceso																Desv. estándar
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
25000	0	3	5959	7986	0	337	5943	7864	0	337	5850	7775	0	326	5823	7683	3611
50000	8662	3	6292	7989	8586	5506	6242	7886	8641	5398	6134	7798	8583	5393	6101	7697	1364
75000	9761	3	6263	8002	9747	18498	6217	7877	9748	18288	6131	7810	9747	18161	6119	7691	4984
100000	9759	3	6313	7987	9746	32611	6202	7895	9748	32286	6187	7769	9747	32170	6082	7683	11359
125000	13434	3	20991	21140	13945	13904	21345	21298	13926	13929	21336	21459	14400	14166	21711	20684	3858
150000	13438	3	21294	34596	13947	13911	21638	34906	13931	13934	21647	35506	14404	14179	22049	34531	9137
175000	13440	3	21349	48354	13951	13916	21691	48810	13932	13943	21696	49657	14406	14186	22107	48713	15415
200000	13442	3	21364	62197	13955	13921	21705	62767	13936	13946	21710	63733	14411	14193	22120	62992	21860
225000	27270	3	26822	29369	28104	27632	27255	29310	28092	27655	27276	29937	28831	28150	27698	29408	928
250000	27271	3	26825	29361	28107	27634	27257	29307	28093	27658	27277	29907	28832	28153	27701	29422	922
275000	27271	3	26825	29359	28108	27634	27258	29307	28093	27659	27277	29908	28832	28154	27702	29419	922
300000	27271	3	26825	29361	28107	27634	27257	29308	28093	27658	27277	29909	28832	28153	27701	29421	923
325000	27271	3	27136	28706	28107	27976	27585	28639	28093	28001	27594	29249	28832	28503	28036	28736	543
350000	27271	3	27136	28706	28107	27975	27585	28639	28093	28000	27594	29250	28832	28502	28036	28737	543
375000	27255	3	27195	27884	28061	27981	27641	27810	28031	28025	27652	28416	28812	28525	28080	27900	376
400000	27129	3	27210	27171	27875	27987	27647	27092	27892	27985	27667	27678	28690	28473	28086	27173	478
425000	26563	3	27015	27606	27329	27395	27452	27542	27346	27412	27469	28104	28125	27913	27891	27611	313
450000	26451	3	27009	26853	27215	27376	27457	26796	27242	27394	27461	27342	28007	27879	27902	26855	393
475000	26396	3	27007	26063	27139	27356	27450	26026	27178	27347	27458	26558	27940	27837	27890	26090	666
500000	26330	3	27010	25240	27088	27332	27449	25230	27127	27339	27462	25717	27884	27823	27892	25268	1015
525000	25878	3	26237	26533	26628	26366	26659	26572	26649	26359	26675	27058	27415	26833	27091	26636	333
550000	25868	3	26216	25710	26610	26355	26647	25735	26630	26332	26666	26216	27382	26817	27088	25802	501
575000	25845	3	26216	24859	26595	26325	26624	24918	26599	26311	26665	25372	27378	26777	27083	24975	825

Paso de tiempo	Proceso																Desv. estándar
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
600000	25859	3	26225	23999	26590	26321	26635	24040	26585	26308	26651	24495	27375	26770	27070	24106	1202
625000	25021	3	25265	25578	25752	25600	25672	25670	25746	25602	25692	26101	26511	26065	26103	25747	304
650000	25009	3	25262	24730	25755	25576	25673	24798	25754	25593	25695	25213	26517	26049	26095	24871	518
675000	25013	3	25248	23858	25758	25572	25670	23922	25758	25571	25683	24331	26512	26045	26084	23994	878
700000	25010	3	25278	22962	25739	25551	25686	23046	25753	25565	25676	23440	26510	26050	26080	23117	1269
725000	24269	3	24415	24656	24981	24710	24806	24792	24885	24648	24752	25115	25623	25104	25153	24833	307
750000	24268	3	24404	23796	24993	24705	24793	23928	24882	24635	24750	24231	25634	25090	25156	23951	533
775000	24269	3	24401	22911	25004	24699	24809	23022	24877	24639	24749	23345	25627	25103	25154	23057	899
800000	24262	3	24415	22016	24996	24701	24804	22148	24876	24636	24753	22453	25630	25104	25138	22191	1288
825000	23176	3	23697	23658	23887	24001	24082	23836	23734	23911	24014	24118	24467	24361	24401	23906	269
850000	23166	3	23696	22795	23882	24007	24097	22946	23720	23907	24010	23238	24448	24363	24398	23038	533
875000	23170	3	23687	21919	23885	24010	24087	22072	23717	23913	24008	22334	24450	24375	24389	22135	921
900000	23179	3	23689	21045	23864	24001	24088	21197	23717	23899	24010	21451	24442	24374	24400	21250	1318
925000	22453	3	22712	22821	23132	23161	23075	23028	22892	23004	22949	23203	23602	23454	23320	23061	255
950000	22445	3	22712	21943	23137	23164	23088	22130	22884	23012	22943	22309	23610	23443	23338	22164	516
975000	22445	3	22694	21079	23121	23151	23077	21278	22895	22999	22946	21425	23587	23437	23327	21298	886
1000000	22449	3	22701	20180	23132	23156	23082	20379	22894	23019	22947	20507	23594	23442	23336	20388	1300
1025000	21763	3	21907	21862	22433	22098	22251	22133	22154	21935	22130	22200	22849	22351	22485	22150	281
1050000	21748	3	21886	20994	22424	22084	22263	21256	22145	21935	22112	21327	22844	22345	22487	21286	540
1075000	21739	3	21889	20109	22432	22087	22278	20360	22147	21940	22112	20433	22842	22343	22487	20407	917
1100000	21754	3	21892	19223	22433	22105	22253	19466	22162	21943	22116	19525	22839	22356	22484	19500	1326
1125000	20676	3	21014	20929	21310	21297	21378	21222	21179	21232	21294	21312	21840	21614	21658	21343	238
1150000	20673	3	21006	20067	21312	21304	21372	20341	21185	21228	21294	20420	21835	21613	21654	20447	530

Paso de tiempo	Proceso															Desv. estándar
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1175000	20669	3	21008	19178	21317	21298	21370	19456	21180	21233	21289	19539	21837	21623	21634	19570	913
1200000	20655	3	21004	18299	21324	21311	21369	18561	21174	21224	21293	18666	21848	21602	21645	18680	1313
1225000	19910	3	20133	19995	20555	20282	20485	20296	20453	20222	20439	20406	21103	20593	20776	20465	271
1250000	19914	3	20141	19115	20556	20287	20489	19396	20460	20218	20429	19528	21100	20603	20765	19581	548
1275000	19913	3	20140	18234	20558	20285	20462	18541	20455	20223	20430	18630	21113	20595	20762	18696	933
1300000	19908	3	20127	17376	20552	20273	20468	17664	20463	20214	20429	17755	21102	20608	20748	17808	1328
1325000	19160	3	19340	19100	19756	19496	19653	19385	19506	19330	19520	19404	20100	19683	19818	19476	252
1350000	19175	3	19346	18215	19757	19488	19653	18508	19496	19325	19531	18505	20092	19685	19821	18583	565

Tabla B.7: Promedio de partículas por proceso según el paso de tiempo para la versión bf_mp_450000part del problema *binflow*

Paso de tiempo	Proceso							
	0	1	2	3	4	5	6	7
25000	0	323	5209	6930	0	316	5138	3758
50000	5487	6847	5347	7020	5587	6929	5253	3713
75000	5520	19072	5294	6926	5611	18911	5260	3717
100000	5520	25105	11400	7042	5612	25090	11137	3704
125000	5517	25384	19238	0	5606	25362	18915	1874
150000	5519	25403	19230	0	5606	25384	18871	1882
175000	5519	25404	19231	0	5606	25385	18867	1871
200000	5519	25404	19231	0	5606	25385	18867	1867
225000	5519	25404	19231	0	5606	25385	18867	0
250000	5519	25404	19231	0	5606	25385	18867	0
275000	5519	25404	19231	0	5606	25385	18867	0
300000	5519	25404	19231	0	5606	25385	18867	0
325000	5519	25404	19231	0	5606	25385	18867	0
350000	5519	25404	19231	0	5606	25385	18867	0
375000	5519	25404	19231	0	5606	25385	18867	0
400000	5519	25404	19231	0	5606	25385	18867	0

(a) Promedio de partículas de los procesos 0 a 7

Paso de tiempo	Proceso								Desviación estándar
	8	9	10	11	12	13	14	15	
25000	0	312	5031	6937	0	285	5072	6896	3200
50000	5418	6795	5322	6945	5412	6793	5294	6846	795
75000	5430	19070	5224	6916	5440	18957	5238	6909	6129
100000	5432	25259	11221	6986	5440	25196	11230	6836	8356
125000	5423	25505	19156	0	5436	25427	19031	0	10926
150000	5425	25522	19099	0	5438	25449	19053	0	10929
175000	5426	25521	19097	0	5438	25449	19057	0	10929
200000	5426	25521	19097	0	5438	25449	19057	0	10929
225000	5426	25521	19098	0	5438	25449	19056	0	10929
250000	5426	25521	19097	0	5438	25449	19057	0	10929
275000	5426	25521	19098	0	5438	25449	19056	0	10929
300000	5426	25521	19098	0	5438	25449	19056	0	10929
325000	5426	25521	19098	0	5438	25449	19056	0	10929
350000	5426	25521	19098	0	5438	25449	19056	0	10929
375000	5426	25521	19098	0	5438	25449	19056	0	10929
400000	5426	25521	19098	0	5438	25449	19056	0	10929

(b) Promedio de partículas de los procesos 8 a 15

Tabla B.8: Promedio de partículas por procesos según el paso de tiempo para la versión bf_16procs_mpart del problema *binflow*

Bibliografía

[1] JAEGER, H. M.; NAGEL, S. R.; BEHRINGER, R. P. Granular solids, liquids, and gases. *Review of Modern Physics*, v. 68, n. 4, p. 1259, 1996.

[2] RICHARD, P.; NICODEMI, M.; DELANNAY, R.; RIBIeRE, P.; BIDEAU, D. Slow relaxation and compaction of granular systems. *Nature Materials*, v. 4, n. 2, p. 121–128, 2005.

[3] DERESIEWICZ, H.; MINDLIN, R. D. *Elastic spheres in contact under varying oblique forces*. Columbia University. Department of Civil Engineering, 1952.

[4] CUNDALL, P. A.; STRACK, O. D. A discrete numerical model for granular assemblies. *Géotechnique*, v. 29, n. 1, p. 47–65, 1979.

[5] NIELSEN, J.; ASKEGAARD, V. Scale errors in model tests on granular media with special reference to silo models. *Powder Technology*, v. 16, p. 123–130, 1977.

[6] QUENTREC, B.; BROT, C. New method for searching for neighbors in molecular dynamics computations. *Journal of Computational Physics*, v. 13, n. 3, p. 430–432, 1973.

[7] VERLET, L. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, New York, v. 159, p. 98–103, 1967.

[8] PENG, B. *Discrete element method (DEM) contact models applied to pavement simulation*. Tesis de Maestría – Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2014.

[9] ARÉVALO-MENDOZA, G.; RAMOS-CAÑÓN, A. M.; PRADA-SARMIENTO, L. F. Análisis de confiabilidad en un modelo de descarga de silos de almacenamiento mediante el método de elementos discretos DEM. (Spanish). *Obras y Proyectos*, v. 15, p. 21–30, 2014.

[10] FERREZ, J. A.; LIEBLING, T. M. Parallel DEM simulations of granular materials. En: High-Performance Computing and Networking – HPCN-Europe, 2001, Berlin, Heidelberg. Editors HERTZBERGER, B.; HOEKSTRA, A.; WILLIAMS, R. Springer Berlin Heidelberg. p. 211–220.

[11] RAKOTONIRINA, A. D.; WACHS, A. Grains3D, a flexible DEM approach for particles of arbitrary convex shape. Part II: Parallel implementation and scalable performance. *Powder Technology*, v. 324, p. 18–35, 2018.

[12] AMRITKAR, A.; DEB, S.; TAFTI, D. Efficient parallel CFD-DEM simulations using OpenMP. *Journal of Computational Physics*, v. 256, p. 501–519, 2014.

[13] SHI, Z.; JIANG, T.; JIANG, M.; LIU, F.; ZHANG, N. DEM investigation of weathered rocks using a novel bond contact model. *Journal of Rock Mechanics and Geotechnical Engineering*, v. 7, n. 3, p. 327–336, 2015.

[14] SCHOLTÈS, L.; DONZÉ, F. V. A DEM model for soft and hard rocks: Role of grain interlocking on strength. *Journal of the Mechanics and Physics of Solids*, v. 61, p. 352–369, 2013.

[15] RENZO, A. D.; DI MAIO, F. P. Comparison of contact-force models for the simulation of collisions in DEM-based granular flow codes. *Chemical Engineering Science*, v. 59, n. 3, p. 525–541, 2004.

[16] SHEN, Z.; JIANG, M.; THORNTON, C. DEM simulation of bonded granular material. Part I: Contact model and application to cemented sand. *Computers and Geotechnics*, v. 75, p. 192–209, 2016.

[17] BURNS, S. J.; HANLEY, K. J. Establishing stable time-steps for DEM simulations of non-collinear planar collisions with linear contact laws. *International Journal for Numerical Methods in Engineering*, v. 100, p. 186–200, 2017.

[18] GOPALAKRISHNAN, P.; TAFTI, D. Development of parallel DEM for the open source code MFIX. *Powder Technology*, v. 235, p. 33–41, 2013.

[19] BERGER, R.; KLOSS, C.; KOHLMEYER, A.; PIRKERA, S. Hybrid parallelization of the LIGGGHTS open-source DEM code. *Powder Technology*, v. 278, p. 234–247, 2015.

[20] YAN, B.; REGUEIRO, R. A. A comprehensive study of MPI parallelism in three-dimensional Discrete Element Method (DEM) simulation of complex-shaped granular particles. *Computational Particle Mechanics*, p. 1–25, 2018.

[21] LIU, H.; TAFTI, D. K.; LI, T. Hybrid parallelism in MFIX CFD-DEM using OpenMP. *Powder Technology*, v. 259, p. 22–29, 2014.

[22] KLOSS, C.; GONIVA, C.; HAGER, A.; AMBERGER, S.; PIRKER, S. Models, algorithms and validation for opensource DEM and CFD-DEM. *Powder Technology*, v. 12, n. 2/3, p. 140–152, 2012.

[23] MAKNICKAS, A.; KAČENIAUSKAS, A.; KAČIANAUSKAS, R.; BALEVIČIUS, R.; DŽIUGYS, A. Parallel DEM software for simulation of granular media. *Informatica*, v. 17, n. 2, p. 207, 2006.

[24] KAČIANUSKAS, R.; MAKNICKAS, A.; KAČENIAUSKAS, A.; MARKAUSKAS, D.; BALEVIČIUS, R. Parallel discrete element simulation of poly-dispersed granular material. *Advances in Engineering Software*, v. 41, p. 52–63, 2010.

[25] SHIGETO, Y.; SAKAI, M. Parallel computing of discrete element method on multi-core processors. *Particuology*, v. 9, n. Multiscale Modeling and Simulation of Complex Particulate Systems, p. 398–405, 2011.

[26] AMRITKAR, A.; TAFTI, D.; LIU, R.; KUFRIN, R.; CHAPMAN, B. OpenMP parallelism for fluid and fluid-particulate systems. *Parallel Computing*, v. 38, n. 9, p. 501–517, 2012.

[27] PLIMPTON, S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, v. 117, p. 1–19, 1995.

[28] DEM Solutions. *EDEM - The leading Discrete Element Method (DEM) software*, visitado el 4 de diciembre de 2017. `https://www.edemsimulation.com/`.

[29] National Energy Technology Laboratory. *MFIX - Multiphase Flow with Interphase eXchanges*, visitado el 5 de diciembre de 2017. `https://mfix.netl.doe.gov/`.

[30] MARKAUSKAS, D.; KAČENIAUSKAS, A.; MAKNICKAS, A. Dynamic domain decomposition applied to hopper discharge simulation by discrete element method. *Information Technology and Control*, v. 40, n. 4, p. 286–292, 2011.

[31] BERGER, M. J.; BOKHARI, S. H. A Partitioning Strategy for Nonuniform Problems on Multiprocessors. *IEEE Transactions on Computers*, v. C-36, n. 5, p. 570–580, 1987.

[32] ANDREOTTI, B.; FORTERRE, Y.; POULIQUEN, O. *Granular Media: Between Fluid and Solid*. Cambridge University Press, 2013.

[33] WIKIPEDIA. *Silo*, visitado el 2 de enero de 2018. `https://en.wikipedia.org/wiki/Silo`.

[34] CHEN, J.; J.M.ROTTER; J.Y.OOI; Z.ZHONG. Flow pattern measurement in a full scale silo containing iron ore. *Chemical Engineering Science*, v. 60, p. 3029–3041, 2005.

[35] CHEN, J. *Interpretation of flow and pressures in full scale silos*. Tesis de Doctorado, University of Edinburgh, Edinburgh, Scotland, 1996.

[36] CHEN, J. F.; ROTTER, J. M.; OOI, J. Y.; ZHONG, Z. Correlation between the flow pattern and wall pressures in a full scale experimental silo. *Engineering Structures*, v. 29, n. 9, p. 2308–2320, 2007.

[37] HORABIK, J.; PARAFINIUK, P.; MOLENDA, M. Experiments and discrete element method simulations of distribution of static load of grain bedding at bottom of shallow model silo. *Biosystems Engineering*, v. 149, p. 60–71, 2016.

[38] NIELSEN, J. Pressures from flowing granular solids in silos. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, v. 1747, p. 2667, 1998.

[39] PAUL, C. W. Industrial particle flow modelling using discrete element method. *Engineering Computations*, v. 6, p. 698–743, 2009.

[40] JENIKE, A. W.; JOHANSON, J. R.; CARSON, J. W. Bin loads—parts 2, 3 and 4: Concepts, mass-flow bins, funnel-flow bins. *Journal of Engineering for Industry*, v. 95, n. 1, p. 1–16, 1973.

[41] ARNOLD, P. C. .; ROBERTS, A. W.; MCLEAN, A. G.; Tunra Bulk Solids Handling Research Associates. *Bulk solids: storage.* 2nd. ed. Newcastle, N.S.W.: TUNRA Bulk Solids Handling Research Associates, 1980.

[42] NEDDERMAN, R. M. *Statics and kinematics of granular materials.* Cambridge University Press, 1992.

[43] SCHURICHT, T.; FüRLL, C.; ENSTAD, G. Full scale silo tests and numerical simulations of the "cone in cone" concept for mass flow. En: LEVY, A.; KALMAN, H. (Eds.) *Handbook of Conveying and Handling of Particulate Solids.* Elsevier Science B.V., 2001. v. 10 de *Handbooks of Powder Technology*, p. 175–180.

[44] OOI, J. Y.; CHEN, J. F.; ROTTER, J. M.; ZHONG, Z.; HÄRTL, J.; JOHNSTONE, M.; FORSMO, S. F.; ANDREASSON, B.; THULIN, D. Experimental investigation of full scale silo with large inner tube. En: International Symposium "Reliable Flow of Particulate Solids IV", 2008, Tromsø, Norway.

[45] El Observador. *Hallan muertos a los dos operarios de fadisol*, visitado el 8 de abril de 2018. `https://www.elobservador.com.uy/hallan-muertos-los-dos-operarios-fadisol-n297567`.

[46] TU, J.; AHMADI, G.; INTHAVONG, K. *Computational fluid and particle dynamics in the human respiratory system.* Biological and Medical Physics, Biomedical Engineering. Springer, 2013.

[47] TU, J.; YEOH, G. H.; LIU, C. *Computational fluid dynamics: A practical approach.* 2nd. ed. Butterworth-Heinemann, 2013.

[48] TUCKER, P. G. *Unsteady computational fluid dynamics in aeronautics.* Number v.104 in Fluid Mechanics and Its Applications. Springer, 2014.

[49] WANFANG, S.; LIANG, G. On effective stochastic Galerkin finite element method for stochastic optimal control governed by integral-differential equations with random coefficients. *Journal of Computational Mathematics*, v. 36, n. 2, p. 183–201, 2018.

[50] GUOJIN, T.; ZIYU, L.; JINGHUI, S.; CHUNLI, W. Direct and inverse problems on free vibration analysis of cracked non-uniform beams carrying spring-mass systems by finite element method. *Journal of Vibroengineering*, v. 19, p. 7–12, 2017.

[51] GHOSH, S. *Distributed systems - an algorithmic approach.* Chapman & Hall/CRC, 2007.

[52] LYNCH, N. A. *Distributed algorithms.* Morgan Kaufmann, 1996.

[53] NESMACHNOW, S. Computación científica de alto desempeño en la Facultad de Ingeniería, Universidad de la República. *Revista de la Asociación de Ingenieros del Uruguay*, v. 1, p. 12–15, 2010.

[54] ANDRADE, N.; CIRNE, W.; BRASILEIRO, F.; ROISENBERG, P. OurGrid: A approach to easily assemble grids with equitable resource sharing. En: Job Scheduling Strategies for Parallel Processing, 2003, Berlin, Heidelberg. Editors FEITELSON, D.; RUDOLPH, L.; SCHWIEGELSHOHN, U. Springer Berlin Heidelberg. p. 61–86.

[55] ANDERSON, D. P. BOINC: A system for public-resource computing and storage. En: Fifth IEEE/ACM International Workshop on Grid Computing, 2004, Pittsburgh, USA. p. 4–10.

[56] AMAZON WEB SERVICES. *Amazon Elastic Compute Cloud (Amazon EC2)*, visitado el 15 de mayo de 2018. `https://aws.amazon.com/ec2/`.

[57] HASSAN, H. A.; MOHAMED, S. A.; SHETA, W. M. Scalability and communication performance of HPC on Azure Cloud. *Egyptian Informatics Journal*, v. 17, n. 2, p. 175–182, 2016.

[58] MARIANI, G.; ANGHEL, A.; JONGERIUS, R.; DITTMANN, G. Predicting cloud performance for HPC applications before deployment. *Future Generation Computer Systems*, v. 87, p. 618–628, 2018.

[59] BELGACEM, M. B.; CHOPARD, B. A hybrid HPC/cloud distributed infrastructure: Coupling EC2 cloud resources with HPC clusters to run large tightly coupled multiscale applications. *Future Generation Computer Systems*, v. 42, p. 11–21, 2015.

[60] The OpenMP Architecture Review Boards. *The OpenMP API specification for parallel programming*, visitado el 7 de abril de 2018. `https://www.openmp.org/`.

[61] MPI Forum. *MPI Forum*, visitado el 7 de abril de 2018. `http://mpi-forum.org/`.

[62] GABRIEL, E.; FAGG, G. E.; BOSILCA, G.; ANGSKUN, T.; DONGARRA, J. J.; SQUYRES, J. M.; SAHAY, V.; KAMBADUR, P.; BARRETT, B.; LUMSDAINE, A.; CASTAIN, R. H.; DANIEL, D. J.; GRAHAM, R. L.; WOODALL, T. S. Open MPI: Goals, concept, and design of a next generation MPI implementation. En: 11th European PVM/MPI Users'Group Meeting, 2004, Budapest, Hungary. p. 97–104.

[63] Argonne National Laboratory. *MPICH - High-Performance Portable MPI*, visitado el 7 de abril de 2018. `https://www.mpich.org/`.

[64] HANXLEDEN, R. V.; SCOTT, L. R. Load balancing on message passing architectures. *Journal of Parallel and Distributed Computing*, v. 13, n. 3, p. 312–324, 1991.

[65] COMPUTING, D. *Dcs computing*, visitado el 12 de diciembre de 2017. `https://www.dcs-computing.com/`.

[66] ZHANG, D.; JIANG, C.; LI, S. A fast adaptive load balancing method for parallel particle-based simulations. *Simulation Modelling Practice and Theory*, v. 17, n. 6, p. 1032–1042, 2009.

[67] HENDRICKSON, B.; DEVINE, K. Dynamic load balancing in computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, v. 184, n. 2, p. 485–500, 2000.

[68] RATHBONE, D.; MARIGO, M.; DINI, D.; VAN WACHEM, B. An accurate force–displacement law for the modelling of elastic–plastic contacts in discrete element simulations. *Powder Technology*, v. 282, n. Supplement C, p. 2–9, 2015.

[69] GARG, R.; GALVIN, J.; LI, T.; PANNALA, S. Open-source MFIX-DEM software for gas–solids flows: Part I – Verification studies. *Powder Technology*, v. 220, n. Selected Papers from the 2010 NETL Multiphase Flow Workshop, p. 122–137, 2012.

[70] O'SULLIVAN, C.; BRAY, J. D. Selecting a suitable time step for discrete element simulations that use the central difference time integration scheme. *Engineering Computations*, v. 21, n. 2-4, p. 278–303, 2004.

[71] TSUJI, Y.; TANAKA, T.; ISHIDA, T. Lagrangian numerical simulation of plug flow of cohesionless particles in a horizontal pipe. *Powder Technology*, v. 71, n. 3, p. 239–250, 1992.

[72] OTSUBO, M.; O'SULLIVAN, C.; SHIRE, T. Empirical assessment of the critical time increment in explicit particulate discrete element method simulations. *Computers and Geotechnics*, v. 86, p. 67–79, 2017.

[73] ALLEN, M. P.; TILDESLEY, D. J. *Computer simulation of liquids*. New York, NY, USA: Clarendon Press, 1989.

[74] HOCKNEY, R.; EASTWOOD, J. *Computer Simulation Using Particles*. Bristol, PA, USA: Taylor & Francis, 1988.

[75] CHIALVO, A. A.; DEBENEDETTI, P. G. On the use of the Verlet neighbor list in molecular dynamics. *Computer Physics Communications*, v. 60, n. 2, p. 215–224, 1990.

[76] LI, W.; YING, T.; JIAN, W.; YU, D. Comparison research on the neighbor list algorithms: Verlet table and linked-cell. *Computer Physics Communications*, v. 181, n. 10, p. 1682–1686, 2010.

[77] WELLING, U.; GERMANO, G. Efficiency of linked cell algorithms. *Computer Physics Communications*, v. 182, n. 3, p. 611–615, 2011.

[78] FERREZ, J. A.; LIEBLING, T. M. Dynamic triangulations for efficient detection of collisions between spheres with applications in granular media simulations. *Philosophical Magazine B: Physics of Condensed Matter; Statistical Mechanics, Electronic, Optical and Magnetic Properties*, v. 82, n. 8, p. 905–929, 2002.

[79] YAZDCHI, K.; BERTOLDI, K.; LUDING, S. Application of Delaunay Triangulation to discrete particle simulation. En: Twelfth Engineering Mechanics Symposium, 2009, De Werelt, Lunteren, The Netherlands.

[80] CHEW, L. P. Constrained Delaunay triangulations. *Algorithmica*, v. 4, n. 1, p. 97–108, 1989.

[81] LEE, D. T.; SCHACHTER, B. J. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences*, v. 9, n. 3, p. 219–242, 1980.

[82] FERREZ, J. A. *Dynamic triangulations for efficient 3D simulation of granular materials*. Tesis de Doctorado, École Polytechnic Fédérale de Lausanne, Instituts de Mathématiques, Lausanne, Suiza, 2001.

[83] FAZEKAS, S. *Distinct element simulations of granular materials*. Tesis de Doctorado, Budapest University of Technology and Economics, Department of Theoretical Physics, Budapest, Hungary, 2007.

[84] Sandia National Laboratories. *LAMMPS - Molecular Dynamics Simulator*, visitado el 4 de diciembre de 2017. `http://lammps.sandia.gov/index.html`.

[85] SWOPE, W. C.; ANDERSEN, H. C.; BERENS, P. H.; WILSON, K. R. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, v. 76, n. 1, p. 637–649, 1982.

[86] TUCKERMAN, M. E.; BERNE, B. J.; MARTYNA, G. J. Reversible multiple time scale molecular dynamics. *The Journal of Chemical Physics*, v. 97, n. 3, p. 1990–2001, 1992.

[87] TUCKERMAN, M. E.; BERNE, B. J.; ROSSI, A. Molecular dynamics algorithm for multiple time scales: Systems with disparate masses. *The Journal of Chemical Physics*, v. 94, n. 2, p. 1465–1469, 1991.

[88] LUEHR, N.; MARKLAND, T. E.; MARTíNEZ, T. J. Multiple time step integrators in ab initio molecular dynamics. *The Journal of Chemical Physics*, v. 140, n. 8, p. 084116, 2014.

[89] MÜLLER, D. *Techniques informatiques efficaces pour la simulation de milieux granulaires par des méthodes d'éléments distincts*. Tesis de Doctorado, École Polytechnic Fédérale de Lausanne, Instituts de Mathématiques, Lausanne, Suiza, 1996.

[90] ROUGIER, E.; MUNJIZA, A.; JOHN, N. W. M. Numerical comparison of some explicit time integration schemes used in DEM, FEM/DEM and molecular dynamics. *International Journal for Numerical Methods in Engineering*, v. 61, n. 6, p. 856–879, 2004.

[91] FRANKLIN, W. R.; NARAYANASWAML, C.; KANKANHALLL, M.; SUN, D.; ZHOU, M.; WU, P. Y. Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines. En: Ninth International Symposium on Computer-Assisted Cartography, 1989, Baltimore, USA.

[92] MÜLLER, C. R.; HOLLAND, D. J.; SEDERMAN, A. J.; SCOTT, S. A.; DENNIS, J. S.; GLADDEN, L. F. Granular temperature: Comparison of magnetic resonance measurements with discrete element model simulations. *Powder Technology*, v. 184, n. 2, p. 241–253, 2008.

[93] MÜLLER, C. R.; SCOTT, S. A.; HOLLAND, D. J.; CLARKE, B. C.; SEDERMAN, A. J.; DENNIS, J. S.; GLADDEN, L. F. Validation of a discrete element model using magnetic resonance measurements. *Particuology*, v. 7, n. 4, p. 297–306, 2009.

[94] SHENDE, S. S.; MALONY, A. D. The TAU Parallel Performance System. *The International Journal of High Performance Computing Applications*, v. 20, n. 2, p. 287–311, 2006.

[95] LINDLAN, K. A.; CUNY, J.; MALONY, A. D.; SHENDE, S.; MOHR, B.; RIVENBURGH, R.; RASMUSSEN, C. A tool framework for static and dynamic analysis of object–oriented software with templates. En: Supercomputing, ACM/IEEE 2000 Conference, Dallas, TX, USA. p. 49.

[96] JOHNSTON, J. F.; HUNT, F. A. Solutions for silo asymmetric flow problems. En: Second International Conference on Design of silos for strength and flow, 1983, Stratford-upon-Avon, England. p. 1–13.

[97] HARTLEN, J. *The wall pressure in large grain silos.* Swedish Council for Building Research, 1984.

[98] MUNCH-ANDERSEN, J. Shocks in coal silos. a case study. *Powder Handling & Processing*, v. 6, n. 4, p. 385–388, 1994.

[99] SCHWAB, C. V.; ROSS, J. I.; WHITE, G. M.; COLLIVER, D. G. Wheat loads and vertical pressure distribution in a full-scale bin. Part I — Filling. *Transactions of the American Society of Agricultural Engineers*, v. 37, n. 5, p. 1613–1619, 1994.

[100] JENIKE, A. W. Denting of circular bins with eccentric drawpoints. *Journal of the Structural Division*, v. 93, n. 1, p. 27–36, 1967.

[101] WOOD, J. G. The analysis of silo structures subject to eccentric discharge. En: Second International Conference on Desing of silos for Strength and Flow, 1983, Stratford-upon-Avon, England. p. 132–144.

[102] ROTTER, J. M. The analysis of steel bins subject to eccentric discharge. En: Second International Conference on Bulk Materials Storage, Handling and Transportation, 1986, Wollongong, Australia. p. 264–271.

[103] 3D Systems. *What Is An STL File?*, visitado el 22 de abril de 2018. `https://www.3dsystems.com/quickparts/learning-center/what-is-stl-file`.

[104] SCHROEDER, W.; MARTIN, K.; LORENSEN, B. *The visualization toolkit (4th ed.).* Kitware, 2006.

[105] AHRENS, J.; GEVECI, B.; LAW, C. ParaView: An End-User Tool for Large-Data Visualization. En: HANSEN, C. D.; JOHNSON, C. R. (Eds.) *Visualization Handbook.* Burlington: Butterworth-Heinemann, 2005. p. 717–731.

[106] ROCHA, L. M. *Comparação dos efeitos produzidos pelo consumo do grão de amaranto (Amaranthus cruentus L.) e de aveia (Avena sativa) em parâmetros da síndrome metabólica.* Tesis de Doctorado, Universidade Estadual de Campinas, Faculdade de Engenharia de Alimentos, Departamento de Alimentos e Nutrição, Campinas, Brasil, 2010.

[107] WIKIPEDIA. *Amaranthus cruentus L.*, visitado el 14 de febrero de 2018. `https://es.wikipedia.org/wiki/Archivo:Amaranthus_cruentus_L.,_seeds.JPG`.

[108] GAIA. *Gaia: Front Page*, visitado el 12 de diciembre de 2017. `http://www.nature-of-gaia.com/index.php`.

[109] KUDOS, S. A.; SOLANKI, C. Evaluation of physical properties of amaranth grain (*amaranthus paniculatus*). *International Journal of Chemical Studies*, v. 6, n. 2, p. 2197–2201, 2018.

[110] NAVARRO, H. A.; BRAUM, M. P. D. S. Linear and nonlinear hertizian contact models for materials in multibody dynamics. En: 22nd International Congress of Mechanical Engineering (COBEM 2013), 2013, Ribeirão Preto. Brazil: ABCM.

[111] HORABIK, J.; MOLENDA, M. Parameters and contact models for DEM simulations of agricultural granular materials: A review. *Biosystems Engineering*, v. 147, p. 206 – 225, 2016.

[112] SADOWSKI, A. *Modelling of failures in thin-walled metal silos under eccentric discharge*. Tesis de Doctorado, University of Edinburgh, Edinburgh, Scotland, 2010.

[113] GOVENDER, N.; WILKE, D. N.; KOK, S. Blaze-DEMGPU: Modular high performance DEM framework for the GPU architecture. *SoftwareX*, v. 5, p. 62–66, 2016.