168
2ej.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE QUIMICA

"IMPLEMENTACION DE ALGORITMOS DE ACELERACION EN UN PROGRAMA DE DINAMICA MOLECULAR DE CADENAS POLIMERICAS"

TESIS

QUE PARA OBTENER EL TITULO DE:

INGENIERO QUIMICO

P R E S E N T A :

JUAN ANDRES TORRES ROBLES

FACULTAD DE QUIMICA

MEXICO, D. F. UNAM. 1998.

PRESIDENTE	PROF. CASTAÑO MENESES VÍCTOR MANUEL
VOCAL	PROF. RAMOS MEJÍA AURORA DE LOS ANGELES
SECRETARIO	PROF. CRUZ GOMEZ MODESTO JAVIER
1ER SUPLENTE	PROF. JIMÉNEZ BEDOLLA JUAN CARLOS
2DO SUPLENTE	PROF. RIVERA MUÑOZ ERIC MAURICIO

SITIO DONDE SE DESARROLLÓ EL TEMA

FACULTAD DE QUÍMICA LAB 212 EDIF. E UNAM
CIUDAD UNIVERSITARIA
Y
DEPARTAMENTO DE INGENIERIA QUIMICA 1415 ENGINEERING DRIVE
UNIVERSITY OF WISCONSIN-MADISON

Dr. M. JAVIER CRUZ GOMEZ
ASESOR

JUAN ANDRÉS TORRES ROBLES
SUSTENTANTE

A mis padres.
Porque, en primer lugar,
yo sin ellos no
estaría aquí
en primer lugar.

A mi hermano.
Que me ha enseñado a
disfrutar la vida desde
otro punto de vista.

A Caro.
Por toda su ayuda, amor y comprensión.
(Ya veremos tú que pones en tu dedicatoria
... yo tampoco he perdido mi chispa).

A mi abuelita Anita.
Que me ha ayudado a ser un
mejor hombre a través de
sus oraciones.

A todos mis abuelos,
que no están conmigo físicamente,
pero que sus consejos
no abandonan mi mente.

Al resto de mi familia,
que no enumero,
más que nada
por el sólo hecho,
de que tendría qué escribir
otra tesis para justificar el espacio.

A mis padrinos Ceci y Salvador (Miki),
con quienes siempre me ha gustado
compartir las metas que voy alcanzando,
y siempre han estado ahí cuando los he necesitado.

A todos mis queridos profesores.
Que compartieron sus conocimiento con gran calidez.

A todas las personas amigas.
Que de algún modo me ayudaron a llegar aquí.

A los doctores
M. Javier Cruz Gómez
Juan J. de Pablo
por ayudarme a cumplir mis metas
y no olvidar que sólo con trabajo
se puede alcanzar
la excelencia.

A mis viejos y nuevos amigos.
Espero nunca perder contacto con ustedes,
o por lo menos,
volver a encontrarlos un día.
En orden de aparición en mi vida:
Hiram, Roberto, Rocío, Lilian, Gabriel, Ulises, Jorge,
Rubén, Norma, Ericka, Sandra, Julio, Braulio y todos
aquellos niños con los que jugué encantado
en el Colegio Vilaseca Esparza.

Tacho, Hugo, Miguel (Pats), Edgar, Miguel (Paz),
Nahum, Igor y demás miembros anónimos de la BLDX.

Gabriela y resto de compañeros de la
Preparatoria No. 1 "Gabino Barreda".

A Juan Manuel, Liz, Eva, Aldo (mi excelente equipo de proyectos).

Heidi, Fernando, Todd, Shyamal, Paul, Dave, Dan, Richie, Adam
And the rest of this huge de Pablo's group.

AL H. JURADO PROFESIONAL

ÍNDICE.

ÍNDICE DE FIGURAS, TABLA Y DIAGRAMAS

ÍNDICE DE GRÁFICOS

1. INTRODUCCIÓN

La dinámica molecular ha sido empleada desde hace varios años, para simular, a nivel atómico, el comportamiento de diversos materiales, estudiar sus características y formular nuevas ecuaciones de estado.

Ha sido hasta el advenimiento de computadoras más veloces, que ha sido posible tener acceso a los tiempos de simulación que se requieren para determinar propiedades y estudios de configuraciones de los sistemas de interés. Aún así, es necesario mantener programas en proceso por periodos de horas, incluso días, para simular fenómenos que en la naturaleza tienen duraciones del orden de fracciones de segundo.

Existen varios grupos actualmente trabajando en este ramo de la ciencia, tratando de hacer los algoritmos más eficientes, y no necesitar equipos de supercómputo para la obtención de los resultados deseados.

Esta tesis, codificará y validará algoritmos de aceleración, tales como:

- Árboles binarios para la organización de eventos. Este algoritmo fue propuesto por Rapaport D.C.[1] y permite un acceso más rápido a los eventos previamente calculados (un evento se define como las

colisiones entre partículas o bien el cambio de subcelda en donde se localiza la partícula). Este acceso más expedito se debe a la estructura del árbol binario y las leyes de organización que lo rigen, permitiendo un número mínimo de búsquedas para cualquier elemento (nodo) deseado y algoritmos sencillos de identificación de los elementos máximo y mínimo.

- División de la celda primaria de simulación en subceldas para determinar un vecindario de partículas. Este algoritmo descrito por Allen [2] permite reducir el número de parejas de partículas que "pueden" sufrir una colisión, tomando en cuenta que sólo las partículas más cercanas pueden interactuar entre sí.

- El uso extensivo de apuntadores. Debido a la naturaleza del problema, es posible asignar a cada partícula un número consecutivo el cual puede ser usado para evitar al máximo búsquedas sobre los indicios de las variables. El ejemplo más claro de esto corresponde a las subrutinas de borrado de partículas. Cada vez que un par de partículas ha sufrido una colisión, todos los eventos programados en donde se presentan cualquiera de estas dos partículas deben de ser eliminados, dado que esos cálculos están basados en las trayectorias y velocidades anteriores a la colisión. Así pues, el uso de apuntadores evita hacer búsquedas sobre

el árbol para eliminar estos nodos, pues cada partícula tiene una lista en donde se incluyen todos los eventos programados para ella y de donde se puede extraer la dirección de cada uno de los nodos involucrados de forma más eficiente.

El objetivo de la elaboración de esta tesis es el tratar de dar a conocer e instrumentar los algoritmos de aceleración antes descritos y que se han dado de forma aislada e integrarlos para generar un código que permita la simulación rápida de sistemas moleculares de gran tamaño.

De tal forma se contará con un código de simulación que servirá como punto de partida a un código mejorado, para ser utilizado en un proyecto patrocinado por la industria química e informática, para diseñar un nuevo polímero que será empleado en la fabricación de circuitos integrados de alto desempeño.

La forma en que este programa ayudará al desarrollo del nuevo polímero, una vez que se instrumenten interacciones partícula-partícula y partícula-superficie, será en la visualización de posibles estructuras e interacciones que pueden suceder en las películas monomoleculares. Debido a que el objetivo final es un polímero que permita cubrir zonas específicas en una superficie, es necesario saber cuál es la resolución máxima que se puede obtener, variando la naturaleza de

la molécula (interacciones y tamaño). Debido a la dificultad que existe en la preparación de polímeros, se espera que la simulación molecular indique qué tipo de polímero se debe intentar sintetizar y cuáles polímeros deber ser descartados.

El modelo que se empleará en la elaboración de este trabajo es el propuesto por Rapaport, el cual consiste en cadenas de esferas duras unidas por un enlace, que no presenta ninguna resistencia conforme las partículas se alejan o se acercan entre sí, pero sufre una tensión elástica cuando la distancia entre dos esferas vecinas que se encuentran en la misma molécula alcanzan la distancia de enlace, evitando así, que las partículas se separen una distancia mayor a δ (ver anexo II).

Los resultados obtenidos se validarán con información previamente publicada, tal como la función de distribución radial para esferas duras, y el coeficiente de compresibilidad.

Una vez que se haya confirmado que el programa devuelve resultados correctos, la siguiente fase será el comparar cuánto tiempo le lleva a este código el llegar a estos resultados, comparando éstos con códigos desarrollados por otros grupos de investigación y cuyos resultados han sido publicados.

El siguiente capítulo, describe de forma más profunda los trabajos publicados previos a la elaboración de esta tesis, de las ideas que servirán para

comprender algunos conceptos de mecánica estadística, así como de la importancia de la función de distribución radial –g(r)-, y la justificación por la que fue elegida como parámetro para determinar la validez del programa.

En el capítulo de Instrumentación, se mostrará básicamente la estructura del programa, así como una breve descripción de las subrutinas que lo conforman, haciendo hincapié en las ecuaciones de movimiento, tanto para el cálculo de tiempos de colisión, como de las ecuaciones para modificar las velocidades después de una colisión entre partículas.

En la parte correspondiente a Resultados, se incluirán las gráficas y tablas necesarias para comprender y comparar los resultados obtenidos de la simulación y verificar que el programa desarrollado es de calidad, al menos comparable a los programas que actualmente varios equipos de investigación tienen a su alcance.

En las Conclusiones, se incluirá además, un breve resumen de las mejoras que son necesarias para poder emplear este programa en la solución de problemas técnicos de frontera, como es el desarrollo de un nuevo polímero.

2. ANTECEDENTES.

2.1. *Breve descripción de un algoritmo típico de Dinámica Molecular.*

La simulación molecular puede ser llevada a cabo de distintas maneras, con una amplia gama de tipos de cálculo, siendo los más utilizados, los llamados métodos de tipo Montecarlo y los basados en Dinámica Molecular.

Los métodos del tipo Montecarlo se basan en la probabilidad de éxito de insertar una partícula en una configuración dada.

Esto es, los movimientos de partículas no siguen las leyes de movimiento de Newton, sino que, en cualquier momento, una partícula "desaparece" del sistema y cambia de lugar "apareciendo" en un lugar al azar en el sistema. Es aquí en donde se evalúa un factor de probabilidad que depende de las condiciones y naturaleza del fluido, el cual permite determinar el rechazo o aceptación del movimiento de la partícula.

La ventaja que tiene la Dinámica Molecular sobre cálculos del tipo Montecarlo, es que los movimientos de las partículas se hacen de forma dinámica, esto es, se basa en la evolución del sistema por medio de la solución sucesiva de

las ecuaciones de movimiento de Newton que gobiernan a cada una de las partículas del sistema y no artificial, permitiendo así estudiar propiedades de transporte y no quedar limitada a sólo propiedades en equilibrio como en el caso de las simulaciones basadas en el método Montecarlo, el cual, si bien es cierto, obtiene resultados de forma más rápida, lo hace a costa de ser específico a este tipo de problemas en donde lo que importan son las propiedades macroscópicas basadas en el promedio de las configuraciones instantáneas del sistema en cuestión.

Actualmente, se está tratando de desarrollar un polímero para la fabricación de circuitos integrados de alto desempeño, y se desea simular el comportamiento de este polímero sobre superficies de silicio, para poder determinar la resolución que se puede lograr en el grabado del circuito. Es por ello, que es necesario recurrir a métodos de Dinámica Molecular para cumplir este objetivo.

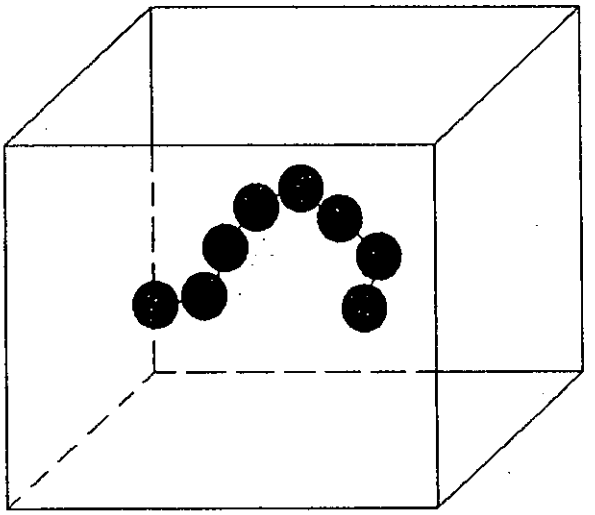
A lo largo de este trabajo, se compararán los resultados aquí logrados, con diversos valores obtenidos por medio de distintos métodos -incluyendo métodos tipo Montecarlo- pero no se hará ninguna comparación entre el tiempo de proceso requerido para llegar a esos resultados, a menos que el método corresponda a una Dinámica Molecular. Esto debe quedar claro, pues como ya se dijo previamente, los métodos Montecarlo obtienen resultados de forma más rápida, pero, dado que

la meta a largo plazo es calcular propiedades de transporte, estos métodos no pueden ser aplicados.

La Dinámica Molecular se basa en el cálculo de trayectorias de sistemas que varían desde cientos, hasta cientos de miles de átomos, y que evolucionan por medio de la solución repetida de las ecuaciones de movimiento de Newton.

La forma usual de llevar a cabo una simulación de Dinámica Molecular, es partir de una configuración inicial, como la que se muestra en la siguiente figura.

Figura 1. "Configuración inicial"



Ésta será la celda de simulación principal, en donde se fija temperatura y densidad del sistema.

La temperatura fija la energía cinética del sistema, así pues, se asignan velocidades a cada partícula de acuerdo a una distribución gaussiana, en donde la media corresponde a:

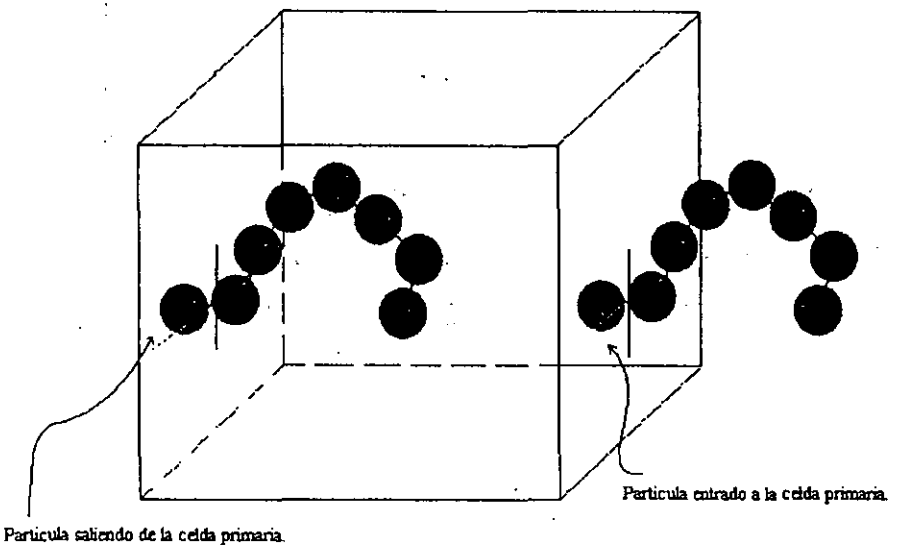
$$v = \sqrt{\frac{3kT}{m}}$$

Y donde las direcciones de la velocidad se fijan al azar y arbitrariamente. Cabe mencionar que, para el caso de esferas duras, la temperatura es por definición infinita, lo cual implica que, para este caso en particular, la distribución de velocidad en realidad no importa.

Pero debido a que no se desean imponer como límites a los bordes de nuestra celda de simulación, es necesario adoptar condiciones periódicas de frontera, esto es, cuando una partícula salga por un lado de la caja, otra entrará por el lado contrario a ella, manteniendo así fijo, el número de partículas presentes en la celda primaria, permitiendo de esta forma simulaciones a densidad constante.

El uso de condiciones periódicas de frontera, se representa en la siguiente figura.

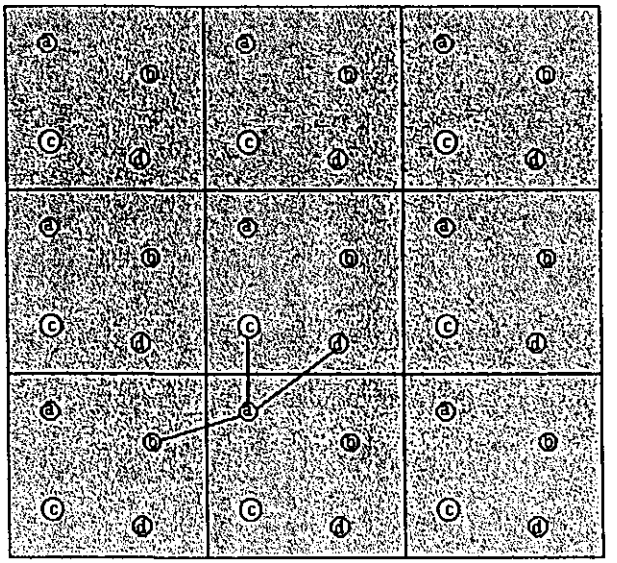
Figura 2. "Empleo de condiciones periódicas de frontera"



El primer intento de hacer más eficientes estos métodos y que por un tiempo fue un rasgo común en este tipo de simulaciones, es el empleo de la convención de la imagen mínima.

La convención de imagen mínima, establece que cada partícula puede interaccionar con la imagen más cercana a ella. Eso quiere decir que, no importa que la partícula más cercana se encuentre localizada en una celda distinta, siempre y cuando la distancia entre ésta y la partícula de interés sea mínima. Esto quedará claro con la ayuda del siguiente gráfico.

Figura 3. "Condiciones de imagen mínima"



Esta figura corresponde a una proyección en dos dimensiones de la celda primaria, y del resto de celdas imágenes a ella.

Como se puede observar en la figura, existen pares de partículas que si se toma una sola celda estarían demasiado alejadas como para poder interaccionar en un tiempo corto, pero las imágenes de éstas en las celdas vecinas, hacen que se deba de tomar en cuenta para el cálculo de un evento próximo a ocurrir.

Para poder obtener las propiedades macroscópicas que nos interesan, se obtiene el promedio de ciertas cantidades instantáneas, que se irán muestreando

durante el tiempo de simulación, siendo necesario conocer de antemano los tipos de interacciones que existirán entre las partículas; de hecho, el establecer el tipo de interacción que pueden sufrir las partículas nos permite hacer la siguiente clasificación para las simulaciones de Dinámica Molecular:

1.- **Dinámica Molecular continua,** en la cual la fuerza que actúa sobre las partículas es debido a un potencial continuo, por ejemplo del tipo Lennard-Jones.

2.- **Dinámica Molecular discontinua,** en la que, como su nombre lo indica, el potencial al que están sujetas las partículas es discontinuo, como el caso del modelo de esferas duras.

En el caso del empleo de potenciales continuos, las ecuaciones de movimiento se van resolviendo a intervalos regulares de tiempo, por medio de técnicas de diferencia finita [3]. Aunque es posible estudiar fenómenos por periodos prolongados usando equipos de supercómputo para emplear potenciales continuos, el hecho de que el paso entre evento y evento es fijo y muy pequeño, y que el costo de tiempo de proceso es elevado para este tipo de equipos, hace que sea más atractivo el empleo de los potenciales discontinuos.

Una simulación de Dinámica Molecular discontinua, evoluciona en una base evento-a-evento, esto es, una vez que se ha calculado el tiempo al que

sucederán todos los siguientes eventos, se toma el más próximo, se avanza el sistema a ese punto en el tiempo, se calculan nuevos eventos (como colisiones o cambios de celda, como se explicará más adelante) y se repite el proceso.

2.2. *Modelo empleado y conceptos básicos de mecánica estadística.*

Esta tesis se ha basado en el modelo de Rapaport [1], para simular cadenas de esferas duras. Es el modelo más simple que se puede encontrar para el caso de cadenas poliméricas. En este modelo, se permite que la distancia entre partículas unidas varíe libremente en un intervalo entre σ y $\sigma + \delta$, donde δ es mucho menor que σ (típicamente en el rango de 0.02 a 0.1).

La cadena se compone de n esferas de diámetro σ unidas unas a otras como un "collar de perlas" y con los enlaces que se pueden deslizar por la superficie. Así pues, las esferas adyacentes pueden tener dos tipos de colisiones intramoleculares; una es la colisión entre la superficie de las esferas, y la otra es la atracción súbita que sufren ambas partículas una vez que la distancia entre ellas es igual a $(\sigma+\delta)$.

Al momento en que el enlace está completamente tenso, debido a que las partículas se están alejando, se le llamará también colisión, debido a que el cálculo para este evento, es prácticamente idéntico a una colisión elástica entre esferas, y

además es ampliamente usado en las publicaciones utilizadas como referencia para el desarrollo de esta tesis.

Es oportuno en este momento, hablar de la importancia de la mecánica estadística, esto con el fin de establecer en la mente del lector el porqué se eligió algo tan poco "usual" como pudiera ser la función de distribución radial, como un parámetro para determinar la validez del programa de cómputo desarrollado.

Citando a Donald McQuarrie [4] "La mecánica estadística es la rama de la física que estudia desde un punto de vista microscópico a los sistemas macroscópicos. La meta de la mecánica estadística es el entendimiento y la predicción de fenómenos macroscópicos y el cálculo de propiedades macroscópicas a partir de las propiedades de moléculas individuales que forman el sistema".

Actualmente, la mecánica estadística se emplea en muy diversos ramos de la ciencia, resolviendo problemas sobre gases, líquidos, soluciones, polímeros, adsorción, metales, espectroscopia, teoría de transporte, DNA, propiedades eléctricas de la materia, entre otros. Y es en la mecánica estadística donde información como g(r), es de suma importancia para el entendimiento sobre todo de líquidos, por dos razones:

1. Si se asume que la energía potencial total de un sistema de N-cuerpos es aditiva con respecto al número de parejas presentes en el sistema, entonces TODAS las funciones termodinámicas del sistema pueden escribirse en términos de g(r).

2. Es una cantidad que puede ser determinada por difracción de rayos X, y sirve para determinar el orden de corto alcance en un fluido.

El sentido físico de la función g(r), corresponde a la probabilidad de encontrar a cierta distancia (tomando cualquier partícula como punto de referencia), una partícula vecina a la distancia (r) elegida. El sólo observar esta función y conociendo su significado, da una idea de la estructura del material, de acuerdo al arreglo que tienen las partículas en el bulto, así como de la densidad del mismo.

Quiero agradecer al profesor Arun Yethiraj del Departamento de Química de la Universidad de Wisconsin-Madison, por permitirme usar su código para determinar g(r), a diversas fracciones de empaquetamiento, así como al Dr. Fernando Escobedo, del Departamento de Ingeniería Química, de la mencionada universidad, por permitirme usar su código para el cálculo de g(r) a partir de archivos de configuraciones.

El código del profesor Yethiraj está basado en la ecuación de Percus-Yevic, la cual fue elegida debido a que devuelve los resultados de forma rápida y, a pesar de que no se trata de una teoría exacta, se ha encontrado que los resultados que predice son muy similares a los resultados encontrados de forma exacta por Barker y Henderson quienes emplearon métodos del tipo Montecarlo [4].

La forma de calcular la función de distribución radial sigue el siguiente algoritmo. Se fija una distancia desde una partícula central hacia el resto del fluido. Esta distancia define el radio de una esfera alrededor de esa partícula en especial. Posteriormente se hace un conteo del número de partículas cuyo centro de masa cae dentro de esta esfera y se determina la densidad local que corresponde a la distancia r elegida. Esta operación se repite para todas las partículas en la celda de simulación, obteniéndose así un promedio a la distancia definida inicialmente, y se repetirá esta operación incrementando desde cero hasta la mitad de la celda de simulación el valor del radio de la esfera. La razón por la que sólo se puede medir g(r) hasta una distancia menor o igual al lado de la celda de simulación (en el caso de una celda cúbica), corresponde al empleo de las condiciones periódicas de frontera.

En el caso de cadenas, una vez que se ha elegido una partícula en especial, las partículas vecinas a ésta que se localizan en la misma cadena son descartadas. De esta forma se evita un "pico" de grandes dimensiones a la distancia lσ que sólo

indica que las cadenas se encuentran interconectadas. El resto del algoritmo es el mismo que el descrito anteriormente.

El otro valor que servirá como punto de comparación será el factor de compresibilidad, al cual estamos más familiarizados. Para poder calcular este factor, se instrumentó en el programa la siguiente fórmula, la cual se escribirá en su forma vectorial.

$$Z = n + \frac{\sum_{colisiones} \left(r_{ij} \cdot \Delta v_i \right)}{t_s N_c v_s^2}$$

donde:

Z : Factor de compresibilidad

n : Número de monómeros en la cadena

Δv_i : Cambio de velocidad de una partícula debido a la colisión

$r_{ij} = r_i - r_j$: Diferencia en el vector posición

t_s : Tiempo total de simulación

N_c : Número de cadenas poliméricas

v_s : Suma de las normas de las velocidades de cada partícula.

2.3. Limitaciones *de la Dinámica Molecular.*

Se usan métodos como la simulación molecular para la solución de problemas debido a que existen aún muchos aspectos que han sido observados experimentalmente en el bulto de polímeros, y que no pueden ser medidos directamente de forma experimental.

Lamentablemente, no ha sido sino hasta la fecha que, gracias a equipos de cómputo más veloces y de menor costo, se ha permitido alcanzar los tiempos de simulación requeridos para poder extraer del programa alguna información útil.

Por ejemplo, para cadenas "largas" -mayor a 50 monómeros- y con el uso de potenciales del tipo Lennard-Jones, se ha estudiado el sistema de cadenas de metileno en el cual el uso de este tipo de potencial restringe las simulaciones a unos cuantos picosegundos de tiempo real y por tanto no se pueden encontrar propiedades como la viscosidad [5] debido a que, para calcular esta propiedad, es necesario tener información de la transferencia de momento a través de planos imaginarios en la celda de simulación y 10^{-12} segundos es un tiempo insuficiente para determinar de forma precisa la viscosidad, pues se llega al orden de las fluctuaciones intrínsicas del sistema.

Para tener una idea más clara, de acuerdo a Alder y Wainwright [6], se requieren temperaturas bajas (menor energía cinética de las partículas), del orden de cientos de miles de colisiones, para simular tiempos reales del orden de milimicrosegundos (nanosegundos), a densidades bajas y simulando partículas discretas. Esto corresponde al mejor de los casos, de tal forma que podemos decir que con las computadoras actuales es posible simular en tiempos razonables de cómputo desde 10^{-12} (picosegundos para sistemas densos polimoleculares) hasta 10^{-9} (nanosegundos para sistemas poco densos y de partículas discretas).

Es por ello que, ha existido una gran variedad de intentos por optimar los métodos de simulación molecular, y el objetivo de esta tesis es el tratar de dar a conocer e instrumentar todos estos intentos que se habían dado de forma aislada e integrarlos para generar un código que permita la simulación rápida de sistemas moleculares por medio de la Dinámica Molecular.

2.4. *Técnicas de optimización actuales.*

El trabajo más completo en la actualidad es la reciente publicación de Steven W. Smith y Carol. K. Hall [3], en donde plantean la inquietud de desarrollar programas más eficientes en los que se pueda hacer uso de estaciones de trabajo convencionales y evitar así el uso de supercomputadoras para este tipo de aplicaciones.

Algunos de los métodos de optimización implementados por Smith y Hall, son los siguientes.

a) *Listas de Tiempo.*

La necesidad de emplear este tipo de listas, es debido a que permite evolucionar el sistema, por periodos más largos, sin que se tenga que llamar a una subrutina para actualizar a todas las partículas en cada evento.

Así pues, cada partícula tendrá un tiempo local asociado, que corresponde a la última vez que su posición fue actualizada. De esta forma, cada vez que se hace el cálculo de un evento, en lugar de actualizar todas las partículas del sistema, simplemente se realiza una actualización temporal entre las partículas involucradas en la colisión. Además, el avance sólo lo realizará una de las partículas (la que no ha sido actualizada en mayor tiempo).

b) *Listas de partículas vecinas.*

Para poder determinar los eventos que habrán de realizarse, es necesario hacer el cálculo entre todas las parejas de partículas del sistema.

Dado que ya mencionamos que los sistemas pueden ir desde cientos hasta cientos de miles de partículas, es necesario, reducir al máximo, el número de partículas para hacer este cálculo.

Para sistemas en donde no existe un gran número de partículas (menor a varios cientos de partículas), la forma de integrar estas listas de "vecinos", es el establecer un radio mayor al diámetro de la partícula, y hacer una búsqueda individual, para establecer cuáles partículas se encuentran en el interior del vecindario fijado de esta forma.

c) *Listas de subceldas.*

Cuando el número de partículas excede a varios cientos de ellas, el uso de listas de vecinos, resulta inadecuado, debido al gran esfuerzo computacional que se debe de hacer para mantener esas listas actualizadas, pues se debe recordar que la lista de partículas vecinas irá cambiando conforme avanza el tiempo, y para determinar cuáles partículas serán incluidas en la lista es necesario determinar la distancia entre cada par del sistema.

La forma de evitar este problema es dividiendo la celda principal en subceldas. Estas subceldas, tendrán una longitud de arista mayor al diámetro de

las partículas, pero nunca mayor a dos diámetros atómicos, si es que se desea tener un óptimo desempeño en el uso de este algoritmo.

El uso de subceldas restringe el dominio de búsqueda a las 26 subceldas contiguas. Aun así, se debe de recordar que cada celda puede tener asociada a más de una partícula. Esto es debido a que, si el centro de masa de la partícula tiene coordenadas tales que están contenidas en el interior de la subcelda, es posible acomodar hasta dos partículas en una celda, si el tamaño de la arista de la subcelda fuera un diámetro atómico (σ).

Este procedimiento no requiere de búsquedas individuales de los pares más cercanos (pues incluye un evento más, el cual corresponde al cruce de partículas en las celdas). De esta forma, cada partícula tiene asociada una celda y los procedimientos se encargan de mantener listas actualizadas a cada momento de las partículas presentes en cada subcelda, y sólo se requiere una búsqueda no muy larga sobre la celda para determinar a los posibles candidatos para sufrir una colisión.

Otra ventaja es que en este caso, las subceldas vecinas, permanecen constantes, de tal forma que sólo se crea la estructura de celdas vecinas al inicio de la simulación.

d) *Programación de los eventos.*

La búsqueda del evento más próximo, requiere un gran gasto de tiempo de proceso para sistemas muy grandes, si es que se usan listas normales para programar estos eventos. Es aquí donde el empleo de un árbol binario logra reducir este gasto computacional. De acuerdo a Rapaport [1], se alcanza un código al menos 30% más rápido en comparación con códigos similares que emplean listas para su programación de eventos.

e) *Falso posicionamiento.*

Cada partícula tendrá asociada un tiempo local, que corresponde a la última vez que su posición fue actualizada.

Durante la simulación, existen varias formas de actualizar las coordenadas de las partículas.

1.- Debido a que se encuentran a una distancia σ y sucede una colisión entre esferas duras.

2.- Existió un cambio de subcelda.

3.- Se ejecutó un muestreo de las configuraciones del sistema.

Pero para poder hacer el cálculo de los eventos es necesario conocer la posición en la que se encuentra cada partícula en cada instante, así pues, cada vez que se desea calcular un evento, se avanza temporalmente a la partícula o partículas involucradas a un mismo tiempo, que lo determina la partícula que ha sido actualizada a un tiempo más cercano al real.

Una forma de pensar en lo anterior es el siguiente ejemplo:

Supongamos que estamos al inicio de la simulación de tal forma que el tiempo transcurrido es t=0.

Una vez que se calculan los eventos por primera vez, supongamos que tenemos un sistema de 2 partículas (**a**, **b**) con los siguientes eventos programados.

Tabla 1. "Eventos programados"

Tipo de evento	partícula a	partícula b
Colisión	t=3	t=3
cambio de celda	t=1.5	t=4
contracción de enlace	t=10	t=10

Como se puede observar, el primer evento que sucederá, será el cambio de celda por parte de la partícula **a**, así pues el tiempo real, será t=1.5 una vez que suceda este evento. Pero no se han actualizado las coordenadas de la partícula **b**, por lo que, cuando suceda la colisión entre ambas partículas, programadas a ocurrir en el tiempo t=4, sólo se requerirá que la partícula **b** se avance temporalmente a un tiempo t=1.5, y así, ambas partículas se encontrarán en las posiciones que les corresponde a un tiempo dado.

3. INSTRUMENTACION.

La parte medular del programa es la integración exitosa de todos los algoritmos de optimización. El programa debe de ser pensado como un todo, y no una colección de subrutinas aisladas, para aprovechar al máximo las oportunidades de modificación de las variables que se dan en algunos procedimientos y que puedan servir después durante la ejecución del mismo..

El programa fue desarrollado en Fortran 77, debido a que por experiencia previa, en el grupo de investigación en donde se desarrolló esta tesis, es un lenguaje probado, y que la comunidad científica actual, entiende de forma general.

Cabe mencionar que las nuevas versiones de Fortran 77, permiten estructuras DO y la declaración explícita de las variables, esto hace al programa más amigable y una herramienta adecuada en aplicaciones en donde no se desea una interfase gráfica y la única preocupación es llevar a cabo operaciones matemáticas a gran velocidad y precisión.

A continuación se hace una breve descripción de las estructuras empleadas en este programa.

3.1. *Árbol Binario.*

Un árbol binario es una estructura de datos en donde se encuentran estos mismos, organizados por medio de un algoritmo sencillo.

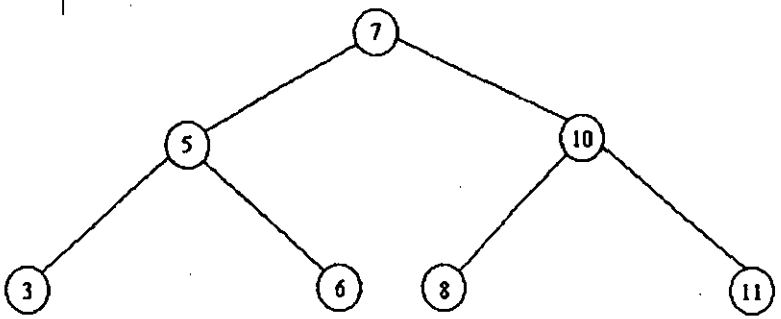
Una descripción bastante adecuada y completa de un árbol binario, es la descrita por Cormen T.H. [7].

"Un árbol binario es una estructura definida por un número finito de conjuntos de nodos que, o bien:

- no contiene nodos o,
- está compuesto de tres conjuntos de nodos: Un nodo raíz, un árbol binario llamado subárbol derecho y un árbol binario llamado subárbol izquierdo.

El árbol binario que no contiene nodos es llamado árbol vacío o árbol nulo, que denotaremos como NIL. Si el árbol izquierdo es no vacío, su raíz es llamada el hijo izquierdo de la raíz de todo el árbol binario, de la misma forma que la raíz no nula del subárbol derecho es el hijo derecho de la raíz del árbol binario. Si un subárbol es el árbol nulo NIL, decimos que el hijo está ausente o perdido."

Diagrama 1. "Árbol binario típico"



Como podemos ver de esta definición, un nodo es la unidad fundamental del árbol binario, y es en cada nodo en donde se ubican cada uno de los eventos que se calculan durante la Dinámica Molecular.

Todo nodo tiene por regla general un antecesor y varios sucesores. En el caso de la estructura del árbol binario, cada nodo tiene sólo dos sucesores, uno que contendrá información que ocurrirá después, y otro que ocurrirá a un tiempo más próximo.

El campo que contendrá el tiempo programado en que sucede cada evento, servirá en este caso para formar la estructura del árbol binario.

El arreglo de los números en el diagrama No. 1, no se realizó de forma aleatoria, sino fueron puestos en esa posición para cumplir con las condiciones de búsqueda de un árbol binario.

Para realizar una búsqueda sobre un árbol binario es necesario recordar que éste está formado, además del campo principal que contiene la información de interés, de tres campos más al menos que indican al hijo izquierdo, hijo derecho (sucesores) y al padre (antecesor). Si un hijo o un padre no existen para un nodo dado, el campo contendrá un valor de NIL. Sólo se permite a la raíz del árbol binario principal el tener su campo de padre con un valor de NIL.

El campo clave o principal, de un árbol binario será insertado en el árbol de tal forma que satisfaga la propiedad de búsqueda de los árboles binarios.

"Sea x un nodo en una búsqueda sobre un árbol binario. Si y es un nodo en el subárbol a la izquierda de x, entonces el campo clave de x será mayor al campo clave de y. Si y es un nodo en el subárbol a la derecha de x, entonces el campo clave de x será menor al campo clave de y." [7]

La organización de datos de esta forma nos permite ejecutar fácilmente operaciones en el manejo de árboles, tales como encontrar el máximo elemento (será el nodo más a la derecha del árbol), el mínimo elemento (será el nodo más a

la izquierda del árbol, y el cual será una de las subrutinas más empleadas en este trabajo), así como operaciones para establecer la relación sucesor-antecesor entre nodos, que será necesaria para realizar operaciones de inserción, borrado y actualización.

Para finalizar, la información que debe de contener cada nodo está de acuerdo a la estructura propuesta por Rapaport [1], que se muestra en el diagrama 2.

Diagrama 2. "Estructura nodal"

	Padre	
Hijo izquierdo	Tipo de evento	Hijo derecho
Lista circular de la partícula A (nodo previo)	Número de la partícula A (tipo A)	Lista circular de la partícula A (nodo posterior)
Lista circular de la partícula B (nodo previo)	Número de la partícula B (tipo B)	Lista circular de la partícula B (nodo posterior)

La partícula tipo A corresponde a la partícula a la que alrededor de ella se hace la búsqueda de partículas vecinas en el caso de una colisión. Siendo así B la partícula que corresponde a una partícula "vecina" de la partícula A.

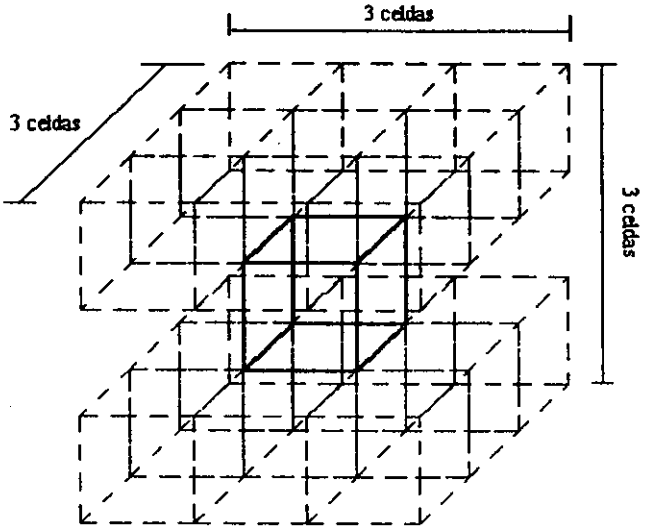
Debe quedar claro que todas las partículas pueden quedar incluidas en cualquiera de estas dos clasificaciones.

Actualmente, cada vez que sucede un evento, todos los nodos en donde se encuentran las partículas o partícula involucrada son borrados, esto se evitó en el presente trabajo, al borrar sólo los nodos de los eventos que ya no fueran físicamente posibles. La desventaja de este método es que es necesario hacer una búsqueda en cada nodo y verificar en qué campo se encuentra la partícula. Se espera que esta situación se compense con la rapidez ganada, al no tener que volver a calcular eventos que fueron generados anteriormente.

3.2. *Arreglos de subceldas.*

Imaginemos por el momento, un arreglo tridimensional de celdas.

Diagrama 3. "Subceldas".



Como se puede observar, cada subcelda tendrá 26 subceldas vecinas, de tal forma que podemos asignar a cada celda un número y elaborar una lista de subceldas vecinas una vez que se ha creado la estructura.

De esta forma, y con la instrumentación de un evento llamado cambio de celda, podemos saber en cualquier momento, cuál es la ubicación de cada una de las partículas.

Existen algoritmos que plantean la posibilidad de reducir el número de celdas vecinas de 26 a 13, pues de esta forma se eliminaría el "duplicar" la búsqueda de parejas.

En un principio esto se intentó, pero se encontró que, para temperaturas altas y densidades bajas, dado que las partículas se desplazan a una gran distancia antes de que se calcule una colisión, es posible tener una superposición de partículas.

Así pues, se adoptó hacer la búsqueda en las 26 celdas vecinas, pero para evitar el manejar el doble de eventos, se introdujo la siguiente condición.

Una vez que un evento ha sido calculado, dado que la inserción en el árbol implica una búsqueda, se aprovecha esta búsqueda para ver si existe ya insertado previamente este evento en el árbol.

Debido a que el manejo numérico en las computadoras no es exacto, y que cada vez que se modifica una posición, ésta varía ligeramente en su valor ($\pm 10^{-11}$), es necesario establecer una tolerancia en la búsqueda de estas cantidades.

Afortunadamente, la diferencia entre eventos distintos es mucho mayor que la diferencia introducida por el error en el manejo numérico, de tal forma que no

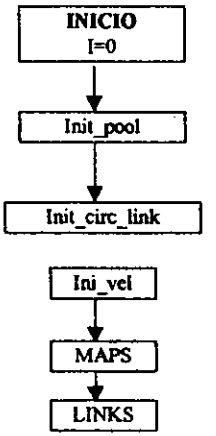
es necesario comparar el tipo de partículas involucradas, lo cual incrementaría el número de operaciones que se realizarían por ciclo de inserción.

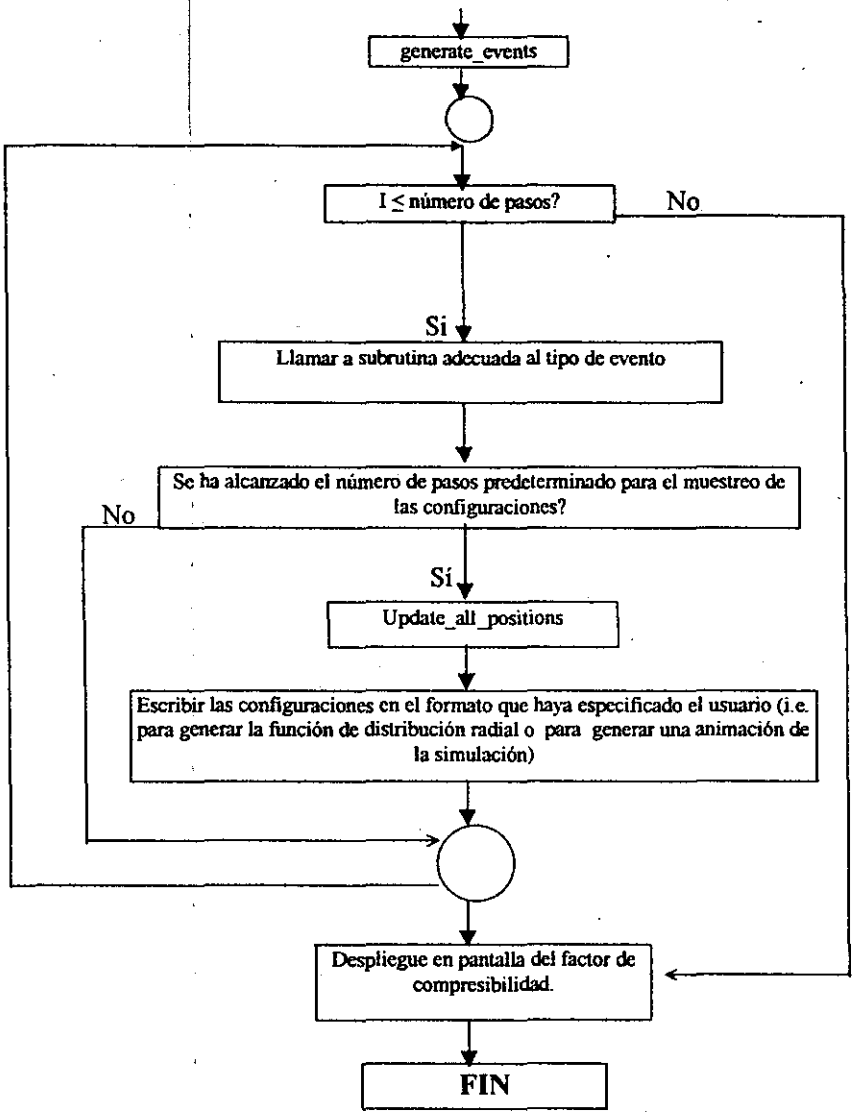
3.3. *Diagrama de bloques.*

Las subrutinas en las que se dividió el programa están organizadas de la siguiente forma.

Para lo que corresponde al programa principal, el diagrama de flujo resultó ser el siguiente:

Diagrama 4. "Diagrama de flujo del programa principal"




generate_events
I ≤ número de pasos?
No
Si
Llamar a subrutina adecuada al tipo de evento
Se ha alcanzado el número de pasos predeterminado para el muestreo de las configuraciones?
No
Sí
Update_all_positions
Escribir las configuraciones en el formato que haya especificado el usuario (i.e. para generar la función de distribución radial o para generar una animación de la simulación)
Despliegue en pantalla del factor de compresibilidad.
FIN

3.4. *Descripción de las subrutinas.*

Una vez que se han cubierto los rasgos generales de los métodos de optimización, a continuación se presenta una descripción de cada una de las subrutinas que se encuentran en el código, y si es el caso, cuáles fueron las modificaciones que se realizaron, para disminuir el número de operaciones por ciclo.

Para mayor claridad, no se entrará en la descripción detallada de todas las variables, a menos que se juzgue pertinente, y se hará la descripción de las rutinas de acuerdo a como van apareciendo en el código y no en el orden conforme se van ejecutando en el programa.

- *Principal.*

En esta parte del programa, se hace la asignación de los valores iniciales de la simulación:

Se inicializa el tiempo a cero (abs_time), así como otras variables que se requerirán para hacer el cálculo del factor de compresibilidad. (rijDvi, num_coll, vs).

Se hace la lectura de las siguientes variables:

- temperature: Temperatura.
- num_steps: Número de pasos.
- max_chain: Número de cadenas.
- max_site: Número de monómeros en cada cadena.
- ic_conf_file: Nombre del archivo que contiene la configuración inicial.
- bond_length: Longitud de enlace.
- sampling_rate: Intervalo de muestreo de las configuraciones del sistema.
- divisions: Número de subceldas que contendrá cada celda primaria.
- xmol_var: Decide si se desea generar la información necesaria para ver la simulación en un programa visualizador llamado xmol.
- gr_var : Decide si la ejecución del programa desea generar la información de la función de distribución radial.
- subcellsize: Tamaño de la subcelda.

Se elabora una lista de vecinos enlazados, la cual no será modificada durante la ejecución del programa, de esta forma se cuenta con una forma rápida de saber si dos partículas son vecinas y en la misma cadena, la forma de accesar a esta variable es simplemente con el número de partícula, y se definen dos campos más, uno que establece el número de partícula anterior, y otro el número de

partícula posterior. Para los extremos, se establece que son vecinos de la partícula cero.

Se llama a las *subrutinas init_pool, init_circ_link, ini_vel, MAPS, LINKS y generate_events*. Más tarde se hablará de cada una de ellas profundamente.

Se establece un ciclo donde se busca el evento más próximo a ocurrir por medio de la función *get_min_node*, se determina qué tipo de evento es y dependiendo de esto se llama a las subrutina *cell_cross*, en caso de cambio de celda, o *collision* en caso de ser alguna colisión. Se debe recordar que estamos llamando colisión a toda aquella modificación en la velocidad que sufran dos partículas al interactuar.

Si se ha alcanzado un número de paso tal que se desea escribir la configuración que existe en ese momento, se llama a la subrutina *update_all_positions*, la cual avanza a todas las partículas del sistema al mismo tiempo global, y se hace la escritura del archivo.

- Subrutina *erase_node(d)*

Esta subrutina se encarga de hacer el borrado del nodo *d*. Esta subrutina, se solicita una vez que se ha localizado un evento a ocurrir, es necesario eliminar del árbol

ese evento y todos los eventos que están asociados con las partículas involucradas.

Esto se debe de hacer por la siguiente razón:

Debido a la estructura del programa, cada evento es tratado de forma individual, por lo que es posible que una partícula tenga 2 ó más posibilidades de interactuar. Pero todos estos eventos que se calcularon y que tienen tiempos de sucesión mayores, ya no sucederán debido a que, después de la colisión que acaba de ocurrir, las velocidades de las partículas cambiaron, de tal forma que todos los eventos programados a suceder después de este evento, y en donde intervengan estas mismas partículas ya no serán válidos, pues fueron calculados a partir de velocidades "falsas".

Este procedimiento hace más eficientes los diferentes casos de borrado que son necesarios para mantener la estructura del árbol binario, y son ampliamente descritos en [1] y [3].

- Subrutina *erase_root_node(d)*

Dado que la forma de instrumentar el árbol binario fue un tanto distinta a la propuesta por Rapaport, fue necesario escribir una subrutina especial para el borrado del nodo raíz. Afortunadamente, no sólo no aumenta el número de

operaciones a desarrollar en cada ciclo, sino que, dependiendo de la estructura instantánea del árbol, puede reducir el número de operaciones que se realizan cada vez que se ejecuta esta rutina.

Básicamente, lo que hace este procedimiento es buscar el elemento máximo del subárbol izquierdo, y reemplazar con este nodo al nodo raíz. De esta forma se mantiene la estructura de búsqueda deseada.

- Subrutina *erase_nodes (part)*

Esta subrutina llama las veces necesarias al procedimiento erase_node(d), para eliminar todos lo eventos que tienen asociados a la partícula fijada por la variable *part*.

- Subrutina *init_pool*

Es deseado que, una vez que se ha borrado un nodo, éste pueda ser utilizado de nuevo, evitando así el uso ineficiente de memoria, pero, debido a que de antemano no se sabe el número de eventos que se van a calcular, se crea una lista en donde se guardarán los números asociados a cada nodo. Esta lista se maneja de la forma UEPS (últimas entradas primeras salidas).

Así pues, esta subrutina se encarga de iniciar esta lista, para que cada vez que se requiera un nodo, se pueda llamar sin problema alguno. Como se dijo anteriormente, dado que no se sabe el número de eventos que sucederán a un mismo tiempo, es necesario tener un mensaje de alerta cuando se quiera accesar a un nodo, y ya no haya nodos disponibles en esta lista.

- Subrutina *send_pool*

Cada vez que se hace el borrado de un nodo se envía a la lista en donde se encuentran los nodos disponibles. Este procedimiento, sólo actualiza el apuntador de la posición del siguiente nodo libre en la lista.

- Subrutina *get_pool*

Cuando se ha generado un evento y se desea incluir dentro del árbol binario, el primer paso es tener acceso a un nodo disponible, y es precisamente lo que esta subrutina realiza, tomando el primer nodo disponible y actualizando el apuntador a la siguiente posición.

- Subrutina *insert_node(d)*

Esta rutina se encarga de insertar en la posición adecuada al evento generado, guardado en el nodo *d*, de acuerdo a la regla de búsqueda del árbol binario.

- Subrutina *generate_events*

Esta rutina se posiciona sobre todas las partículas, y busca qué partículas están a su alrededor, en el caso de que se encuentre a una partícula en el vecindario, se hace el cálculo del tiempo en que sucederá la colisión.

El cálculo de este tiempo de colisión se hace en una función llamada *colltime*. La razón por la que se decidió separar esta función de este programa, es debido a que *generate_events*, sólo es ejecutado una sola vez durante la ejecución del programa, precisamente al principio de la simulación, posteriormente la función *colltime*, será llamada por otros procedimientos conforme la simulación progresa.

Esta subrutina, también crea por primera vez la estructura del árbol binario, ayudado principalmente del procedimiento *insert_node*.

- Subrutina *init_circ_link*

Esta rutina sólo inicializa un campo de una variable, que corresponde al nodo siguiente de una lista de nodos relacionados por la misma partícula.

Esta lista que se menciona aquí, es la forma de incluir en cada nodo, la información de qué eventos están relacionados con la misma partícula, para que cuando se desee hacer el borrado de ella, exista una lista de "eventos a borrar", y así se evite hacer una búsqueda por todo el árbol binario para determinar qué eventos se deben eliminar.

- Subrutina *update_circ_link(plink,node_num,type)*

Esta subrutina se encarga de realizar la actualización de la lista de eventos con partículas en común.

La variable *plink* corresponde al número de partícula de interés, *node_num* al nodo que va a ser removido de la lista y *type* si se trata de una partícula de tipo A o B.

- Subrutina *ini_vel(tempe)*

En esta subrutina se fijan las velocidades iniciales del sistema, de acuerdo a la temperatura del mismo. Se debe recordar que, para el caso de esferas duras, la temperatura no tiene ningún sentido físico, pero, dado que este programa es la base de un código mejorado, en donde la temperatura sí es de suma importancia, se decidió incluir esta subrutina.

- Subrutina *collision(sub_node)*

En esta subrutina se hace la modificación de las velocidades de las partículas involucradas en una colisión, así como la actualización de las coordenadas y del tiempo local de cada una de ellas.

Las fórmulas que se emplean se derivan de las ecuaciones de movimiento de Newton, y se hace la aproximación de que las partículas tienen la misma masa.

A diferencia de la función *colltime()*, esta subrutina, no requiere determinar si las partículas se alejan o se acercan, pues debido al producto punto entre los vectores posición y velocidad, los signos se ajustan de forma automática.

Estas ecuaciones son escritas en su forma vectorial y se muestran a continuación:

$$\Delta v_i = -\Delta v_j = \frac{-(v_{ij} \cdot r_{ij})}{\left|r_{ij}\right|^2}$$

donde:

Δv_i : Es el cambio en el vector velocidad de la partícula i

Δv_j : Es el cambio en el vector velocidad de la partícula j

v_{ij} : Es la diferencia entre los vectores velocidad v_i y v_j

r_{ij} : Es la diferencia entre los vectores posición r_i y r_j

La forma en que esta ecuación fue implementada, puede ser vista en el código del programa que se incluye en el Anexo I de esta tesis.

- Subrutina *MAPS*

Esta rutina genera el vecindario de cada una de las subceldas, que contendrán a las partículas durante la simulación.

- Subrutina *LINKS*

Asigna a cada una de las partículas a la subcelda que le corresponde debido a su posición.

- Subrutina *cell_crossing(p)*

Calcula el tiempo en que la partícula *p*, cambiará de subcelda. En este caso, para aumentar la rapidez de ejecución de programa, se escribió un subcódigo para cada tipo de cruce de celda, esto hace que se disminuya el número de operaciones necesarias para el cálculo del tiempo de cambio de celda.

- Subrutina *cell_cross(node)*

Actualiza la posición de la partícula en la celda de simulación, hace el borrado del nodo que se acaba de ejecutar y modifica la lista de partículas de la subcelda a la que ahora pertenece la partícula en cuestión.

- Subrutina *re_linking(partb, no)*

Mantiene actualizadas las uniones de los nodos con partículas en común, es llamado cada vez que sucede un evento, antes de que se realice el borrado de los

nodos involucrados.

- Subrutina *calculate_single_part(pa)*

Es el equivalente de *generate_events*, pero específico a calcular los posibles eventos de la partícula definida por la variable *pa*.

- Subrutina *erase_nodes_cc(part,no)*

Dado que en un cambio de celda no se desean borrar todos los eventos previos calculados de la partícula, pues la velocidad de la partícula es la misma, sólo se hace el borrado específico de este evento. Esto es con el fin de no repetir un cálculo que ya se ha hecho previamente.

- Subrutina *update_all_positions()*

Avanza a todas las partículas del sistema, una delta de tiempo tal que los tiempos locales de todas las partículas coincidan con el tiempo global de simulación, para así poder obtener una configuración que puede ser empleada después para su visualización, o bien para el cálculo de g(r).

- Función *gausran(v, m)*

Da como resultado un número aleatorio dentro de una distribución gaussiana con varianza *v* y media *m*.

- Función *colltime(parta, partb)*

Regresa el tiempo de colisión entre dos partículas, este cálculo requiere determinar qué tipo de situación se está llevando a cabo. Primero, definiremos algunas cantidades para facilitar la notación en las fórmulas siguientes.

$r_{ij} = r_i - r_j$ (Diferencia en el vector posición)

$v_{ij} = v_i - v_j$ (Diferencia en el vector velocidad)

$b_{ij} = r_{ij} \cdot v_{ij}$ (Producto punto para determinar las componentes involucradas en la colisión)

$C_{ij} = r_{ij}^2 - \sigma^2$ (Indica la distancia entre dos partículas al momento de la colisión)

$C^{\delta}_{ij} = r_{ij}^2 - (\sigma+\delta)^2$ (Indica la distancia a la que sucede una atracción súbita debido al enlace entre dos partículas).

I. Partículas acercándose. ($b_{ij}<0$)

si ($b_{ij}^2 - v_{ij}^2 C_{ij} > 0$) Sucede la colisión al tiempo:

$$t_{ij} = \frac{-b_{ij} - (b_{ij}^{\,2} - v_{ij}^{\,2} C_{ij})^{1/2}}{v_{ij}^{\,2}}$$

II. Partículas alejándose ($b_{ij}>0$)

si ($b_{ij}^{\,2}-v_{ij}^{\,2} C^{\delta}_{ij} > 0$) Sucede la colisión al tiempo:

$$t_{ij} = \frac{-b_{ij} + (b_{ij}^{\,2} - v_{ij}^{\,2} C^{\delta}_{ij})^{1/2}}{v_{ij}^{\,2}}$$

siempre y cuando el tiempo t_{ij} calculado sea positivo.

- Función *ICELL(IX,IY,IZ,MC)*

Función usada para determinar de forma rápida las celdas vecinas a la celda localizada en la posición *(IX, IY, IZ)*.

- Función *get_min_node()*

Función para determinar el evento más próximo.

- Función *get_v2()*

Función que suma la norma de las velocidades de todas las partículas en el sistema, para hacer el cálculo del factor de compresibilidad.

4. RESULTADOS

En este capítulo, se mostrarán los resultados obtenidos con el programa desarrollado. Primero se incluye la verificación de que el programa funcione adecuadamente.

Como primer caso, se pensó en un sistema de partículas discretas, pues para el caso de esferas duras, existe mucha información y datos publicados, lo cual facilitó la validación del código.

Para poder calcular la función de distribución radial, es necesario hacer un muestreo de diferentes configuraciones del sistema a intervalos regulares. Así pues, se eligieron 4 fracciones de empaquetamiento.

La fracción de empaquetamiento queda definido como la fracción de volumen que las partículas en su totalidad ocupan dentro del volumen de la celda.

De forma general, la fracción de empaquetamiento (η) tiene como fórmula:

$$\eta = \frac{N \cdot V_e}{V_c}$$

donde:

η : Fracción de empaquetamiento.

N : Número de partículas en el sistema.

V_e : Volumen de cada partícula.

V_c : Volumen de la celda de simulación.

En este caso las partículas tenían geometría esférica, y la celda es cúbica, por tanto, se puede definir η como:

$$\eta = \frac{4 \cdot \pi \cdot r_e^{\,3} \cdot N}{3 \cdot L^3}$$

donde:

η : Fracción de empaquetamiento.

N : Número de partículas en el sistema.

r_e : Radio de cada partícula.

L : Longitud de una de las aristas de la celda de simulación.

4.1. Validación de la función de distribución radial para esferas duras discretas.

Gráfico 1.

"Fracción de empaquetamiento : 0.19"

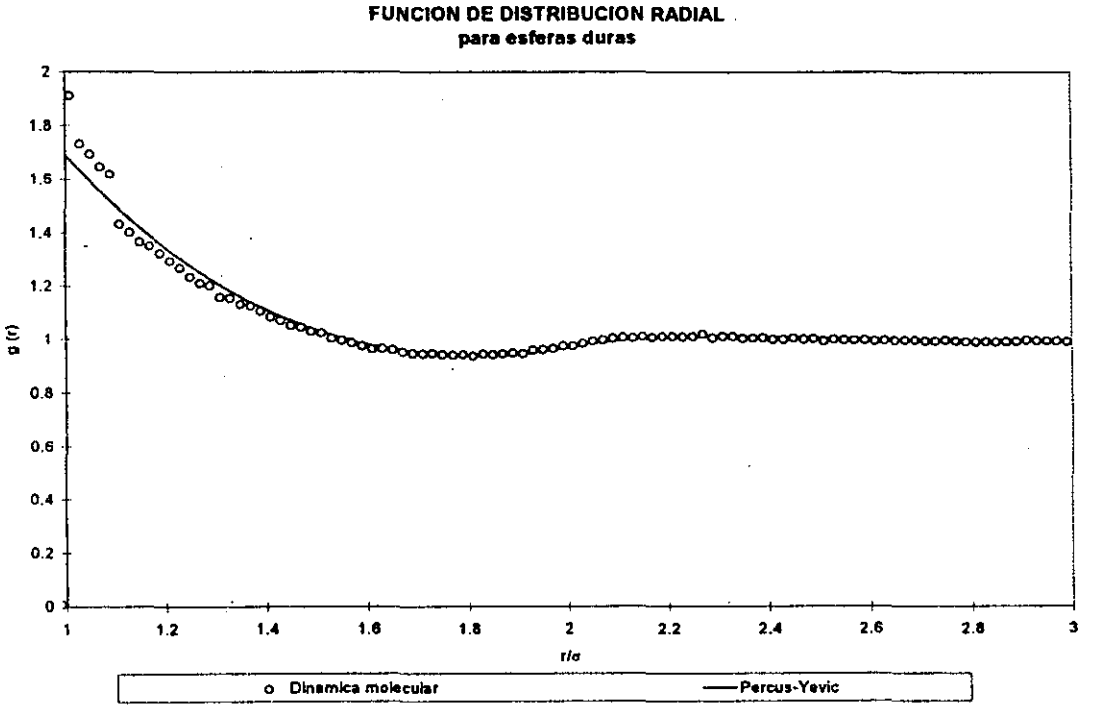


Gráfico 2.

"Fracción de empaquetamiento : 0.24"

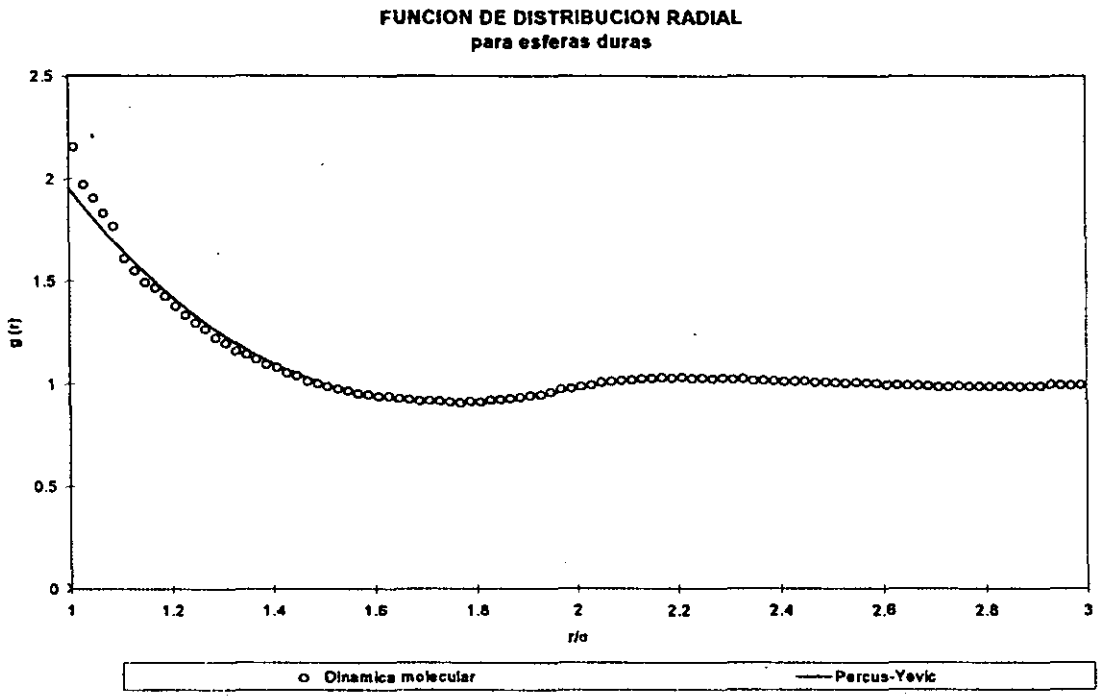


Gráfico 3

"Fracción de empaquetamiento : 0.29"

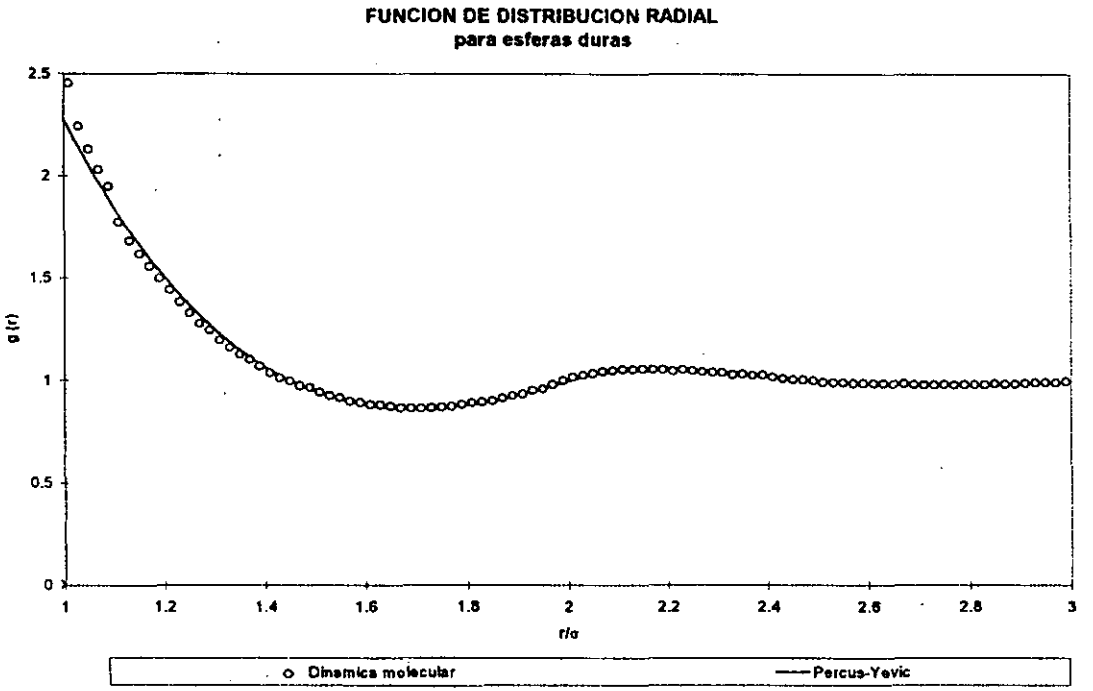
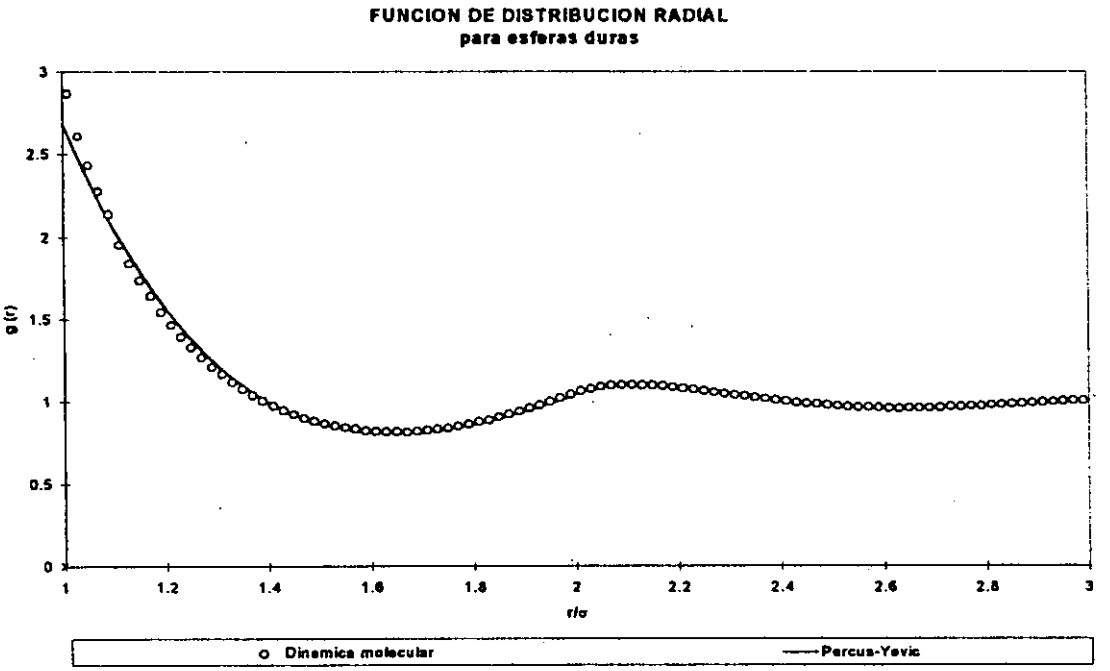


Gráfico 4.

"Fracción de empaquetamiento : 0.33"



Como se puede observar en el cálculo de g(r), conforme se incrementa la fracción de empaquetamiento (que es directamente proporcional a la densidad del sistema), se puede observar una mayor estructura en el fluido. La estructura del fluido queda definida por la posición y el tamaño de los picos que se presentan a distancias mayores a un diámetro atómico σ.

Una vez que se comprobó que la función de distribución radial para esferas duras discretas correspondía a valores cercanos a los predichos por la teoría y simulaciones hechas por medio de otros métodos, se decidió hacer la validación para las cadenas de esferas duras.

Como se puede observar, la mayor desviación de los resultados calculados por medio de la simulación molecular y los obtenidos por medio de la ecuación de Percus-Yevic corresponde a distancias menores a 1.1 diámetros atómicos. Esta desviación es debido a la forma en que se extraen las configuraciones durante la ejecución del programa.

Debido a que este programa está basado en un algoritmo evento-evento, y no a un incremento de tiempo arbitrario, en promedio, las partículas estarán unas más cerca de otras, pues dado que un evento corresponde al cálculo de colisiones, éstas suceden cuando la distancia entre ellas es igual a un diámetro atómico.

Es por ello que se puede observar que este programa sobreestima la distribución radial de partículas, pero, a medida que la densidad aumenta, este error se ve disminuido debido a que en realidad las partículas se encuentran más juntas en promedio.

Afortunadamente, a distancias mayores, el efecto del muestreo se cancela prácticamente, lo cual se observa fácilmente por la correspondencia entre los resultados de Dinámica Molecular y le ecuación de Percus-Yevic.

4.2. Validación de la función de distribución radial para cadenas de esferas duras.

Como primer ejemplo, se incluye la función de distribución radial para un octámero de esferas duras a diversas fracciones de empaquetamiento (η= 0.3390, 0.2988, 0.2011, 0.0977).

Estos resultados fueron comparados con los resultados publicados por Chiew Y.C [8]. Desafortunadamente, estos valores no pudieron ser comparados en el mismo gráfico por dos razones:

- La información de estos datos, fue publicada únicamente en forma de gráficos, de forma que no se quiso introducir un error de lectura.

- Por limitaciones del código desarrollado no es posible generar resultados para las fracciones exactas de empaquetamiento que se dan en la publicación. Esto es debido a que la fracción de empaquetamiento depende de un número discreto de partículas dentro de la celda de simulación, así pues, para poder llegar a una fracción de empaquetamiento tal como 0.1 por ejemplo, la longitud de la arista de la celda de simulación tendría que ser muy grande y por tanto un gran número de partículas. Aun así, se trató de acercar lo más posible a los resultados publicados.

A pesar de ello, los resultados de los datos previos se incluyen en una imagen extraída directamente del artículo. Para permitir al lector asegurarse de que, si bien las curvas no son semejantes completamente, sí se obtienen los rasgos generales. Como se puede observar por simple inspección, los datos no presentan una gran desviación entre sí.

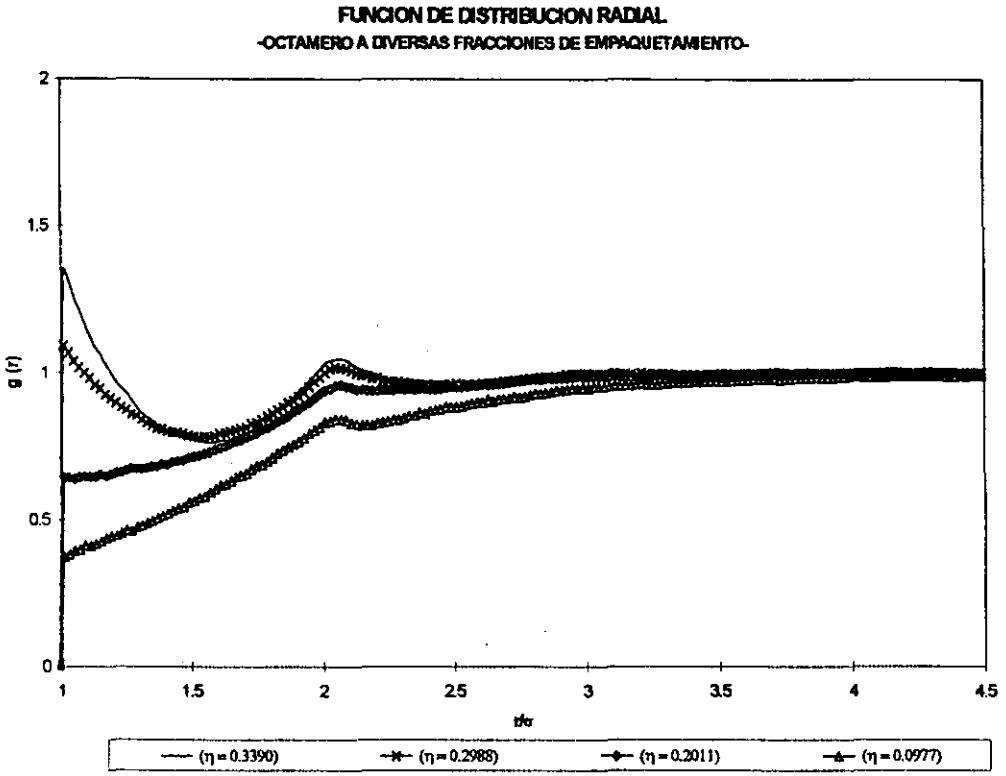
En el caso de las simulaciones de este trabajo, se agregó una curva más a una fracción de empaquetamiento mayor a las publicadas, sólo con el fin de saber si el código puede soportar altas densidades sin sufrir rompimientos de cadena, como se observó en versiones Beta del código aquí desarrollado.

La forma de saber si existen rompimientos de cadena, es al observar g(r) y no encontrar a la distancia $r/\sigma=2$ un pico tan pronunciado. Este pico se da principalmente, debido a que las esferas en una misma cadena están unidas por un enlace.

La razón por la que se hace hincapié en la inclusión de esta nueva curva, es debido a que no se debe de confundir con la curva inferior que sí está cerca de los valores publicados, y no ocasione confusión entre dos curvas en donde las fracciones de empaquetamiento sí son bastante diferentes.

Gráfico 5.

"Función de distribución radial para cadenas"

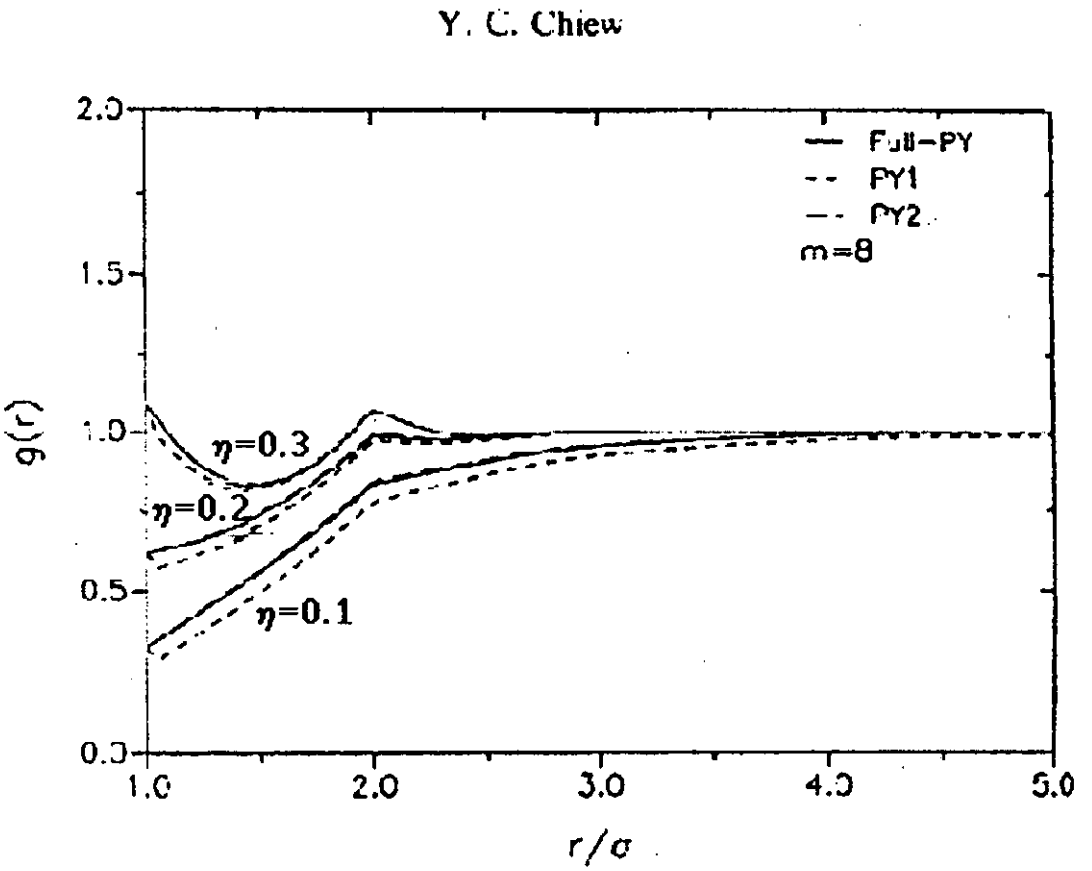


La razón por la que en el siguiente gráfico se muestran tres diferentes curvas es debido a que en ese artículo, se busca hacer aproximaciones a la ecuación de Percus-Yevic.

De esta forma, los resultados que se consideran "correctos", son los representados por medio de la línea continua, que está formada por la solución de la ecuación de Percus-Yevic sin ninguna aproximación adicional.

Gráfico 6.

"Función de distribución radial para cadenas (resultados publicados)"



Se debe recordar que el factor de compresibilidad es una desviación a la idelidad del sistema. En los gráficos mostrados anteriormente, podemos osbservar que, a medida que aumenta la densidad, el factor de compresibilidad se alejará de la unidad (valor que corresponde a gas ideal) y también sufrirá un incremento conforme aumenta la longitud de la cadena.

4.3. Validación del factor de compresibilidad para cadenas de esferas duras.

En la última etapa de validación del código, se desean comparar los resultados obtenidos para el factor de compresibilidad en cadenas duras reportado por Mark Denlinger y Carol K. Hall [9].

Se debe hacer mención que las curvas continuas que se muestran corresponden a una regresión estadística, que ajusta no sólo los valores obtenidos por estos autores, sino también de información para este sistema que ha sido generada por diversos trabajos previos.

Gráfico 7.

"Factor de compresibilidad para cadenas duras"

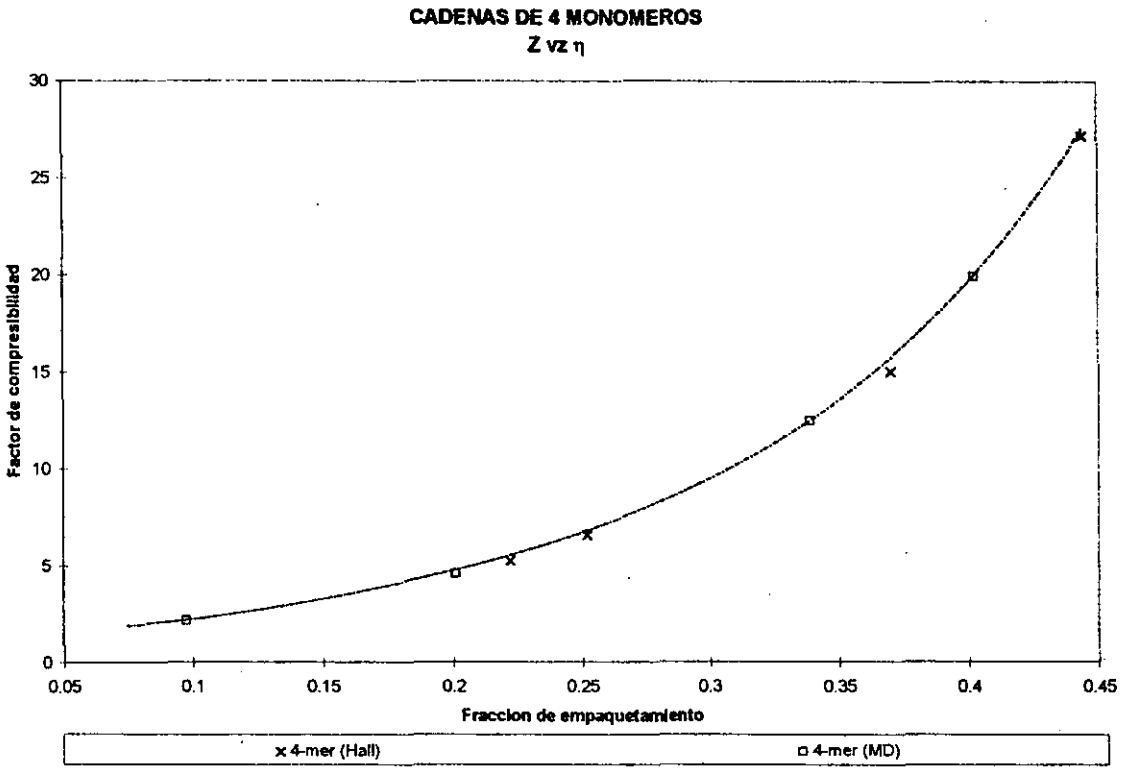


Gráfico 8.

"Factor de compresibilidad para cadenas duras"

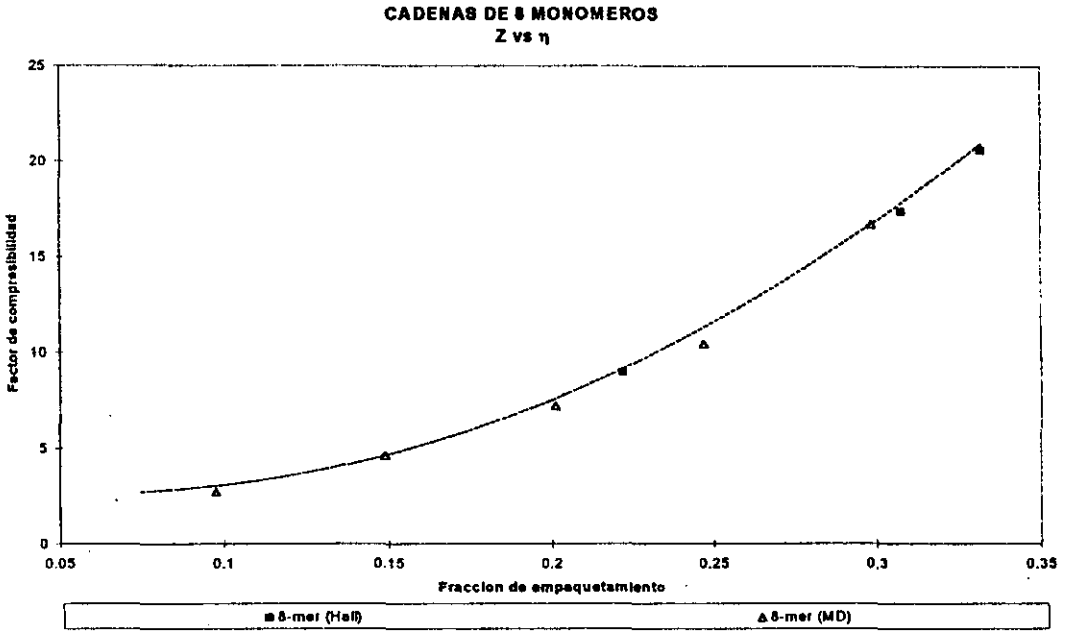


Gráfico 9.

"Factor de compresibilidad para cadenas duras"

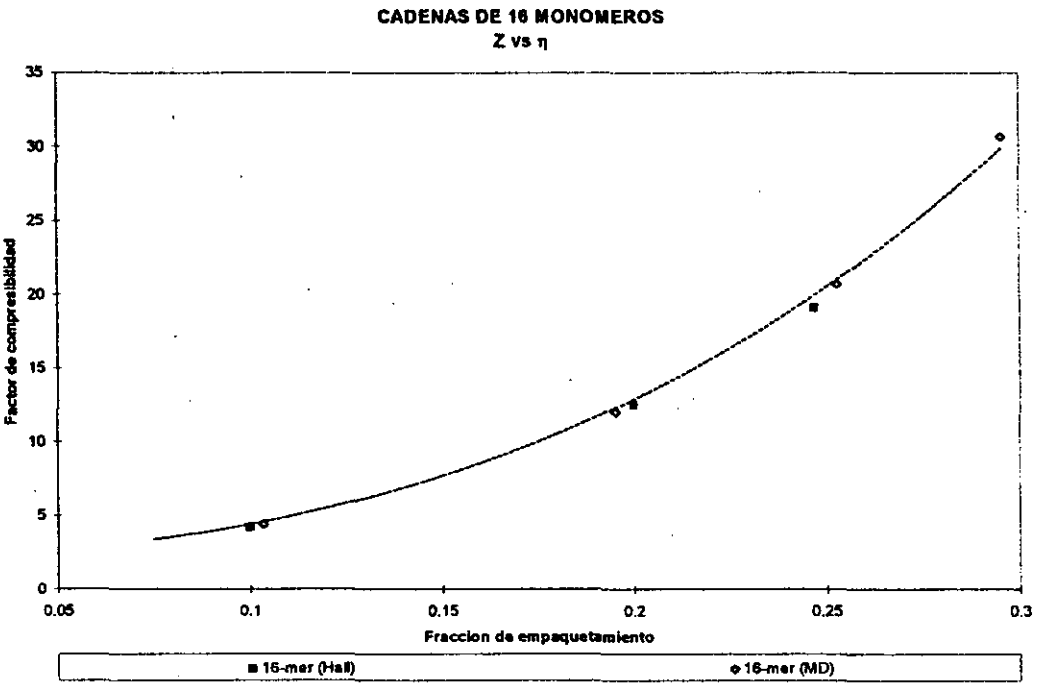
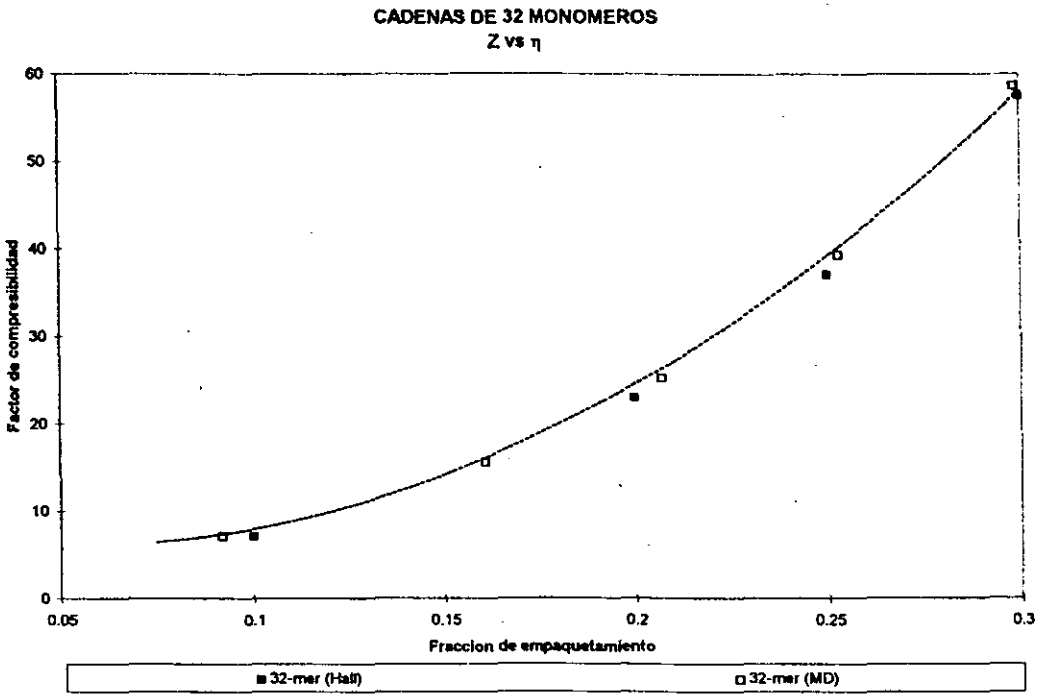


Gráfico 10.

"Factor de compresibilidad para cadenas duras"



La razón del por qué sólo se muestra la comparación entre los resultados obtenidos a partir del código desarrollado en esta tesis y los valores de Denlinger y Hall, es debido a que emplearon también un algoritmo de Dinámica Molecular para la obtención de ellos.

Como se puede observar, los resultados producidos por el código desarrollado, generan factores de compresibilidad que están acorde con resultados obtenidos previamente y calculados por similares y distintos métodos.

4.4. Comparación en el desempeño del código desarrollado con trabajos similares.

En lo que corresponde a la parte del desempeño del trabajo, en donde se muestran gráficos del número de colisiones y el tiempo de UCP (unidad central de proceso):

Todas las corridas del programa desarrollado en esta tesis se llevaron a cabo en un equipo DEC alpha 3000/166.

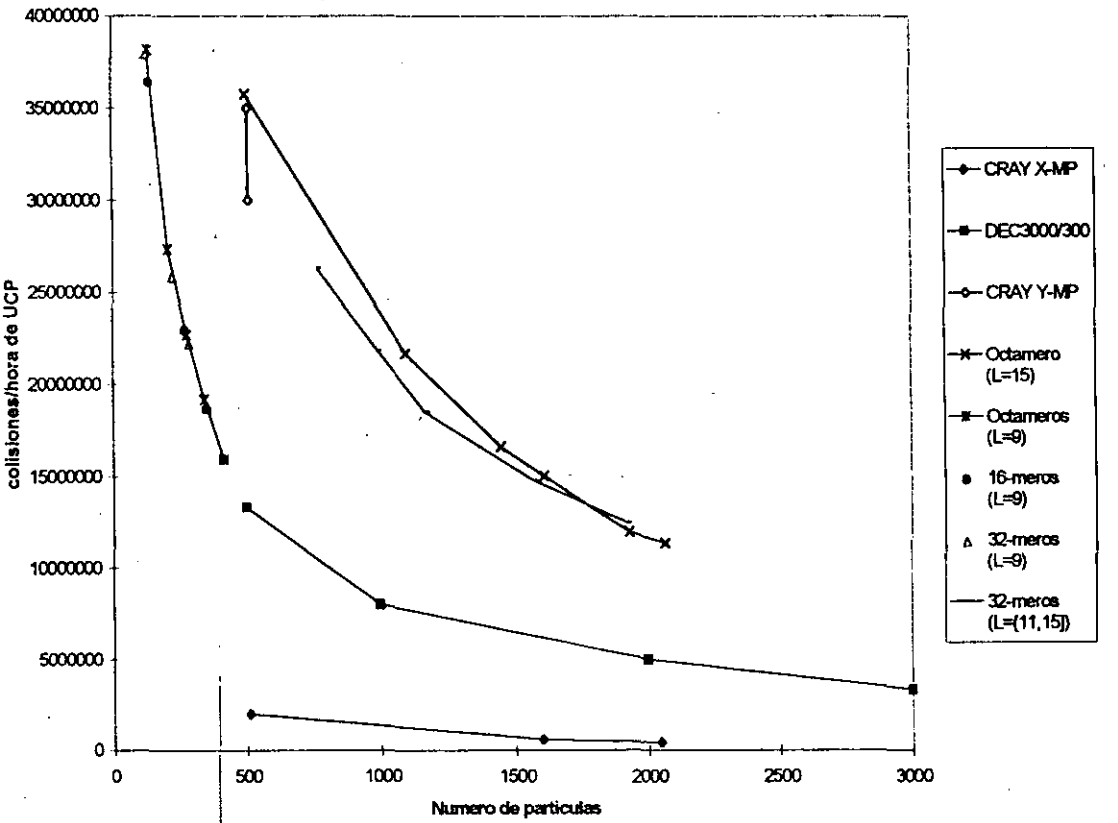
Es de hacer notar, que Hall. C.K. y colaboradores han sido las personas más activas en este ramo, de tal forma que en la gráfica de desempeño, se incluyen versiones de resultados anteriores en donde no se empleaban algoritmos de

optimización, así como los resultados de su última versión, en donde se instrumentaron los algoritmos antes descritos en esta tesis.

El incluir estos resultados nos permite tener una idea de la increíble mejora en la rapidez de cálculo, por el solo hecho de manejar árboles binarios, listas, falsos posicionamientos, etc.

Gráfico 11.

"Comparación en el desempeño de distintos códigos de Dinámica Molecular"



En el gráfico anterior se desearon introducir los siguientes aspectos que se encontraron al término de esta tesis.

La versión sin las rutinas de optimización que corrió en un equipo de supercómputo CRAY modelo X-MP, resulta del orden de 10 veces más lento comparado con el código que emplea rutinas de optimización y que fue corrido en un equipo de velocidad similar (CRAY Y-MP).

En el gráfico se incluye la longitud de la celda empleada para la simulación de los sistemas, de esta forma es posible tener una idea de las fracciones de empaquetamiento empleadas. Se puede ver en el gráfico que, a medida que la fracción de empaquetamiento aumenta, la velocidad de proceso disminuye dramáticamente; esto se debe a que, debido a la alta densidad del sistema, las rutinas de búsqueda de vecinos se hacen más lentas, pues se deben accesar listas más largas. Además, el número de colisiones que se gráfica, se refiere al número de colisiones efectivas.

Cada vez que se encuentran pares de partículas cercanas, se hace el cálculo de colisión entre ellas, de tal forma que, a mayor número de parejas posibles, mayor el número de colisiones calculadas; esto no sólo afecta el tiempo que se gasta en el cálculo directo del tiempo de colisión, sino que, al ser insertado este "posible evento", posteriormente se requerirá que las rutinas de borrado de nodos,

tengan duraciones más largas, debido a la mayor longitud de las listas asociadas a las partículas.

Como se puede observar, la longitud de cadena para misma fracción de empaquetamiento no afecta el desempeño del programa, pues se trataron cadenas de 8, 16 y 32 monómeros, y, como se puede observar, todos los resultados caen dentro de la misma línea de comportamiento.

Aunque no se cuenta con datos suficientes para comparar el programa con un equipo de supercómputo, al menos se conoce el rango de la velocidad de proceso del código optimado para el equipo de este tipo, y como es posible observar, el código que aquí se presenta es ligeramente más rápido, para el caso de 512 partículas.

5. CONCLUSIONES

Haciendo uso de algoritmos de optimización propuestos anteriormente por un gran número de investigadores, es posible instrumentar un código que genere resultados de forma más eficiente, en comparación con códigos similares que hacen uso de la Dinámica Molecular.

Se hace notar que, a pesar de que este código y el desarrollado por Hall y colaboradores tienen como base los mismos antecedentes y básicamente las mismas rutinas de optimización, la forma de hacer la instrumentación de los algoritmos, permite generar códigos con mejor o más pobre desempeño.

Aun con las velocidades alcanzadas, no es posible simular cadenas de las dimensiones que realmente suceden en la naturaleza, en las que cada molécula polimérica puede contener hasta millones de monómeros.

Aun así, este tipo de algoritmos, puede ser de utilidad para moléculas de tamaño pequeño. Es por esta razón, el interés de generar un código que incluyera todos los algoritmos de simulación presentes, e integrarlos en un programa de Dinámica Molecular.

El siguiente aspecto en el desarrollo del nuevo polímero, corresponde la inclusión de potenciales atractivos del tipo "pozo de potencial", para permitir la interacción entre moléculas de distinto tipo.

Las modificaciones que sufrirá este código serán las siguientes:

- Permitir distintos tamaños de partículas.
- Permitir por lo tanto distintas masas de partículas.
- Agregar un potencial atractivo para poder simular interacciones entre partículas de distinta naturaleza (i.e. hidrofílicas e hidrofóbicas).

La parte de mayor interés es la elección del modelo a utilizar para el potencial atractivo. El más simple es el pozo de potencial, pero, de ser necesario, es posible generar potenciales más complicados haciéndolos discretos, por medio de funciones más complicadas [10]. Aun así, el código generado en esta tesis puede ser modificado, y de hecho fue planeado para poder instrumentar estas características con mayor facilidad.

6. BIBLIOGRAFÍA.

[1] **Rapaport D.C.**,"The event scheduling problem in molecular dynamic simulation". Journal of Computational Physics. 1980. Vol. 34. p.p. 184-201.

[2] **Allen M.P, Tildesley D.J.** "Computer Simulation of liquids". Oxford science publications. 1987.

[3] **Smith S.W., Hall C.K.** et al. "Molecular Dynamics for Polymeric Fluids Using Discontinuous Potentials". Journal of Computational Physics. 1997. Vol. 134. p.p. 16-30.

[4] **McQuarrie. Donald A.** "Statistical Mechanics". Harper chemistry publishers. 1976. p.p. 1-3,282.

[5] **Smith S.W., Hall C.K.** et al. "Molecular Dynamics study of transport coefficients for hard-chain fluids". Journal of Chemical Physics. 1995. Vol. 102. p.p. 1057-1073.

[6] **Alder B.J. y Wainwright T.E.** "Studies in Molecular Dynamics. I. General Method". Journal of Chemical Physics. 1959. Vol. 31. p.p. 459-466.

[7] **Cormen T.H., Leiserson C.E. et.al.** "Introduction to Algorithms". The MIT press. 1996. p.p. 94,95,244-246.

[8] **Chiew Y.C.** "Percus-Yevic integral equation theory for athermal hard-sphere chains. II Average intermolecular correlation functions". Molecular Physics. Vol 73, No.2. p.p. 359-373.

[9] **Denlinger M.A., Hall Carol K.** "Molecular-dynamics simulation results for the pressure of hard-chain fluids". Molecular Physics. Vol. 71. No. 3. p.p. 541-559.

[10] **Chapela G.A. Martinez-Casas S.E.** "Molecular dynamics for discontinuous potentials I. General method and simulation of hard polyatomic molecules". Molecular physics. Vol 53. No. 1. p.p. 139-159.

ANEXO I. "Código desarrollado"

```
      program md
    implicit none

C   VAR
    integer*4       max_nodes
    parameter       (max_nodes=10000)
    integer*4       time_list(max_nodes,4)
    double precision   eventtime(max_nodes)
    integer*4       node_type (max_nodes,10)
    integer*4       i,j
    integer*4       counter
    double precision   new_event_time
    integer*4       afected_node, afected_particle
    integer*4       free_node, pointer
    integer*4       pool_list(max_nodes)
    integer*4       max_site, max_chain
    double precision   min_time
        integer*4       max_particle
    parameter       (ma4x_particle=10000)
    double precision   particle(max_particle,9)
    integer*4       particle2(max_particle,3)
    integer*4       tot_num_par
    double precision   packing, xbox, ybox, zbox
    double precision   temperature
    integer*4       num_steps
C   ------------------------------------------------------------------
    integer*4       chain(max_particle,2)
    integer*4       event_counter
    character*20      ic_conf_file
    INTEGER           N, M, NCELL, MAPSIZ
    PARAMETER         ( N = 10000, M = 15, NCELL = M * M * M )
    PARAMETER         ( MAPSIZ = 26 * NCELL )
    INTEGER           LIST(N), HEAD(NCELL), MAP(MAPSIZ)
    double precision   bond_length
    integer*4       get_min_node, root_var
    double precision   abs_time, rijDvi, get_v2
    double precision   vs
    integer*4       divisions
    integer*4       sampling_rate, num_coll
    integer*4       xmol_var, gr_var
    double precision   subcellsize

C   GLOBAL
    common /sorting_var/ eventtime
    common /list/ time_list
    common /tree_structure/ node_type
    common /new_events/ new_event_time, afected_node,
   >          afected_particle
    common /pool/ pool_list, free_node, pointer
    common /box_dim/ packing, xbox, ybox, zbox
    common /new_events2/  particle
    common /new_events3/ particle2
```

```
      common /min_times/ min_time
      common /gen_events/ tot_num_par
      common /options/ temperature,num_steps
      common /chain_structure/ chain
      common /global_events/ event_counter
      COMMON / BLOCK2 / LIST, HEAD, MAP
      common /particle_bond/bond_length
      common /absolute_time/ abs_time
      common /num/divisions
      common /root/ root_var
      common /ini_calc/ i
      common /pressure/ rijDvi, vs
      common /sub_cell_size/ subcellsize

C     BEGIN
      abs_time=0d0
      rijDvi=0
      num_coll=0
      vs=0
      open(unit=1, file='options',status='unknown')
      read(1,*) temperature
      read(1,*) num_steps
      read(1,*) max_chain
      read(1,*) max_site
      read(1,*) ic_conf_file
      read(1,*) bond_length
      read(1,*) sampling_rate
      read(1,*) divisions
      read(1,*) xmol_var
      read(1,*) gr_var
      read(1,*) subcellsize
      close(unit=1,status='keep')
      counter=0
      do i=1, max_chain
       do j=1, max_site
        counter=counter+1
        particle2(counter,3)=i
        if(j.lt.max_site)then
         if(j.eq.1)then
          chain(counter,1)=0
          chain(counter,2)=counter+1
         else
          chain(counter,1)=counter-1
          chain(counter,2)=counter+1
         endif
        else
         chain(counter,1)=counter-1
         chain(counter,2)=0
        endif
       enddo
      enddo
      open(unit=1, file=ic_conf_file,status='unknown')
      read(1,*) tot_num_par
      read(1,*) packing, xbox, ybox, zbox
      do i=1, tot_num_par
       read(1,*) particle(i,1),particle(i,1),particle(i,2)
```

```
>    ,particle(i,3)
     particle(i,1)=particle(i,1)*xbox
     particle(i,2)=particle(i,2)*ybox
     particle(i,3)=particle(i,3)*zbox
    enddo

    close(unit=1, status='keep')
    write(*,*) 'initializing pool and circular links'
    call init_pool()
    call init_circ_link()
    call ini_vel(temperature)
    write(*,*) 'creating mapping structure'
    call MAPS
    write(*,*) 'set the particles inside cells'
    call LINKS
    write(*,*) 'constructing the binary tree'
    i=0
    call generate_events()

    open(unit=9,file='ocfile.dat',status='unknown')
    open(unit=10,file='oc_xmol.dat',status='unknown')
    open(unit=11, file='pressure.dat',status='unknown')

    do i=1, num_steps
     afected_node=get_min_node()
     if(node_type(afected_node,6).eq.3)then
      call cell_cross(afected_node)
     else
      call collision(afected_node)
      num_coll=num_coll+1
     endif
     if(mod(i,sampling_rate).eq.0)then
      write(*,*)'step number: ',i,'time: ',abs_time,'root: ',
     :          root_var
      call update_all_positions()
      write(11,*)abs_time, rijDvi, rijDvi/abs_time
c_____________________________________________________________________
      if(gr_var.eq.1)then
      write(9,*)tot_num_par  !Archivo de configuración para g( r )
      write(9,*)
      do j=1, tot_num_par
       write(9,*)particle(j,1)/xbox,
     :           particle(j,2)/ybox,
     :           particle(j,3)/zbox
      enddo
      endif
C_____________________________________________________________________
      if(xmol_var.eq.1)then
      write(10,*)tot_num_par+8 !Archivo de configuración para xmol
      write(10,*)
      write(10,*)'O ', 0.5d0,-0.5d0, 0.5d0
      write(10,*)'O ',-0.5d0,-0.5d0, 0.5d0
      write(10,*)'O ',-0.5d0, 0.5d0, 0.5d0
      write(10,*)'O ', 0.5d0, 0.5d0, 0.5d0
      write(10,*)'O ', 0.5d0,-0.5d0,-0.5d0
      write(10,*)'O ',-0.5d0,-0.5d0,-0.5d0
```

```
      write(10,*)'O ',-0.5d0, 0.5d0,-0.5d0
      write(10,*)'O ', 0.5d0, 0.5d0,-0.5d0
      do j=1, tot_num_par
       write(10,*)'I ',particle(j,1)/xbox,
     :          particle(j,2)/ybox,
     :          particle(j,3)/zbox
      enddo
      endif
C_____________________________________________________________________
       endif
      enddo
      write(*,*) 'writing a result file'
      close(unit=9, status='keep')
      close(unit=10, status='keep')
      close(unit=11, status='keep')
      write(*,*) '=========================================='
      write(*,*) '| Some useful quantities'
      write(*,*) '|collisions       = ',num_coll
      write(*,*) '|rate (col/events) = ',real(num_coll)/num_steps
      write(*,*) '|<v2>              = ',get_v2()
      write(*,*) '|Nc               = ',max_chain
      write(*,*) '|n                = ',max_site
      write(*,*) '|Z                = ',max_site+
     :          (rijDvi/abs_time)/(get_v2()*max_chain)

      write(*,*) 'end of program'
      end



C     ----------------------------------------
C     ==========Procedures=====================
C     ----------------------------------------
C     Las siguientes subrutinas son para el manejo del árbol
C     ----------------------------------------
      subroutine erase_node(d)
      implicit none

C     VAR
      integer*4  max_nodes
      parameter  (max_nodes=10000).
      integer*4  node_type(max_nodes,10)
      integer*4  s, p ,d
      integer*4  root_var

C     GLOBAL
      common /tree_structure/node_type
      common /root/ root_var

C     BEGIN
      if (node_type(d,4).eq.0) then
       s=node_type(d,1)
      else
       if (node_type(d,1).eq.0) then
        s=node_type(d,4)
```

```
      else
       if (node_type(node_type(d,4),1).eq.0) then
        s=node_type(d,4)
       else
        s=node_type(node_type(d,4),1)
50       continue
        if (node_type(s,1).ne.0) then
         s=node_type(s,1)
         goto 50
        endif
        if (node_type(s,4).ne.0) then
         node_type(node_type(s,4),3)=node_type(s,3)
        endif
        node_type(node_type(s,3),1)=node_type(s,4)
        node_type(node_type(d,4),3)=s
        node_type(s,4)=node_type(d,4)
       endif
       node_type(node_type(d,1),3)=s
       node_type(s,1)=node_type(d,1)
      endif
     endif
     p=node_type(d,3)
     if (s.ne.0) node_type(s,3)=p
     if (p.ne.0) then
      if (node_type(p,1).eq.d) then
       node_type(p,1)=s
      else
       node_type(p,4)=s
      endif
     else
      root_var=s
     endif
     end
C    ------------------------------------------------
     subroutine erase_root_node(d)
     implicit none

C    VAR
     integer*4  max_nodes
     parameter  (max_nodes=10000)
     integer*4  node_type(max_nodes,10)
     integer*4  d, left_son, right_son
     integer*4  root_var, test_node

C    GLOBAL
     common /tree_structure/node_type
     common /root/ root_var

C    BEGIN
     test_node=node_type(d,1)
     left_son=test_node
     right_son=node_type(d,4)
     if(left_son.ne.0)then
150    continue
      if(node_type(test_node,4).ne.0)then
       test_node=node_type(test_node,4)
```

```
        goto 150
       else
        if(test_node.eq.node_type(d,1))then
         node_type(test_node,3)=0
         node_type(test_node,4)=node_type(d,4)
         node_type(right_son,3)=test_node
        else
         node_type(node_type(test_node,3),4)=node_type(test_node,1)
         if(node_type(test_node,1).ne.0)then
          node_type(node_type(test_node,1),3)=node_type(test_node,3)
         endif
         node_type(test_node,3)=0
         node_type(test_node,4)=node_type(d,4)
         node_type(test_node,1)=node_type(d,1)
         node_type(right_son,3)=test_node
         node_type(left_son,3)=test_node
        endif
       endif
       root_var=test_node
      else
       root_var=right_son
       node_type(right_son,3)=0
      endif
      end
C     ------------------------------------------------
      subroutine erase_nodes(part)
      implicit none

C     Var
      integer*4  part, partb
      integer*4  max_particle
      parameter  (max_particle=10000)
      integer*4  particle2(max_particle,3)
      integer*4  test_node, next_node
      integer*4  max_nodes
      parameter  (max_nodes=10000)
      integer*4  node_type(max_nodes,10)
      integer*4  root_var

C     GLOBAL
      common /tree_structure/node_type
      common /new_events3/  particle2
      common /root/ root_var

C     Begin
      test_node=particle2(part,1)
      if(test_node.ne.0)then
35      continue
        if(node_type(test_node,5).eq.part)then
          if(node_type(test_node,8).eq.0)then
            partb=node_type(test_node,7)
            if(partb.ne.0)then
              call re_linking(partb, test_node)
            endif
            if(root_var.eq.test_node)then
              call erase_root_node(test_node)
```

```
      else
       call erase_node(test_node)
      endif
      call send_pool(test_node)
     else
      next_node=node_type(test_node,8)
      partb=node_type(test_node,7)
      if(partb.ne.0)then
       call re_linking(partb, test_node)
      endif
      if(root_var.eq.test_node)then
       call erase_root_node(test_node)
      else
       call erase_node(test_node)
      endif
      call send_pool(test_node)
      test_node=next_node
      goto 35
     endif
    else
     if(node_type(test_node,7).eq.part)then
      if(node_type(test_node,9).eq.0)then
       partb=node_type(test_node,5)
       if(partb.ne.0)then
        call re_linking(partb, test_node)
       endif
       if(root_var.eq.test_node)then
        call erase_root_node(test_node)
       else
        call erase_node(test_node)
       endif
       call send_pool(test_node)
      else
       next_node=node_type(test_node,9)
       partb=node_type(test_node,5)
       if(partb.ne.0)then
        call re_linking(partb, test_node)
       endif
       if(root_var.eq.test_node)then
        call erase_root_node(test_node)
       else
        call erase_node(test_node)
       endif
       call send_pool(test_node)
       test_node=next_node
       goto 35
      endif
     endif
    endif
    particle2(part,1)=0
   endif
  end
C  ------------------------------------------------------
  subroutine init_pool()
  implicit none
```

```
C    Var
     integer*4  max_nodes
     parameter  (max_nodes=10000)
     integer*4  free_node, pointer
     integer*4  pool_list(max_nodes)
     integer*4  is

C    GLOBAL
     common /pool/ pool_list, free_node, pointer

C    BEGIN
     pointer=0
     do is=1, max_nodes
       call send_pool(is)
     enddo
     end
C    ------------------------------------------------
     subroutine send_pool(d)
     implicit none

C    VAR
     integer*4  max_nodes
     parameter  (max_nodes=10000)
     integer*4  node_type(max_nodes,10)
     integer*4  free_node, pointer
     integer*4  pool_list(max_nodes)
     double precision  eventtime(max_nodes)
     integer*4  d, is

C    GLOBAL
     common /pool/ pool_list, free_node, pointer
     common /tree_structure/node_type
     common /sorting_var/eventtime

C    BEGIN
     do is=1,10
       node_type(d,is)=0
     enddo
     pointer=pointer+1
     pool_list(pointer)=d
     eventtime(d)=0
     end
C    ------------------------------------------------
     subroutine get_pool()
     implicit none

C    VAR
     integer*4  max_nodes
     parameter  (max_nodes=10000)
     integer*4  free_node, pointer
     integer*4  pool_list(max_nodes)

C    GLOBAL
     common /pool/ pool_list, free_node, pointer

C    BEGIN
```

```
      free_node=pool_list(pointer)
      pool_list(pointer)=0
      pointer=pointer-1
      if (pointer.le.0) then
        write(*,*)'the error was caused because an empty pool'
      endif
      end
C     ------------------------------------------------------
      subroutine insert_node(d)
      implicit none

C     VAR
      integer*4          max_nodes
      parameter          (max_nodes=10000)
      integer*4          root_var
      integer*4          q ,d
      double precision   time
      integer*4          node_type(max_nodes,10)
      double precision   eventtime(max_nodes)
      integer*4          pool_list(max_nodes)
      integer*4          free_node, pointer
      integer*4          error_i
      double precision   td

C     GLOBAL
      common /root/ root_var
      common /pool/ pool_list, free_node, pointer
      common /tree_structure/node_type
      common /sorting_var/ eventtime
      common /avoid_double/ error_i

C     BEGIN
      error_i=0
      time=eventtime(d)
      q=root_var
70    continue
      if (time.gt.eventtime(q)) then
        td=eventtime(q)/time
        if(td.gt.0.99999999999999D0) goto 71
        if (node_type(q,4).eq.0) then
          node_type(d,3)=q
          node_type(q,4)=d
        else
          q=node_type(q,4)
          goto 70
        endif
      else
       if(time.le.eventtime(q))then
        if(time.ne.0)then
          td=time/eventtime(q)
          if(td.gt.0.99999999999999D0) goto 71
        endif
        if (node_type(q,1).eq.0) then
          node_type(d,3)=q
          node_type(q,1)=d
        else
```

```
      q=node_type(q,1)
      goto 70
     endif
    else
71    continue
     call send_pool(d)
     error_i=1
    endif
   endif
   end
C  ------------------------------------------
C  Este es la creación por primera vez del árbol binario
C  ------------------------------------------
   subroutine generate_events()
   implicit none

C  VAR
   INTEGER          N, M, NCELL, MAPSIZ
   PARAMETER        ( N = 10000, M = 15, NCELL = M * M * M )
   PARAMETER        ( MAPSIZ = 26 * NCELL )
   INTEGER          LIST(N), HEAD(NCELL), MAP(MAPSIZ)
   integer*4        max_nodes
   parameter        (max_nodes=10000)
   integer*4        is, js, ks, IMAP, center_cell
   integer*4        tot_num_par
   integer*4        max_particle
   parameter        (max_particle=10000)
   integer*4        free_node, pointer
   integer*4        pool_list(max_nodes)
   integer*4        particle2(max_particle,3)
   double precision   particle(max_particle,9)
   integer*4        coll_type, neighbour, partb
   double precision   colltime, coll_time, last_time
   integer*4        node_type(max_nodes,10)
   integer*4        event_counter
   double precision   eventtime(max_nodes)
   double precision   c_c_time
   double precision   subcellsize
   integer*4        root_var
   integer*4        error_i
   integer*4        divisions
   integer*4        i

C  GLOBAL
   common /gen_events/ tot_num_par
   common /new_events2/ particle
   common /new_events3/ particle2
   common /pool/ pool_list, free_node, pointer
   common /collision_Type/coll_type
   COMMON / BLOCK2 / LIST, HEAD, MAP
   common /tree_structure/node_type
   common /sorting_var/ eventtime
   common /global_events/ event_counter
   common /cell_crosss/ c_c_time
   common /root/ root_var
   common /avoid_double/ error_i
```

```
      common /num/divisions
      common /ini_calc/ i
      common /sub_cell_size/ subcellsize

C     BEGIN
      event_counter=0
      do is=1, tot_num_par
       center_cell = particle2(is,2)
       IMAP=(center_cell-1)*26
       last_time=0.5D50
       do js=1,27
        ks=1
600       continue
        if(js.lt.27)then
         if(ks.eq.1)then
          neighbour=MAP(IMAP+js)
          partb=HEAD(neighbour)
          ks=2
         else
          partb=LIST(partb)
         endif
        else
         if(ks.eq.1)then
          partb=HEAD(center_cell)
          ks=2
         else
          partb=LIST(partb)
         endif
         if(partb.eq.is)then
          goto 500
         endif
        endif

        if (partb.ne.0) then
         coll_time=colltime(is,partb)
         if(coll_time.lt.1.0D50)then
          call get_pool()
          event_counter=event_counter+1
          node_type(free_node,5)=is
          node_type(free_node,6)=coll_type
          node_type(free_node,7)=partb
          eventtime(free_node)=coll_time
          if(root_var.eq.0)then
           root_var=free_node
          else
           call insert_node(free_node)
          endif
          if(error_i.ne.1)then
           call update_circ_link(is,free_node,1)
           call update_circ_link(partb,free_node,2)
          endif
         endif
        endif

500     continue
        if(partb.ne.0)then
```

```
      if(LIST(partb).ne.0)then
        goto 600
      endif
    endif
  enddo
  call cell_crossing(is)
  call get_pool()
  event_counter=event_counter+1
  node_type(free_node,5)=is
  node_type(free_node,6)=3
  node_type(free_node,7)=0
  eventtime(free_node)=c_c_time
  if(root_var.eq.0)then
    root_var=free_node
  else
    call insert_node(free_node)
  endif
  if(error_i.ne.1)then
    call update_circ_link(is,free_node,1)
  endif
 enddo
end
C  ------------------------------------------------------
 subroutine init_circ_link()
 implicit none

C   var
 integer*4        tot_num_par
 integer*4        is
 integer*4        max_particle
 parameter        (max_particle=10000)
 integer*4        particle2(max_particle,3)

C   Global
 common /new_events3/ particle2
 common /gen_events/ tot_num_par

C   Begin
 do is=1,tot_num_par
   particle2(is,1)=0
 enddo
end
C  ------------------------------------------------------
 subroutine update_circ_link(plink,node_num,type)
 implicit none

C   Var
 integer*4          max_particle
 integer*4          max_nodes
 parameter          (max_nodes=10000)
 parameter          (max_particle=10000)
 integer*4          particle2(max_particle,3)
 integer*4          plink, node_num, test_node
 integer*4          node_type(max_nodes,10)
 integer*4          type
 double precision   eventtime(max_nodes)
```

```
C     Global
      common /new_events3/ particle2
      common /tree_structure/ node_type
      common /sorting_var/ eventtime

C     Begin
      if(particle2(plink,1).eq.0)then
       if(type.eq.1)then
        node_type(node_num,2)=0
       else
        node_type(node_num,10)=0
       endif
      else
       test_node=particle2(plink,1)
       if(node_type(test_node,5).eq.plink)then
        node_type(test_node,2)=node_num
        if(type.eq.1)then
         node_type(node_num,2)=0
         node_type(node_num,8)=test_node
        else
         node_type(node_num,9)=test_node
         node_type(node_num,10)=0
        endif
       else
        if(node_type(test_node,7).eq.plink)then
         node_type(test_node,10)=node_num
         if(type.eq.2)then
          node_type(node_num,9)=test_node
          node_type(node_num,10)=0
         else
          node_type(node_num,2)=0
          node_type(node_num,8)=test_node
         endif
        else
         write(*,*)'in this node the particle',plink,
     :          'is not present'
        endif
       endif
      endif
      particle2(plink,1)=node_num
      end
C     ------------------------------------------------------------------
      subroutine ini_vel(tempe)
      implicit none

      external drand48

C     VAR
      integer*4         is
      integer*4         tot_num_par
      double precision  drand48
      double precision  m, v
      double precision  tempe
      double precision  gausran
      integer*4         max_particle
```

```
      parameter         (max_particle=10000)
      double precision    particle(max_particle,9)

C     GLOBAL
      common /gen_events/ tot_num_par
      common /new_events2/  particle

C     BEGIN
      m=0.0d0
      v=sqrt(tempe)*0.001d0
      write(*,*)'Initializing trajectories'
      do is=1, tot_num_par
        particle(is,4)=gausran(v,m)
        particle(is,5)=gausran(v,m)
        particle(is,6)=gausran(v,m)
      enddo
      end
C     ---------------------------------------
      subroutine collision (sub_node)
      implicit none

C     VAR
      INTEGER           N, M, NCELL, MAPSIZ
      PARAMETER           (N=10000,M=15,NCELL=M*M*M)
      PARAMETER           (MAPSIZ=26*NCELL)
      INTEGER           LIST(N), HEAD(NCELL), MAP(MAPSIZ)
      integer*4         is
      integer*4         max_particle, partb
      integer*4         max_nodes
      parameter         (max_particle=10000)
      parameter         (max_nodes=10000)
      double precision    particle(max_particle,9)
      integer*4         particle2(max_particle,3)
      integer*4         node_type(max_nodes,10)
      double precision    dt1, dt2
      double precision    rx1, ry1, rz1, rx2, ry2, rz2
      double precision    packing, xbox, ybox, zbox
      double precision    abs_time
      double precision    eventtime(max_nodes)

      integer*4         sub_node
      double precision    rijx, rijy, rijz
      double precision    vijx, vijy, vijz, dia2
      double precision    bij, bond_length
      double precision    dvcollx, dvcolly, dvcollz
      integer*4         event_type
      integer*4         old_cell, new_cell

      integer*4         s, d, p
      integer*4         divisions
      double precision    divt, rijDvi, vs

C     GLOBAL
      common /new_events2/ particle
      common /new_events3/ particle2
      COMMON / BLOCK2 / LIST, HEAD, MAP
```

```
      common /box_dim/ packing, xbox, ybox, zbox
      common /tree_structure/ node_type
      common /absolute_time/ abs_time
      common /sorting_var/ eventtime
      common /particle_bond/bond_length
      common /num/divisions
      common /pressure/ rijDvi, vs

C     BEGIN
      divt=dble(divisions)
      is=node_type(sub_node,5)
      partb=node_type(sub_node,7)
      abs_time=eventtime(sub_node)
      event_type=node_type(sub_node,6)
      dt1=abs_time-particle(is,7)
      dt2=abs_time-particle(partb,7)

  ! update positions
      rx1=particle(is,1)+particle(is,4)*dt1
      ry1=particle(is,2)+particle(is,5)*dt1
      rz1=particle(is,3)+particle(is,6)*dt1
      rx2=particle(partb,1)+particle(partb,4)*dt2
      ry2=particle(partb,2)+particle(partb,5)*dt2
      rz2=particle(partb,3)+particle(partb,6)*dt2
  ! minimum image (remember that the coordinates are reduced)
  ! coordinates (from -0.5 to 0.5)
      rx1=(rx1/xbox-anint(rx1/xbox))*xbox
      ry1=(ry1/ybox-anint(ry1/ybox))*ybox
      rz1=(rz1/zbox-anint(rz1/zbox))*zbox
      rx2=(rx2/xbox-anint(rx2/xbox))*xbox
      ry2=(ry2/ybox-anint(ry2/ybox))*ybox
      rz2=(rz2/zbox-anint(rz2/zbox))*zbox
  ! Storing the new positions in the particle array
      particle(is,1)=rx1
      particle(is,2)=ry1
      particle(is,3)=rz1
      particle(partb,1)=rx2
      particle(partb,2)=ry2
      particle(partb,3)=rz2
  ! Set in new cells, and empty old cells
      old_cell=particle2(is,2)

      s=0
      d=HEAD(old_cell)
      p=LIST(d)
200   continue
      if(d.ne.is)then
       s=d
       d=p
       p=LIST(d)
       goto 200
      endif
      if(s.eq.0)then
       HEAD(old_cell)=p
      else
       LIST(s)=p
```

```
      endif
      LIST(d)=0
      particle2(is,2)=1+INT((rx1/xbox+0.5d0)*divt)
     :               +INT((ry1/ybox+0.5d0)*divt)*divisions
     :               +INT((rz1/zbox+0.5d0)*divt)*divisions**2
      new_cell=particle2(is,2)
      d=HEAD(new_cell)
      if(d.eq.0)then
       HEAD(new_cell)=is
      else
201    continue
       if(LIST(d).ne.0)then
        d=LIST(d)
        goto 201
       else
        LIST(d)=is
       endif
      endif

      old_cell=particle2(partb,2)
      s=0
      d=HEAD(old_cell)
      p=LIST(d)
300    continue
      if(d.ne.partb)then
       s=d
       d=p
       p=LIST(d)
       goto 300
      endif
      if(s.eq.0)then
       HEAD(old_cell)=p
      else
       LIST(s)=p
      endif
      LIST(d)=0
      particle2(partb,2)=1+INT((rx2/xbox+0.5d0)*divt)
     :                   +INT((ry2/ybox+0.5d0)*divt)*divisions
     :                   +INT((rz2/zbox+0.5d0)*divt)*divisions**2
      new_cell=particle2(partb,2)
      d=HEAD(new_cell)
      if(d.eq.0)then
       HEAD(new_cell)=partb
      else
301    continue
       if(LIST(d).ne.0)then
        d=LIST(d)
        goto 301
       else
        LIST(d)=partb
       endif
      endif

  ! Set new local times
      particle(is,7)=abs_time
      particle(partb,7)=abs_time
```

```
! calculate the change in velocity for a collision
      rijx=(rx1-rx2)/xbox
      rijy=(ry1-ry2)/ybox
      rijz=(rz1-rz2)/zbox

      rijx=(rijx-anint(rijx))*xbox
      rijy=(rijy-anint(rijy))*ybox
      rijz=(rijz-anint(rijz))*zbox

      vijx=particle(is,4)-particle(partb,4)
      vijy=particle(is,5)-particle(partb,5)
      vijz=particle(is,6)-particle(partb,6)

      bij=rijx*vijx+rijy*vijy+rijz*vijz

      if(event_type.eq.1)then
       dvcollx=-(bij*rijx)
       dvcolly=-(bij*rijy)
       dvcollz=-(bij*rijz)
      else
       dia2=1.0d0+bond_length
       dia2=1.0d0/dia2**2
       dvcollx=-(bij*rijx*dia2)
       dvcolly=-(bij*rijy*dia2)
       dvcollz=-(bij*rijz*dia2)
      endif

      particle(is,4)=particle(is,4)+dvcollx
      particle(partb,4)=particle(partb,4)-dvcollx
      particle(is,5)=particle(is,5)+dvcolly
      particle(partb,5)=particle(partb,5)-dvcolly
      particle(is,6)=particle(is,6)+dvcollz
      particle(partb,6)=particle(partb,6)-dvcollz

C-------------pressure calculation-----------------

      rijDvi=rijDvi+(rijx*dvcollx+rijy*dvcolly+rijz*dvcollz)
      vs=vs+particle(is,4)**2+particle(is,5)**2+particle(is,6)**2

      call erase_nodes(is)
      call erase_nodes(partb)
      call calculate_single_part(is)
      call calculate_single_part(partb)

      end
C     ------------------------------------------
C     Las subrutinas MPAS & LINKS son de un archivo de utilerias
C     de Allen (referencia incluida en bibliografia) y la
C     función ICELL, esta tomada de archivos del Profesor
C     Juan dePablo de la Universidad de Wisconsin-Madison
C     ------------------------------------------
      SUBROUTINE MAPS
      implicit none

C     Var
      INTEGER          N, M, NCELL, MAPSIZ
```

```
      PARAMETER         (N=10000,M=15,NCELL=M*M*M)
      PARAMETER         (MAPSIZ = 26 * NCELL )
      INTEGER           HEAD(NCELL), LIST(N), MAP(MAPSIZ)
      INTEGER           IX, IY, IZ, IMAP
      INTEGER           ICELL
      double precision  packing, xbox, ybox, zbox
      integer*4         divisions

C     Global
      COMMON / BLOCK2 / LIST, HEAD, MAP
      common /box_dim/ packing, xbox, ybox, zbox
      common /num/divisions

C     Begin
      DO IZ = 1,divisions
        DO IY = 1,divisions
          DO IX = 1,divisions
            IMAP = ( ICELL (IX,IY,IZ,divisions) - 1 ) * 26
            MAP(IMAP+1 ) = ICELL(IX+1,IY  ,IZ  ,divisions)
            MAP(IMAP+2 ) = ICELL(IX+1,IY+1,IZ  ,divisions)
            MAP(IMAP+3 ) = ICELL(IX  ,IY+1,IZ  ,divisions)
            MAP(IMAP+4 ) = ICELL(IX-1,IY+1,IZ  ,divisions)
            MAP(IMAP+5 ) = ICELL(IX+1,IY  ,IZ-1,divisions)
            MAP(IMAP+6 ) = ICELL(IX+1,IY+1,IZ-1,divisions)
            MAP(IMAP+7 ) = ICELL(IX  ,IY+1,IZ-1,divisions)
            MAP(IMAP+8 ) = ICELL(IX-1,IY+1,IZ-1,divisions)
            MAP(IMAP+9 ) = ICELL(IX+1,IY  ,IZ+1,divisions)
            MAP(IMAP+10) = ICELL(IX+1,IY+1,IZ+1,divisions)
            MAP(IMAP+11) = ICELL(IX  ,IY+1,IZ+1,divisions)
            MAP(IMAP+12) = ICELL(IX-1,IY+1,IZ+1,divisions)
            MAP(IMAP+13) = ICELL(IX  ,IY  ,IZ+1,divisions)
            MAP(IMAP+14) = ICELL(IX  ,IY  ,IZ-1,divisions)
            MAP(IMAP+15) = ICELL(IX-1,IY  ,IZ-1,divisions)
            MAP(IMAP+16) = ICELL(IX-1,IY  ,IZ  ,divisions)
            MAP(IMAP+17) = ICELL(IX-1,IY  ,IZ+1,divisions)
            MAP(IMAP+18) = ICELL(IX+1,IY-1,IZ-1,divisions)
            MAP(IMAP+19) = ICELL(IX+1,IY-1,IZ  ,divisions)
            MAP(IMAP+20) = ICELL(IX+1,IY-1,IZ+1,divisions)
            MAP(IMAP+21) = ICELL(IX  ,IY-1,IZ-1,divisions)
            MAP(IMAP+22) = ICELL(IX  ,IY-1,IZ  ,divisions)
            MAP(IMAP+23) = ICELL(IX  ,IY-1,IZ+1,divisions)
            MAP(IMAP+24) = ICELL(IX-1,IY-1,IZ-1,divisions)
            MAP(IMAP+25) = ICELL(IX-1,IY-1,IZ  ,divisions)
            MAP(IMAP+26) = ICELL(IX-1,IY-1,IZ+1,divisions)
          ENDDO
        ENDDO
      ENDDO
      write(*,*)'Mapping done'
      END
C     ------------------------------------------------------
      SUBROUTINE LINKS
      implicit none

C     VAR
      INTEGER           N, M, NCELL, MAPSIZ
      PARAMETER         (N=10000,M=15,NCELL=M*M*M)
```

```
      PARAMETER         ( MAPSIZ = 26 * NCELL )
      INTEGER           HEAD(NCELL), LIST(N), MAP(MAPSIZ)
      double precision  CELLI
      INTEGER           ICELL, I
      integer*4         tot_num_par
      integer*4         max_particle
      parameter         (max_particle=10000)
      double precision  particle(max_particle,9)
      integer*4         particle2(max_particle,3)
      double precision  packing, xbox, ybox, zbox
      integer*4         divisions, d

C     GLOBAL
      COMMON / BLOCK2 / LIST, HEAD, MAP
      common /gen_events/ tot_num_par
      common /new_events2/ particle
      common /new_events3/ particle2
      common /box_dim/ packing, xbox, ybox, zbox
      common /num/divisions

C     BEGIN
      DO ICELL = 1, NCELL
        HEAD(ICELL) = 0
      ENDDO
      CELLI = dble(divisions)
C      ** SORT ALL ATOMS **
      DO I = 1, tot_num_par
        ICELL=1+INT((particle(I,1)/xbox+0.5d0)*CELLI)
     :       +INT((particle(I,2)/ybox+0.5d0)*CELLI)*divisions
     :       +INT((particle(I,3)/zbox+0.5d0)*CELLI)*divisions**2
        if(HEAD(ICELL).ne.0)then
         d=HEAD(ICELL)
110       continue
         if(LIST(d).eq.0)then
          LIST(d) = I
         else
          d=LIST(d)
          goto 110
         endif
        else
         HEAD(ICELL) = I
        endif
        particle2(I,2)=ICELL
      ENDDO
      END
C     ------------------------------------------------------------
      subroutine cell_crossing(p)
      implicit none

C     Var
      integer*4         p
      integer*4         max_particle
      parameter         (max_particle=10000)
      double precision  wall, c_c_time, dist
      double precision  particle(max_particle,9)
      double precision  timex, timey, timez
```

```
      double precision   subcellsize

C     Global
      common /new_events2/ particle
      common /cell_crosss/ c_c_time
      common /sub_cell_size/ subcellsize

C     Begin
C     ------------X coordinate----------------------
      if(particle(p,4).lt.0)then
       if(particle(p,1).gt.0)then
        wall=int(particle(p,1)/subcellsize)*subcellsize
       else
        wall=int(particle(p,1)/subcellsize-1.0d0)*subcellsize
       endif
      else
       if(particle(p,1).gt.0)then
        wall=int(particle(p,1)/subcellsize+1.0d0)*subcellsize
       else
        wall=int(particle(p,1)/subcellsize)*subcellsize
       endif
      endif
      dist=1.0d-10+abs(wall-particle(p,1))
      timex=dist/abs(particle(p,4))
C     ------------Y coordinate----------------------
      if(particle(p,5).lt.0)then
       if(particle(p,2).gt.0)then
        wall=int(particle(p,2)/subcellsize)*subcellsize
       else
        wall=int(particle(p,2)/subcellsize-1.0d0)*subcellsize
       endif
      else
       if(particle(p,2).gt.0)then
        wall=int(particle(p,2)/subcellsize+1.0d0)*subcellsize
       else
        wall=int(particle(p,2)/subcellsize)*subcellsize
       endif
      endif
      dist=1.0d-10+abs(wall-particle(p,2))
      timey=dist/abs(particle(p,5))

      if(timex.lt.timey)then
       c_c_time=timex
      else
       c_c_time=timey
      endif
C     ------------Z coordinate----------------------
      if(particle(p,6).lt.0)then
       if(particle(p,3).gt.0)then
        wall=int(particle(p,3)/subcellsize)*subcellsize
       else
        wall=int(particle(p,3)/subcellsize-1.0d0)*subcellsize
       endif
      else
       if(particle(p,3).gt.0)then
        wall=int(particle(p,3)/subcellsize+1.0d0)*subcellsize
```

```
      else
        wall=int(particle(p,3)/subcellsize)*subcellsize
      endif
    endif
    dist=1.0d-10+abs(wall-particle(p,3))
    timez=dist/abs(particle(p,6))
    if(timez.lt.c_c_time)then
     c_c_time=timez
    endif
    c_c_time=c_c_time+particle(p,7)
    end
C   ------------------------------------------------
    subroutine cell_cross(node)
    implicit none

C   Var
    INTEGER          N, M, NCELL, MAPSIZ
    PARAMETER          (N=10000,M=15,NCELL=M*M*M)
    PARAMETER          ( MAPSIZ = 26 * NCELL )
    INTEGER          HEAD(NCELL), LIST(N), MAP(MAPSIZ)
    integer*4        node
    integer*4        max_particle
    integer*4        max_nodes
    parameter        (max_particle=10000)
    parameter        (max_nodes=10000)
    integer*4        particle2(max_particle,3)
    integer*4        node_type(max_nodes,10)
    double precision   particle(max_particle,9)
    double precision   eventtime(max_nodes)
    integer*4        root_var
    integer*4        pool_list(max_nodes)
    integer*4        free_node, pointer

    double precision   rx, ry, rz
    integer*4        part
    integer*4        old_cell, new_cell
    double precision   abs_time
    double precision   tdiff
    double precision   packing, xbox, ybox, zbox

    integer*4        s, d, p
    integer*4        divisions
    double precision   divt

C   Global
    common /new_events2/ particle
    common /new_events3/ particle2
    common /tree_structure/ node_type
    common /sorting_var/ eventtime
    common /absolute_time/ abs_time
    common /BLOCK2/ LIST, HEAD, MAP
    common /box_dim/ packing, xbox, ybox, zbox
    common /root/ root_var
    common /pool/ pool_list, free_node, pointer
    common /num/divisions
```

```
C     Begin
      divt=dble(divisions)
      part=node_type(node,5)
      tdiff=eventtime(node)-particle(part,7)
      abs_time=eventtime(node)
      particle(part,7)=eventtime(node)
      rx=particle(part,1)+particle(part,4)*tdiff
      ry=particle(part,2)+particle(part,5)*tdiff
      rz=particle(part,3)+particle(part,6)*tdiff
      rx=(rx/xbox-anint(rx/xbox))*xbox
      ry=(ry/ybox-anint(ry/ybox))*ybox
      rz=(rz/xbox-anint(rz/zbox))*zbox
      particle(part,1)=rx
      particle(part,2)=ry
      particle(part,3)=rz
      old_cell=particle2(part,2)

      s=0
      d=HEAD(old_cell)
      p=LIST(d)
400    continue
      if(d.ne.part)then
       s=d
       d=p
       p=LIST(d)
       goto 400
      endif
      if(s.eq.0)then
       HEAD(old_cell)=p
      else
       LIST(s)=p
      endif
      LIST(d)=0
      particle2(part,2)=1+INT((rx/xbox+0.5d0)*divt)
     :                 +INT((ry/ybox+0.5d0)*divt)*divisions
     :                 +INT((rz/zbox+0.5d0)*divt)*divisions**2
      new_cell=particle2(part,2)
      d=HEAD(new_cell)
      if(d.eq.0)then
       HEAD(new_cell)=part
      else
401    continue
       if(LIST(d).ne.0)then
        d=LIST(d)
        goto 401
       else
        LIST(d)=part
       endif
      endif

      call erase_nodes_cc(part,node)
      call calculate_single_part(part)
      end
C     ------------------------------------------------------
      subroutine re_linking(partb, no)
      implicit none
```

```
C    VAR
     integer*4    max_nodes
     integer*4    max_particle
     parameter    (max_particle=10000)
     parameter    (max_nodes=10000)
     integer*4    node_type(max_nodes,10)
     integer*4    particle2(max_particle,3)
     double precision  eventtime(max_nodes)
     integer*4    no, partb
     integer*4    ol_ne_li
     integer*4    ol_la_li
     integer*4    test_node
     integer*4    chk_lnk, chk_typ

C    Global
     common /tree_structure/ node_type
     common /new_events3/ particle2
     common /sorting_var/ eventtime

C    Begin
     test_node=particle2(partb,1)
     chk_lnk=0
25    continue
     if(test_node.ne.0)then
      if(test_node.eq.no)then
       if(node_type(test_node,5).eq.partb)then
        ol_ne_li=node_type(test_node,8)
        ol_la_li=node_type(test_node,2)
       else
        if(node_type(test_node,7).eq.partb)then
         ol_ne_li=node_type(test_node,9)
         ol_la_li=node_type(test_node,10)
        else
         write(*,*)'parta: ',node_type(test_node,5),
     :             'partb: ',node_type(test_node,7),
     :             'should not be here in part: ',partb,
     :             'link'
        endif
       endif

       if(chk_lnk.eq.0)then
        particle2(partb,1)=ol_ne_li
       else
        if(chk_typ.eq.1)then
         node_type(chk_lnk,8)=ol_ne_li
        else
         if(chk_typ.eq.2)then
          node_type(chk_lnk,9)=ol_ne_li
         else
          write(*,*)'node corrections doesnt work'
         endif
        endif
       endif

       if(ol_ne_li.ne.0)then
```

```
      if(node_type(ol_ne_li,5).eq.partb)then
       node_type(ol_ne_li,2)=ol_la_li
      else
       if(node_type(ol_ne_li,7).eq.partb)then
        node_type(ol_ne_li,10)=ol_la_li
       else
        write(*,*)'Error in next link, part: ',
     :        node_type(ol_ne_li,5),'or part',
     :        node_type(ol_ne_li,7),'must be part: ',partb,
     :        'time :',eventtime(ol_ne_li)
        write(*,*)'last error relinking node: ',no
       endif
      endif
     endif
    else
     if(node_type(test_node,5).eq.partb)then
      ol_ne_li=node_type(test_node,8)
      chk_typ=1
     else
      ol_ne_li=node_type(test_node,9)
      chk_typ=2
      if(node_type(test_node,7).ne.partb)then
       write(*,*) 'sending to a unknown node'
      endif
     endif
     chk_lnk=test_node
     test_node=ol_ne_li
     goto 25
    endif
   else
    write(*,*)'hey, end reached, without finding partb'
   endif
   end
C  ------------------------------------------------
   subroutine calculate_single_part(pa)
   implicit none

C   VAR
   INTEGER        N, M, NCELL, MAPSIZ
   PARAMETER      ( N = 10000, M = 15, NCELL = M * M * M )
   PARAMETER      ( MAPSIZ = 26 * NCELL )
   INTEGER        LIST(N), HEAD(NCELL), MAP(MAPSIZ)
   integer*4      max_nodes
   parameter      (max_nodes=10000)
   integer*4      js, ks, IMAP, center_cell
   integer*4      tot_num_par
   integer*4      max_particle
   parameter      (max_particle=10000)
   integer*4      free_node, pointer
   integer*4      pool_list(max_nodes)
   integer*4      particle2(max_particle,3)
   integer*4      coll_type, neighbour, partb
   double precision colltime, coll_time, last_time
   integer*4      node_type(max_nodes,10)
   integer*4      event_counter
   double precision eventtime(max_nodes)
```

```
      double precision c_c_time
      double precision abs_time, td
      double precision subcellsize
      integer*4        root_var
      integer*4        pa
      integer*4        error_i

C     GLOBAL
      common /gen_events/ tot_num_par
      common /new_events3/ particle2
      common /pool/ pool_list, free_node, pointer
      common /collision_Type/coll_type
      COMMON / BLOCK2 / LIST, HEAD, MAP
      common /tree_structure/node_type
      common /sorting_var/ eventtime
      common /global_events/ event_counter
      common /cell_crosss/ c_c_time
      common /root/ root_var
      common /avoid_double/ error_i
      common /absolute_time/ abs_time
      common /sub_cell_size/ subcellsize

C     BEGIN
      center_cell = particle2(pa,2)
      IMAP=(center_cell-1)*26
      last_time=0.5D50
      do js=1,27
       ks=1
601     continue
       if(js.lt.27)then
        if(ks.eq.1)then
         neighbour=MAP(IMAP+js)
         partb=HEAD(neighbour)
         ks=2
        else
         partb=LIST(partb)
        endif
       else
        if(ks.eq.1)then
         partb=HEAD(center_cell)
         ks=2
        else
         partb=LIST(partb)
        endif
        if(partb.eq.pa)then
         goto 501
        endif
       endif
       if (partb.ne.0) then
        coll_time=colltime(pa,partb)
        if(abs_time.eq.0)then
         td=0.5
        else
         td=coll_time/abs_time
        endif
        if((td.lt.0.999999999d0).or.(td.gt.1.000000001))then
```

```
        if(coll_time.lt.1.0D50)then
         call get_pool()
         event_counter=event_counter+1
         node_type(free_node,5)=pa
         node_type(free_node,6)=coll_type
         node_type(free_node,7)=partb
         eventtime(free_node)=coll_time
         call insert_node(free_node)
         if(error_i.ne.1)then
          call update_circ_link(pa,free_node,1)
          call update_circ_link(partb,free_node,2)
         endif
        endif
       endif
      endif
501     continue
      if(partb.ne.0)then
       if(LIST(partb).ne.0)then
        goto 601
       endif
      endif
     enddo
     call cell_crossing(pa)
     call get_pool()
     event_counter=event_counter+1
     node_type(free_node,5)=pa
     node_type(free_node,6)=3
     node_type(free_node,7)=0
     eventtime(free_node)=c_c_time
     call insert_node(free_node)
     if(error_i.ne.1)then
      call update_circ_link(pa,free_node,1)
     endif
     end
C    ------------------------------------------
     subroutine erase_nodes_cc(part,no)
     implicit none

C    Var
     integer*4  part
     integer*4  max_particle
     parameter  (max_particle=10000)
     integer*4  particle2(max_particle,3)
     integer*4  test_node
     integer*4  max_nodes
     parameter  (max_nodes=10000)
     integer*4  node_type(max_nodes,10)
     integer*4  pool_list(max_nodes)
     integer*4  root_var,no
     integer*4  free_node, pointer


C    GLOBAL
     common /tree_structure/node_type
     common /new_events3/  particle2
     common /root/ root_var
```

```
      common /pool/ pool_list, free_node, pointer

C     Begin
      test_node=particle2(part,1)
      if(test_node.ne.0)then
75     continue
      if(node_type(test_node,5).eq.part)then
       if(test_node.eq.no)then
        call re_linking(part,no)
        if(root_var.eq.test_node)then
         call erase_root_node(test_node)
        else
         call erase_node(test_node)
        endif
        call send_pool(test_node)
       else
        test_node=node_type(test_node,8)
        if(test_node.eq.0)then
         write(*,*)'hey sending to node zero'
        endif
        goto 75
       endif
      else
       if(node_type(test_node,7).eq.part)then
         test_node=node_type(test_node,9)
         if(test_node.eq.0)then
          write(*,*)'hey sending to node zero'
         endif
         goto 75
       endif
      endif
      endif
      end
C     ------------------------------------------------
      subroutine update_all_positions()
      implicit none

C     Var
      integer*4         max_particle
      parameter         (max_particle=10000)
      double precision  particle(max_particle,9)
      double precision  rx, ry, rz
      double precision  abs_time
      double precision  tdiff
      double precision  packing, xbox, ybox, zbox
      integer*4         tot_num_par, is

C     Global
      common /box_dim/ packing, xbox, ybox, zbox
      common /new_events2/ particle
      common /absolute_time/ abs_time
      common /gen_events/ tot_num_par

C     Begin
      do is=1, tot_num_par
       tdiff=abs_time-particle(is,7)
```

```
      rx=particle(is,1)+particle(is,4)*tdiff
      ry=particle(is,2)+particle(is,5)*tdiff
      rz=particle(is,3)+particle(is,6)*tdiff
      rx=(rx/xbox-anint(rx/xbox))*xbox
      ry=(ry/ybox-anint(ry/ybox))*ybox
      rz=(rz/xbox-anint(rz/zbox))*zbox
      particle(is,1)=rx
      particle(is,2)=ry
      particle(is,3)=rz
      particle(is,7)=abs_time
     enddo
     end
C    ------------------------------------------
C    ======================FUNCIONES===========================
C    ------------------------------------------
     double precision function gausran(v, m)
     implicit none

     external drand48

C    VAR
     double precision    drand48
     double precision    m, v
     double precision    fac, gsct, rsq, v1, v2

C    BEGIN
1    v1=2.0*drand48()-1.0d0
     v2=2.0*drand48()-1.0d0
     rsq=v1**2+v2**2
     if (rsq.ge.1.0 .or. rsq.eq.0.0) goto 1
     fac=sqrt(-2.0d0*dlog(rsq)/rsq)
     gsct=v1*fac
     gausran=v2*fac*v+m
     end
C    ------------------------------------------
     double precision function colltime(parta, partb)
     implicit none

C    Var
     integer*4          max_particle
     parameter          (max_particle=10000)
     double precision   particle(max_particle,9)
     double precision   vijx, vijy, vijz
     double precision   rijx, rijy, rijz
     double precision   tdiff, bij, rij2, vij2
     integer*4          parta, partb
     integer*4          coll_type
     integer*4          chain(max_particle,2)
     double precision   bond_length, t1, min_time, t
     double precision   packing, xbox, ybox, zbox
     integer*4          i

C    Global
     common /box_dim/ packing, xbox, ybox, zbox
     common /new_events2/  particle
     common /collision_Type/coll_type
```

```
      common /particle_bond/ bond_length
      common /chain_structure/ chain
      common /ini_calc/ i

C     Begin
      min_time=1.0D50
      tdiff=particle(parta,7)-particle(partb,7)
      if (tdiff.lt.0.0d0) then
       tdiff=-tdiff
       rijx=particle(parta,1)+particle(parta,4)*tdiff-
     >      particle(partb,1)
       rijy=particle(parta,2)+particle(parta,5)*tdiff-
     >      particle(partb,2)
       rijz=particle(parta,3)+particle(parta,6)*tdiff-
     >      particle(partb,3)
       colltime=particle(partb,7)
      else
       rijx=particle(parta,1)-(particle(partb,4)*tdiff+
     >      particle(partb,1))
       rijy=particle(parta,2)-(particle(partb,5)*tdiff+
     >      particle(partb,2))
       rijz=particle(parta,3)-(particle(partb,6)*tdiff+
     >      particle(partb,3))
       colltime=particle(parta,7)
      endif
      rijx=(rijx/xbox-anint(rijx/xbox))*xbox
      rijy=(rijy/ybox-anint(rijy/ybox))*ybox
      rijz=(rijz/zbox-anint(rijz/zbox))*zbox

      vijx=particle(parta,4)-particle(partb,4)
      vijy=particle(parta,5)-particle(partb,5)
      vijz=particle(parta,6)-particle(partb,6)

      bij = rijx*vijx+rijy*vijy+rijz*vijz
      rij2=rijx**2+rijy**2+rijz**2
      vij2=vijx**2+vijy**2+vijz**2

      if(bij.lt.0.0d0)then
       t = bij**2-vij2*(rij2-0.999999999999d0)
       if(t.ge.0.0d0)then
        t =(-bij-dsqrt(t))/vij2
        if((t.ge.0.0d0).and.(t.lt.min_time))then
         min_time=t
         coll_type=1
        endif
       endif
      endif

      if((partb.eq.chain(parta,1)).or.
     >   (partb.eq.chain(parta,2)))then
       t1=(1.0d0+bond_length)**2
       t1=bij**2-vij2*(rij2-t1)
       if(t1.ge.0.0d0)then
        t1=(-bij+dsqrt(t1))/vij2
        if((t1.ge.0.0d0).and.(t1.lt.min_time))then
         min_time=t1
```

```
        coll_type=2
       endif
      endif
     endif
     colltime=colltime+min_time
     end
C    ------------------------------------------
     INTEGER FUNCTION ICELL(IX,IY,IZ,MC)
     implicit none
C     VAR
     integer*4 IX, IY, IZ, MC
     ICELL  = 1 + MOD ( IX - 1 + MC, MC )
    >        + MOD ( IY - 1 + MC, MC ) * MC
    >        + MOD ( IZ - 1 + MC, MC ) * MC * MC
C     RETURN
     END
C    ------------------------------------------
     integer*4 function get_min_node()
     implicit none

C    Var
     integer*4  max_nodes
     parameter  (max_nodes=10000)
     integer*4  node_type(max_nodes,10)
     integer*4  root_var, q
     double precision eventtime(max_nodes)

C    Global
     common /tree_structure/node_type
     common /root/root_var
     common /sorting_var/ eventtime

C    Begin
     q=root_var
120  continue
     if(node_type(q,1).ne.0)then
      q=node_type(q,1)
      goto 120
     endif
     get_min_node=q
     end
C    ------------------------------------------
     double precision function get_v2()
     implicit none

C    Var
     integer*4          max_particle
     parameter          (max_particle=10000)
     double precision   particle(max_particle,9)
     integer*4          tot_num_par, is
     integer*4          num_coll

C    Global
     common /new_events2/  particle
     common /gen_events/ tot_num_par
```

```
C     Begin
      get_v2=0
      do is=1, tot_num_par
        get_v2=get_v2+(particle(is,4)**2+particle(is,5)**2+
     :        particle(is,6)**2)/tot_num_par
      enddo
      end
```