UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PARALELIZACIÓN DE ALGORITMOS PARA EL ESTUDIO DE LA INTERACCIÓN DE AGREGADOS

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

JESÚS ANTONIO SOSA HERRERA

DIRECTORA DE TESIS: DRA. SUEMI RODRÍGUEZ ROMO

MÉXICO, D.F. **2008.**

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos:

Agradezco a la Univesidad Nacional Autónoma de México el haberme dado la formación profesional que ahora con orgullo y dedicación desempeño todos los días en beneficion de mi país, mi comunidad, mi familia, y de mi persona.

Agradezco de manera muy especial a la Dra. Suemi Rodríguez Romo por haberme introducido al mundo de la investigación científica. Gracias Dra. por todo su apoyo.

INDICE

Pag.

INDICE DE FIGURAS

Pag.

INDICE DE TABLAS

Pag.

I. INTRODUCCION

I.1 Objetos Fractales.

Se conocen como objetos fractales aquellos con formas geométricas irregularmente fragmentadas que, al ser subdivididos en partes presentan cierto grado de semejanza con respecto al objeto original. Su nivel de fragmentación no tiende a desvanecerse o a fluctuar a escalas mayores o menores. Los objetos fractales exhiben también la propiedad de auto-similaridad que consiste en que presentan la misma –o más o menos la misma– estructura cuando se observan a diferentes escalas.

La rama de las matemáticas que nos permite estudiar los objetos fractales es la geometría fractal. El concepto de geometría fractal fue introducido a mediados de los 70's por Benoit Mandelbrot (Mandelbrot, 1988), derivándola de estudios de objetos, considerados hasta entonces como "*monstruos matemáticos*", que están definidos por descripciones algorítmicas sencillas que requieren entradas simples, pero que proporcionan salidas excepcionalmente complicadas. En la geometría fractal, un objeto fractal se caracteriza porque su *dimensión euclidiana* suele ser menor a su *dimensión de Hausdorff*.

La *dimensión de Hausdorff* se obtiene a partir de la *medida de Hausdorff* que, a muy grandes rasgos, determina las extensiones de subconjuntos de $\Re^n$ que presentan geometrías complicadas, haciendo uso de cubiertas cerradas.

Las definiciones formales de la medida y dimensión de Hausdorff, así como de los conceptos necesarios para su definición, se expresan como sigue (Falconer, 2003):

Sea $U \subseteq \Re^n$ con $U \neq \varnothing$, el *diámetro* de U se define como:

$$|U| = \sup\{|x - y| : x, y \in U\} \qquad \ldots(I.1)$$

Si $\{U_i\}$ es una colección contable (o finita) de conjuntos con diámetros a lo más de ε, que cubren a $F \subseteq \Re^n$, esto es $F \subset \cup_{i=1}^{\infty} U_i$, con $0 < |U_i| \leq \varepsilon$ para cada i, entonces se dice que $\{U_i\}$ es una $\varepsilon - cobertura$ de F.
Supóngase que s es un número no negativo, entonces, para cada $\varepsilon > 0$ se define:

$$\mathrm{H}^s_\varepsilon(F) = \inf\{\sum_{i=1}^{\infty} |U_i|^s : \{U_i\} \; es \; una \; \varepsilon - cobertura \; de \; F\} \qquad \ldots(I.2)$$

La *medida* s *- dimensional de Hausdorff* del conjunto F se define como:

$$\mathrm{H}^s(F) = \lim_{\varepsilon \to 0}(\mathrm{H}^s_\varepsilon(F)) \qquad \ldots(\mathrm{I}.3)$$

Una propiedad importante de la *medida de Hausdorff* es la *propiedad de escalamiento*, que consiste en que si $S : \Re^n \to \Re^n$ es una transformación tal que si $\lambda > 0$, se cumple $|S(x) - S(y)| = \lambda |x - y|$, entonces se cumple también que

$$\mathrm{H}^s(S(F)) = \lambda^s \mathrm{H}^s(F) \qquad \ldots(\mathrm{I}.4)$$

La *dimensión de Hausdorff*, también conocida como *dimensión Hausdorff-Besicovitch* está dada por:

$$\dim_H F = \inf\{s \geq 0 : \mathrm{H}^s(F) = 0\} = \sup\{s : \mathrm{H}^s(F) = \infty\} \quad \ldots(\mathrm{I}.5)$$

Además de la *dimensión de Hausdorff*, existen otras definiciones alternativas de dimensionalidad para objetos fractales, que son ampliamente utilizadas debido a que, aunque $\dim_H F$ se define para cualquier subconjunto de $\Re^n$ en general, esta definición no siempre es aplicable en la práctica, por lo que para propósitos de cálculo y experimentación, suelen utilizarse otras definiciones de dimensión, como la siguiente, inspirada en la propiedad (1.4) (Barnsley, 1988):

Sea $F \subseteq \Re^n$, para cada $\varepsilon > 0$ sea $N_\varepsilon(F)$ el menor número de bolas cerradas de radio ε que se necesitan cubrir a F, si

$$D = \lim_{\varepsilon \to 0} \frac{\log(N_\varepsilon(F))}{\log(1/\varepsilon)} \qquad \ldots(\mathrm{I}.6)$$

existe, entonces a D se le llama la *dimensión fractal* de F.

Los valores de dimensión derivados de cada una de las definiciones anteriores, no necesariamente son iguales para todos los conjuntos sobre los cuales se aplican, lo que si se cumple (Falconer, 2003) es que:

$$\dim_H F \leq D \qquad \ldots(\mathrm{I}.7)$$

para un mismo $F \subseteq \Re^n$.

Los fractales pueden ser divididos en dos categorías, aleatorios y no aleatorios, según intervengan o no procesos estocásticos en su formación (Bunde, 1996). En la naturaleza, solamente se observan fractales aleatorios.

I.2 Fractal DLA.

El proceso de Agregación Limitada por Difusión (DLA) descrito por primera vez por T. Witten y L. Sanders (Witten & Sanders, 1981) como un modelo para la agregación coloidal irreversible. Poco después, se comprobó que dicho modelo también servía para simular el comportamiento de varios fenómenos físicos, como son la acumulación de electrolitos, difusión de fluidos en medios porosos, crecimiento bacteriano, agregación de cristales, coalescencia de partículas, comportamiento de reactivos, entre otros (Vicsek, 1992).

El alcance que tiene este modelo queda plasmado por la siguiente aserción:
"El modelo DLA es aplicable a la agregación en cualquier sistema en el que la difusión es el medio primario de transporte en el sistema." (Halsey, 2001).

Lo anterior ha llevado a que el conocimiento de las propiedades del DLA sea de gran importancia para establecer la exactitud con que se pronostican o analizan tales fenómenos. Muchas de estas propiedades son apreciables solamente cuando se han simulado agregados con un gran número de partículas.

La generación de agregados fractales implica procesos estocásticos en los cuales no es aplicable la utilización de métodos determinísticos, por lo que se recurre a la simulación. Tales procesos de simulación consumen una gran cantidad de recursos de cómputo y, a diferencia de algunos otros tipos de fractales, los agregados por difusión limitada no son trivialmente paralelizables, por lo que hay pocas técnicas prácticas del uso de multiprocesamiento en dichos procesos.

Investigaciones más recientes (Tchijov, *et al*. 1996) han introducido reglas que permiten la interacción entre agregados creciendo simultáneamente. El estudio de estas variantes del modelo DLA permitirá darle más aplicaciones, como puede ser el estudio de una variedad más amplia de reacciones, varios tipos de crecimiento, difusión de sustancias heterogéneas, estudios evolutivos, diagnóstico de tumores malignos, entre otros.

I.3 Objetivo de Tesis.

El presente trabajo tiene como objetivo llevar los algoritmos para la generación y estudio de la interacción de agregados fractales por difusión limitada a una implementación de procesamiento paralelo confiable, que utilice de manera razonable recursos como son el tiempo de procesamiento y la memoria disponible; evaluando su desempeño y determinando así, la viabilidad de su aplicación al análisis de la interacción entre múltiples agregados.

Lo anterior se considera una tarea importante porque a la fecha no se encuentra literatura en la que se analice de manera experimental la interacción de agregados utilizando procesamiento paralelo con la finalidad de manejar un número elevado de partículas. Como se sabe, al manejar agregados fractales de mayor tamaño, se tiene mayor precisión en las estimaciones de las propiedades asintóticas de los mismos, que es lo que se pretende lograr en el trabajo aquí descrito.

I.4 Esquema de Trabajo.

Se parte de un algoritmo secuencial, optimizado mediante el uso de estructuras de datos multidimensionales, como son el *kd-tree* y el *quad-tree*, encontrándose que el *quad-tree* presenta mayor simplicidad para la codificación de técnicas de aceleración de búsquedas.

Para hacer un uso eficiente de memoria, se utiliza la versión *bucket* del *quad-tree*. Las búsquedas sobre datos es *primero en profundidad*, haciendo uso de heurísticas pertenecientes a técnicas *primero el mejor* (Samet 2006). Para mejorar los tiempos de búsqueda se recurre al uso de búsquedas aproximadas y, cuando es necesario, se realizan búsquedas exactas.

La técnica de multiprocesamiento utilizada es la creación de hilos a través de la interfaz *pthreads* (Butenhof, 1997), además del uso de *OpenMP* (Dagum, 1997) como una alternativa de codificación. Al ejecutarse los hilos sobre una plataforma de multiprocesamiento, se consigue el paralelismo de las operaciones implicadas en la generación del agregado fractal en cuestión. La sincronización entre los procesadores se lleva a cabo mediante el uso de mutexes para el caso de *pthreads* y de definición de secciones críticas para el caso de *OpenMP*.

Para la observación, se utilizan métodos de visualización que permiten manejar los agregados en formatos gráficos. Con esto se crean representaciones que permiten observar propiedades cualitativas de los agregados. Las técnicas de visualización aquí empleadas hacen uso de las librerías gráficas *OpenCV* (Intel Corp, 2001) y de los formatos de dibujo *DXF* (Autodesk, 2000).

Se realizan también análisis de dimensionalidad fractal para tener la seguridad de que los agregados creados corresponden exactamente al modelo DLA.

II. MODELO DE DIFUSION POR AGREGACION LIMITADA.

II.1 Algoritmo DLA

El modelo de difusión por agregación limitada (DLA), descrito originalmente por Witten & Sanders (Witten & Sanders, 1981), que produce agregados fractales, se puede resumir en los siguientes pasos:

1. Una partícula inicial o “semilla” en el origen conforma el agregado inicial.

2. Si el número de partículas que conforman el agregado es mayor o igual que n, terminar, si no, continuar.

3. Otra partícula se inicia a cierta distancia al origen denominada “radio de nacimiento” y comienza a realizar una sucesión de saltos aleatorios generando así un “movimiento browniano”.

4. Si la partícula en movimiento se encuentra con una de las partículas que conforman el agregado, esta también pasa a formar parte del agregado y se regresa al paso 2.

5. Si la partícula excede cierta distancia al origen llamada “radio de muerte”, la partícula se elimina y se regresa al paso 3.

En la figura II.1 se esquematiza el fenómeno físico que se simula con el modelo.

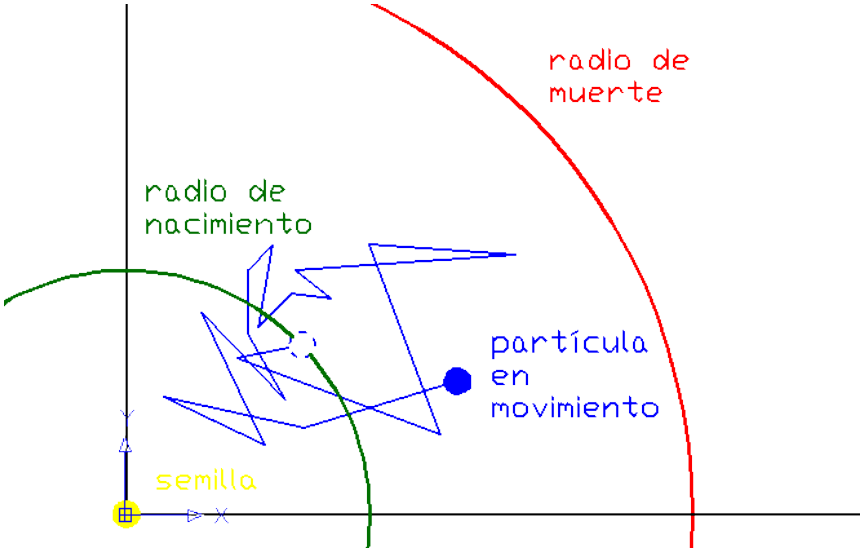


Figura II.1 Algoritmo DLA

En la figura II.2 podemos observar una estructura típica obtenida mediante la aplicación del algoritmo DLA con un número de partículas n=250,000. El agregado fractal fue construido con el software elaborado para esta tesis siguiendo la metodología para un solo procesador descrita en el siguiente capítulo.

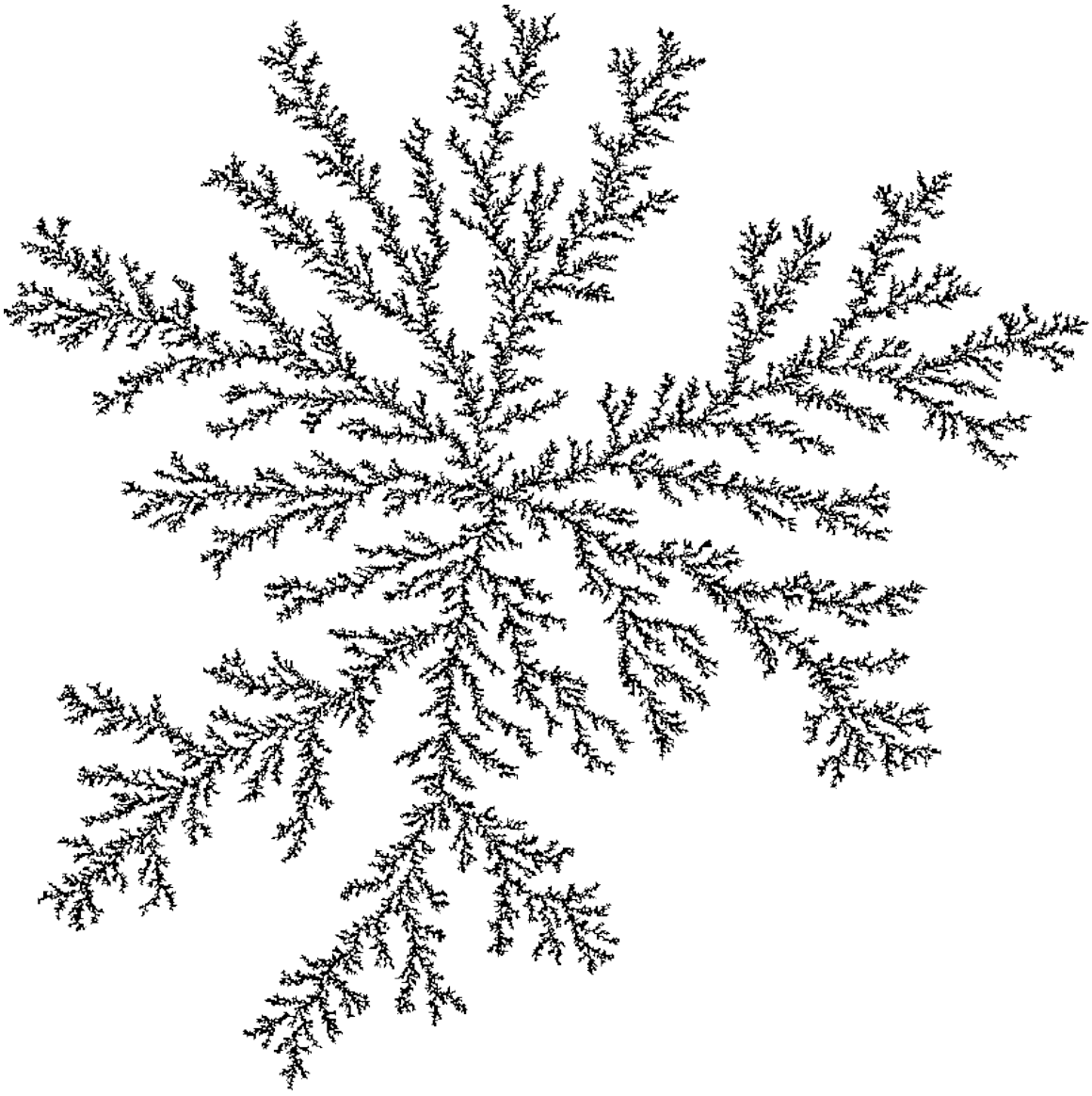


Figura II.2 Agregado DLA con n=250,000 partículas.

II.2 Modelos "*Lattice"* y "*Off-lattice"*.

Si los saltos aleatorios que dan las partículas en el modelo DLA son de longitud fija y se asume que las partículas solo pueden estar en posiciones con una disposición determinada por dicha longitud se dice que tiene el modelo "*lattice"* o de rejilla.

La figura II.3 ejemplifica una distribución de partículas sobre una rejilla cuadrada. La geometría de la rejilla puede ser, sin embargo, cualquier figura geométrica regular que tienda a llenar el espacio.

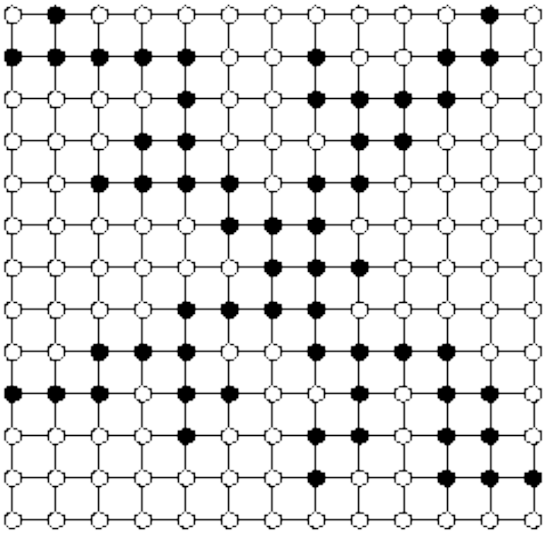


Figura II.3 Modelo "*lattice"*.

Si se asume que las partículas no están confinadas a una estructura, sino que pueden ocupar cualquier lugar en el espacio, sin interferir con la demás partículas, se dice que se tiene el modelo "*off-lattice*". Este esquema se muestra en la figura II.4.

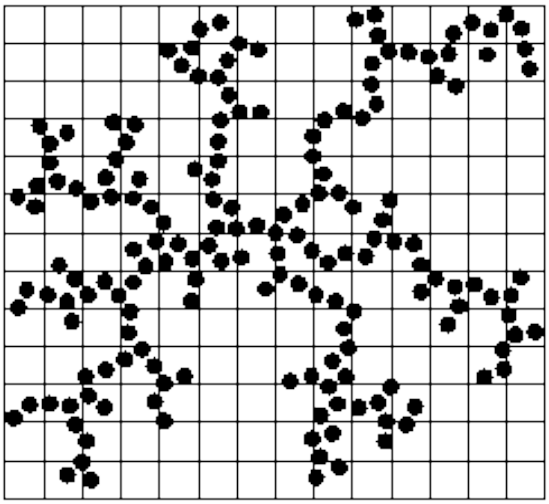


Figura II.4 Modelo *off-lattice*.

Cabe destacar que el modelo de rejilla puede obtenerse a partir del modelo *off-lattice*, mediante un ajuste de coordenadas al momento de adherir la partícula al resto del agregado. Además, la geometría de la rejilla puede alterar la morfología y otras propiedades a gran escala del agregado (Meakin, 1986).

II.3 Complejidad Computacional del Algoritmo DLA.

La complejidad computacional del algoritmo DLA está principalmente asociada al cálculo de las caminatas aleatorias de las partículas.

Los autores Moriarty y Machta, muestran la forma de evaluar la complejidad de este tipo de procesos (Moriarty & Machta, 1997).

Considerando el modelo de rejilla, tenemos lo siguiente: El valor esperado $E(x)$ de la longitud de la caminata aleatoria, esto es, el número de saltos que dará una partícula en movimiento antes de adherirse al cluster, es proporcional al área que ha de cubrir dicha caminata. Dada la naturaleza radial del modelo simulado, para caminatas en dos dimensiones obtenemos:

$$E(x) \propto r_c^{\,2} \qquad \text{…(II.1)}$$

donde r_c es el radio del agregado, dado por la distancia entre la partícula más alejada al origen y la semilla del agregado. Sabemos también que

$$r_c \cong r_g \qquad \text{…(II.2)}$$

con r_g denotando al *radio de giro* de nuestro agregado. El radio de giro representa la máxima distancia del centro de inercia del agregado a las partículas que conforman el miso. El *radio de giro* r_g puede calcularse así:

$$r_g = \sqrt{\frac{1}{n}\sum_{i=1}^{n} |\vec{x}_i - \vec{x}_0|^2} \qquad \text{…(II.3)}$$

donde n es el número total de partículas, x_i es la posición de la *i*-ésima partícula que conforma el agregado, y x_0 es una posición fija, pudiendo ser, por ejemplo, la posición de la semilla, o bien el origen.

Una de las propiedades más conocidas y aceptadas del DLA (Bunde & Shlomo, 1996) es la siguiente:

$$n \sim r_g^{\,D} \qquad \ldots(II.4)$$

con D igual a la dimensión fractal del agregado. De aquí obtenemos

$$r_g \sim n^{1/D} \qquad \ldots(II.5)$$

considerando las relaciones (II.1), (II.2) y (II.5) vemos que

$$E(x) \propto r_c^{\,2} \cong r_g^{\,2} \sim n^{2/D} \qquad \ldots(II.6)$$

teniendo en cuenta que las n partículas realizan una caminata aleatoria con un valor esperado de número de saltos descrito por (II.6), llegamos a la conclusión de que el algoritmo DLA tiene una complejidad para el caso promedio de:

$$O(n \cdot n^{2/D}) = O(n^{1+2/D}) \qquad \ldots(II.7)$$

II.4 Modelos Multicolor en el Proceso de Difusión Limitada por Agregación.

Una variante significativa al modelo DLA original fue propuesta por los investigadores V. Tchijov, S. Nechaev y S. Rodríguez Romo (Tchijov *et al*, 1996), en la cual es posible distinguir entre diferentes tipos de partículas en difusión, asignándoles un identificador (color) correspondiente a cada tipo. El propósito de manejar más de un color en las partículas en movimiento, va más allá de la enumeración de las partículas que se adhieren al agregado originado por cierta semilla. Los colores permiten analizar la interacción de los agregados bajo ciertas reglas, lo que ha llevado al descubrimiento de nuevas propiedades del modelo DLA (Rodríguez-Romo *et al*, 2004).

Aunque los agregados con interacción suelen tener la misma dimensión fractal que el modelo DLA original, su morfología cambia. Estos cambios aparecen de manera aleatoria, o bien debido a diferentes parámetros, como son la separación entre las semillas iniciales, o por cambios en la reglas de interacción.

A la fecha, las investigaciones sobre este tipo de modelos se encuentran aún en desarrollo.

Una forma de simular la interacción de diferentes agregados es a través de la adición de las siguientes reglas al modelo DLA tradicional (Rodríguez-Romo *et al*, 2004) citadas textualmente:

1. *“Si una partícula en movimiento aleatorio con color A(B), alcanza un punto de la rejilla vecino a la semilla A(B), entonces se adhiere para formar un nuevo agregado inicial de dos partículas A(B). Este proceso es recursivo hasta alcanzar agregados de n partículas”.*

2. *“Si una partícula en movimiento aleatorio con color A(B) alcanza un punto de la rejilla vecino a la semilla B(A), entonces muere”.*

Estas reglas pueden extenderse a más colores, con el consecuente incremento de complejidad de la interacción entre los agregados. Es posible también modificar las reglas de interacción. Los fenómenos resultantes con tales modificaciones siguen siendo temas abiertos de investigación.

Las figuras II.5 y II.6 muestran, respectivamente, diferentes casos de interacción de agregados referentes a los modelos de dos y tres colores. En éstas se puede apreciar que la complejidad de la interacción produce morfologías heterogéneas para diversas simulaciones del mismo fenómeno. Los fractales mostrados se elaboraron con el software obtenido para este trabajo siguiendo la metodología descrita en el capítulo siguiente con n=100,000 partículas para cada caso.

Figura II.5 Modelo DLA dos colores.

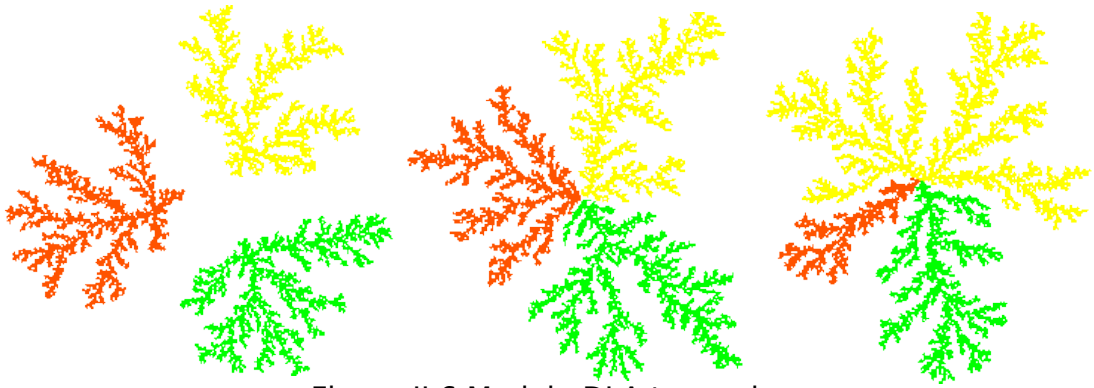


Figura II.6 Modelo DLA tres colores.

III. METODOLOGIA.

III.1 Herramientas de Software y Hardware.

A pesar de la simplicidad de la descripción del proceso con el que se obtienen los agregados fractales por difusión limitada, la generación del código que simule agregados lo suficientemente grandes para realizar análisis confiables sobre éstos, es una tarea complicada.

Dado que la velocidad de ejecución y el uso eficiente de memoria están entre los factores más importantes a tener en cuenta en la implementación del algoritmo, y que el programa debe realizar tareas con cierto nivel de complejidad, se escoge el lenguaje *C estándar* para desarrollar el código.

Teniendo en cuenta la gran cantidad de cálculos necesarios para la simulación del fenómeno DLA, el primer aspecto a considerar en el desarrollo del software que ejecute el algoritmo DLA es la reducción del tiempo de procesamiento. Una técnica que reduce considerablemente el tiempo de procesamiento es la utilización de multiprocesamiento, que consiste, a grandes rasgos, en la utilización de varios procesadores ejecutando en forma paralela el mismo, o parte del mismo algoritmo, comunicando entre si los resultados parciales obtenidos para obtener un resultado final. Debido a la gran dependencia histórica en el proceso de agregación limitada por difusión, el algoritmo DLA, a diferencia de otros procesos de crecimiento fractal, no es trivialmente paralelizable (Matchta,2006). Más aún, el multiprocesamiento genera una distorsión sobre el modelo DLA que tiende a comprimir la estructura del objeto en simulación, aumentando su dimensionalidad fractal, haciendo que el agregado pase a formar parte de otra clase de universalidad estudiada en el proceso MPDA –*Multi-Particle Diffusive Aggregation*–(Voss, 1984). Es posible, sin embargo, para un número determinado de procesadores, lograr que el modelo MPDA converja hacia el modelo DLA si el número de partículas que conforman el agregado es suficientemente grande (Kaufman *et al*. 1995).

El programa desarrollado se elaboró con la capacidad de ejecutarse en paralelo sobre un número variable de procesadores, con ciertas restricciones, para hacer converger el modelo MPDA hacia el modelo DLA, como se expone más adelante.

El compilador seleccionado es *GNU Compiler Collection*, mejor conocido por las siglas *gcc*. Esto debido al grado de estabilidad que ha presentado en el desarrollo de aplicaciones que requieren el uso de procesamiento paralelo (Lastovetsky, 2003).

Durante la codificación del programa para el presente trabajo, el único inconveniente encontrado con la elección de *gcc* fue la ausencia de una interfaz amigable para el usuario, sobre todo en la ejecución del comando de depuración

gdb. Lo anterior sólo afectó a la etapa de desarrollo, no así al desempeño del código obtenido.
Se utiliza el sistema operativo *Linux* como plataforma de programación. *Linux* ofrece desde la versión 2.4 de su núcleo, soporte para multiprocesamiento simétrico (SMP), además de que cumple de manera más que satisfactoria los estándares *POSIX* (Kurt Wall, *et al*, 2008). Muchos servidores importantes de procesamiento masivo utilizan *Linux*, o bien, un sistema operativo compatible con los estándares *POSIX (Unix, Aix, Solaris*, entre otros). Esto garantiza que al utilizar *Linux* como plataforma de programación se tiene un buen grado de portabilidad hacia diferentes equipos de multiprocesamiento.
A pesar de utilizar una plataforma fija, el código generado no queda inevitablemente ligado a esta plataforma, como se verá más adelante.

Para el control del multiprocesamiento se utiliza la interfaz estándar *POSIX* (*Portable Operating System Interfaces*) conocida como *POSIX Thread*s, o *pthread*s. Asimismo, se hace uso de las librerías *OpenMP*, en una codificación del programa independiente a la que utiliza *pthreads*, obteniendo así una portabilidad mayor.

El hardware en el que se realizan las pruebas del programa obtenido, consiste tanto de un equipo estándar con un procesador *Intel Celeron M* (1.4 GHz, 1.2 GB *RAM*), así como de un servidor de alto rendimiento con ocho procesadores *Intel Xeon* (3.2 GHz, 4 GB *RAM*). Como se ha expuesto, el programa está diseñado para ejecutarse en una gran variedad de equipos con soporte de multiprocesamiento simétrico, y no se encuentra particularmente restringido a las características de la plataforma y los equipos en los que se realizaron las pruebas aquí presentadas.

III.2 Algoritmo Secuencial.

El paso previo a la paralelización del algoritmo DLA, es la creación de código secuencial que genere agregados fractales de manera eficiente. En este caso, la propiedad *'secuencial'* se refiere a que el código se ejecutará solamente por un procesador y, por tanto, de manera absolutamente sincrónica, en el orden establecido por las instrucciones y estructuras del lenguaje de programación. A partir del código obtenido de esta manera se genera la versión del código en paralelo.

III.2.1 Técnicas para Acelerar la Simulación.

Al analizar el modelo de rejilla, se puede observar que, mientras se realiza la simulación del movimiento browniano, el resultado del movimiento de varios saltos aleatorios fijos puede sustituirse por un solo salto de longitud equivalente, con respecto al punto de inicio de la caminata aleatoria. Esto sólo es posible si la partícula se mueve en un espacio libre, esto es, si no hay otras partículas que

interfieran en la trayectoria. Como la dirección del salto es aleatoria, la aplicación de un salto libre solamente es posible si se sabe que existe una vecindad de radio igual a la longitud del salto en la que no existe ninguna partícula perteneciente al agregado. La figura III.1 ejemplifica la sustitución de varios saltos aleatorios por la sustitución de un solo salto resultante.

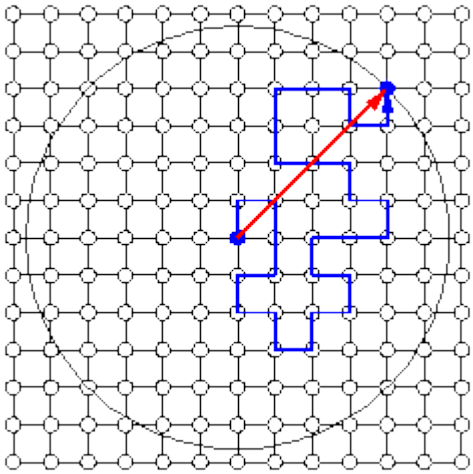


Figura III.1 Varios saltos como uno solo.

La sustitución de varios pasos por un solo paso equivalente, es más eficientemente implementada en el modelo "*off-lattice"*, pues se evitan muchos ajustes intermedios de coordenadas. Dadas las ventajas obtenidas por el cambio de modelo, se toma al modelo "*off-lattice"* como la versión a codificar.

El diagrama de flujo detallado correspondiente a la implementación algorítmica del modelo "*off-lattice"* se muestra en la figura III.2. En esta figura, *N* representa el número de partículas, *A* es el conjunto que conforma el agregado, r_n y r_m son los radios de nacimiento y muerte respectivamente, y *tol* es la tolerancia de adherencia al cluster, con $\varepsilon << tol$; *d(p,A)* representa la distancia del punto p al conjunto *A*. Todas las coordenadas se consideran pertenecientes al espacio continuo bidimensional, representadas por variables referentes a tipos de datos reales.

La entrada del algoritmo la constituye el número de partículas *N* que se desea conformen el agregado. La salida será el conjunto *A* que consta de todas las partículas que son parte del agregado.

El ciclo principal del algoritmo está identificado por la etiqueta *1*, que realiza la simulación del movimiento browniano de la partícula actual. El bucle identificado por la etiqueta *2*, inicia nuevamente el movimiento browniano desde el radio de nacimiento r_n para las partículas que se alejaron más allá del radio de muerte r_m. El ciclo identificado con la etiqueta *3*, realiza la adición de una nueva partícula al agregado fractal, después de lo cual el proceso, si aún no se ha llegado al tamaño deseado del fractal, se vuelve a iniciar.

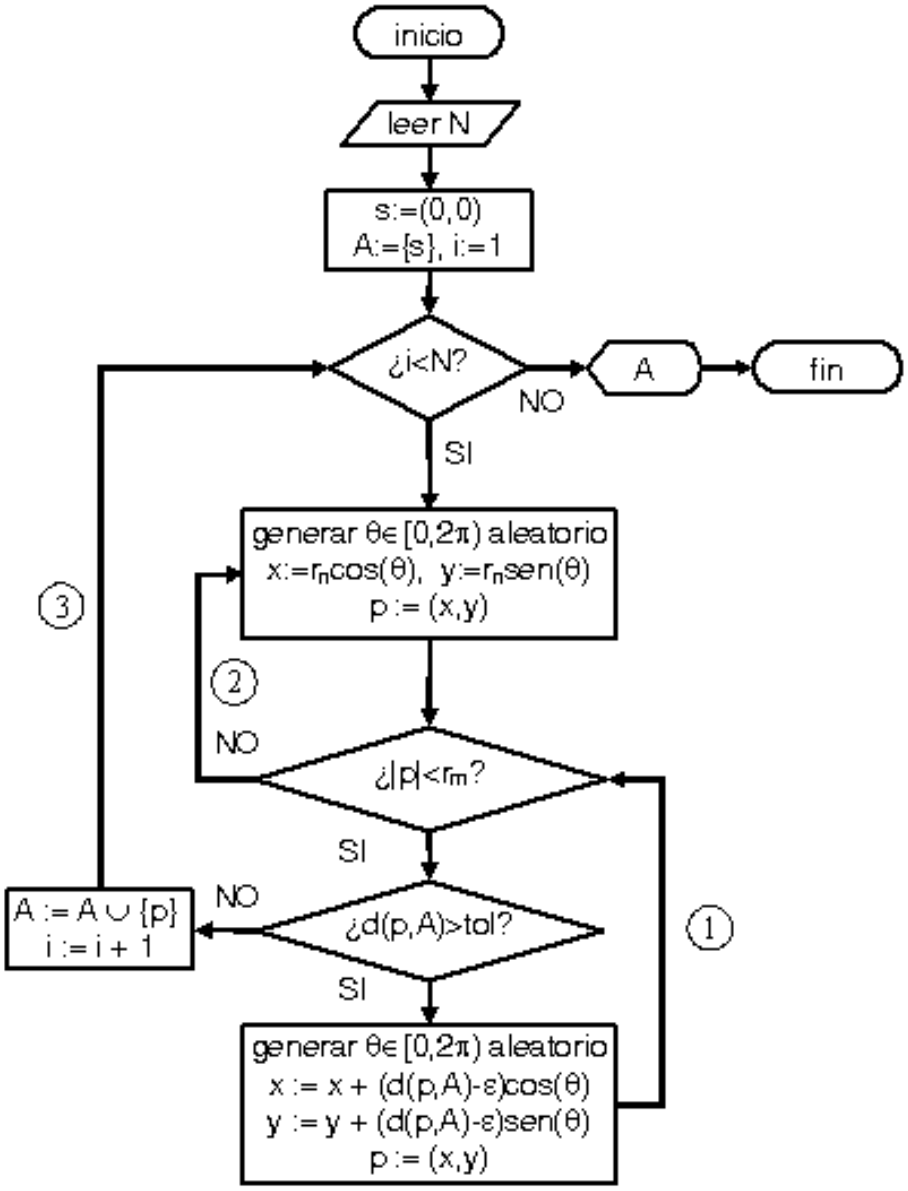


Figura III.2 Diagrama de flujo del Algoritmo DLA para el modelo "*off-lattice*".

Este proceso siempre terminará, aunque posiblemente, después de un número de pasos bastante grande. La probabilidad de que una partícula se encuentre con el agregado puede llegar a ser muy baja, pero nunca es nula debido a que el radio de muerte r_m se toma habitualmente dentro de un rango relativamente corto con respecto al radio de nacimiento r_n. La elección de valores apropiados para los radios r_n y r_m puede ayudar a disminuir el valor esperado en el número de saltos que tiene que dar una partícula antes de encontrarse con el agregado.

En una simulación eficiente, se debe inducir a la partícula en movimiento a saltar la mayor distancia posible sin traslaparse ni chocar con ninguna de las partículas que conforman el agregado.

Se tiene, por tanto, que verificar la distancia de la partícula en movimiento, al agregado, la cual está dada por la mínima distancia entre la partícula en movimiento *p* y todas las partículas del agregado *A*. (ecuación III.1).

$$d(p, A) = \min_{a \in A}(d(p, a)) \qquad \text{...(III.1)}$$

en donde $d(p, a)$ representa la distancia de la partícula a a la partícula p.

La figura III.3 muestra un ejemplo gráfico del tipo de distribución en el espacio típica de un subconjunto de puntos sobre los cuales se realiza la búsqueda.

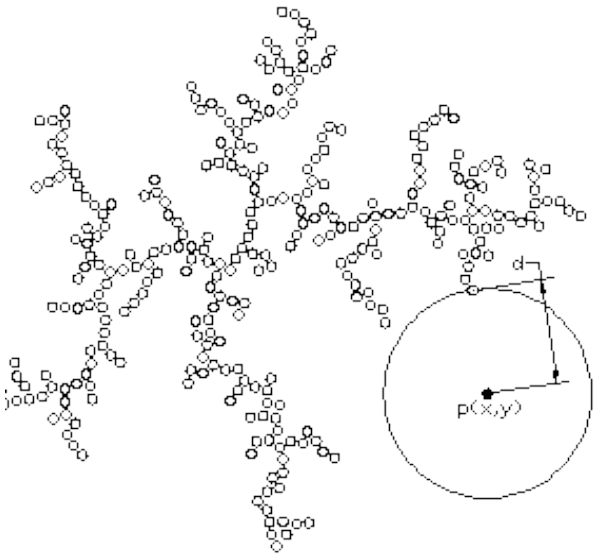


Figura III.3 Distancia de un punto a un agregado.

Considerando que se maneja un número bastante elevado de partículas (del orden de 10^8), se requiere de una técnica de búsqueda y una estructura de almacenamiento en memoria adecuadas.

En las secciones siguientes, se muestra la forma en la que se utilizaron diferentes técnicas de búsqueda y almacenamiento de datos para realizar la codificación de varias versiones del algoritmo DLA. En el capítulo dedicado al análisis de resultado se muestra el rendimiento de cada versión, con lo que es posible apreciar las ventajas de una técnica en particular con respecto a las otras.

III.2.2 Uso de Listas Ligadas y Árboles Binarios.

Dado que las partículas que conforman el agregado se van obteniendo en forma dinámica, y que el espacio que cubren cuando aumenta su cantidad es demasiado grande, se utilizan estructuras de datos dinámicas para almacenarlas, ya que con el almacenamiento estático no se aprovecharía la memoria de manera eficiente.

La estructura de datos más sencilla empleada es la lista ligada. Se puede, por ejemplo, realizar una búsqueda secuencial sobre la lista ligada, en la que se van guardando las partículas del cluster. Si la lista esta ordenada por radios, se puede utilizar la desigualdad del triángulo para limitar el espacio de búsqueda, de la siguiente manera:

Se empieza a buscar desde el extremo de la lista que contiene los puntos más externos del agregado, es decir, con mayor radio. Si se tiene que la distancia más cercana encontrada al momento es menor que la diferencia de los radios entre la partícula en movimiento y la partícula de la lista que se está examinando, entonces la búsqueda se detiene.

Una forma más efectiva para realizar una búsqueda sobre un conjunto resulta ser una búsqueda binaria. La estructura que almacene los datos puede entonces ser un árbol binario balanceado. El índice de búsqueda puede ser nuevamente el radio de la partícula con respecto al origen.

Conforme aumenta el número de partículas, nos damos cuenta de que, tanto la lista ligada, como el árbol binario, resultan ser ineficaces. El empleo de estructuras y técnicas de búsqueda tradicionales (Knuth, 1979; Aho *et al*., 1983; Langsman *et. al*.,1997), como son árboles *B*, tablas de dispersión *Hash*, entre otros, nos lleva a resultados similares.

La razón de lo anterior, es que se realiza el ordenamiento de los datos con respecto a un solo índice, y nuestros datos se encuentran dispersos en un espacio bidimensional. Partículas con un mismo radio no necesariamente están cerca una de la otra, lo que lleva a buscar una mínima distancia sobre datos que se encuentran bastante separados, sin una indicación extra que permita distinguir dicha situación.

Afortunadamente, existen estructuras de datos especializadas (o adaptaciones a las estructuras tradicionales) para manejar información multidimensional. Tales estructuras han sido exitosamente empleadas en aplicaciones de bases de datos espaciales, visión computacional, sistemas de información geográfica, programación de videojuegos, diseño VLSI, entre otros. (Samet, 2006).

III.2.3. Empleo de *KD-Tree*.

El *kd-Tree* es una estructura con un balance espacial ordenando en un árbol binario y alternando en cada nivel la dimensión con respecto a la cual se ordena. Esta estructura fue introducido por Bentley (Bentley, 1975). La '*k*' representa el número de dimensiones o índices con respecto a los cuales se ordena la información. La figura III.4 muestra el almacenamiento de tres puntos en un *2d-tree*. Las letras representan nodos, cada nodo puede tener dos hijos. Un hijo puede ser un nodo, o un dato. Cada nodo divide una dimensión a la mitad, alternándose en cada nivel del árbol.

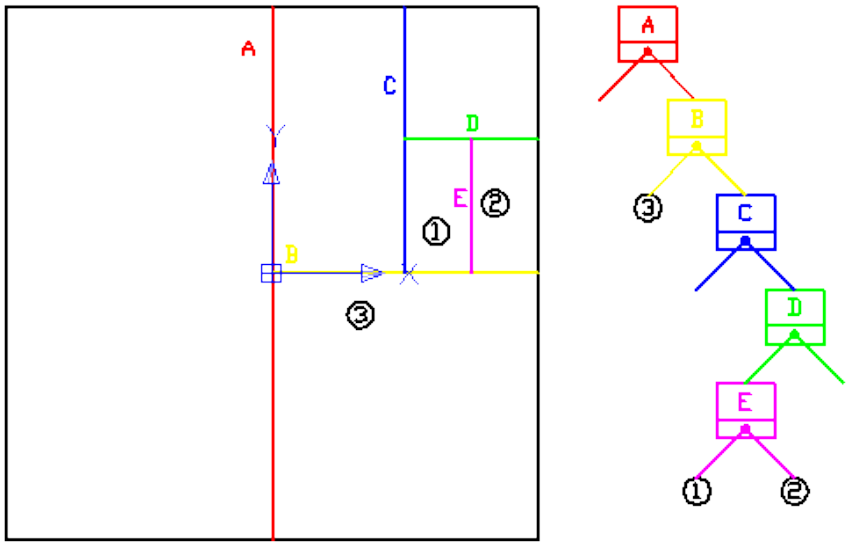


Figura III.4 KD-Tree.

Una forma de balancear el árbol *kd-tree* que contiene los puntos del agregado, es tomando la dimensión con máxima amplitud y dividirla en dos para cada nivel, esto después de que se han agregado cierto número de partículas. Con este esquema no se obtuvieron resultados satisfactorios, pues nuestra generación de datos es demasiado dinámica.

El método de balanceo que resultó adecuado, fue tomar inicialmente una magnitud máxima fija de expansión para cada dimensión, la cual se sabe de antemano no será excedida por los puntos que conforman el agregado, y así, realizar la división alternada de cada dimensión hasta obtener un espacio reducido, en el que solamente cabe un limitado número de partículas. Esta variante se conoce como método de "*bucket*" (Samet, 2006). El método de *bucket* también ayuda a agrupar partículas en conjuntos que tienen en común coordenadas cercanas, y ha demostrado ser eficiente cuando los datos a manejar se encuentran muy esparcidos (Samet, 2006).

III.2.4 Uso de *Quad-Tree*.

El *quad-tree* es una estructura que realiza un balanceo espacial en dos dimensiones, de forma que puntos dentro de un mismo sub-cuadrante están dentro de un mismo hijo de un nodo. El *quad-tree* fue inventado por Finkel y Bentley (Finkel & Bentley, 1974) Cada nodo del *quad-tree* tiene cuatro hijos, que pueden contener un dato u otro nodo. La figura III.5 ejemplifica el almacenamiento de tres puntos sobre un *quad-tree*.

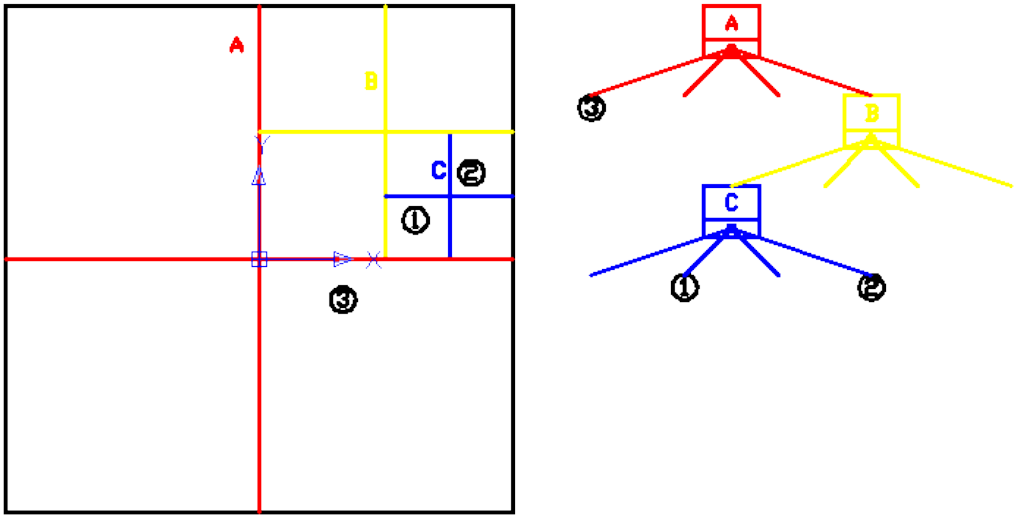


Figura III.5 Quad-Tree.

Una de las ventajas que presenta el uso del *quad-tree*, es que divide a ambas dimensiones simultáneamente, por lo que las búsquedas a través de esta estructura suelen ser un poco más rápidas que en el *kd-tree*. Esto es debido a que la división del espacio en dos direcciones se obtiene al inspeccionar un solo nivel del *quad-tree*, mientras que en el *kd-tree* se requiere la inspección de dos niveles para realizar la misma división. Como puede apreciarse en las figuras III.4 y III.5, el *quad-tree* suele tener, en general, una profundidad menor que el *kd-tree* para el almacenamiento de la misma cantidad de datos.

Una desventaja del *quad-tree* sobre el *kd-tree*, es que no se puede escalar tan fácilmente a otras dimensiones. La versión tridimensional denominada *oct-tree*, no presenta grandes dificultades, sin embargo, para dimensiones mayores, la codificación de tales estructuras resulta complicada, puesto que cada nodo debe tener un número fijo de hijos que crece en proporción 2^d, en donde d es la dimensión. En cambio, cada nodo de un *kd-tree* solamente tiene dos hijos, sin importar la dimensión de los datos que se estén manejando.

La búsqueda a través de esta estructura puede realizarse en profundidad (*Depth Search*), o bien con estrategia primero el mejor (*Best First Search*) (Samet, 2006).

III.2.5 Búsqueda Aproximada en el *QuadTree*.

La creación de agregados fractales de gran tamaño requiere todavía de más elementos de aceleración para generar un gran número de partículas. Podemos encontrar que una estimación, más que un cálculo exacto en la distancia máxima que puede saltar una partícula, da como resultado un incremento de velocidad significativo en simulación del proceso. Algunos autores utilizan mapas de ocupación de partículas (Kauffman, et al. 1995; Tolman & Meakin 1989; Menshutin & Schur 2006), con varios niveles de profundidad y, por tanto, de precisión en la distancia de salto.

La idea básica detrás de este tipo de búsqueda es la siguiente:

- Si una partícula se encuentra alejada del agregado, entonces se buscará cuál es el nodo ocupado más cercano, y se tomará como distancia de salto la distancia entre partícula y los límites del área que cubre el nodo dentro del *quad-tree*.
- Si la partícula se encuentra cercana al agregado, entonces se debe realizar una búsqueda entre las partículas que conforman el agregado, sacando provecho de la estructura del *quad-tree* para buscar solamente entre las partículas cercanas.

Como se ha mencionado, las búsquedas aproximadas suelen hacerse mediante mapas de bits sobre varios niveles del *quad-tree*.

La sustitución de una distancia aproximada en lugar de una exacta incrementa el número de saltos que una partícula tiene que dar para alcanzar al agregado, pero la ganancia en velocidad de búsqueda es mucho mayor. Los autores Henry Kauffman y Benoit Mandelbrot (Kauffman *et al*. 1995), utilizan el esquema de *quad-tree* como estructura de almacenamiento y búsqueda, haciendo referencia a los mapas de partículas utilizados por Susan Tolman y Paul Meakin (Tolman & Meakin 1989); sin embargo, no explican la forma en que implementan tales mapas, o la manera en que hacen uso del *quad-tree* para obtener los saltos.

El presente trabajo, explota la estructura del *quad-tree* no solamente al realizar la búsqueda exacta entre las partículas, sino también para determinar los saltos aproximados que dan las partículas cuando se encuentran a cierta distancia del resto del agregado. Al no hacer uso de mapas externos a la estructura, no hay necesidad de actualizar información adicional al hecho de agregar una partícula al *quad-tree*.

La forma detallada en que se realiza la búsqueda de una partícula al agregado, haciendo uso de las características del *quad-tree* tanto para realizar búsquedas exactas como aproximadas, se describe mediante el pseudo-código siguiente:

```
precondition: dist_minima=∞

recursive real function BuscaDistancia(nodo,dist_minima)
/*busca la distancia exacta o aproximada de un punto al agregado*/
value pointer node nodo
reference real dist_minima
pointer node hijo, real d
if not esNulo(nodo) then
   if dist(p,centro(nodo)) > diagonal(padre(nodo)) then
      /*si es el caso, devuelve una distancia aproximada*/
      if dist(p,centro(nodo) -diagonal(nodo)/2) < dist_minima then
         return dist(p,centro(nodo))- diagonal(nodo)/2
      endif
   else
      if  esHoja(nodo) then
         /*si se trata de una hoja, busca exactamente en todos los puntos*/
         d:=dist_minima
         for each punto in nodo do
            if dist(p,punto)< d then d:=dist(punto)
          enddo
          if d< dist_minima then return d endif
      else
         /*si no es hoja, determina hacia que dirección se encuentra el punto*/
         if estaHaciaNE(p,nodo) then
            hijo:=hijoNE(nodo)
         elseif estaHaciaNO (p,nodo) then
            hijo:=hijoNO(nodo)
         elseif estaHaciaSO (p,nodo) then
            hijo:=hijoSO(nodo)
         elseif estaHaciaSE (p,nodo) then
            hijo:=hijoSE(nodo)
         endif
        /*busca recursivamente en la estructura, el orden es importante*/
         dist_minima:=BuscaDistancia(hijo,dist_minima)
         if intersecta(circulo(p,dist_minima),vecinoIzq(hijo)) then
            dist_minima:=BuscaDistancia(vecinoIzq(hijo),dist_minima)
         endif
         if intersecta(circulo(p,dist_minima),vecinoDer(hijo)) then
            dist_minima:=BuscaDistacia(vecinoDer(hijo),dist_minima)
         endif
         if intersecta(circulo(p,dist_minima),vecinoOpuesto(hijo)) then
            dist_minima:=BuscaDistancia(vecinoOp(hijo),dist_minima)
         endif
       endif
    endif
endif
return  dist_minima
```

Nótese que se hace uso de la información implícita en la estructura en la que se guardan los datos: El hecho de que un nodo contenga un apuntador nulo, significa que el área correspondiente está libre de partículas.

Con el propósito de aclarar un poco más el pseudo-código anterior, se da a continuación una descripción de las tareas realizadas por las funciones utilizadas en el mismo:

esNulo(nodo)	Función booleana que determina si la referencia *nodo* es un apuntador nulo.
centro(nodo)	Función vectorial que devuelve el punto correspondiente al centro del espacio cubierto por la referencia *nodo* dentro del *quad-tree*.
diagonal(nodo)	Función real que devuelve la longitud de la diagonal que atraviesa el área del quad-tree correspondiente a la referencia *nodo*. Devuelve *infinito* si nodo es apuntador nulo
dist(punto1,punto2)	Función real que devuelve la distancia entre los puntos *punto1* y *punto2*.
padre(nodo)	Función que regresa una referencia al padre del nodo referenciada por *nodo* dentro de la jerarquía del *quad-tree*.
esHoja(nodo)	Función booleana que dice si la referencia *nodo* es un apuntador a una hoja dentro de la jerarquía del *quad-tree*.
estaHaciaNE(p,nodo) estaHaciaNO(p,nodo) estaHaciaSO(p,nodo) estaHaciaSE(p,nodo)	Funciones booleanas que indican si el punto *p* se encuentra hacia el noreste, noroeste, suroeste o sureste del centro del área en el *quad-tree* referenciada por *nodo*, respectivamente.
circulo(p, r)	Función que devuelve una referencia a la representación de un círculo con centro en el punto *p* y radio *r*.
intersecta(circ,nodo)	Función booleana que determina si el circulo referenciado por *circ* intersecta al área correspondiente a la referencia *nodo* en el *quad-tree*.
vecinoIzq(nodo) vecinoDer(nodo)	Funciones que devuelven, respectivamente, una referencia a los nodos izquierdo y derecho, dentro del mismo subcuadrante, del área del *quad-tree* referenciada por *nodo*.
vecinoOpuesto(nodo)	Función que devuelve, una referencia al nodo opuesto, dentro del mismo subcuadrante, del área del *quad-tree* referenciada por *nodo*.

La figura III.6 ejemplifica, en forma gráfica, los casos en los que se dan saltos exactos y aproximados, mostrando dos partículas en movimiento *a* y *b*. A la partícula *a,* se asigna una distancia de salto *r,* de acuerdo con la simple inspección de su distancia con el nodo cuyo centro está indicado por el punto *1*. La diagonal indica las distancias que se toman en el criterio de comparación, obsérvese que la diagonal atraviesa tres niveles de jerarquía. A la partícula *b*, debido a su cercanía con el resto de las partículas del agregado, se le asigna una distancia de salto igual a su distancia con el agregado. Las líneas muestran los límites de los nodos del *quad-tree*.

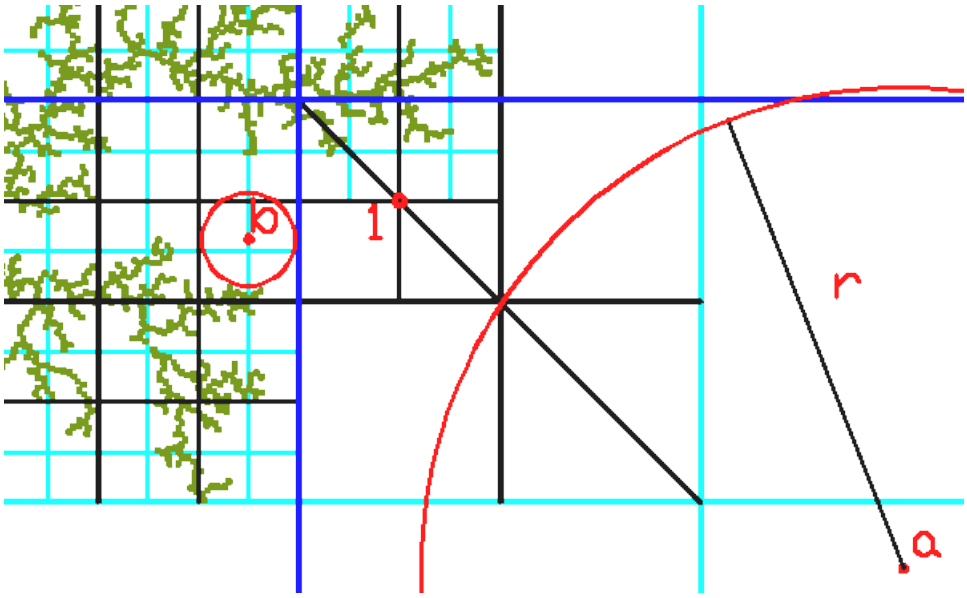


Figura III.6 Saltos aproximado (a) y exacto (b).

III.2.6 Determinación de Saltos a Distancias Mayores al *Radio Exterior*.

Uno de los pasos del algoritmo DLA, consiste en eliminar a la partículas que viajan más allá de una determinada distancia con respecto al origen, a dicha distancia se le llama "*radio de muerte*". Idealmente, las partículas eliminadas representan a aquellas que viajan hacia el infinito y cuya probabilidad de encontrarse con el agregado es nula. Si la distancia tomada para el *radio de muerte* es relativamente pequeña, se estarían eliminando también partículas que tienen cierta probabilidad de reencontrarse con el agregado, aunque, posiblemente, después de un gran número de saltos. Por otro lado, tomar una distancia para el radio de muerte demasiado grande, provoca que el tiempo de simulación sea excesivo.

Para solucionar este problema, en algunas publicaciones (Kauffman, et al. 1995; Menshutin & Schur 2006) se evita la eliminación de las partículas que sobrepasan el *radio de muerte*, el cual es ignorado. En su lugar, las partículas que van más allá de cierta distancia del llamado "*radio de nacimiento"*, son proyectadas de nuevo al mismo, mediante un mapeo de la recta real sobre un circulo unitario en el plano complejo, y utilizando las propiedades de frontera del modelo DLA.

Al poner en práctica el mecanismo mencionado, se observa, que la probabilidad con la que una partícula regresa al *radio de nacimiento,* r_n, depende de la distancia a dicho radio en la que se realice la proyección. Si tal distancia es muy pequeña con respecto a r_n, la partícula regresará en un punto casi idéntico a la posición en la que abandono la circunferencia descrita por r_n. Esto ocasiona el efecto de que las partículas que van más allá de la distancia r_n sean atraídas '*magnéticamente*' hacia el agregado, produciendo fractales más compactos. Por supuesto, el efecto tiende a disminuir conforme se realiza la proyección a distancias mayores de r_n.

Teniendo en cuenta lo anterior, en esta investigación se toma un punto de vista algo diferente para tratar con las partículas se alejan del agregado, considerando a aquellas que van más allá del "*radio exterior"*, tal radio se refiere a la distancia entre la semilla y la partícula más alejada de la misma.

En nuestro enfoque, a las partículas p que se alejan a una distancia mayor a d_1 del *radio exterior* r_e, se les asigna una longitud de salto l_s, que con seguridad se conoce pueden moverse sin interferir con el resto de la estructura, en particular, la longitud asignada es:

$$l_s = |p| - (r_e + d_2) \qquad \text{...(III.2)}$$

con $0 < d_2 < d_1$.

$|p|$ es la distancia a la que se encuentra la partícula del centro del agregado, considerado en este caso como el origen (0,0). De esta manera sabemos que en el círculo con centro en p y radio l_s se encuentra libre de partículas del agregado, lo que permite dar un salto libre en cualquier dirección, pero sin exceder la distancia l_s. La distancia d_1 puede ser de magnitud de unos cuantos diámetros de las partículas en movimiento. La distancia d_2 puede ser, por ejemplo $0.5d_1$. Cuando las partículas vuelven a estar a una distancia menor a $r_e + d_1$ del origen, entonces la distancias de saltos se calculan como se describió primeramente en secciones anteriores.

La figura III.7 muestra, a manera de esquema, un ejemplo de las magnitudes de las distancias involucradas en la ecuación III.2.

La distancia d_2 se utiliza para evitar una sucesión infinita de saltos que podría ser inducida por errores en las comparaciones en punto flotante de simple o doble precisión, por lo que la diferencia $d_1 - d_2$ puede considerarse como un margen de error para operaciones de comparación de radios. Si no consideramos este margen, la distancia de salto asignada podría ser cero en los casos en que la partícula se encuentre muy cerca de r_e, y en el cálculo de l_s se perdiera precisión debido a las operaciones de punto flotante.

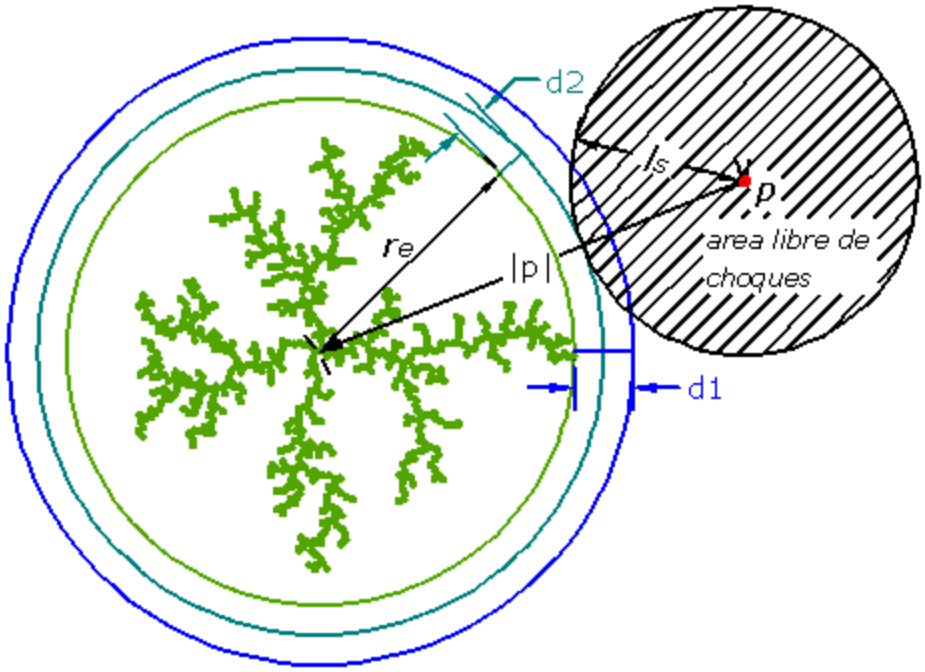


Figura III.7 Saltos a distancias mayores al *radio exterior*.

La aplicación de esta técnica reduce el tiempo de simulación del movimiento browniano para las partículas que se alejan de a una distancia mayor a $r_e + d_1$ del origen, sin comprometer la dimensión fractal de los objetos así generados, pues de esta manera, se puede tomar un *radio de muerte* significativamente alejado del *radio de nacimiento*.

Lo que se consigue con lo anterior es básicamente extender la simulación del movimiento browniano ignorando a las partículas que conforman el agregado, sabiendo que en los saltos a cierta distancia la partícula no chocará con el agregado; si existe la posibilidad de choque, entonces se buscará la máxima distancia que se puede saltar sin chocar, como ya se ha descrito anteriormente.

III.2.7 Generación de Números Pseudo-Aleatorios.

Al observar el diagrama de flujo de la figura III.2, nos damos cuenta de la necesidad de generar números aleatorios en el rango [0,2π) para la codificación del algoritmo DLA. Dado que las computadoras a nuestro alcance se basan en modelos de cálculo deterministas, proponemos utilizar generadores de números pseudo-aleatorios. La elección de un buen generador de tales números determina la exactitud de la simulación de nuestro modelo.

Una forma de obtener números pseudo-aleatorios que puedan considerase aceptables, es mediante llamadas a la función *drand48()*, que genera números pseudo-aleatorios en el intervalo [0,1) con distribución uniforme, basándose en el método de congruencia lineal y utilizando aritmética entera de 48 bits (*Free Software Foundation*, 1997). El valor así obtenido es escalado al intervalo [0,2π).

Como nuestro algoritmo se va a ejecutar en paralelo, es necesario tener en cuenta el siguiente detalle: Ninguna función que utilice variables estáticas, como es el caso de la función *drand48()*, se considera segura para su utilización en un programa multihilos (*thread safe*). (Butenhof, 1997).

Lo anterior es debido a que las variables estáticas deben conservar su valor para la subsiguiente llamada a la función. Si se tuviera que más de un hilo hace una llamada a una función que utiliza variables estáticas, esta llamada, y por tanto, la modificación de la variable estática, pasaría inadvertida por la lógica de ejecución de cualquiera de los otros hilos. Las funciones de la librería estándar como rand(), rand48(), drand() y drand48() hacen uso de variables estáticas para rastrear el valor actual de iteración en el método de congruencia lineal (*Free Software Foundation*, 1997).

Para evitar tal situación, el compilador obliga a que solamente un hilo a la vez pueda acceder a una función que utilice variables estáticas. Como nuestro programa hace un uso intensivo de la generación de números aleatorios, manejando funciones con variables estáticas, el rendimiento de la ejecución del código en paralelo podría llegar a ser igual, o menor, que el rendimiento utilizando un solo procesador, debido a que el compilador forzaría a la modificación de la variable estática por un solo procesador a la vez, bloqueando al resto de los procesadores.

Muchas librerías que consideran el uso del multiprocesamiento, suministran versiones re-entrantes (*reentrant*) para las funciones que manejan variables estáticas. En el caso de *drand48()*, la versión re-entrante es *drand48_r()*. En esta última, la variable estática correspondiente a la semilla del generador, es sustituida por un apuntador al valor inicial que se tomará como semilla, y que será modificado en las llamadas subsecuentes a la función. De esta forma, la versión re-entrante,

drand48_r(), puede ser ejecutada simultáneamente por varios hilos sin pérdida de paralelismo.

Para los modelos con color, la generación aleatoria del color de la partícula en movimiento se realiza con llamadas a la función *lrand48_r()*, que devuelve un número entero aleatorio entre 0 y 2^{31}.

III.2.8 Manejo de Modelos DLA con Dos y Tres Colores.

La meta principal de esta tesis, que es, como ya se ha mencionado, la elaboración de software que facilite la creación y análisis de agregados de los modelos multicolor utilizando procesamiento paralelo, se consiguió llevando a cabo la codificación del modelo DLA manejando partículas a las que se les asocia una característica denominada color, que permite la aplicación de las reglas de interacción estándar (Rodríguez Romo, *et al*, 2004).

La simulación del modelo multicolor es inherentemente más lenta, debido a que, por ejemplo, en el caso del modelo bicolor, una partícula tiene la misma probabilidad de adherirse a un agregado del mismo color, que de ser destruida al encontrarse con partículas de un agregado de color diferente, es decir, aproximadamente la mitad de las partículas son destruidas después de haber realizado los cálculos correspondientes a sus trayectorias, lo cual consume tiempo de proceso. Para el caso del modelo tricolor, la probabilidad de que una partícula sea destruida es mayor que la probabilidad de que pase a formar parte de un agregado. Las probabilidades anteriores varían según la evolución que presenten los agregados (Rodríguez-Romo, *et al*. 2004).

A las partículas que conforman el agregado, se les puede identificar agregando un *byte* adicional a la estructura que las define. Con este enfoque, las partículas de diferentes colores pueden estar dentro de una misma estructura de almacenamiento. De esta forma se logran búsquedas muy rápidas en la determinación del vecino más cercano, equivalentes en velocidad al modelo de un solo color. Una desventaja que puede presentar este esquema, es la cantidad de memoria adicional requerida. Aun cuando el identificador de color consume solamente un *byte*, la cantidad extra de memoria para un agregado de 10^8 partículas es aproximadamente de 100 MB. Si se deseará escalar la cantidad de partículas solamente a un orden de magnitud mayor, la cantidad de memoria extra requerida sería de 1GB aproximadamente.

Si no se quiere agregar un identificador de color sobre cada partícula, se puede en su lugar, crear varias estructuras para almacenarlas, una correspondiente a cada color. Esto evita el requerimiento de memoria adicional proporcional al número de partículas. Sin embargo el efecto colateral que aparece es un aumento en el tiempo de simulación proporcional al número de colores que se estén manejando, pues las búsquedas se tienen que hacer por separado sobre cada estructura, para

determinar la distancia de una sola partícula a los agregados. A pesar de que el número de puntos contenido en cada una de las estructuras es menor, esto no ayuda mucho al desempeño, pues no se saca provecho enteramente de las sub-búsquedas a través de la estructura de almacenamiento. La búsqueda completa tiene que comenzar de nuevo desde la raíz del *quad-tree* respectivo. Ambas técnicas mencionadas se codificaron para realizar los experimentos de esta investigación.

Es importante seleccionar cual de las dos técnicas anteriores se seguirá de acuerdo al tamaño de los agregados que se pretenda generar. En las corridas del algoritmo ejecutadas para la obtención de los datos presentados en el siguiente capítulo, se utilizaron dos o tres estructuras *quad-tree* según se manejaran dos o tres colores, puesto que se cuenta con suficiente memoria en el hardware utilizado para alcanzar un numero de partículas del orden de 10^8.

Se prefirió en estos casos hacer un uso eficiente de la memoria debido a que cada partícula se etiquetó con un entero adicional de 4 *bytes*, indicando en esta forma el orden en que las partículas fueron añadidas al agregado. La mayor pérdida en la velocidad de simulación no fue inducida por esta decisión, sino por la complejidad inherente al modelo multicolor. Cabe mencionar que en nuestros experimentos se observó un aumento en el tiempo de ejecución directamente proporcional al número de colores manejados, como se muestra en la sección IV.I.2. por tanto, tenemos que la complejidad computacional no se ve afectada por la introducción de colores en el modelo.

III.3 Algoritmo Paralelo.

Ya que se cuenta con el algoritmo secuencial que cumple con las expectativas requeridas para los propósitos establecidos, como se muestra en la sección IV.I.I, se procede a la paralelización del mismo. En este trabajo consideramos imprescindible el empleo de técnicas de multiprocesamiento debido a la necesidad de realizar cálculos intensivos para la simulación de una cantidad conveniente de fractales DLA, de forma tal que se puedan extraer datos confiables que describan sus características. La reciente introducción en el mercado masivo de computadoras con tecnología "*multicore*", nos lleva a buscar soluciones que aprovechen al máximo estas tecnologías que actualmente tenemos de una u otra forma a nuestro alcance. El esquema de paralelización aquí utilizado se basa en el modelo PDLA (*Parallel Diffusion Limited Aggregation*) (Kauffman, *et al*. 1995) tomando ciertas medidas para evitar el efecto que tiene el número de procesadores utilizados sobre la dimensión fractal.

Los fractales construidos con el algoritmo paralelo se comparan con los resultados obtenidos por diversos investigadores con anterioridad (ver siguiente sección). Esto nos da la seguridad de afirmar que los resultados obtenidos para los modelos multicolor son también válidos, pues al momento de escribir esta tesis, no existe literatura publicada que muestre resultados sobre la paralelización de los modelos multicolor, o inclusive, aunque se tiene determinado el comportamiento de estos (Rodríguez Romo, *et al*, 2005), no se han publicado experimentos con grandes números de partículas para los modelos multicolor.

III.3.1 Esquema de Paralelización.

Una manera efectiva de conseguir agregados de tamaño considerable mediante el uso de múltiples procesadores, es el modelo PDLA (Kauffman, *et al*. 1995). Como sus autores lo indican, este procedimiento no es equivalente al modelo DLA. Estrictamente hablando, el modelo PDLA es equivalente al modelo de difusión de multi-partículas MPDA (R. F. Voss, 1993) que genera agregados fractales más compactos. No obstante, se tiene que, para valores muy grandes del número de partículas que conforman el agregado, el modelo PDLA tiende a converger al modelo DLA.

En esta tesis tomaremos el modelo PDLA para generar aproximaciones al DLA con múltiples procesadores pues este modelo es con el que hasta la fecha se han conseguido DLA's con un mayor número de partículas; los investigadores Kauffman y Maldelbrot, reportan haber creado con este método, 20 agregados DLA de 100,000,000 de partículas con un tiempo de ejecución promedio de 13 horas por agregado, utilizando 32 procesadores trabajando en paralelo (Kauffman, *et al*. 1995).
Con tecnología de hardware más reciente, los autores Menshutin y Schur (Menshutin & Schur, 2006) informan haber generado 100 agregados DLA con $5*10^7$ partículas, sin reportar el tiempo de procesamiento; tampoco mencionan haber generado agregados del orden de 10^8.

Empíricamente se ha encontrado que, haciendo uso de 32 procesadores, el modelo PDLA converge al DLA cuando se alcanza un número de partículas del orden de 10^6. Para 8 procesadores, que es el número máximo de procesadores usado en los experimentos aquí descritos, el orden de convergencia es de 10^5.

Se ha publicado un algoritmo paralelo que corresponde exactamente al modelo DLA. Los autores K. Moriarty J. Machta y R. Greenlaw, describen una versión del proceso DLA que hace uso de el modelo PRAM –*Parallel Random Access Machine*– (Moriarty *et al*, 1997). Desafortunadamente, tal procedimiento sirve más bien para propósitos teóricos, pues en la práctica tiene un desempeño muy pobre. Entre los resultados teóricos encontrados en dicha publicación, se determina una alta dependencia histórica en el proceso de crecimiento de un agregado DLA, tal dependencia limita la paralelización eficiente del mismo.

Una forma de lograr una gran similitud del modelo PDLA con el DLA, es generar las primeras *k* partículas utilizando un único procesador y, a partir de entonces, empezar a utilizar el resto de los procesadores.

Con la metodología aquí descrita, es posible generar núcleos de DLA con un solo procesador de hasta 10^7 partículas en menos de 15 minutos. Lo anterior nos da un margen dos órdenes de magnitud arriba del punto de transición del modelo PDLA hacia el modelo DLA para 8 procesadores (Kauffman, et al, 1995).

En nuestro proyecto de paralelización, después de la aplicación respectiva del modelo lineal, cada procesador maneja una partícula en movimiento, que llegado su momento, es agregada a una estructura global, compartida por todos los procesadores.

III.3.2 Paralelización con *POSIX Threads*.

Con el fin de exponer las herramientas de software utilizada en nuestro proceso de paralelización, en esta sección se describen las funciones que fueron utilizadas para incorporar multiprocesamiento en nuestros algoritmos.

Las interfaces de programación de aplicaciones (API's) definidas en el estándar internacional *POSIX* 1003.1c-1995 definen aspectos de uso de características multihilos en plataformas compatibles con el estándar. Por tanto, en *Linux, AIX, Unix* y otros sistemas compatibles, se cuenta con tal interfase para la creación de programas multihilos. Si un programa multihilos se ejecuta sobre una plataforma con múltiples procesadores, es posible sacar provecho del paralelismo para procesamiento simultáneo o asíncrono de varias tareas (Butenhof, 1997).

Para codificar una primera versión paralelizada del proceso DLA, se utiliza la librería de creación de hilos del estándar *pthreads*, con el fin de que la ejecución paralela comparta memoria y múltiples procesadores puedan tener acceso a toda la estructura del *quad-tree* al mismo tiempo.

Los hilos en *pthreads* son creados mediante una llamada a la función:

```
int ptread_create(pthread_t *hilo, const ptread_attr_t *atributos,
                  void *(*inicio)(void *), void *arg);
```

donde *hilo* es un apuntador al identificador del hilo creado, este identificador pertenece al tipo de datos opaco *pthread_t*, *atributos* es un apuntador a la estructura que contiene los atributos del hilo en el momento de su creación, *inicio* es la dirección de la función inicial que ejecutará el hilo y *arg* es un apuntador a los argumentos que serán pasados a la función inicial.

La dirección de la función que se pasa a cada hilo a través de la variable *inicio*, es la que realiza la simulación del movimiento browniano. El parámetro que se pasa mediante la variable arg, es un apuntador a la estructura que contiene, además de una referencia al nodo raíz del *quad-tree*, datos adicionales respecto a la simulación del DLA, como son número de partículas, radio máximo, radio de muerte, radio de nacimiento, entre otros. El apuntador que se pasa es el mismo para todos los hilos, que de esta forma comparten la estructura de información.

En nuestro código, se crea un hilo nuevo por cada procesador detectado en el sistema. En *Linux*, la detección de los procesadores puede hacerse con una llamada a la función:

```
long int sysconf(int nombre);
```

que devuelve el estado de la configuración del sistema correspondiente a la variable de entorno identificada por *nombre*. El nombre correspondiente al número de procesadores es _SC_NPROCESSORS_CONF.

Una vez que el hilo principal definido por la función *main()* ha creado los hilos para cada procesador, tiene que esperar a la terminación de éstos antes de comenzar a evaluar datos estadísticos, que han sido en su mayor parte, precalculados por los hilos creados. La forma de esperar a la terminación de un hilo en *pthreads* es llamando a la función:

```
int pthread_join(pthread_t hilo, void **ap_valor);
```

donde *hilo* es el hilo del cual se espera su terminación y *valor* es una referencia a un apuntador con el valor de terminación del hilo.

Para evitar colisiones de acceso a datos entre los procesadores, se utilizan mutexes para delimitar las secciones críticas en las cuales más de un procesador pudiera estar escribiendo en una misma localidad de memoria al mismo tiempo, generando de otra manera datos inconsistentes.

Una situación que potencialmente propicia condiciones de carrera (*race conditions*) en el algoritmo, ocurre cuando más de un procesador intenta agregar una partícula al cluster al mismo tiempo. Esto lleva a la modificación estructura de datos en forma simultánea, que dependiendo del área del *quad-tree* en la que escriba cada procesador, puede no tener consecuencia alguna, o bien provocar la sobre-escritura de la sección la estructura que contiene a una partícula, en cuyo caso, por el hecho de tratarse de memoria dinámica, crear una pequeña fuga de memoria, con la consecuente pérdida de la o las partículas que se sobrescriban. Otra consecuencia posible es que las partículas queden parcial o totalmente traslapadas, es decir, se tendrían dos o más partículas ocupando el mismo lugar en el espacio al mismo tiempo, con lo que nuestro agregado contendría datos incorrectos. También se corre el riesgo de dejar a la estructura en un estado inconsistente, que podría llegar

a la detención por completo de la ejecución del programa. Un ejemplo de cómo puede llegar a ocurrir este último caso puede encontrarse en el libro de J. Beveridge (Beveridge, 1997).

La solución al problema anterior consiste en agregar un mutex que se bloquea al comenzar la operación de escritura sobre la estructura y se desbloquea al terminar dicha operación. En el algoritmo DLA que utiliza un *quad-tree* como estructura de en la versión *bucket*, la operación de escritura va más allá de la simple asignación de un valor a una variable, pues puede implicar la modificación tanto del árbol, como del *bucket* que contiene a las partículas. Por supuesto, con esto se pierde un poco de paralelización, pero el porcentaje de código ejecutándose en paralelo se mantiene dentro de un nivel aceptable, como se muestra en la sección IV.I.3.

Los mutex son creados definiendo variables del tipo opaco pthread_mutex_t. Antes de poder utilizar un mutex, es necesario inicializarlo. Una forma de conseguir esta inicialización es llamando a la función:

```
int pthread_mutex_init(pthread_mutex_t *mutex,
                       pthread_mutexattr *atributos);
```

donde *mutex* es un apuntador a la variable del tipo *pthread_mutex_t* definida, y *atributos* es un apuntador a la estructura que especifica los atributos del mutex en el momento de su inicialización.

Las instrucciones que bloquean y desbloquean a un mutex son las siguientes:

```
int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Cuando un mutex ya no se necesita más, debe ser destruido utilizando la instrucción:

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Un hilo detendrá su ejecución si trata de bloquear un mutex que ya ha sido previamente bloqueado, el desbloqueo del mutex por parte del hilo propietario activa la verificación de bloqueo para todos los demás hilos que están esperando por el mutex, estos intentarán bloquearlo nuevamente pero solamente uno lo conseguirá, continuando de esta manera su ejecución. Una vez terminada la ejecución de la sección crítica protegida por el mutex, el hilo propietario tiene la obligación de desbloquearlo.

En la codificación de nuestro algoritmo en paralelo, se utilizan tres mutex que abarcan diferentes secciones iterativas de la operación de escritura, con esto reducen los tiempos de espera entre bloqueos, manteniendo la exclusividad de la

operación. Para la sección más crítica del código, es necesario un chequeo redundante de las partículas cercanas. Esto es con el fin de evitar errores en la posición de las partículas sin aumentar excesivamente los tiempos de bloqueo. Lo anterior no implica una condición de carrera, pues el chequeo se realiza en la parte de código protegida por el mutex. La verificación consiste en un simple recorrido en profundidad sobre el *quad-tree*.

El estándar *POSIX* define también los candados de lectura-escritura. La simulación de un agregado DLA requiere un acceso de lectura-escritura intensivo por parte de todos los procesadores sobre la estructura compartida. Cuando se usan sub-búsquedas aproximadas se da una tendencia hacia las operaciones de escritura no muy significativa.

Lo anterior lleva a que, en el caso particular de nuestros experimentos, el uso de candados lectura-escritura resulte ineficaz, pues, como se comprobó experimentalmente en nuestras primeras implementaciones paralelas del algoritmo DLA, se propicia que los procesadores permanezcan bloqueados durante una buena parte del tiempo de proceso, en el que constantemente ejecutan tanto instrucciones de lectura como de escritura sobre la estructura de datos compartida, con una mayor recurrencia sobre las operaciones de lectura, pero no lo suficiente como para justificar el uso del candado lectura-escritura. Esperar a la terminación de todas las lecturas, antes de comenzar una escritura, no presenta, en nuestro caso, una gran ventaja sobre la seguridad con la que se realizan las operaciones, esto debido a que previamente se toman precauciones en la forma en la que se realiza la operación de escritura, evitando dejar a la misma en un estado incoherente en caso de que la lectura por parte de un procesador ocurra durante una etapa intermedia del proceso de escritura por parte de otro procesador.

Otro aspecto importante que se permite manipular con las librerías *pthreads* es la utilización de los recursos a nivel sistema operativo.

En *Linux*, los hilos llevan una calendarización (*scheduling*) con el esquema definido por el estándar *POSIX SCHED_FIFO* (*First In First Out*), o bien *SCHED_RR* (*Round Robin*). En el primer esquema, se permite la ejecución de los hilos por tiempo indefinido, hasta que estos terminen su trabajo, o cedan voluntariamente el control del procesador, ya sea por una instrucción específica o mediante la espera ocasionada por algún bloqueo, los hilos con este esquema no pueden ser interrumpidos externamente, ni aún por hilos de mayor prioridad. En el segundo esquema, a cada hilo se le asigna cierta cantidad de tiempo para su ejecución, después de la cual, el hilo es inevitablemente interrumpido por el sistema operativo, que decidirá el momento de ejecutar el hilo nuevamente, considerando todas las demás tareas a realizar, con sus respectivas prioridades.

La utilización de *SCHED_FIFO* dificulta la concurrencia, pero permite el uso óptimo del tiempo de procesamiento.

Las librerías *pthreads*, admiten también la especificación del nivel de contención de dominio, que puede ser *PTHREAD_SCOPE_SYSTEM*, en el cual los hilos compiten por los recursos a nivel sistema, o bien *PTHREAD_SCOPE_PROCESS*, en el cual los hilos solamente compiten por los recursos dentro del mismo proceso, este último esquema existe para proporcionar concurrencia. El nivel de contención que permite un mejor uso del multiprocesamiento es *PTHREAD_SCOPE_SYSTEM*.

Una forma de establecer los esquemas mencionados en *pthreads* es mediante:

```
int pthread_setschedparam (pthread_t hilo, int política,
                           const struct sched_param *parámetro);
```

donde *hilo* es el hilo al cual se le aplica el esquema, *política* indica la opción del esquema a seguir y *parámetro* es un apuntador a la estructura que contiene información del esquema.

Para que los cambios de calendarización tengan efecto sobre un hilo, es necesario ejecutar previamente la siguiente instrucción:

```
pthread_attr_setinheritsched(&atribHilos ,PTHREAD_EXPLICIT_SCHED);
```

que indica que la calendarización no será heredada del hilo creador, sino que será definida explícitamente, la variable *atribHilos* guardará tal indicación.

Las extensiones NPTL(*Native POSIX Threads Library*) de *Linux* permiten manipular la afinidad de los hilos o procesos sobre un procesador determinado. En el mapeo efectuado para de paralelización del algoritmo DLA aquí presentado, no es indispensable cumplir con algún requisito de afinidad, sin embargo, por cuestiones de eficiencia en el proceso de calendarización, la afinidad es deseable.

La afinidad con las extensiones NPTL se codifica mediante la llamada a la función:

```
pthread_setaffinity_np(pthread_t hilo,size_t tamaño, cpu_set_t *máscara);
```

donde *hilo* nuevamente indica el hilo sobre el cual se realiza la operación, *tamaño* es el tamaño de la máscara binaria que representa el conjunto de procesadores sobre el cual se restringe la ejecución del hilo y *máscara* es un apuntador a dicha máscara.

La importancia del uso adecuado de recursos se puede apreciar en la figura III.8. En el primer recuadro, observamos como una calendarización inapropiada y tiempos de bloqueo excesivos resultan en un mal aprovechamiento de un sistema de multiproceso. Mientras que, en el segundo recuadro, se logra la utilización de los procesadores a su máxima capacidad; el porcentaje de uso del último procesador (en rojo), en este caso, corresponde a la interfaz de usuario del visor de rendimiento que permite la toma de la instantánea mostrada.

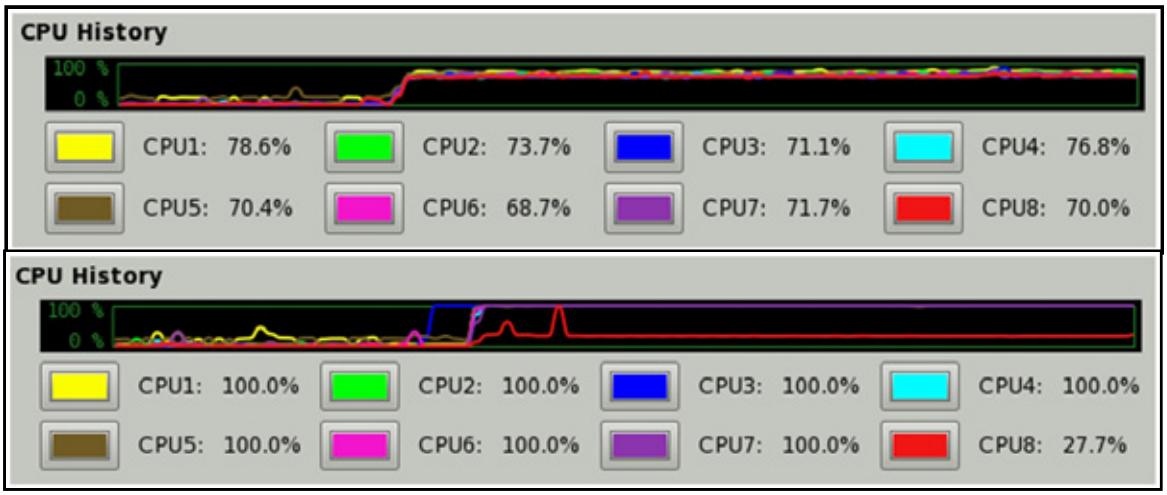


Figura III.8 Utilización de recursos en paralelo.

El mismo esquema de paralelización se sigue tanto para los agregados de un solo color, como para las versiones multicolor del modelo.

Para la codificación de los modelos bicolor y tricolor, el número de mutexes utilizados aumenta, en el caso de usar estructuras diferentes para cada color, a seis y nueve respectivamente. Esto favorece a la paralelización, pues los bloqueos ocurren con menos frecuencia.

III.3.3 Paralelización con *OpenMP*.

Otra forma en que se consiguió el uso efectivo de multiprocesamiento para la simulación del fenómeno DLA, fue con la utilización de la biblioteca *OpenMP*. El estándar de programación *OpenMP* facilita la programación multihilo utilizando la estrategia maestro-esclavo, en donde un solo hilo principal lleva el control sobre los demás hilos que se van creando (Dagum, 1997).

La forma en que se aplican las instrucciones de *OpenMP* sobre los lenguajes *C* y *C++* es a través de directivas de compilación *#pragma*. Las directivas de *OpenMP* se emplean sobre bloques estructurados que tengan sólo un punto de entrada y solamente un punto de salida.

De manera simultánea al código basado en *pthreads*, se elaboró un código de paralelización para el algoritmo DLA utilizando las librerías *OpenMP*. La versión utilizada fue *GNU OpenMP*, identificada por la abreviatura *GOMP*. Esta librería es parte de algunas distribuciones de *Linux* como son *Red Hat* y *Fedora*, por lo que no fue necesario realizar ninguna instalación o configuración especial sobre la plataforma utilizada.

El código que utiliza *OpenMP* se generó a partir del código que hace uso de *pthreads*. Para esto, se sustituyó el código de creación de hilos y paso de parámetros hacia los mismos por una simple indicación de la directiva *#pragma omp parallel* de *OpenMP* en el momento de llamar a la función que simula el movimiento browniano de las partículas en torno a una semilla.

Los mutex fueron sustituidos por directivas *#pragma omp crítical* con una etiqueta correspondiente al mutex que se sustituye. Estas directivas se indicaron en las secciones de código comprendidos entre el bloqueo y la liberación de un mutex.

El número *n* de hilos de ejecución paralela se estableció mediante la llamada a la función cuyo prototipo es el siguiente:

```
int omp_set_num_threads(int n);
```

y se encuentra declarada en el archivo *omp.h*.

El código para *OpenMP* no presentó gran diferencia en cuanto a velocidad de ejecución o la eficiencia en el uso de procesadores con respecto al código de *pthreads*. La ejecución del código con librerías *OpenMP* es entre un 7% y un 10% más lenta que el código con *pthreads* cuando se utiliza la calendarización *SCHED_FIFO*. Estos valores se obtuvieron con 10 ejecuciones del algoritmo DLA con 8 procesdores generando agregados de 10^7 partículas, para cada uno de los casos de paralelización usando *pthreads* con el esquema *SCHED_FIFO, SCHED_RR* y *OpenMP*. La calendarización proporcionada por *OpenMP* es comparable con el esquema *SCHED_RR* de *pthreads*, en el cual no se pierde la respuesta instantánea de la interfaz de usuario cuando se hace uso del total de los procesadores disponibles, y en el que no obstante, un proceso puede ser interrumpido o desplazado por otro proceso con mayor prioridad.

La diferencia principal encontrada entre ambos métodos, consiste en que la versión de *OpenMP* requiere mucho menos líneas de codificación, pues no es necesario crear, inicializar, calendarizar, mapear, o definir propiedades de los hilos de manera explicita. A su vez, el hecho de que el código de *pthreads* requiera de tales instrucciones, permite mayor flexibilidad en el control del multiprocesamiento.

Al momento de ejecutar las corridas de prueba y obtención de datos para el algoritmo DLA, que se presentan el capítulo siguiente, se disponía de momento de todos los recursos del servidor para la ejecución con multiprocesamiento, por lo que se utilizó la versión de *pthreads* con el esquema *SCHED_FIFO*, consiguiendo una velocidad de ejecución ligeramente mayor al de las otras versiones con *OpenMP* o *SCHED_RR*.

III.3.4 El Modelo DLA y MPI.

El objetivo de esta sección es explicar el porqué se tomó para la presente investigación un esquema de tecnología *multicore*, en lugar de usar un esquema con base a la división de dominios con memoria distribuida.

Un recurso de paralelización de algoritmos muy recurrido es la comunicación interproceso. Una forma habitual de implementar esta comunicación es mediante el uso de MPI. Las librerías MPI (*Message Passing Interface*), constituyen un estándar bastante aceptado para la paralelización de algoritmos (U. of Tennessee, 2003). MPI utiliza un mecanismo de intercambio de mensajes para la comunicación entre procesos. Tales procesos pueden ejecutarse en paralelo sobre sistemas de multiprocesamiento simétrico, o bien sobre sistemas levemente acoplados no heterogéneos, pues MPI no es tolerante a fallos. Una aplicación ideal para MPI son los sistemas de multiprocesamiento masivo.

En una etapa inicial de esta investigación, se propuso utilizar las librerías de paralelización MPI. Lo anterior se desarrolló mediante una serie de procesos productores-consumidores, que determinaban las distancias de las partículas al agregado para ciertas secciones. Dichas secciones estaban separadas mediante círculos concéntricos de radios progresivamente mayores. Esto nos permitía una división de dominio que reducía proporcionalmente la cantidad *n* de partículas a manejar. No obstante, se encontró que la notificación entre procesos era demasiado lenta con respecto a una operación lectura-escritura sobre memoria dinámica, y la comunicación global entre los mismos, muy demandante debido al salto de las partículas de una sección a otra. Dado lo anterior, se prefirió conducir la investigación hacia las técnicas que proporcionan uso compartido de memoria, evitando una comunicación excesiva entre procesos. De aquí se descartó la utilización de MPI en nuestro código paralelizado.

III.4 Visualización de los Agregados.

Aunque la graficación de los agregados no es el objetivo principal de esta tesis, también se desarrolló código de visualización para obtener una representación que permita observar las características cualitativas del fenómeno simulado. Mediante la visualización del fractal DLA podemos darnos una idea de las características cualitativas fenómeno simulado. Con la finalidad de poder visualizar los agregados en una forma dinámica, que permita el manejo de vistas a escala en tiempo real de secciones del agregado, se transforman los datos generados al formato DXF. El formato DXF (*Drawing Format Interchange*) fue introducido originalmente por la empresa *Autodesk*, con la finalidad de contar con un formato que permitiera el intercambio de archivos entre diferentes aplicaciones de diseño gráfico (*Autodesk Inc*, 2000). Con el tiempo, este formato se convirtió en un estándar para la industria.

Para lograr la compatibilidad con el formato DXF, se requiere de la especificación de una serie de códigos en un archivo binario. Tales códigos corresponden a las propiedades de las entidades gráficas elementales que conforman un dibujo. Adicionalmente, hay que agregar un '*sentinela*' al archivo binario, de forma que alguna aplicación que sea compatible con el formato pueda reconocer el archivo.
Los valores de las propiedades se establecen como enteros o como números de punto flotante de doble precisión representados en forma binaria. Existe también la opción de especificar los códigos en formato ASCII, pero el espacio de disco ocupado y el tiempo de carga son mucho mayores.

Al usar este formato, no es necesario contar con una aplicación comercial para crear un diseño gráfico del DLA, pues el agregado es '*dibujado*' mediante el código en lenguaje *C* desarrollado en este trabajo. Tal procedimiento de dibujo consiste, principalmente, en la conversión de los datos numéricos al formato gráfico DXF. El agregado transformado de esta manera, puede observarse usando alguno de los visualizadores que pueden descargarse gratuitamente desde *Internet*, como son *DWGTrueView, Qcad, FreeDWFviewer*, entre otros.

La figura III.9 muestra el uso del *DWGTrueView* para visualizar un agregado DLA con opciones en tiempo real, como son: *Zoom in, zoom out, pan, regen, select box*, *layers*, *coordinates*, entre otras.

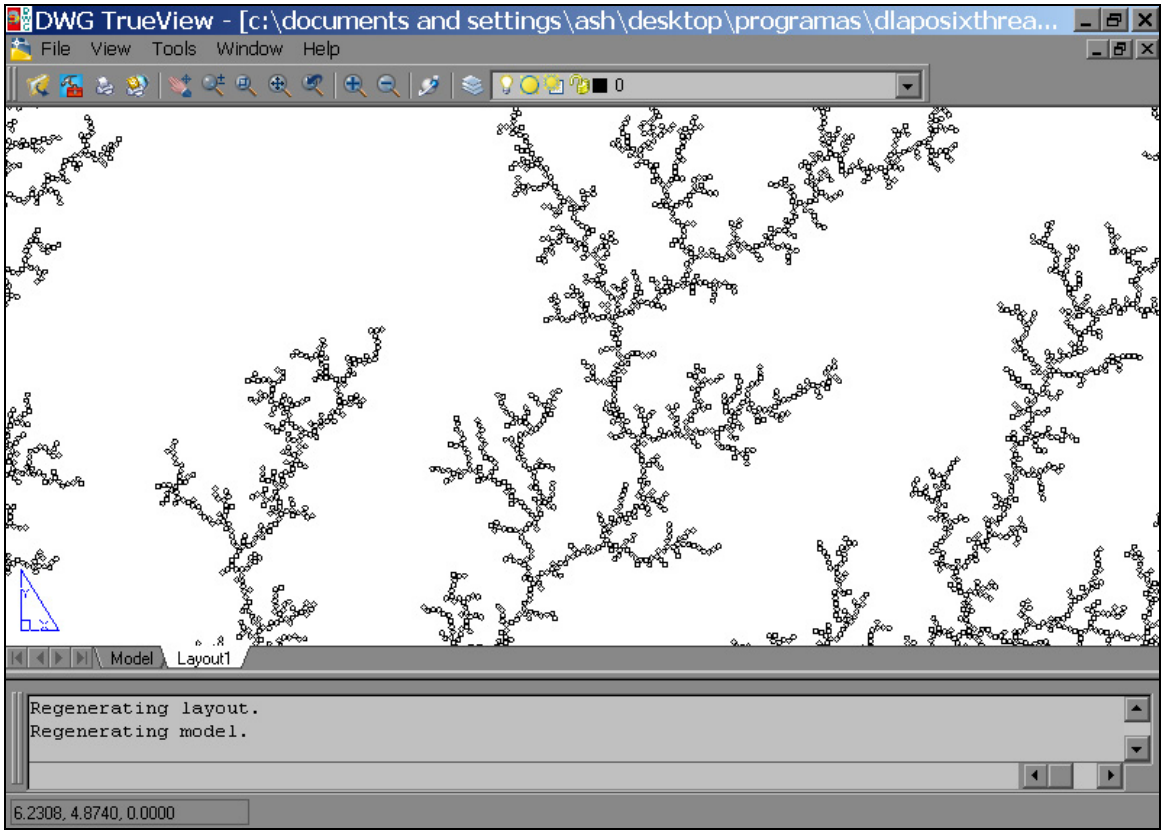


Figura III.9 Visualización con formato DXF.

La técnica de visualización con el formato DXF nos permite el manejo de cantidades hasta de n=2,500,000 partículas en tiempo real.

Para la visualización de agregados fractales con un mayor número de partículas, digamos, del orden de 10^8, lo que se hace es una proyección del fractal sobre el área correspondiente a una imagen de alta resolución, por ejemplo, de 8000x8000 píxeles (64Mpx). La figura III.10 muestra un agregado fractal DLA con n=100,000,000 de partículas, generado con las técnicas aquí descritas. La proyección se guardó en un archivo de imagen con formato *JPEG*. El pequeño recuadro muestra una comparativa, a la misma escala, con respecto al fractal de n=250,000 partículas presentado anteriormente en la figura II.2.

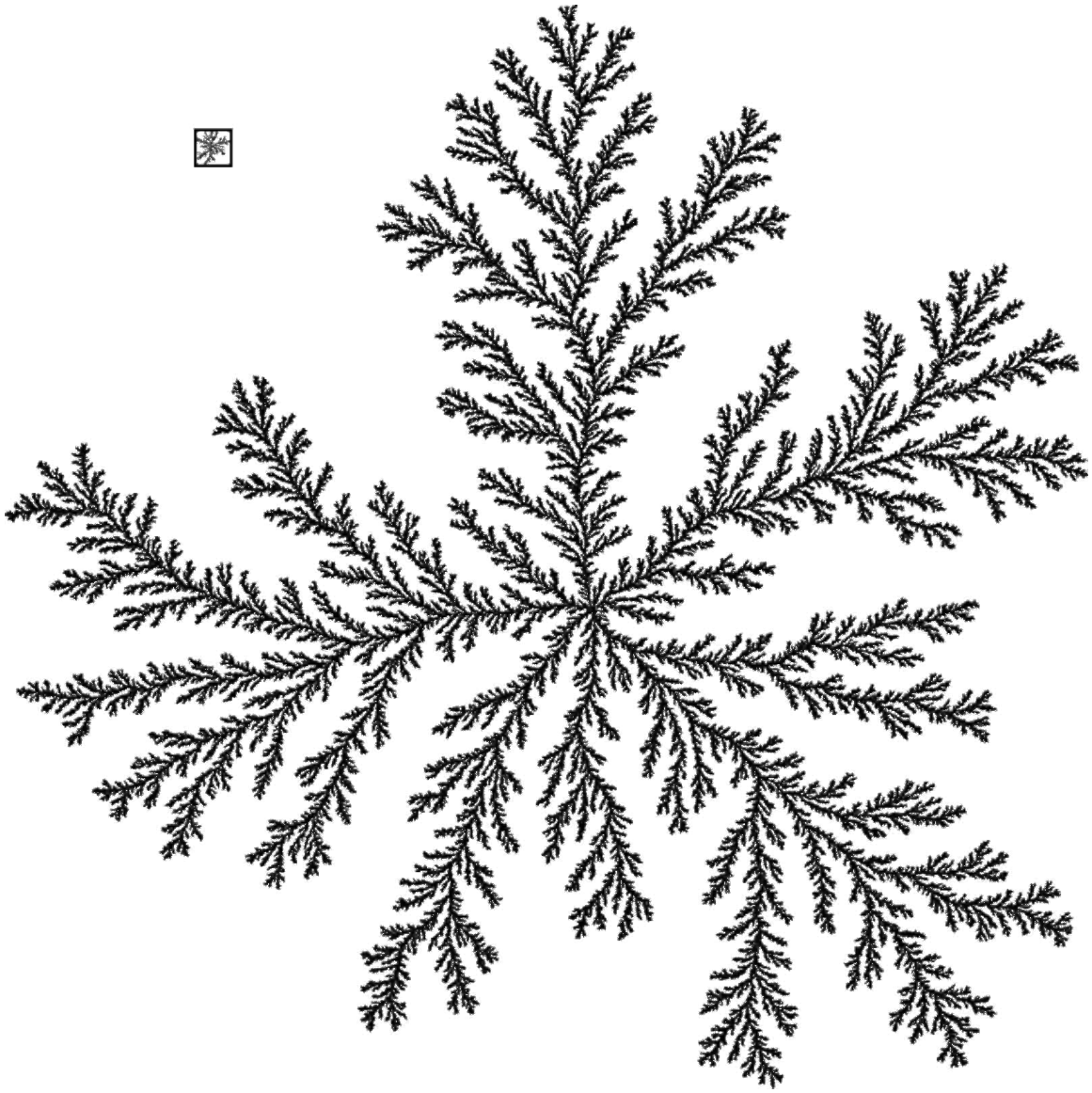


Figura III.10 DLA con n=100,000,000 de partículas.

La proyección mostrada anteriormente se llevó a cabo mediante código programado en lenguaje *C*, que hace uso de las librerías gráficas *OpenCV* (*Intel Corp*., 2001) las cuales permiten el manejo de formatos gráficos con una gran facilidad.

Por supuesto, al proyectar los puntos sobre un área de cobertura menor, se tiene una pérdida sustancial de información, pero este procedimiento se utiliza solamente con fines de visualización. Los cálculos para análisis estadísticos se siguen realizando sobre los datos completos en formato binario.

El archivo en formato binario consume, por cada partícula, 8 *bytes* para la coordenada *x*, 8 *bytes* para la coordenada *y*, y 4 *bytes* para indicar el orden de agregación de las partículas. Ocupando un espacio en disco de 1.86 GB para un fractal con n=100,000,000 de partículas. En memoria RAM, el espacio ocupado fue mayor, pues se agregan 4 *bytes* para el apuntador al siguiente elemento dentro del *bucket* delimitado por una hoja del *quad-tree*; además, el mismo *quad-tree* ocupa 0.8 GB de memoria consumida por los nodos que no son hojas y sus respectivos apuntadores. Es decir, se requirieron 3.2 GB de memoria RAM para crear el fractal mostrado en la figura III.10 utilizando coordenadas de punto flotante con doble precisión. Para la representación de coordenadas utilizando un almacenamiento de punto flotante de precisión simple, la memoria RAM requerida fue de 2.4 GB. De aquí vemos el porqué se necesita la conversión a un formato comprimido como lo es *JPEG*.

La imagen obtenida de 64Mpx fue, como es de esperarse, nuevamente escalada para quedar dentro del formato manejado por el editor de textos con el que se elaboró la presente tesis.

Las rutinas de visualización se compilaron en un archivo ejecutable diferente al del programa que genera los agregados, En particular, el código que proyecta el DLA hacia una imagen *JPEG*, al no requerir gran capacidad de cálculo, ni de memoria para su ejecución, puede procesar los datos generados por la simulación trabajando en una computadora personal.

Con lo anterior se concluye la descripción del hardware y las técnicas utilizadas para crear el software creado para elaborar los análisis de interacción de agregados obtenidos mediante la difusión limitada. Como se ha expuesto, la técnica que presenta mayores ventajas para nuestra aplicación es el procesamiento paralelo con memoria compartida, implementada mediante las librerías *pthreads*, haciendo uso del modelo PDLA. En el siguiente capítulo se muestran los resultados obtenidos con la aplicación de este software.

IV. ANALISIS DE RESULTADOS.

IV.1 Comportamiento de la Ejecución de los Algoritmos.

Con la finalidad de determinar si los programas codificados sirven para los propósitos planteados, se procedió a realizar pruebas sobre su desempeño, además de la elaboración de estadísticas para comprobar que el modelo generado corresponde exactamente al DLA.

IV.1.1 Desempeño del Algoritmo Secuencial.

Los tiempos de ejecución en un solo procesador, de programas que utilizan las diferentes técnicas descritas en el capítulo correspondiente a la metodología, son mostrados en la tabla IV.1. Como puede observarse, el uso de búsquedas aproximadas por los nodos intermedios del *quad-tree* presenta una gran ventaja sobre todos los demás métodos probados. Cuando el programa de generación de agregados se ejecuta creando un solo DLA, en el modelo clásico y por tanto sin interacción de ningún tipo, se obtienen tiempos de ejecución relativamente cortos.

Tiempos de ejecución.	
Modelo *Off-lattice* n=100,000 partículas	
Técnica	**Tiempo (seg.)**
Lista Ligada	681
Lista Ligada con Desigualdad Triangular	354
Árbol Binario balanceado	197
KDTREE	44
KDTREE *Bucket Depth Search*	16
KDTREE *Bucket Best First Search*	7
QUADTREE *Pre-Balanceado Best First Search*	5
QUADTREE *Pre-Balanceado* Sub-Búsquedas Aproximadas.	1

Tabla IV.1. Desempeño de diferentes técnicas para el modelo *off-lattice*.

Los tiempos mostrados corresponden al tiempo real transcurrido entre el inicio y el final de la ejecución del algoritmo (*wall clock time*). Siguiendo con la evaluación del

desempeño, en la tabla IV.2 se muestra el tiempo de cálculo necesario para la generación de fractales DLA con un mayor número de partículas, utilizando un solo procesador. El método utilizado se tomará de aquí en adelante es el de sub-búsquedas aproximadas con el *quad-tree*, como se describe en el pseudocódigo de la sección III.2.5.

Tiempos de ejecución.		
***Quad-tree* con sub-búsquedas.**		
No. de partículas.	**Tiempo (seg.)**	**Tiempo (min.)**
100,000	1	0.016
500,000	35	0.583
1,000,000	101	1.683
2,500,000	190	3.166
5,000,000	424	7.066
10,000,000	1112	18.53
25,000,000	3300	55.00
50,000,000	5759	95.98
100,000,000	12785	213.08

Tabla IV.2. Tiempos de ejecución con DLA's de diferentes tamaños.

Los tiempos de ejecución indicados en la tabla anterior, sugieren que no solamente se ha conseguido una mejora de proporción constante con respecto al modelo de rejilla del DLA, sino que incluso se ha disminuido la complejidad computacional del procedimiento. Un análisis formal de esta disminución queda fuera de los objetivos de la tesis aquí presentada.

IV.1.2 Rendimiento de los Algoritmos con Color.

Al tener en los modelos multicolor un aumento en los pasos de simulación del modelo, inducido por la interacción de las figuras fractales, los tiempos de las corridas fueron significativamente mayores, pero conservándose dentro de los límites prácticos para nuestros objetivos.

Entre factores que más contribuyen al retardo de las corridas, está la simulación del movimiento browniano de partículas que son destruidas al encontrarse con un agregado de diferente color al de ellas mismas. Tales partículas, junto con las que se ignoran al ir más allá del radio de muerte, no contribuyen a incrementar ninguno de los agregados. Además, como se mencionaba, en las simulaciones con un gran número de partículas, se utilizan, según el corresponda, dos o tres estructuras de almacenamiento, aumentando el tiempo de ejecución secuencial, pero favoreciendo al paralelismo y a la eficiencia en el uso de memoria.
La tabla IV.3 expone los tiempos de proceso requerido por un solo procesador para la simulación de fractales con interacción para dos y tres colores, con diferentes

valores para el número de partículas, utilizando un solo procesador. Se toman los promedios para tres corridas.

Tiempos de ejecución.		
Modelo Bicolor		
No. de partículas.	**Tiempo (seg.)**	**Tiempo (min.)**
100,000	17	0.28
250,000	43	0.71
500,000	89	1.48
1,000,000	215	2.58
2,500,000	657	10.95
5,000,000	1676	27.93
10,000,000	3936	65.60
Modelo Tricolor		
No. de partículas.	**Tiempo (seg.)**	**Tiempo (min.)**
100,000	30	0.50
250,000	84	1.40
500,000	183	3.05
1,000,000	425	7.08
2,500,000	1322	22.03
5,000,000	3176	52.93
10,000,000	7124	118.73

Tabla IV.3. Tiempos de ejecución con DLA’s coloreados.

IV.1.3 Desempeño del Algoritmo Paralelo.

En nuestro código paralelo, un factor que limita la ganancia en tiempo de ejecución son las operaciones de sincronización que se describen en la sección III.3.2, es decir los bloqueos que evitan la escritura simultánea de la estructura de datos por parte de los procesadores. Las mejoras obtenidas en los tiempos de ejecución al incrementar el número de procesadores se muestran en la tabla IV.4.

Los tiempos de ejecución presentados son el promedio tomado de tres corridas para el número correspondiente de partículas, y sobre el número de procesadores indicado. En todos los casos, la desviación fue menor del 5%.

Tiempos de ejecución.								
Multiprocesamiento								
DLA 1,000,000 partículas.								
No. Procs.	1	2	3	4	5	6	7	8
Tiempo seg.	72	38	24	18	17	16	15	13
DLA 2,500,000 partículas.								
No. Procs.	1	2	3	4	5	6	7	8
Tiempo seg.	190	100	65	51	44	41	37	35
DLA 5,000,000 partículas.								
No. Procs.	1	2	3	4	5	6	7	8
Tiempo seg.	424	212	148	109	100	88	83	74
DLA 10,000,000 partículas.								
No. Procs.	1	2	3	4	5	6	7	8
Tiempo seg.	883	447	325	236	212	184	164	157
Tiempo min.	14.7	7.45	5.41	3.93	3.53	3.06	2.73	2.61
DLA 25,000,000 partículas.								
No. Procs.	1	2	3	4	5	6	7	8
Tiempo min.	55	29.9	23.6	18.9	14.9	12.1	10.6	10.1
DLA 50,000,000 partículas.								
No. Procs.	1	2	3	4	5	6	7	8
Tiempo min.	95.9	50.9	43	34.3	22.5	20.4	18.8	17.7
DLA 100,000,000 partículas.								
No. Procs.	1	2	3	4	5	6	7	8
Tiempo min.	213	112	73.6	64.5	53.9	48.1	44.9	39.1

Tabla IV.4. Tiempos de ejecución DLA's con multiprocesamiento.

Como puede verse, la ganancia, aunque no es estrictamente lineal con respecto al número de procesadores, es bastante significativa. Observaciones teóricas, como es la *Ley de Amdahl* (Culler, *et al*. 1997), nos indican que el nivel de paralelización obtenido es aceptable. Dicha ley está expresada por la ecuación IV.1.

$$G_{vel} = \frac{1}{(1-p) + \frac{p}{n}} \qquad \text{...(IV.1)}$$

en donde *n* representa el número de procesadores en los que se realiza la paralelización del código, *p* simboliza la proporción de código paralelizable, por lo que *(1-p)* nos indica la fracción de código que no se puede paralelizar; *p/n* es la cantidad de código distribuida entre los *n* procesadores y G_{vel} es la ganancia en velocidad conseguida.

La figura IV.1 es representativa de esta observación. En IV.1a) se indican los porcentajes de paralelización derivados de la aplicación de la ecuación IV.1 sobre datos resultantes de las simulaciones hasta con 10^8 partículas. Las variaciones en los porcentajes se deben a la diferencias del consumo de tiempo en operaciones de sincronización. Con un mayor número de procesadores suelen ocurrir más bloqueos y por tanto se requiere mayor porcentaje de tiempo de proceso utilizado para sincronía. En IV.1b) se muestra la ganancia de velocidad obtenida para distinto número de procesadores.

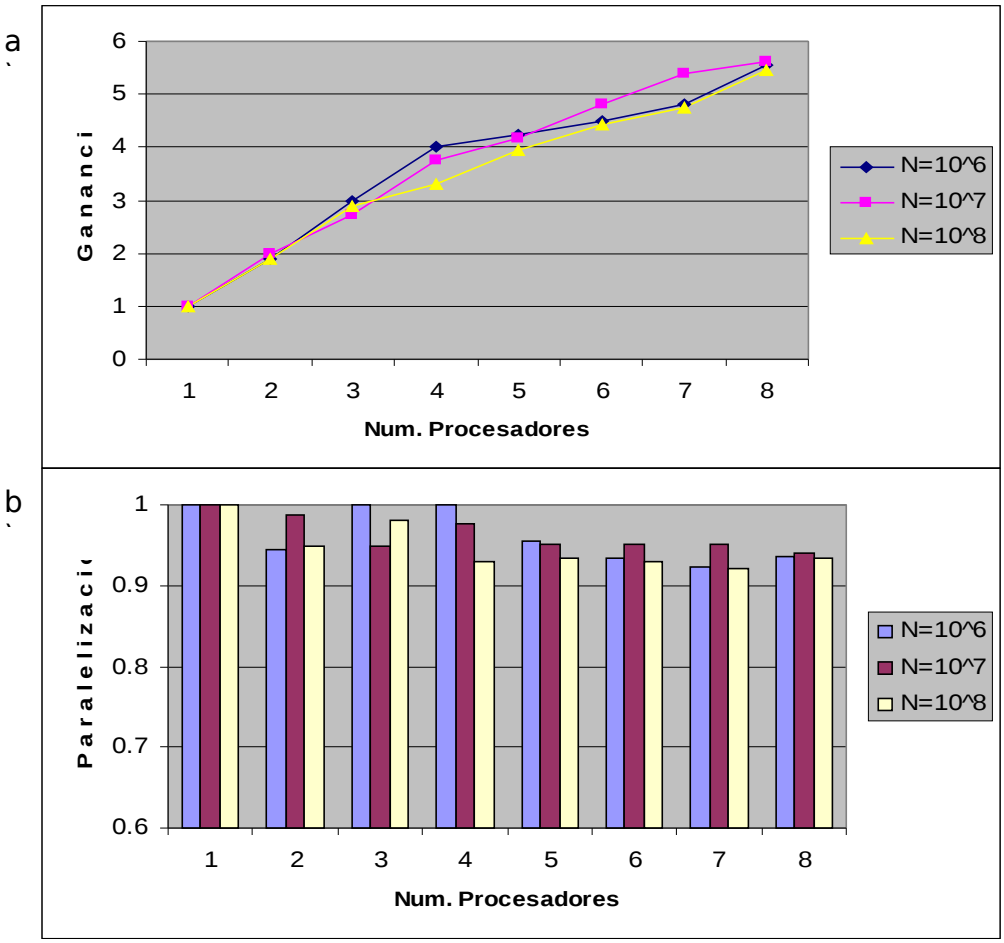


Figura IV.1. Porcentaje de paralelización y ganancia en velocidad.

IV.2 Análisis de Dimensionalidad Fractal.

La verificación de que el algoritmo generado corresponde exactamente al modelo DLA, se lleva a cabo con la determinación de la dimensionalidad fractal de los agregados producidos para el caso de un DLA obtenido a partir de una sola semilla. Con esto se tiene la seguridad de que las técnicas de aceleración aquí utilizados no afectan al modelo, y que resultan confiables para explorar nuevas propiedades que pudieran derivarse de los modelos multicolor, o bien del modelo original sobre un gran número de datos.

Una forma de determinar la dimensión fractal de un DLA a partir de datos estadísticos discretos es hacer uso de la propiedad:

$$N \sim R_g^{\,D} \qquad \text{…(IV.2)}$$

que para los fines prácticos descritos a continuación puede ser transformada en:

$$N = kR_g^{\,D} \qquad \text{…(IV.3)}$$

en el modelo *"off-lattice"*, *k* es una constante de ajuste que depende del diámetro de las partículas, y en el caso del modelo sobre "*lattice"*, de la separación de los elementos de la rejilla y de la forma estructural de la misma. Aplicando la función logaritmo en ambos lados de la ecuación (IV.3) obtenemos:

$$\log(N) = \log(kR_g^{\,D}) \qquad \text{…(IV.4)}$$

$$\log(N) = \log(k) + \log(R_g^{\,D}) \qquad \text{…(IV.5)}$$

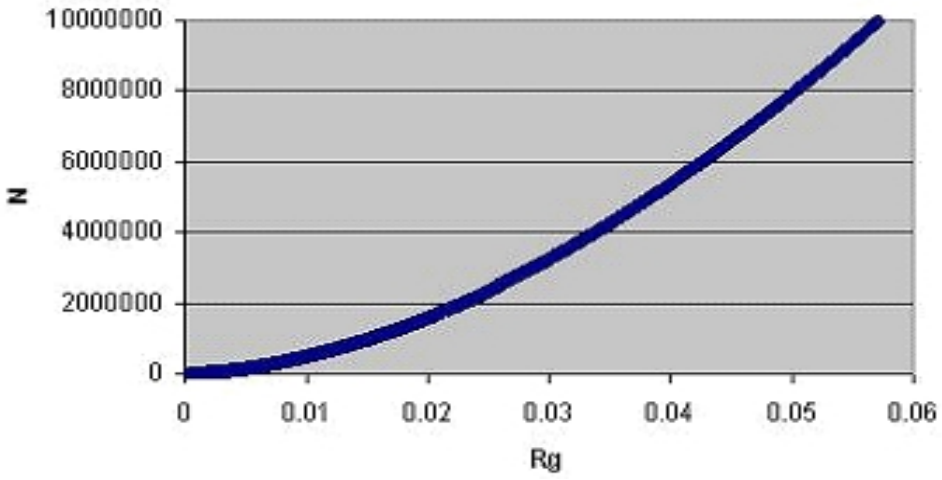
$$\log(N) = c + D\log(R_g) \qquad \text{…(IV.6)}$$

De donde vemos una dependencia lineal de $\log(N)$ con respecto a $\log(R_g)$ determinada por la dimensión fractal *D* y alguna constante aditiva *c*.

Con los datos conseguidos, un ajuste de estos por el método de mínimos cuadrados hacia una expresión lineal nos permite encontrar las constantes *c* y *D*, siendo de nuestro interés *D*, que corresponde a la dimensión fractal del agregado descrito por los datos.

La figura IV.2 (a) muestra la graficación del número de partículas *N* contra los radios de giro R_g encontrados para un conjunto de datos derivados de la ejecución de nuestro algoritmo. La figura IV.2 (b) muestra $\log(N)$ contra $\log(R_g)$, que permite hacer el ajuste lineal para la determinación de la dimensión fractal *D*.

a)



b)

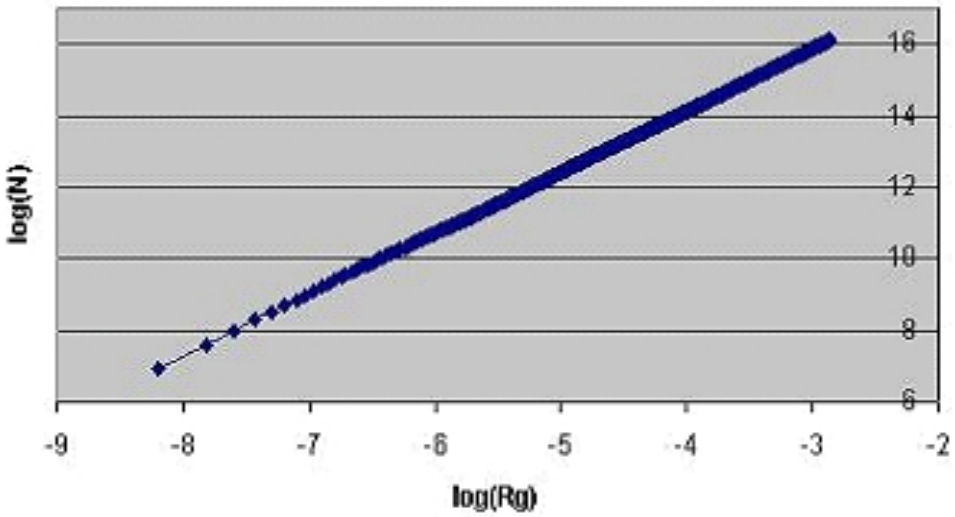


Figura IV.2. Relaciones radio de giro contra número de partículas.

Es muy importante determinar de manera cuantitativa si la simulación del fenómeno DLA no se ve afectada por el multiprocesamiento, que como ya se ha mencionado cae dentro del modelo PDLA con una convergencia al modelo DLA para agregados de gran tamaño. En la tabla IV.5 muestra la dimensionalidad fractal calculada para un promedio de diez corridas del algoritmo con un valor de n=10,000,000 de partículas para cada uno de los valores indicados en el número de procesadores utilizados. El renglón etiquetado con *Rg* corresponde al radio de giro y el renglón marcado con *sd* presenta la desviación estándar de las mediciones. Se utilizó en cada caso un núcleo con n=1,000,000 de partículas generadas por un solo procesador. Esto nos permite afirmar que la introducción del multiprocesamiento, hasta con 8 procesadores y siguiendo la metodología aquí descrita, no afecta de manera importante a la dimensionalidad fractal de los agregados generados.

Dimensión Fractal y Radio de Giro				
DLA 10,000,000 partículas				
No. Procs.	**1**	**2**	**3**	**4**
Rg.	0.05829042	0.057433	0.057872	0.058172
sd	0.00170285	0.00062	0.002436	0.00124
Dim.	1.70654038	1.724243	1.714188	1.720475
sd	0.04553881	0.038541	0.033376	0.031555
No. Procs.	**5**	**6**	**7**	**8**
Rg.	0.058343	0.0576171	0.057769	0.057201
sd	0.001646	0.0015945	0.002763	0.000976
Dim.	1.712779	1.7053001	1.70974	1.721508
sd	0.03928	0.0467081	0.05338	0.034934

Tabla IV.5. Dimensión fractal con multiprocesamiento.

También se tiene que la dimensionalidad varía con respecto al número de partículas *N*. Formalmente, la dimensionalidad fractal se define en el límite cuando $N \to \infty$, por lo que agregados con un mayor número de partículas nos dará una aproximación más exacta a tal dimensionalidad. La tabla IV.6 muestra el promedio de las dimensiones fractales encontradas en tres corridas del algoritmo DLA sobre 8 procesadores con diferentes números de partículas. En cada caso se tomo un núcleo de un orden de magnitud menor generado por un solo procesador.

Dimensión Fractal y Radio de Giro			
Ejecución paralela con 8 procesadores			
N	**10^3**	**10^4**	**10^5**
Rg.	0.00025637	0.001045	0.003953
sd	9.5748E-06	4.88E-05	8.13E-05
Dim.	1.74988502	1.713164	1.699942
sd	0.06857902	0.075513	0.017928
N	**10^6**	**10^7**	**10^8**
Rg.	0.015207	0.057127	0.2215942
sd	0.00035	0.000907	0.0071619
Dim.	1.713488	1.7287	1.7118961
sd	0.009604	0.027698	0.0309011

Tabla IV.6. Dimensión fractal de acuerdo a *N*.

Como puede observarse, los valores obtenidos para la dimensión fractal *D*, están alrededor de 1.70 y 1.71, tomando en cuenta las desviaciones estándar, las medidas pueden siempre considerarse dentro de este rango. Estas observaciones van de acuerdo con el valor encontrado en otros experimentos (Tolman & Meakin, 1989) para el DLA con geometría radial, midiendo la dimensionalidad con respecto al radio de giro, como es el caso que aquí manejamos.

Se han reportado otros valores de dimensionalidad fractal utilizando diferentes parámetros del DLA. Benoit Mandelbrot ha obtenido valores de 1.67 realizando medidas sobre los fiordos (*lacunarity gaps*) que se forman entre las ramas del agregado, pero el valor medido por el radio de giro sigue dando como resultado 1.71. (Maldelbrot *et al*, 2002).

Existe una relación teórica, basada en la interacción de las caminatas aleatorias con la superficie del DLA, conocida como '*fórmula de Muthukumar*', expresada en la ecuación (IV.7) que sugiere un valor para la dimensión fractal del fenómeno DLA, medida con respecto al radio de giro, para distintos valores de la dimensión euclidiana (Hasley, 1992).

$$D_g = (d^2 + 1)/(d + 1) \qquad \text{...(IV.7)}$$

en esta igualdad, d representa la dimensión euclidiana, por lo que para el DLA en dos dimensiones el valor sugerido es $D_g = 5/3 = 1.66\overline{6}$. No obstante, en el artículo de T.C. Hasley se encuentra un valor teórico de 2-1/2+1/5=1.7. La ecuación (IV.7), parece sin embargo, no tener controversias con los resultados experimentales hallados para valores de $d > 2$.

Otros estudios que hacen uso de la *teoría de la renormalización* acotan la dimensión fractal *D* de acuerdo a la siguiente desigualdad (Davitovitch, 2000):

$$1.69 < D < 1.72 \qquad \text{...(IV.8)}$$

Los autores Menshutin y Schur reportan variaciones en los valores de *D*, comprendidos entre 1.70 y 1.74 (Menshutin *et al*, 2006), que ocurren al cambiar el radio de giro en la obtención experimental de la dimensión fractal, por otros parámetros como son el radio promedio, el radio cuadrático medio, el radio efectivo, y el radio con respecto a la semilla del agregado, siendo el valor de *D* conseguido con el radio de giro de 1.71; lo que está de acuerdo con los valores obtenidos en esta tesis.

IV.3 Resultados con los Modelos Multicolor.

Las simulaciones con los modelos multicolor, presentaron la misma dimensionalidad fractal que el modelo DLA original para los casos en los que la distancia entre las semillas es la misma que entre cualquier otro par de partículas, es decir, cuando se encuentran separadas solamente por una distancia igual a la tolerancia de adhesión de cualquier partícula al resto del conjunto. Esto nos asegura que para estos casos, la dimensión fractal de los agregados en conjunto no cambia por la introducción de los modelos multicolor, que se siguen conservando dentro del régimen del modelo DLA no obstante la introducción de reglas de interacción en el algoritmo.

La tabla IV.7 muestra la dimensionalidad fractal encontrada en los modelos de dos y tres colores para un promedio de cinco corridas para cada valor del número de partículas N mostrado, utilizando un solo procesador.

Dimensión Fractal y Radio de Giro			
Modelo Bicolor			
N	**10^6**	**$5x10^6$**	**10^7**
Rg.	0.014815	0.038145	0.055009
Dim.	1.719238	1.715491	1.706197
Modelo Tricolor			
N	**10^6**	**$5x10^6$**	**10^7**
Rg.	0.015134	0.038436	0.052218
Dim.	1.705556	1.719955	1.691604

Tabla IV.7. Dimensión fractal de los modelos multicolor.

Las tablas IV.8 y IV.9 nos muestran la dimensión fractal obtenida con el promedio de cinco ejecuciones al variar el número de procesadores para generar los modelos multicolor de la forma descrita arriba.

Dimensión Fractal y Radio de Giro				
Modelo Bicolor N=10^7				
Procs.	**1**	**2**	**3**	**4**
Rg.	0.055009	0.053375	0.054322	0.051544
Dim.	1.706197	1.721532	1.723867	1.712928
Procs.	**5**	**6**	**7**	**8**
Rg.	0.056013	0.057009	0.055468	0.051633
Dim.	1.729508	1.705887	1.718987	1.723011

Tabla IV.8. Dimensión fractal del modelo bicolor con multiproceso.

Dimensión Fractal y Radio de Giro				
Modelo Tricolor $N=10^7$				
Procs.	**1**	**2**	**3**	**4**
Rg.	0.052218	0.054914	0.054554	0.053097
Dim.	1.691604	1.725758	1.705370	1.741450
Procs.	**5**	**6**	**7**	**8**
Rg.	0.051407	0.050740	0.053656	0.054532
Dim.	1.719702	1.719358	1.724558	1.731177

Tabla IV.9. Dimensión fractal del modelo tricolor con multiproceso.

Con esto verificamos que la introducción del procesamiento en paralelo tampoco afecta a la dimensionalidad fractal de los agregados multicolor.

Aún con la confirmación de las observaciones anteriores, vemos que la morfología adquirida por los agregados de cada color es bastante diferente en cada simulación. Tal variación se debe a la misma aleatoriedad en los modelos multicolor.

El fenómeno observado es que los agregados compiten entre sí, limitando el crecimiento de cualquier otro agregado cercano cuando es posible, o bien, pueden tener un crecimiento limitado por algún agregado cercano. Puede incluso, alcanzarse condiciones más o menos equilibradas, en las que los agregados tienen un crecimiento aproximadamente proporcional. Tales efectos se observan en la figura IV.3, que muestra la interacción entre dos y tres agregados con un número pequeño de partículas n=5,000, pero que permite observar visualmente las interacciones, pues las fluctuaciones en el crecimiento que determinan la estructura a gran escala de los agregados ocurren en las etapas iniciales de la simulación (Rodríguez Romo, *et al*, 2005).

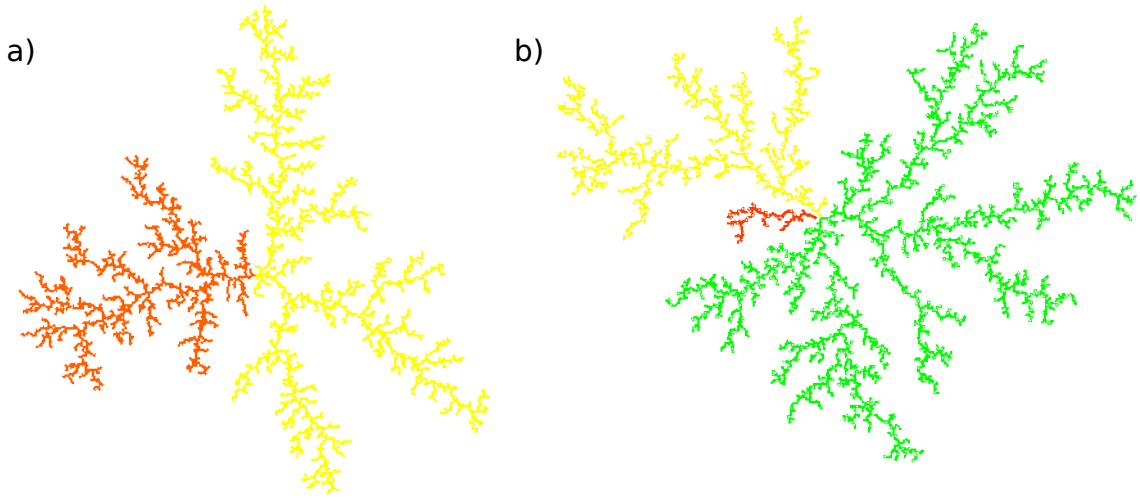


Figura IV.3. Interacción entre dos a) y tres b) agregados.

Un factor que determina la morfología es la separación que existe entre las semillas de diferente color que conforman los agregados iniciales. El proceso que se desarrolla, es además estocástico, por lo que una separación determinada no implica una proporción preestablecida en la forma en que se distribuyen las partículas sobre cada agregado. Los cambios de morfología inducidos por la distancia de separación pueden apreciarse en la figura IV.4, que muestra la interacción de agregados para distintas distancias de separación *d,* medida en unidades equivalentes al diámetro de las partículas. El primer recuadro corresponde al modelo bicolor, el segundo al modelo tricolor. En estos casos se tomó un número bajo de partículas n=10,000 para poder visualizar la interacción a distancias cortas.

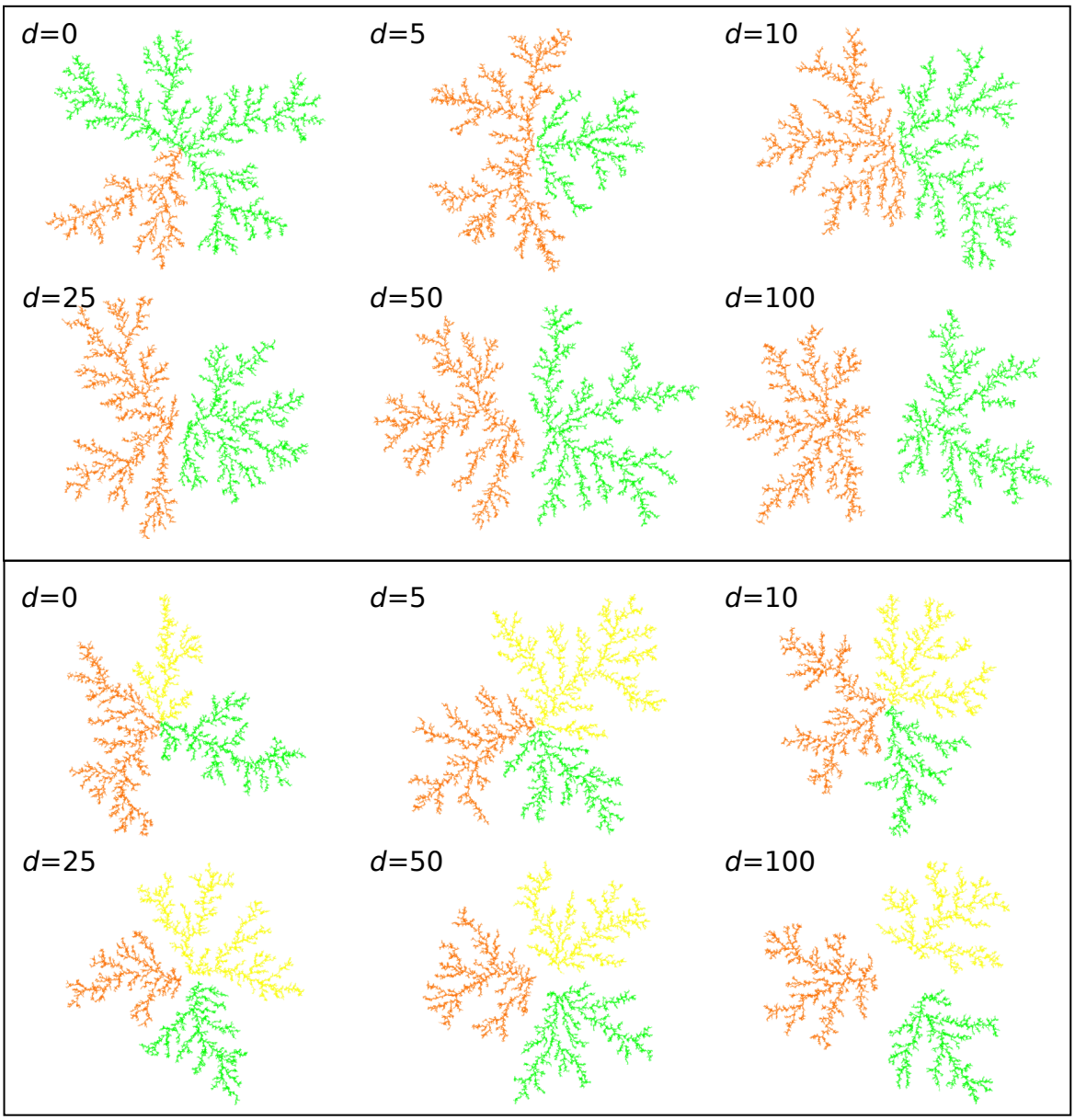


Figura IV.4. Interacción entre dos y tres agregados a diferentes distancias.

Un fenómeno importante a resaltar es la aparición de agregados con dimensionalidad cero cuando se asumen valores pequeños de separación *d*, es decir, se tienen conjuntos, en los que el número de partículas correspondientes a un cierto color permanece constante conforme el número total de partículas en el sistema aumenta.

Un ejemplo de lo anterior puede verse en la figura IV.3 b) para el color rojo. En los ensayos aquí elaborados, este efecto desaparece para valores de $\underline{d = 5}$ diámetros de partícula para el modelo bicolor y $\underline{d = 12}$ diámetros para el modelo tricolor. Estas distancias se determinaron con 100 sondeos por distancia para agregados de n=1,000,000 de partículas en cada modelo. Ensayos para distancias una unidad menor presentaron al menos un conjunto con dimensionalidad cero en menos de 100 repeticiones, mientras que pruebas con las distancias indicadas y también con una unidad mayor, no presentaron conjuntos de dimensionalidad cero después de 100 repeticiones. Por supuesto, el efecto tampoco se volvió a observar en ninguno de los experimentos para valores mayores de *d*.

Con la finalidad de apreciar numéricamente el comportamiento de los agregados con interacción se presentan las tablas IV.10 y IV.11, en las que podemos ver una comparación en la distribución de las partículas por cada color c_1, c_2 y c_3. Estos datos fueron obtenidos en cinco ejecuciones del algoritmo para dos y tres colores en interacción con cada valor de distancia entre semillas indicado por *d*. El número total de partículas en este caso fue de n=10,000,000 por ensayo. Se observa la influencia de la separación inicial de las semillas en la dispersión de las partículas sobre cada color c_i.

Distribución por color de las partículas.						
Modelo Bicolor. N=10^7						
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=0	c_1	4921962	8062695	7186981	8065405	5335272
	c_2	5078038	1937305	2813029	1934595	4664728
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=5	c_1	7262193	4520819	3791834	3969847	2684263
	c_2	2737807	5479181	6208116	6030153	7315727
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=10	c_1	5923182	5410052	3199752	4318355	6006621
	c_2	4076818	4589968	6800268	5681615	3993359
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=50	c_1	5099296	4607429	6116589	5173747	4701468
	c_2	4900704	5392571	3883411	4826253	5298532
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=100	c_1	5231025	3910911	4480369	4503664	5002403
	c_2	4768975	6089089	5519631	5496336	4997597

Tabla IV.10. Distribución por color con interacción. Modelo Bicolor

Distribución por color de las partículas. Modelo Tricolor. $N=10^7$						
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=0	c_1	2630814	4732783	3523811	3314204	44
	c_2	4270304	2774402	2973381	7	5577927
	c_3	3098872	2492825	3502808	6685736	4421633
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=5	c_1	2864406	4996223	3946705	4990402	4387274
	c_2	4198812	2785204	1614835	3769693	2411772
	c_3	2940782	2218573	4438470	1239905	3200954
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=10	c_1	3785563	2843689	2161508	3727290	3282987
	c_2	2695253	3243881	3745291	3856240	3500761
	c_3	3519184	3912430	4093201	2416470	3216262
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=50	c_1	3674608	3760745	3033926	4177012	2767981
	c_2	3386436	3285137	2828253	3540845	3740665
	c_3	2938956	2954118	4137821	2282143	3491354
Ejecución:		**1**	**2**	**3**	**4**	**5**
d=100	c_1	3328206	3569840	3296219	3039456	3311689
	c_2	3253119	3291203	3461146	3670008	3067262
	c_3	3418675	3138957	3242635	3290536	3621049

Tabla IV.11. Distribución por color con interacción. Modelo Tricolor

En la tabla IV.11 podemos observar la aparición de agregados con dimensionalidad cero en la ejecución número 4 con una distancia d=0 para el color c_2; lo mismo ocurre para el color c_1 en la ejecución número 5 con d=0.

Con estos experimentos se confirma el comportamiento señalado en investigaciones previas (Tchijov, *et al*, 1996; Rodríguez Romo, *et al*, 2005), para las simulaciones con un gran número de partículas. La morfología a gran escala está determinada por las fluctuaciones ocurridas en las primeras etapas de crecimiento de los agregados y la interacción tiende a ser menos acentuada cuando la distancia de separación entre las semillas no permite que existan '*coordenadas compartidas*' o '*quasi- compartidas*' (Rodríguez Romo, *et al*, 2005) –o más precisamente, sus respectivos equivalentes en espacio para el modelo *"off-lattice"*– en las configuraciones iniciales de los agregados.

V. CONCLUSIONES.

Al haber creado un software eficiente y confiable para la generación y análisis de interacción de agregados fractales, utilizando procesamiento paralelo, se considera que los objetivos de esta tesis quedaron cubiertos.

Los rangos de dimensionalidad fractal conseguida están dentro de los valores aceptados por la mayoría de otras investigaciones previas, tanto teóricas (Davidovitch *et al*, 2000; Hastings 1997; Halsey *et al*, 1992) como experimentales (Tolman & Meakin, 1989; Kaufman *et al*, 1995).

Una aportación significativa lograda, es que se determinó la forma de implementar los detalles ocultos de codificación que no se especifican en ninguna publicación especializada sobre el fenómeno de Agregación Limitada por Difusión, que tienden a exponer principalmente resultados y teorías, dejando a un lado muchos detalles prácticos. De esta manera, se obtuvieron, sin poner gran presión sobre el tiempo de procesamiento, agregados fractales con un número de elementos del orden de 10^8; que es el tamaño máximo para fractales DLA publicado a la fecha (Kaufman *et al*, 1995). Nuestra implementación, cuya parte central se basa en el pseudocódigo de la sección III.2.5 difiere de las publicadas anteriormente en que no utiliza mapas de bits para determinar la posición relativa de las partículas en movimiento con respecto al agregado. En vez de ello, utilizamos la información implícita en los apuntadores nulos del *quad-tree* en el que se almacenan las partículas.

En la ejecución sobre cualquier sistema que haga uso de la plataforma *Linux* no hay necesidad de modificar el programa pues *pthreads* forma parte de la interfaz estándar de *Linux*. Si se desea utilizar otro sistema operativo, se puede usar la versión con *OpenMP*, también si necesidad de realizar ningún cambio, excepto que se debe asegurar la compatibilidad con dichas librerías, o bien, proporcionar una configuración del sistema que soporte *pthreads* (por ejemplo, existen librerías para la emulación de *pthreads* en win32). Nuestra aplicación es, entonces, también altamente portable. El software creado utiliza tecnologías de sistemas de multiprocesamiento altamente acoplados, como son los sistemas *multicore* y otros sistemas de memoria compartida.

El nivel de paralelización obtenido es igualmente satisfactorio, tomando en cuenta la alta dependencia histórica del proceso DLA sobre cada uno de los pasos de agregación anteriores. Sin embargo, si se requiere incrementar el número de procesadores en forma significativa, puede ser necesario aumentar el orden del número de partículas del núcleo de DLA generado por un solo procesador, para no alterar la dimensionalidad de los fractales así obtenidos.

En el caso de los modelos DLA multicolor con interacción, vemos que los tiempos de generación son mayores, esto está asociado a la misma complejidad inducida por los nuevos modelos. No obstante, tanto el tamaño de los agregados multicolor obtenidos, como los tiempo de ejecución necesarios, son bastante adecuados para su aplicación en investigaciones posteriores concernientes a tales modelos. Los resultados obtenidos experimentalmente confirman observaciones hechas con anterioridad (Rodríguez Romo, *et al*, 2005) que afirman que la morfología a gran escala de estos modelos queda determinada en las primeras etapas del crecimiento, y que se encuentra influenciada por la distancia entre las semillas iniciales y la probabilidad con la que aparecen semillas de cada color.

VI. BIBLIOGRAFIA.

Aho Alfred V., Hopcroft John E., Ullman Jeffrey D. Data Structures and Algorithms. Addison-Wesley Publishing Company. 1983.

Autodesk Inc. DXF Reference. Autodesk Inc. 2000. www.autodesk.com.

Barnsley Michael. Fractals Everywhere. Academic Press Inc. 1988.

Bentley J.L. Multidimensional Binary Search Trees Used for Associative Searching. Comunications of the ACM 18(9). 1975.

Beveridge J. Multithreading Applications in Win32: A Complete Guide to Threads. Addison-Wesley Longman Publishing Co. 1997.

Bunde A., Shlomo Havlin. Fractals and Disordered Systems. Second Edition Springer-Verlag Berlin 1996.

Butenhof David R. Programming with Posix Threads. Addison-Wesley Professional Computing Series. 1997.

Culler David, Singh Jaswinder, Gupta Anoop. Parallel Computer Architecture. A Hardware/Software Approach. Morgan Kaufmann Publishers. 1997.

Dagum Leo. OpenMP: A Proposed Industry Standard API for Shared Memory Programming. 1997. www.openmp.org.

Davidovitch B., Procaccia I. Dimension of Fractal Growth Patterns as Dynamical Exponent. Physical Review Letters Vol. 85 Num. 17. 2000.

Falconer Kenneth. Fractal Geometry. Mathematical Foundations and Applications. Second Edition. Ed. Wiley and Sons Ltd. 2003.

Finkel R.A., Bentley J.L. Quad trees: a Data Structure for Retrieval on Composite Keys. Acta Informatica. 4(1) 1974.

Free Software Foundation. Linux Manual Pages. Free Software Foundation Inc. 1997. www.kernel.org/doc/manpages/

Halsey T.C., Leibig M. Theory on Branched Growth. Physical Review A. Vol. 46. Num. 12. 1992.

Halsey T.C. Diffusion Limited Aggregation. A model for Pattern Formation. Physics Today Online. American Institute of Physics 2001. www.aip.org.

Hastings M. B. Renormalization Theory of Stochastic Growth. Physical Rewiew E. Vol. 55, Num. 1. 1997.

Intel Corporation. Open Source Computer Vision Library. Reference Manual. Intel Corp. 2001. www.intel.com.

Kaufman Henry, Vespignani Alessandro, Maldelbrot Benoit, Woog Lionel. Parallel Diffusion-Limited Aggregation. Physical Review E Vol. 52 Num. 5, 1995.

Knuth D. E. The Art of Computer Programming. Vol. 3. Sorting and Searching. Addison-Wesley Publishing Company. 1979.

Kurt Wall, Watson Mark. Linux Programming Unleashed. 2nd Edition. SAMS 2008.

Langsman Y. Augenstein M.J. Tenenbaum A.M. Data Structures Using C and C++. Second Edition. Prentice-Hall Inc. 1993.

Latstovetsky Alexey. Parallel Computing on Heterogeneous Networks. Willey-Interscience 2003.

Mandelbrot Benoit. Fractal Geometry: What is it, and what does it do?. Proceedings of the Royal Society of London. Princeton University Press 1988.

Mandelbrot Benoit B., Kol Boaz, Aharony Amnon. Angular Gaps in Radial Diffusion-Limited Aggregation: Two Fractal Dimensions and Nontransient Deviations from Linear Self-Similarity. Physical Review Letters Vol. 88 Num. 5 2002.

Machta J. Complexity, Parallel Computation and Statistical Physics. Willey Periodicals, Inc. Vol. 11 No. 5. 2006.

Meakin Paul. Universality, Nonuniversality, and the Effects of Anisotropy on Aiffusion-Limited Aggregation. Physical Review A. Vol. 33 Num. 5. 1986.

Menshutin A.Y. Schur Lev N. Test of multiscaling in a diffusion-limited-aggregation model using an off-lattice killing-free algorithm. Physical Review E. 73, 011407. 2006.

Moriarty K., Machta J., Greenlaw R. Parallel Algorithm and Dynamic Exponent for Diffusion-Limited Aggregation. Physical Review E. Vol. 55 Num. 5 1997.

Rodriguez-Romo Suemi, Tchijov Vladimir, Ibañez-Orozco Oscar, Castaño Victor M. Growth Probability in Bicolored Diffusion Limited Aggregation. Physica A 347 2005.

Samet Hannan. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Series in Computer Graphics. 2006.

Tchijov V., Nechaev S., Rodríguez-Romo S. Colored DLA Model. JETP Letter 64. 498 1996.

Tolman Susan, Meakin Paul. Off-Lattice and Hypercubic-Lattice Models for Difusión-Limited Aggregation in Dimensionalities 2-8. Physical Review A. Vol. 40, No. 1. Jul. 1989.

University of Tennessee. MPI: A Message-Passing Interface Standard. University of Tennessee, Knoxville, Tennessee. Message Passing Interface Forum November 15, 2003

Voss Richard F. Multiparticle Diffusive Fractal Aggregation. Physical Review B Vol. 30 Num 1 1984.

Vicsek T. Fractal Growth Phenomena (Second Edition) World Scientific. 1992.

Witten T. A., Sander L. M. Diffusion-Limited Aggregation, A Kinetic Critical Phenomenon. Physical Review Letters Vol. 47 Num.19 1981.