UNIVERSIDAD AUTÓNOMA METROPOLITANA

Simulación paralela de los procesos de intrusión y retracción de mercurio (Hg) en medios porosos para clusters multicore

Para obtener el grado de
MAESTRO EN CIENCIAS
(Ciencias y Tecnologías de la Información)
PRESENTA

Lic. Carlos Hiram Moreno Montiel

Asesores de la Tesis

Dra. Graciela Román Alonso

Dr. Miguel Alfonso Castro García

Sinodales

Presidente: Dra. Graciela Román Alonso

Secretario: Dr. Fernando Rojas González

Vocal: Dr. Santiago Domínguez Domínguez

México D.F

Febrero del 2010

Resumen

La mayoría de los materiales que se encuentran a nuestro alrededor presentan huecos o poros en su interior, el estudio de poros es de gran importancia pues determinan propiedades especificas de un material. Al determinar las propiedades de un material existen beneficios que son debidos a la distribución de sus poros tales como el determinar la calidad de cementos, catalizadores, cerámicas, plásticos, etc. Una forma de conocer la distribución de los poros en un material, es con la porosimetría de mercurio, principalmente con los fenómenos de intrusión y retracción de mercurio. Sin embargo, en la vida real al manejar directamente, estos fenómenos resultan muy costosos ya que el mercurio es muy tóxico, por lo que se simulan por computadora.

Una manera de simularlos es con el modelo dual de sitios y enlaces, el cual representa un medio poroso (llamado red porosa), de una forma muy real. Sin embargo, al simular los fenómenos mencionados surgen problemas en cuanto a uso de memoria y procesamiento. El uso de memoria se incrementa al representarlos en una red porosa, debido a que algunos materiales reales presentan hasta billones o trillones de poros. Ademas, el tiempo de procesamiento es alto debido a que en cada uno de los poros del material se realizan operaciones que ocurren durante los fenómenos. Una alternativa para disminuir los requerimientos, en memoria y tiempo de procesamiento es con el cómputo paralelo.

En esta tesis se prepone un simulador paralelo para representar los fenómenos de intrusión y de retracción de mercurio en materiales porosos. El simulador busca reducir el consumo de memoria y tiempo de procesamiento para lo cual es elaborada en una primera versión usando memoria compartida con OpenMP y una segunda combinando memoria compartida y paso

de mensajes con OpenMP y MPI. Esta simulación logra disminuir el tiempo de procesamiento y el consumo de memoria.

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología, CONACyT, por la beca que me fue otorgada en mis estudios de posgrado.

A la Universidad Autónoma Metropolitana - Unidad Iztapalapa y a todos los Académicos que la integran ya que son una parte importante de mi formación académica.

También al Departamento de Ingeniería Eléctrica por todas las facilidades que me proporcionaron en el transcurso de mis estudios y los recursos que ofrecieron para el desarrollo de mi trabajo de tesis.

De manera muy especial agradezco a mi asesora la Doctora Graciela Román Alonso por ser una excelente asesora y por aportarme sus conocimientos, y por ser tan excelente persona.

Al Dr. Miguel Castro por la ayuda y los consejos que me ha proporcionado a lo largo de la tesis.

Dedicado a:

Esta tesis está dedicada con todo mi cariño a mi madre

También esta dedicada a mis hermanos por su apoyo incondicional.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Al resolver algún tipo de problema en el que se busca una buena solución sin gastar muchos recursos (naturales, económicos, de espacio etc.), es conveniente auxiliarse con un modelo en el que se represente adecuadamente el problema. Con base en un modelo se pueden realizar diversos experimentos para llegar a su solución deseada, respetando cada una de las limitantes y reglas que presente el problema. Al usar modelos, obtenemos ciertos tipos de métodos de aproximación y de simulación, con los cuales podemos tener una manera más conveniente de resolver nuestro problema real.

Un método de aproximación se utiliza cuando, se requiere de un gran número de iteraciones para alcanzar los objetivos del modelo. Típicamente estos modelos se usan cuando se necesita resolver un problema matemático con muchas variables, requiere de muchos criterios para llegar a su resultado, pues en algunos casos no posee una solución exacta.

No obstante con una simulación, se puede plantear un escenario de la vida real, en el cual es muy difícil generar o reproducir los sucesos que se requieran modelar. Esto debido a que en algunos problemas no se puede tener experimentos directos sobre medios físicos o de algún otro tipo. En algunos casos las limitantes de un problema de la vida real son: el costo económico, la escasez de los recursos para su diseño, el manejo de materiales peligrosos, etc. Por este motivo se utilizan métodos de simulación por computadora, para obtener soluciones, con la ventaja de realizar muchas simulaciones sin que existan riesgos como costo económico o pérdidas físicas, según sea el caso. Un ejemplo de un modelo en el cual se requiere llevar acabo una simulación, es el proceso de intrusión y el de retracción de mercurio en materiales

porosos [1].

El proceso de intrusión consiste en invadir con mercurio un medio poroso mediante un aumento gradual de presión sobre todos los poros presentes. El proceso de retracción consiste en drenar el mercurio de una red porosa llena de este material, disminuyendo la presión en todos los poros del material. Al usar los procesos de intrusión y retracción de mercurio se ha observado principalmente el comportamiento del volumen penetrado por unidad de masa del sólido en función de la presión de penetración. Cuando la presión es aumentada o disminuida continuamente (en la intrusión o la retracción), se pueden obtener datos tales como la superficie específica, el volumen de poros, la distribución del tamaño de poros, etc. Estos datos son de gran utilidad en el área comercial ya que es un método relativamente rápido y sencillo para la determinación de la estructura de los espacios vacíos de los materiales (materiales pétreos, cerámicos, etc.). Los procesos de intrusión y de retracción de mercurio se han utilizado ampliamente para estudiar materiales porosos sólidos como: rocas, adsorbentes, catalizadores, textiles, cuero, polímero, filtros, materiales de refractarios, carbón, hierro poroso y comprimidos farmacéuticos entre otros. Sin embargo, en estos procesos es difícil tener una experimentación directa, porque el mercurio es un elemento que requiere de un manejo especial y cuidadoso principalmente por ser tóxico al inhalarlo, al tener contacto, o al ingerirlo.

Por otro lado, como el mercurio es un material peligroso, resulta muy complicado realizar los procesos de intrusión y de retracción de manera directa en materiales de gran tamaño. Por tal motivo resulta conveniente realizar una simulación por computadora ya que gracias a esto no se desperdiciarían recursos y no habría peligro como en un escenario real. Un elemento importante para llevar acabo la simulación es el representar el material poroso en computadora, con una técnica que represente al material real.

En la UAM Iztapalapa se desarrolló una técnica llamada modelo dual de sitios y enlaces (DSBM) propuesto por Vicente Mayagoitia y Kornhauser en 1984 [3], en el cual se representan los materiales porosos para ser usados por computadora. Este modelo representa de la mejor manera posible cada uno de los huecos o concavidades que tiene un material poroso, principalmente por dos tipos de estructuras geométricas. En los huecos más grandes del mate-

rial se coloca una esfera y en la parte más delgada, un cilindro conectado a la esfera para que se tenga un camino por todos los huecos del material. Con el modelado de materiales porosos se forma una especie de red cúbica de sitios (esferas) y enlaces (cilindros) interconectados entre sí, llamada red porosa. Gracias a este modelado es posible simular en computadora los fenómenos que ocurren en materiales porosos tales como la intrusión y la retracción. El modelado de los fenómenos de intrusión y retracción de mercurio, se facilita con el DSBM pero si las redes porosas son grandes (como normalmente ocurre) el tiempo de procesamiento es alto debido a la gran cantidad de poros y a la memoria que se requiere. Por estos motivos se busca una manera de simular estos procesos de una mejor forma para reducir el tiempo en que se llega a su solución. Una manera es diseñar algoritmos de intrusión y retracción de mercurio en los cuales se ocupen todos los recursos de una o varias computadoras al mismo tiempo. Con el cómputo paralelo se puede solventar ese requerimiento, ya que con el se puede ejecutar el trabajo en varias máquinas al mismo tiempo.

Para poder diseñar un modelo de simulación de los procesos de intrusión y retracción de mercurio, se busca explorar los diferentes esquemas que existen en la programación paralela, y con los cuales se puede obtener un mayor rendimiento. Los esquemas investigados y más convenientes que se usaron en este trabajo fueron los esquemas de memoria compartida y de paso de mensajes. En esta tesis se presenta el diseño de un simulador paralelo de los procesos de intrusión y de retracción de mercurio en medios porosos basado en los esquemas de memoria compartida y de paso de mensajes. Este simulador es diseñado para explotar el paralelismo en clusters y sistemas multiprocesador y multicomputador. La primera versión del simulador explota los recursos de memoria compartida existentes en un sistema multiprocesador. Al usar memoria compartida se define un conjunto de hilos para llevar acabo las tareas de intrusión y retracción en una red porosa. Para este esquema variamos el número de hilos para encontrar la configuración a la cual se obtienen los mejores tiempos dependiendo del número de procesadores que se tengan. El esquema de memoria compartida se implementó con la herramienta OpenMP. La segunda versión del simulador explota los recursos de memoria distribuida de un sistema multicomputador. En esta versión la red porosa es parti-

cionada y en cada una de las sub-redes que resultan se tienen que llevar acabo los procesos de intrusión o retracción. Esto se logra con el esquema de memoria distribuida con ayuda de paso de mensajes, en donde se tiene un conjunto de nodos de un cluster que trabajan en cada una de las sub-redes generadas.

La estructura de esta tesis se organiza de la siguiente forma:

En el Capítulo 2 se presentan las características de un medio poroso así como el modelado en computadora, lo cual nos sirve para entender las estructuras que manejamos a lo largo del proyecto. En el Capítulo 3 se explica la porosimetría de mercurio y se muestran dos procesos importantes (intrusión y retracción) que se aplican en los medios porosos y sus características. En el Capítulo 4 se presenta una descripción general de las herramientas más usadas de programación paralela. Con esto se tiene una referencia de las herramientas existentes así como sus ventajas y desventajas para desarrollar el simulador. En el Capítulo 5 se describe el diseño de los algoritmos paralelos de intrusión y retracción de mercurio en sistemas multiprocesador y multicomputadores. En el Capítulo 6 se muestran las pruebas con las cuales se evaluaron los algoritmos y los resultados obtenidos de cada una de las pruebas realizadas, por último en el Capítulo 7 se presentan las conclusiones y el trabajo a futuro de esta tesis.

Capítulo 2

Materiales porosos y su modelado

En este capítulo se describe un material poroso, así como su modelado por computadora el cual es nombrado red porosa. Este modelado permite simular en computadora de una manera fácil y lo más parecido a lo real los materiales porosos.

2.1. Introducción

La mayoría de las substancias y materiales que se encuentran a nuestro alrededor, vistas a un nivel microscópico, presentan concavidades o huecos, llamados espacios no ocupados por materia como los materiales mostrados en la Figura 2.1.

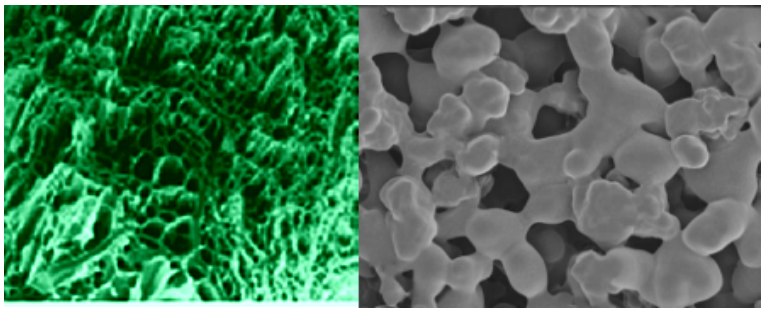


Figura 2.1: Vista microscópica de una superficie con poros.

Una gran variedad de sólidos naturales y sintéticos, los cuales son constituyentes de los suelos de cultivo, los mantos petrolíferos, los materiales de construcción o cementosos [4] (concreto, ladrillo, madera), los adsorbentes[5] (carbón activado, sílice, zeolitas), los catali-

zadores (procesos de reformación, polimerización), el papel, el cuero, etc., presentan huecos. Algunos de estos materiales son considerados altamente porosos, de ahí que sus propiedades en su textura determinan en gran manera su comportamiento. En la mayoría de los casos y aunque el material presente sea muy sólido como un metal, posee huecos en su estructura interna. Cuando estos espacios son lo suficientemente grandes como para ser invadidos por las moléculas de alguna otra substancia, en fisicoquímica, se dice que se tiene un material poroso. Investigadores han estado trabajando desde hace varias décadas en temas relacionados con la descripción, obtención, caracterización y estudio de las propiedades físico-químicas mecánicas, térmicas, eléctricas, adsorción física y química, durabi1idad, etc. de diferentes tipos de materiales en la naturaleza. Este tipo de investigaciones se realizan en base a modelos de simulación en los cuales se tienen presentes todas las propiedades de los materiales para su descripción. En esta tesis ocupamos un modelado sobre materiales porosos, que es descrito en la siguiente sección.

2.2. Modelado

Un medio poroso puede ser representado para su manejo en computadora mediante una estructura llamada red porosa. Una red porosa se genera al identificar los huecos o poros que presenta cualquier estructura, considerando que cada hueco está comunicado con otro hueco mediante concavidades internas. Para cada uno de los huecos se hace una representación mediante dos estructuras geométricas según el tipo de hueco. Los huecos más grandes dentro del material se representan con una esfera o sitio y los huecos más pequeños se representan con un cilindro o enlace como se ve en la Figura 2.2. Este modelado es llamado modelo dual de sitios y enlaces (DSBM)[3, 6].

Entonces se establece que un enlace está conectado a dos sitios de las esferas más cercanas y los sitios de igual manera a los enlaces, como un tipo de tubería de sitios y enlaces que representan todo el material y generan una red cúbica. Con esto se tiene un camino por todos los huecos del material representado por la red compuesta de esferas y cilindros. Al poder representar de esta manera un material, podemos caracterizar todos los huecos que estén

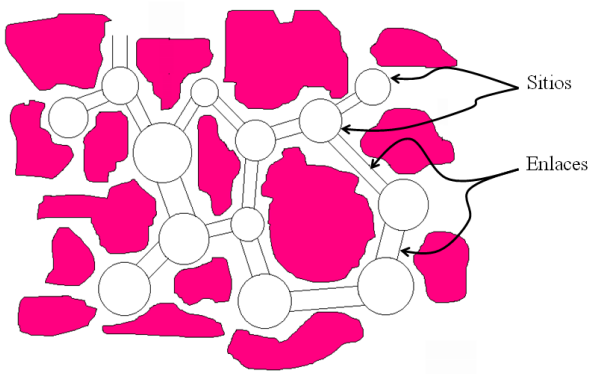


Figura 2.2: Representación del medio poroso mediante esferas y cilindros.

presentes.

2.2.1. Modelo dual de sitios y enlaces

En este modelado de materiales porosos por computadora, se busca representar todos los poros del material mediante una red de una geometría dada e.g. cúbica. En esta red se establecen dos tipos de huecos fundamentales los de mayor tamaño (sitios, cavidades) y los de menor tamaño (enlaces,cuellos capilares). Esta representación de una red porosa Figura 2.3 se conoce como el Modelo Dual de Sitios y Enlaces (DSBM) la cual fue diseñada por Mayagoitia y Kornhauser en la UAM Iztapalapa en 1984.

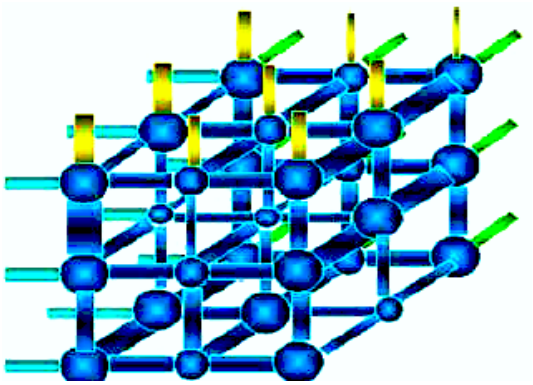


Figura 2.3: Representación de una red.

Este modelo permite de manera sencilla representar por computadora la construcción de

medios porosos de diversa naturaleza [7], siendo la meta principal al diseñar estas redes llevar acabo simulaciones de algunos fenómenos capilares que ocurren en materiales porosos.

2.2.2. Principio de construcción

Al diseñar el modelado, se intenta que la representación sea lo más parecida al material poroso real, por lo que se deben establecer principios en su construcción. Mayagoitia y Kornhauser establecieron los principios para la construcción de una red porosa, donde se debe de tomar en cuenta principalmente la morfología así como su topología de sitios y enlaces que la constituyen. El más importante de estos principios dice:

El tamaño de radio de un sitio debe ser mayor o cuando menos igual al tamaño de cualquiera del radio de los enlaces a los cuales está unido [9].

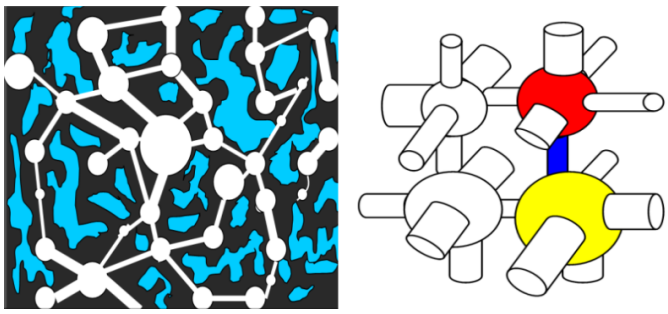


Figura 2.4: Representación del modelo dual de sitios y enlaces(DSBM).

Este principio se aplica sobre toda la red e impone una restricción sobre las funciones de distribución de tamaños de sitios y enlaces como se ve en la Figura 2.4. Así se asegura que haya un número suficiente de enlaces con radios de tamaños adecuados, menores a los tamaños de los radios de los sitios que conecta cada enlace. Este principio se establece porque se usan líquidos en los procesos que son simulados sobre las redes cúbicas modeladas. Si no se observase este principio de construcción, no sería posible interconectar un sitio con otro vecino a través de un el enlace que tuviera un mayor tamaño de radio mayor al de los dos sitios que se intentaran conectar. Por lo anterior se deben manejar distribuciones adecuadas de tamaños de sitios y enlaces, que garanticen que todos los sitios puedan interconectarse

por enlaces de menor o igual al tamaño. En esta tesis usamos este modelado para representar ciertos fenómenos que ocurren en medios porosos principalmente los procesos capilares. En el Capítulo 3 se describen algunos procesos capilares y de manera detallada los que intervienen en la porosimetría de mercurio: el proceso de intrusión y el proceso de retracción de mercurio que son los que simulamos en este trabajo de tesis.

Capítulo 3

Procesos capilares y porosimetría de mercurio

En este capítulo se presenta de manera muy general los procesos capilares que ocurren en los materiales porosos. Como se mencionó anteriormente estos procesos son de gran importancia en la industria, ya que pueden caracterizar materiales para utilizarlos de mejor manera. También se describe la porosimetría de mercurio y dos de sus procesos, la intrusión y la retracción. Estos procesos son pieza fundamental de esta tesis pues la simulación presentada en los siguientes capítulos se realiza sobre estos dos procesos.

3.1. Introducción

Cuando se trata de describir un sistema poroso, se hace en base a un modelo que contenga suficientes parámetros para que describa de manera adecuada al sólido que se pretende estudiar. Algunos de los parámetros que generalmente se toman en cuenta para una descripción global son: la densidad real que es la densidad del sólido en sí, la densidad aparente o densidad del medio poroso en conjunto, y su porosidad que es la relación entre el volumen del hueco y el volumen total. De estos tres parámetros, la porosidad es la que nos importa para estudiar un material poroso que es invadido con una substancia, con la cual se obtienen rasgos únicos de su estructura interna. Cuando un fluido invade un material poroso (por sus huecos) experimenta una variedad de fenómenos fisicoquímicos, los cuales son conocidos

como procesos capilares.

3.2. Procesos capilares

En los procesos capilares dos o más fluidos compiten por la posesión del espacio poroso libre. Sin embargo, no todos los líquidos se utilizan en este tipo de procesos, son usados sólo algunos que cumplen con propiedades específicas tales como la no adsorbancia.

Algunos ejemplos de procesos capilares son:

- Imbibición - llenado espontáneo de la estructura porosa por un líquido que moja al sólido.
- Drenaje - rechazo no espontáneo del líquido anterior de la estructura porosa mediante un gas bajo presión.
- Intrusión o penetración - llenado no espontáneo de una estructura porosa por un fluido que no moja al sólido.
- Retracción - rechazo espontáneo del fluido anterior de la estructura porosa al aliviar la presión del mismo fluido.
- Condensación capilar - transformación espontánea de un vapor en líquido dentro la estructura porosa.
- Evaporación capilar - transformación espontánea de un líquido en vapor dentro de la estructura porosa.

Con procesos capilares se obtiene una gran variedad de fenómenos naturales y de procesos industriales. Específicamente involucran a procesos fisicoquímicos tales como:

1. El mojado y secado tanto en suelos de cultivo como en acuíferos, en la industria química, textil, alimenticia, etc.

2. La recuperación de petróleo, los yacimientos geotérmicos (flujo de un líquido y su vapor).

3. Técnicas de porosimetría de penetración de mercurio (intrusión y retracción).

4. Procesos de adsorción y desorción de gases (condensación y evaporación capilar).

De esta lista de procesos capilares la porosimetría de mercurio es muy importante en la industria ya que este tipo de técnicas sirven para caracterizar materiales que se ocupan cotidianamente. La manera en que se caracteriza un material es por medio de su distribución de poros, con ayuda de la intrusión y retracción de mercurio. El mercurio es utilizado debido a que es un material con ciertas características especificas, pues no se adsorbe, no daña la estructura interna del material, ni la suya propia.

En la siguiente sección se describe la porosimetría de mercurio, de la cual se presentan los procesos de intrusión y retracción de mercurio a detalle.

3.3. Porosimetría de mercurio

La porosimetría de mercurio se considera como un fenómeno capilar, este fenómeno está conformado por los procesos de intrusión (penetración) y retracción (extrusión) de mercurio. Estos fenómenos consisten en forzar al mercurio a penetrar y desalojar las cavidades porosas, aumentando o disminuyendo respectivamente la presión en todo el material.

La porosimetría de mercurio es una técnica muy eficiente con la que se puede evaluar los tamaños de poros en un material. Algunos de los ejemplos donde se utiliza es en la industria productora de cementos, en donde la porosimetría ayuda a determinar el tiempo de curado, así como las proporciones agua/cemento en las formulaciones. La porosimetría también sirve para determinar el poder que tiene algún impermeabilizante [10], en su grado de hidratación, así como la fatiga, en materiales. Con la porosimetría se puede estudiar la fragilidad de los huesos provocada por una pérdida de proteínas y alteraciones en sus minerales. Otra aplicación es en productos farmacéuticos sólidos, aerogeles y otros reactivos, en la recuperación del aceite

en filtros adsorbentes, etc. Para poder llevar acabo la porosimetría de mercurio en estos productos y evaluar propiedades como tamaño de poros, se utilizan los procesos de intrusión y retracción de mercurio.

A continuación se describen a detalle los procesos de intrusión y retracción de mercurio.

3.3.1. Proceso de intrusión

La intrusión también llamada penetración de mercurio[12], es una técnica que tiene como objetivo invadir una estructura porosa con mercurio, por todos sus huecos o poros mediante un incremento gradual de la presión[13]. Inicialmente se rodea con mercurio todo el material poroso (libre de mercurio en su interior) como se ve en la Figura 3.1 (a), de esta manera se garantiza que el material puede empezar a ser intruído con el mercurio por todas sus fronteras o partes externas. Una vez que el material esta rodeado incluso por los poros exteriores, el mercurio comienza a ser intruído, siendo forzado a entrar en el material a una cierta presión inicial.

Posteriormente mediante un aumento gradual de la presión se va intruyendo periódicamente el material hacia la parte central de este (el interior) hasta donde la presión actual lo permita. El material no se intruye todo a una cierta presión ya que existen diferentes tamaños de radios de sitios y enlaces (que respetan el principio de construcción), los cuales son penetrados a una presión específica. Regularmente los sitios y enlaces de mayor tamaño de radio son intruídos con presiones menores, y los de menor tamaño de radio requieren una mayor presión. Se dice que el material está totalmente intruído cuando todos los poros (sitio o enlace) presenten mercurio en su interior como se ve en la Figura 3.1(b).

Al llevar acabo este proceso la cantidad de operaciones depende del número de poros en la red [14], Si se tiene una red cúbica de tamaño L el número de sitios queda expresado por:

$$L * L * L = L^3. \tag{3.1}$$

Como cada sitio presenta 3 enlaces el número de sitios y enlaces es:

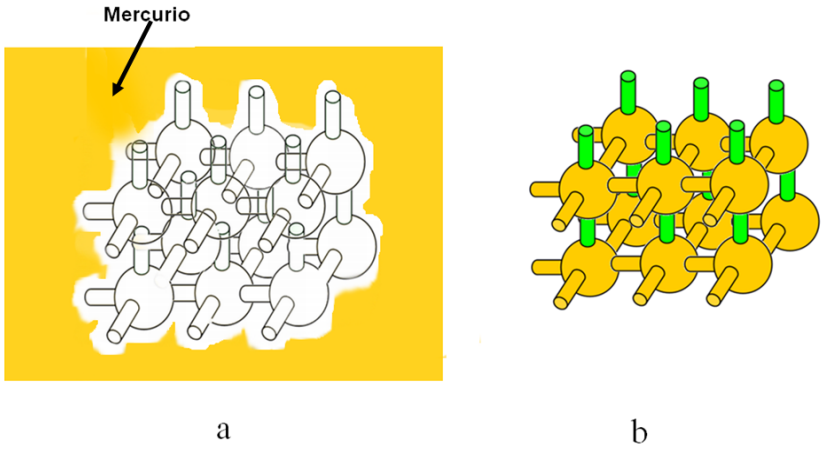


Figura 3.1: Proceso de intrusión a) red vacía. b) red llena por completo de mercurio.

$$L^3 * 3 = 3L^3 \tag{3.2}$$

Durante el proceso de intrusión ocurren diferentes fenómenos los cuales ocurren por el movimiento del mercurio al penetrar los poros del material. Uno de estos fenómenos es la *cantotaxis*, la cual se debe a la resistencia del mercurio al pasar de un hueco pequeño a uno grande (i.e. de un enlace a un sitio); el menisco inicialmente se ancla en el vértice de la conexión sitio-enlace y empieza a hincharse conforme la presión del mercurio aumenta, cuando se alcanza una presión crítica el mercurio invade el sitio. En otras palabras, la resistencia a penetrar el sitio con mercurio es muy grande inicialmente, y el mercurio logra pasar hacia el siguiente poro, cuando la presión es suficientemente alta (Figura 3.2), este fenómeno puede presentarse en diversas partes del material.

Algoritmo de intrusión

Los pasos para llevar acabo el proceso de intrusión son los siguientes:

1. Invasión de enlaces periféricos: Se rodea el medio poroso con mercurio proveniente del exterior, para que por todos los enlaces periféricos se intente intruir el mercurio. Se incrementa la presión hasta un valor correspondiente al radio crítico R_c (El radio crítico es la presión máxima de invasión del sitio; es decir, cuando se vence la cantotaxis).

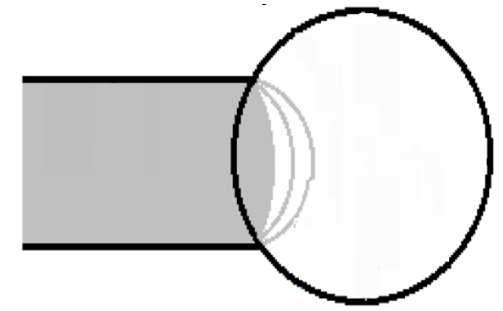


Figura 3.2: Fenómeno de cantotaxis.

Se examinan las caras externas del medio poroso; de tal forma que todos los enlaces periféricos de tamaño de radio mayores o iguales al R_c son invadidos con mercurio.

2. Invasión de sitios: Se invade un sitio con mercurio desde alguno de los enlaces que lo rodean, siempre que esté conectado a la fuente externa de mercurio sobre una trayectoria continua de mercurio. Una trayectoria continua de mercurio es un camino de sitios y enlaces interconectados llenos de mercurio. Como se mencionó anteriormente los sitios son penetrados con mercurio considerando el fenómeno de cantotaxis[15].

3. Invasión de enlaces internos: Los enlaces son intruídos con mercurio de acuerdo a la ecuación de Washburn:

$$P^l = \frac{2\sigma^{lv} cos\theta}{R_B} \tag{3.3}$$

Donde P^l es la presión que se requiere para intruír el mercurio en un poro cilíndrico (enlace) de radio R_B; θ es el ángulo de contacto entre el mercurio y la superficie sólida y σ^{lv} es la tensión interfacial entre el mercurio y su vapor

Es decir, cuando sus tamaños son mayores o iguales que el radio crítico en ese momento, y únicamente si hay una trayectoria continua a la fuente de mercurio.

Estos tres pasos para la intrusión se realizan sobre la red porosa en todos sus poros. Sin embargo, el tiempo que toma en llevarse acabo la intrusión es grande, sobre todo en redes de gran tamaño por el número de elementos descritos en la ecuación 3.2. En trabajos anteriores se ha simulado el proceso de intrusión mediante algoritmos secuenciales sobre redes porosas

bajo el modelo DSBM. Sin embargo, el tiempo en que se lleva acabo el proceso es grande. La razón de esos tiempos es debido a que los poros los recorren de manera exhaustiva uno por uno verificando las condiciones con las que se lleva acabo la intrusión. Al verificar las condiciones el algoritmo esperaba a que se recorrieran todos los enlaces externos de alguna cara para comenzar con los sitios asociados y gradualmente verificar las condiciones para los poros internos. Si la red es demasiado grande al recorrer cada uno de los poros a una presión y al verificar las condiciones, se consumen grandes tiempos de procesamiento y de memoria.

Este algoritmo se mejoró para este trabajo de tesis y se diseñó un algoritmo recursivo para simular el proceso de intrusión de mercurio. En este algoritmo no se recorre de manera exhaustiva uno a uno los poros de la red para ver si cumplen con las condiciones y después profundizar por los poros de la red, sino que desde el primer poro que se comienza a intruir se busca profundizar hacia los poros internos de la red. De esta manera, ya se conoce una posible ruta por donde se puede intruir a una cierta presión, y cuando se intente intruir por el siguiente poro se tiene información de los caminos por donde se puede intruir la red. Con estos caminos se crea una posible ruta que al juntarse con otras rutas de los poros del material, agilizan el proceso de intrusión. El pseudocódigo de este algoritmo se describe en el Apéndice 1.

3.3.2. Proceso de retracción

El proceso de retracción o extrusión, también denominado proceso de inhibición (en el casao de líquidos que mojen la superficie La superficie sólida) es cuando se retira el componente ajeno (en este caso el mercurio) a un sistema mediante una disminución de presión[12] (Figura 3.3a)). Este proceso se lleva acabo una vez que se ha finalizado el proceso de intrusión, sin importar si la red está saturada o no de mercurio. La idea principal para el proceso de retracción, radica en que en la red exista al menos una trayectoria continua de mercurio hacia el exterior. Este camino debe de existir en cada uno de los poros que se encuentren dentro de la misma red, para que el vaciado de mercurio sea completo (Figura 3.3b)). Sin embargo existen fenómenos que se producen durante el vaciado de mercurio en la red, por ejemplo el

snap-off.

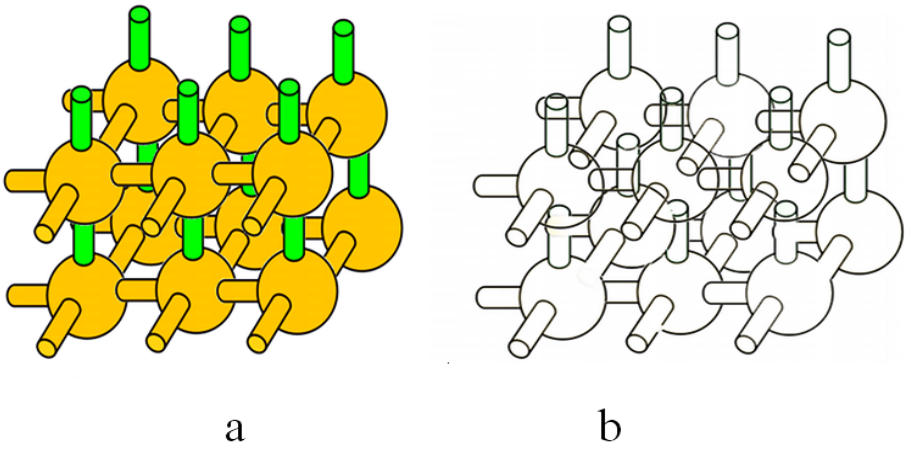


Figura 3.3: Proceso de retracción a) red llena por completo de mercurio. b) red vacía.

Snap-off

El fenómeno de Snap-off (Figura 3.4) ocurre en los enlaces cuando durante el proceso de intrusión quedaron espacios de aire muy comprimidos [16], debido a la presión con la que se intruye el mercurio. Al momento de comenzar con la retracción esos espacios con aire, comienzan a aumentar hacia cualquier dirección al disminuir la presión.

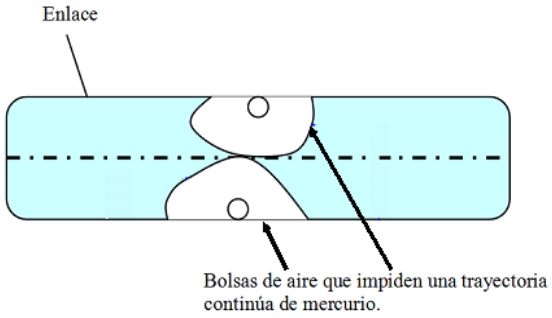


Figura 3.4: Representación del snap off.

Si los espacios con aire son demasiado grandes y dos de ellos se encuentran dentro de un mismo enlace, existe la posibilidad de que el enlace quede con un hueco de tamaño igual a su

radio. Si esto ocurre ya no se cumple la condición inicial para que exista un camino continuo de mercurio, y por lo tanto el enlace no podría vaciarse. El tamaño de radio presente en esta situación es llamado radio crítico $R_{Csnap-off}$. En estos casos el material queda con partes de mercurio que no se pudieron extraer y se les denomina islas[17] como se ve en la Figura 3.5. No obstante, conforme la presión sobre el mercurio se disminuye, el tamaño de las bolsas de aire aumentan y en algunos casos comienzan a ser inestables y se rompen, al romperse las islas se vacía el enlace de mercurio pues ya existe la trayectoria continua de mercurio.

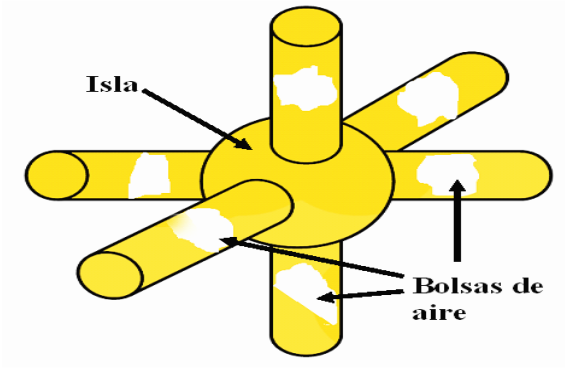


Figura 3.5: Representación de una isla.

Algoritmo de retracción

El algoritmo de retracción es el siguiente:

1. Se fija un radio crítico $R_{Csnap-off}$, (radio crítico hasta donde sucede el estrangulamiento o bloqueo) en donde ciertos enlaces de tamaño R menor que $R_{Csnap-off}$ pueden vaciar el mercurio por el libre paso en sus paredes. La fuente de mercurio debe tener al menos una trayectoria continua de mercurio hacia el exterior.

2. Si se cumple la condición de que R_C es mayor que $R_{Csnap-off}$, los poros de la red se vacían de la siguiente manera:

   - Un enlace lleno se vacía por un extremo si está conectado a un sitio vacío y por el otro a un sitio lleno de mercurio el cual conserva al menos una trayectoria continua de mercurio hacia el exterior de la red.

- Un sitio lleno de radio menor que R_C, se vacía si posee al menos uno de sus enlaces vacío y si está conectado al exterior por al menos una trayectoria continua de mercurio.

Este algoritmo no se encontraba implementado anteriormente para ser simulado por computadora, por lo que usando el mismo principio del algoritmo de intrusión recursivo, se diseñó el algoritmo para la retracción. El pseudocódigo de este algoritmo se describe en el Apéndice 1. El algoritmo de retracción recursivo es mas costoso que el de intrusión, debido a que en este proceso ocurren ciertos fenómenos como por ejemplo el snap-off y las islas que se pueden generar.

Estos dos fenómenos de la porosimetría de mercurio son estudiados en esta tesis y son simulados en computadora. En la simulación, un inconveniente es que por el número de poros (del orden de millones) presentes en la red se ocupan grandes cantidades de memoria para representar los datos. Por otro lado el tiempo de procesamiento es alto pues las operaciones realizadas son del orden del número de poros presentes. Cuando se lleva acabo el proceso de intrusión o de retracción se tienen que verificar 3 condiciones importantes y la ecuación de Washburn, por lo que se tienen 4 operaciones que se realizan por poro de la red. La cantidad de poros en una red de tamaño L mencionados en la ecuación 3.2, asignada a las 4 operaciones, indica que se procesan los siguientes datos:

$$Operaciones = 12 * L^3 \tag{3.4}$$

Este proceso se repite pues en cada iteración se actualiza el valor de presión, aumentándola para que se logre penetrar por todos los poros posibles de la red. Como existen redes de tamaño L= 300, se tienen que procesar $12*300^3 = 8100000000$ operaciones de punto flotante, por lo que el tiempo en procesamiento es alto. Una red de este tamaño ocupa gran cantidad de memoria y la simulación de los procesos a diferentes presiones en algunos casos se consume la memoria en su totalidad. Redes generadas de tamaño de 300 sitios por lado, ocupan un espacio de memoria física de 500 mb y al simular los procesos de intrusión o retracción se consume un total de 4 Gb en RAM por lo que en algunos sistemas se consume en su totalidad

la memoria RAM. Una de las maneras con las que se puede atacar este tipo de problemas es mediante el diseño de algoritmos paralelos.

En el Capítulo 4 se describen algunas de las herramientas de programación paralela de uso común. Estas herramientas permiten implementar algoritmos de manera paralela.

Capítulo 4

Arquitecturas y herramientas de programación paralela

En este capítulo se presentan algunas arquitecturas paralelas, que surgen por la necesidad de mayores velocidades de procesamiento en los sistemas de cómputo. De la misma forma se describen diferentes herramientas de la programación paralela, que explotan los recursos disponibles en una o varias computadoras. Para estas herramientas se presentan sus ventajas así como las desventajas principales para poder diseñar los algoritmos de simulación descritos en el capítulo anterior.

4.1. Introducción

Una computadora puede ser catalogada de alta eficiencia, dependiendo del tiempo en ejecutar una serie de instrucciones, tomando en cuenta el número de datos presentes. Durante décadas se ha buscado una mejor eficiencia en cuanto al tiempo en que se procesan los datos en una computadora, supercomputadoras, mainframes etc. En base a esto, se diseñan arquitecturas que dependen de los avances tecnológicos para un buen rendimiento, sobre todo, en los componentes de los sistemas de cómputo como los microprocesadores. En los procesadores lo que delimita el tiempo para ejecutar cualquier tipo de instrucción, radica en los ciclos del procesador. Sin embargo, aunque el procesador tenga de las mejores tecnologías, depende mucho del tipo de aplicaciones que se manejen lo que hace que su tiempo de ejecución se

incremente. Aplicaciones de áreas de dinámica molecular, de química, de modelado en proteínas requieren del máximo poder de cómputo o procesamiento, por esto se han diseñado sistemas en donde se ocupan varios procesadores simultáneamente. Al ocupar más de un procesador, se tienen sistemas de cómputo paralelo[18], en donde un problema es dividido en varias tareas que se asignan a diferentes procesadores y así se aumenta el poder de cómputo. Con la aparición del cómputo paralelo se pueden resolver problemas considerados complejos, en los que se requiere de muchos recursos computacionales como son la memoria y el procesador. Para el cómputo paralelo existen diferentes arquitecturas, así como herramientas de programación con las que se extienden sus capacidades. En el desarrollo de un simulador paralelo como el que se propone en este trabajo, se debe buscar la mejor manera de tener un sistema en el cual su meta principal sea reducir el costo de tiempo de ejecución, utlizando las herramientas apropiadas.

En la siguiente sección, se hace una breve descripción de arquitecturas y herramientas de programación paralela, que nos sirven para desarrollar aplicaciones en donde se ejecuten de manera simultánea varias tareas.

4.2. Arquitecturas paralelas

La necesidad de establecer diferentes tipos de arquitecturas paralelas, surge por la amplia variedad de aplicaciones. En estas aplicaciones se requiere de algún sistema específico así como de más recursos computacionales como memoria y procesamiento. La meta principal al diseñar estas arquitecturas, es ejecutar varias tareas de manera simultánea, para así obtener soluciones más rápidas a problemas complejos. Los problemas complejos o de mayor tamaño son aquellos que requieren más ciclos de reloj en su ejecución.

La clasificación de arquitecturas paralelas de Flynn Michael [19], en la década de los setentas, muestra como se pueden agrupar el número de instrucciones así como el flujo de datos que es utilizado para el procesamiento paralelo de la información. En esta clasificación encontramos los siguientes tipos. SISD (Single Instruction and Single Data Stream), este modelo sigue la manera tradicional de la programación secuencial,en donde una cadena de

instrucciones es ejecutada, una a una sobre una secuencia de datos extraídos de la memoria. SIMD (Single Instruction and Multiple Data Streams) como su nombre lo dice se tienen varios procesadores que están ejecutando de manera sincróna la misma instrucción, pero la diferencia fundamental es que los datos se recuperan en su propia memoria local. MISD (Multiple Instruction and Single Data Stream) en este caso se pueden ejecutar múltiples instrucciones en un solo procesador. MIMD (Multiple Instruction and Multiple Data Stream), en este caso un grupo de procesadores pueden ejecutar diferentes instrucciones a sus datos.

Hoy en día la clasificación de Flynn queda como una referencia básica, ahora las arquitecturas pueden separarse en dos grandes grupos: multiprocesadores y multicomputadores. En las arquitecturas multiprocesador se tienen varios procesadores ocupando el mismo espacio de memoria haciendo este un sistema de memora compartida. En las arquitecturas multicomputador se tiene un conjunto máquinas con su propio espacio de memoria y utilizan paso de mensajes para la comunicación.

En las siguientes secciones se describen estos dos sistemas.

4.2.1. Multiprocesadores

Un sistema multiprocesador consta de varios procesadores que ejecutan instrucciones en un solo espacio de direcciones, por lo que son considerados como sistemas de memoria compartida (Figura 4.1).

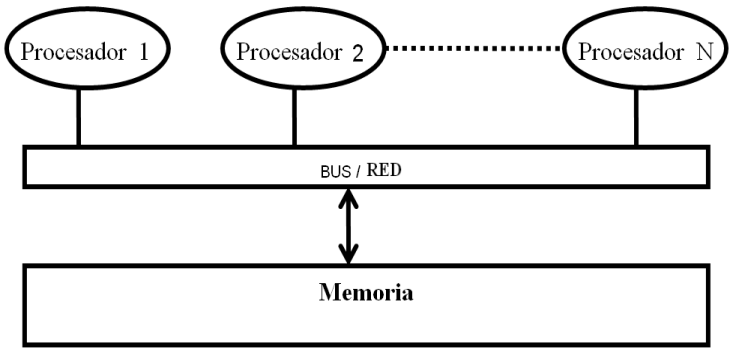


Figura 4.1: Multiprocesador.

Estos sistemas realizan la comunicación de procesadores mediante lectura y escritura en areas compartidas de memoria con tiempos de acceso similares. No obstante, esta comunicación o acceso tiene dos variantes, la primera llamada UMA (Uniform Memory Access) y la segunda NUMA (No Uniform Memory Access). En las UMA los procesadores tienen el mismo tiempo de acceso a la memoria. De esta forma los diferentes procesadores pueden leer y escribir en cualquier parte de la memoria. Sin embargo, este tipo de arquitecturas no es escalable ya que si aumentamos el número de procesadores resulta muy costoso al aumentar también el número de conexiones. No obstante, con las arquitecturas NUMA se puede tener mejor escalabilidad que en las UMA. En esta arquitectura los accesos a memoria del procesador hacia los datos pueden ser locales o remotos . En el acceso remoto existe un bus o red de interconexión que puede hacer más costoso el tiempo el acceso, no obstante en los accesos locales existe un medio directo entre el procesador y la memoria que reduce el costo en tiempo. Algunas arquitecturas donde se utiliza NUMA son en las T3D así como la Cray [20]. En las computadoras Pentium y en distribuciones de Sun Enterprise Server se utiliza UMA

Con estos dos sistemas es posible crear aplicaciones que exploten todos los recursos de las computadoras, y usar modelos de programación dependiendo del tipo de aplicaciones que diseñen.

4.2.2. Multicomputadores

Los sistemas considerados como multicomputadores, se conforman de varios procesadores o nodos, interconectados entre ellos por alguna red como se muestra en la Figura 4.2. Un nodo puede ser a su vez una máquina monoprocesador o bien multiprocesador. El sistema multicomputador es considerado de memoria distribuida ya que cada nodo tiene su propio espacio de direcciones.

En el sistema de multicomputador la comunicación entre cada procesador ocurre mediante intercambio de mensajes, esta comunicación puede ser para sincronizar procesadores los cuales trabajan sobre una aplicación para intercambiar resultados. Una ventaja de estos

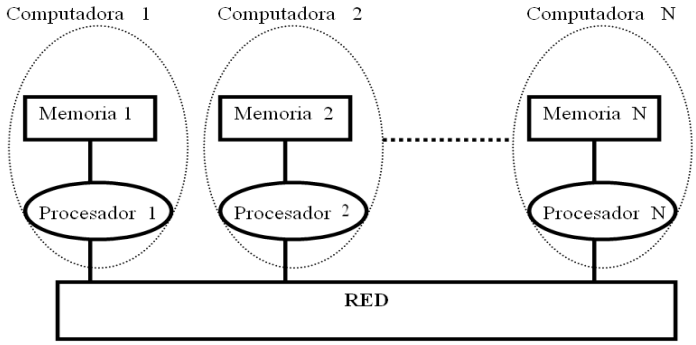


Figura 4.2: Multicomputador.

sistemas es la escalabilidad, pues se pueden añadir más procesadores o nodos con los cuales se obtiene más poder de cómputo al tener más unidades de procesamiento. Los sistemas de varios procesadores pueden ser homogéneos (del mismo tipo), o heterogéneos (con diferentes características). En la siguiente sección se describen de manera breve los modelos básicos para programar en esta arquitectura, así como algunas herramientas para diseñar los algoritmos del trabajo de tesis.

4.3. Modelos y herramientas de programación paralela

Al diseñar un programa paralelo se deben tomar en cuenta tres pasos importantes:

El primero es dividir el problema en varias tareas. Aquí se identifica el problema y se busca la manera en la cual puede ser particionado. Se debe de respetar cada uno de los requerimientos del problema original en cada partición para que el resultado sea el mismo.

El segundo paso es organizar la comunicación y la sincronización. En este paso dependiendo del modelo de programación, es como se lleva acabo la comunicación entre los procesadores que realizan las tareas para intercambiar resultados o para sincronizarse. Al final se asignan las tareas a los procesadores, considerando la comunicación entre procesos para su ejecución.

El tercer paso es asignar el trabajo a un grupo de entidades o nodos. En este paso cada una de las particiones del programa es distribuida y ejecutada simultáneamente por un conjunto de procesadores, según los recursos que se tengan. Al procesar las tareas en diferentes nodos

Modelo de Memoria Compartida	*Modelo de Paso de Mensajes*
Pthreads	*PVM*
OpenMPM	*MPI*

Tabla 4.1: Ejemplo de herramientas de Programación Paralela y su Modelo de Programación.

se puede llegar a la solución en un tiempo menor que si se ejecutara en una sola máquina.

Los modelos de programación paralela presentan una relación directa con el tipo de modelo de la arquitectura en que se ejecutan. En arquitecturas multicomputador se puede utilizar el modelo de paso de mensajes. En este modelo la comunicación entre los procesadores de cada uno de los nodos presentes se lleva acabo mediante el envío y recepción de mensajes. Este modelo es muy bueno ya que las computadoras que conforman el sistema multicomputador cuentan cada una con su espacio de memoria y su propio procesador. En el tipo de arquitectura multiprocesador se puede usar el modelo de programación de memoria compartida. En este tipo de modelo mediante el acceso a un espacio de direcciones compartida, es como se lleva acabo la comunicación entre cada uno de los procesadores presentes. Mediante la lectura y escritura de variables compartidas se realiza la cooperación y comunicación entre los procesos lo cual es una ventaja pues se facilita la programación.

En base a estos dos modelos de programación identificamos algunas herramientas de programación paralela que más se utilizan en la actualidad. Estas herramientas pueden clasificarse en base al modelo de comunicación que usan: de memoria compartida y paso de mensajes. En la Tabla 4.1 se muestran algunas de estas herramientas.

4.3.1. Herramientas de programación con memoria compartida: Pthreads y OpenMP

Pthread (hilos)

Un hilo es una instancia que se encarga de ejecutar una tarea en algún programa. Varios hilos pueden coexistir en el mismo proceso, compartiendo descriptores de archivos, tablas de manejadores de señal y la memoria principal del proceso. Por lo cual las operaciones de

cambio de contexto entre los hilos no son muy costosas. En un programa se pueden utilizar múltiples hilos los cuales pueden encargarse de ejecutar diferentes tareas. Los hilos pertenecen al modelo de programación de memoria compartida ya que cada uno de ellos comparte el mismo espacio de memoria al realizar sus tareas. La comunicación que ocurre entre dos o más hilos se realiza leyendo y escribiendo en el espacio de memoria compartida en el cual se encuentren. Una de las ventajas al estar implementando este tipo de estructuras y llevar acabo las operaciones de forma paralela es el costo pequeño de las comunicaciones[20]. Sin embargo, al utilizar los hilos se necesita un control en la manera de que ellos acceden a la memoria principal para escribir, por lo que deben de hacerlo de manera sincronizada. La librería Pthread [20] es una herramienta que permite crear diferentes hilos de ejecución y los mecanismos más usados para sincronizarse estos hilos son los candados [21].

Algunas de las ventajas que podemos tener en los hilos, es que se implementan de manera muy sencilla, ya que los hilos pueden correr en casi cualquier sistema operativo, solo basta con manejar su biblioteca. Los hilos son muy eficientes dado que mejoran el desempeño en un programa al planificar la ejecución de tareas de manera paralela, dependiendo del algoritmo que se tenga. Los hilos pueden ser usados en varios lenguajes de programación como Java, C, C++, Python, PHP etc[21]. En sistemas operativos como Linux los hilos están fuertemente relacionados en la comunicación y sincronización de las tareas que realiza el usuario, ya que aumentan la planificación de los procesos que se manejan en su kernel [30]. En sistemas operativos como Windows NT y Server 2003 se implementaron para mejorar la planificación de los procesos, a nivel de comunicación en red.

OpenMP (Open Multi-Processing)

OpenMP es un conjunto de directivas y rutinas en una biblioteca, que se usan para desarrollar el cómputo paralelo[22]. OpenMP se usa para implementar el paralelismo dentro de un programa secuencial típico, identificando las secciones donde puede dividirse una tarea en subtareas paralelas. Esta herramienta se diseño específicamente para ser usada donde el procesamiento se lleva acabo mediante el modelo de memoria compartida. En OpenMP se

intenta aprovechar al máximo la manera de asignar el trabajo a diferentes hilos, esto lo realiza el programador identificando las regiones que podrían ser paralelas como se ve en la Figura 4.3.

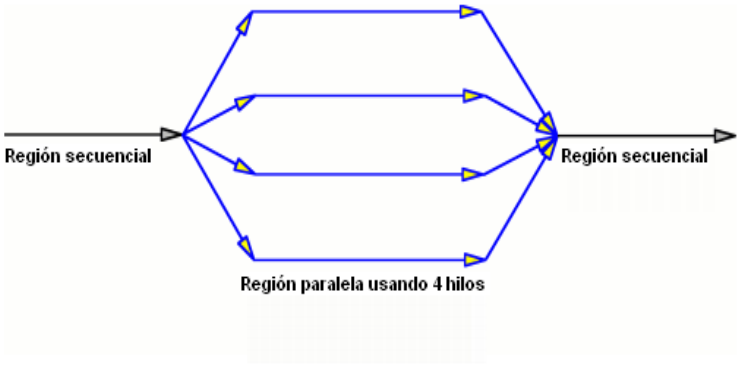


Figura 4.3: Manejo de hilos en OpenMP.

OpenMP está compuesta de dos partes:

- Un grupo de primitivas que son usadas por los programadores para dar las directivas de compilación.
- El otro elemento son el número de variables de entorno limitado, que definen los parámetros del sistema en paralelo. Estos parametros pueden ser los hilos, o regiones paralelas.

Las operaciones comúnmente usadas por OpenMP son principalmente para definir una región paralela y los puntos de sincronización. A continuación se mencionan algunas de ellas

- pragma omp critical : Define una región crítica.
- pragma omp parallel : Se indica que el bloque que le sigue será ejecutado por todos los hilos activos.
- pragma omp parallel sections : Indica las secciones de código que se paralelizan.

OpenMP se usa en computadoras que requieren de un alto rendimiento de sus multiprocesadores. OpenMP se ha puesto en práctica en muchos compiladores comerciales como visual C++ y los compiladores de Intel, también en plataformas no comerciales como Linux (C, C ++, y Fortran).

4.3.2. Herramientas de programación de paso de mensajes PVM y MPI

PVM (Parallel Virtual Machine)

PVM es una máquina virtual que permite crear tareas paralelas dinámicamente, para ello, fragmenta dinámicamente la aplicación y ejecuta las tareas en diferentes máquinas virtuales[24]. PVM se utiliza en monoprocesadores, en máquinas SMP y también en clusters que están conectados en una red[25].

La librería de PVM se basa en dos partes:

El demonio pvmd (pvmd3)

Este demonio es un proceso que se ejecuta en cada una de las máquinas de la máquina virtual, se encarga de la comunicación entre tareas y la conversión de datos entre máquinas[26]. Todas las acciones en el entorno de programación se realizan a través de llamadas a funciones. Las primitivas que permiten crear procesos y establecer la comunicación por mensajes son a través de la biblioteca de primitivas (libpvm3).

La librería de las primitivas de comunicación

Estas funciones son diversas y podemos encontrar aquellas que permiten el envió y recepción de mensajes entre tareas, la coordinación de tareas, inicialización de tareas, modificación de la máquina virtual etc.

PVM se basa en la noción de una aplicación con múltiples tareas, cada una de las tareas es identificada con un TID (task identifier), para que el envió y la recepción de mensajes se

realicen utilizando los tids. Las funciones principales de PVM para el envió y recepción de mensajes son:

- pvm_send: Envia los datos
- pvm_recv: Recibe los datos

MPI (Message Passing Interface)

MPI es una interfaz estandarizada para ser usada en aplicaciones paralelas y está basada en paso de mensajes [27]. Esto último debido a que las tareas dentro de MPI regularmente tienen un espacio de memoria completamente separada. MPI se diseñó para crear aplicaciones del estilo de programación SPMD (Single Process, Multiple Data). La sincronización así como el intercambio de datos, se hacen por medio de paso de mensajes. Se tienen funciones de comunicación punto a punto entre dos procesos, así como de funciones que involucran a múltiples procesos. Esta herramienta usa un modelo de programación fragmentada, es decir, cuando un programa es fragmentado en varios subprogramas estos se ejecutan simultáneamente. En MPI existen dos modelos de comunicación para el paso de mensajes: síncrona y asíncrona. En el paso de mensajes síncrono, el proceso que envía el mensaje espera a que otro proceso lo reciba para seguir su ejecución. Dentro del paso de mensajes síncrono se usan llamadas a procedimientos remotos, y se usan normalmente para arquitecturas cliente/servidor. En el paso de mensajes asíncrono, existe un proceso que envía y no espera a que el mensaje sea recibido para continuar la ejecución de una tarea, generando nuevos mensaje y enviándolos antes de que se haya recibido el anterior. En este esquema se emplea un buffer, en el cual se almacenan los mensajes. La manera en la que se sincroniza MPI es mediante el paso de mensajes explícitos en ciertas funciones. Las funciones de recepción de mensajes son en las que se encuentra la operación de sincronización con el transmisor, ya que es el que espera que haya un mensaje disponible. En MPI se tiene un comunicador que es un grupo de procesos y su contexto de comunicación[28], éste se crea para que las comunicaciones entre dos comunicadores nunca interfieran entre sí. De esta manera no existe riesgo de que se mezclen mensajes del usuario, con mensajes de la biblioteca que son privados ya que se pueden ejecu-

tar funciones de la biblioteca en un comunicador, y en otro diferente el programa del usuario. Las funciones usadas por MPI para el envió y recepción de mensajes entre dos tareas son:

- MPI_Send: Maneja el envío de datos.
- MPI_Rec: Maneja la recepción de datos.

Aplicaciones donde se usa MPI

MPI se implementa principalmente en sistemas donde la memoria esta distribuida entre varias máquinas, además de que se usa el paso de mensajes para la comunicación y sincronización.

En máquinas y equipos de cómputo paralelo esta herramienta es muy efectiva ya que proporciona una interfaz simple y operaciones fáciles de manejar para el usuario.

4.3.3. Herramientas utilizadas en este proyecto

Al analizar las diferentes herramientas de programación que se usan hoy en día, se logra una buena estimación en cuanto al rendimiento que proporcionan ya que se puede reducir los tiempos de ejecución y el uso en memoria. Uno de los objetivos de esta tesis es desarrollar una aplicación paralela la cual explote los recursos de sistemas multiprocesador y multicomputador, así como mejorar el desempeño de la aplicación a desarrollar para reducir el tiempo de ejecución. La aplicación a desarrollar consiste en simular de manera paralela los procesos de intrusión y retracción de mercurio descritos en el Capítulo 3. Los procesos ocurren en una red porosa mencionada en el Capítulo 2. El simulador para estos dos procesos se puede crear ocupando algunas de las herramientas de programación de memoria compartida así como de paso de mensajes mencionadas anteriormente. A continuación se describen las herramientas elegidas para la realización del simulador.

La primera herramienta de programación paralela que se utiliza es OpenMP, con esta herramienta usamos la red como un solo espacio de memoria.

Una de las ventajas de OpenMP es que su modelo de programación paralela es mucho más estructurado que Pthreads. OpenMP es capaz de definir secciones críticas en el código

donde varios hilos pueden escribir un valor, mientras que con pthread deben de ser codificados directamente en la parte del código donde ocurran. Con Pthreads puede que en su implementación todos los subprocesos de trabajo están durmiendo (pthread_wait) hasta que el hilo maestro los despierta, mientras que en OpenMP se maneja de manera automática.

En OpenMP se hacen los ciclos for paralelos con primitivas especializados, mientras que con Pthreads no se tiene esta ventaja aunque se puede implementar. En general, el desarrollo de aplicaciones con OpenMP es mucho más rápido.

Con la ayuda de los hilos que implementa OpenMP podemos llevar acabo los procesos de intrusión y de retracción de mercurio en diferentes lados de la red cúbica. Con OpenMP podemos identificar secciones de código en las cuales ocurren ciclos o iteraciones y paralelizarlas. Debido a que los procesos a simular requieren de iteraciones por aumento o disminución de presiones al simularlos (descritas en el Capítulo 3) resulta ideal ocupar esta herramienta de programación.

Tras haber revisado las funciones y características de ambas herramientas, MPI es ligeramente más compacto, pues require menos código que PVM. Por ejemplo, la realización de un broadcast en MPI requiere únicamente una llamada a función, mientras que en PVM se requieren dos (una para empacar la información y otra para realizar el envío).

PVM es un proyecto con más antiguedad, la API de PVM es más compleja, en ocasiones requiriendo un número de llamadas mayor para ciertas funciones sencillas. Ambas bibliotecas proporcionan aproximadamente la misma funcionalidad, de modo que la elección se puede dejar a criterio del programador; esto, desde luego, teniendo en mente que MPI presenta una implementación más limpia y mejor diseñada.

La segunda herramienta de programación paralela elegida fue MPI, en este caso la red se particiona y se distribuye en diferentes máquinas.

MPI es elegida porque se facilitan de alguna forma todos los pasos de la paralelización (particionamiento, distribución, comunicación y sincronización), no obstante estos pasos los hace el programador. Con MPI cada una de las máquinas puede llevar acabo los procesos de intrusión y de retracción de mercurio de manera independiente con la partición que se le

asigne. Con ayuda del paso de mensajes que implementa MPI, se ejecuta la comunicación, entre las máquinas que llevan acabo los dos procesos (intrusión y retracción) en cada partición. Además de que al utilizar MPI podemos tener tareas que se encarguen de coordinar a los demás tareas, para que los dos procesamientos se comporten de la misma manera que en una red no particionada.

En el siguiente capítulo se describe la implementación de los algoritmos paralelos propuestos para simular los procesos de intrusión y de retracción de mercurio. La implementación se realiza considerando modelos de programación, de memoria compartida y de paso de mensajes.

Capítulo 5

Simulador paralelo de intrusión y retracción de mercurio

En este capítulo se describe la propuesta de un simulador paralelo de los procesos de intrusión y de retracción de mercurio en medios porosos. La manera en la que se diseña consiste en dos versiones. La primera versión es un simulador diseñado en esquema de memoria compartida con OpenMP. En la segunda parte se agrega el modelo de paso de mensajes, para expandir aun más el procesamiento del simulador con ayuda de MPI. A continuación se describe el diseño de estas versiones.

5.1. Algoritmos paralelos de intrusión y retracción de mercurio

La primera versión del simulador paralelo de los procesos de intrusión y de retracción de mercurio se diseñó para ser ejecutado en los sistemas de memoria compartida con multiprocesadores o sistemas con procesadores multicore. El simulador se diseña para procesar una red porosa cúbica (descrita en el Capítulo 3). Esta red cúbica presenta enlaces en el exterior, situados en cada una de las seis caras como se ve en la Figura 5.1.

En esta primera versión del simulador, se utilizan hilos (implementado con OpenMP) que se encargan de los procesos de intrusión y retracción de mercurio. Al diseñar un simulador en el cual puede utilizar diferente número de hilos, es posible determinar su desempeño obtenido

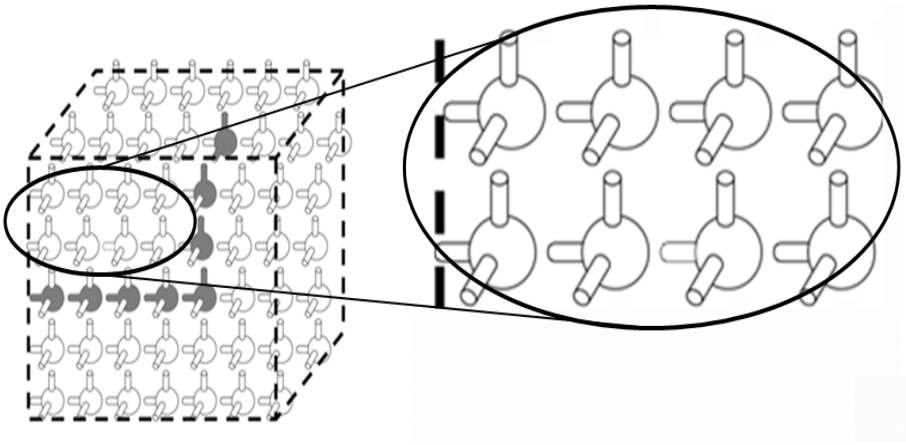


Figura 5.1: Fronteras de la red porosa.

en base al tiempo de ejecución. La configuración de hilos con el menor tiempo obtenido en la ejecución de los procesos de intrusión y retracción, para diferentes tamaños de redes será la mejor. Para obtener esta configuración y considerando las 6 caras externas de una red cúbica, primero se diseñó un modelo para determinar los tiempos cuando se usa un número de hilos menores o iguales a seis, posteriormente para más de 6.

En la primera parte se ejecutan los procesos de intrusión y de retracción con diferente número de hilos y se toman los tiempos de cada uno de ellos para compararlos posteriormente. La manera en la que se implementa esta parte es aprovechando la forma cúbica de la red (con 6 lados o caras), ya que cada una de la cara puede ser intruída o retraída de manera independiente en paralelo por alguno de los hilos en ejecución. Como la red es cúbica se pueden definir hilos que trabajen en cada una de las caras externas de la red, de la siguiente forma:

Cuando se usa un solo hilo, éste se encarga de realizar los procesos de intrusión/retracción desde las seis caras de la red cúbica, y lo hace en diferentes tiempos como se ve en la Figura 5.2, en donde los números indican el orden de procesamiento de cada cara. Una vez iniciada su ejecución comienza por la cara 1 y no es hasta que termina con su trabajo que comienza con el procesamiento de la cara 2, y así sucesivamente.

La manera en que se llevan acabo los procesos de intrusión/retracción es con la ayuda

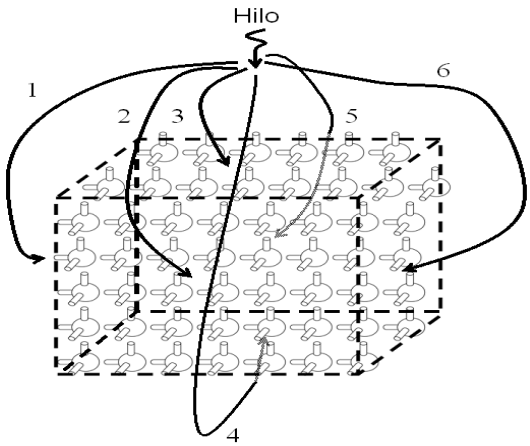


Figura 5.2: Hilo realizando los procesos de intrusión/retracción.

de los algoritmos secuenciales recursivos descritos en el en apéndice 1. En cada proceso de la intrusión y retracción simulada se tiene un registro de la cantidad de sitios intruídos o retraídos, y esta termina hasta que se complete con el número total de los sitios (red totalmente intruída o retraída).

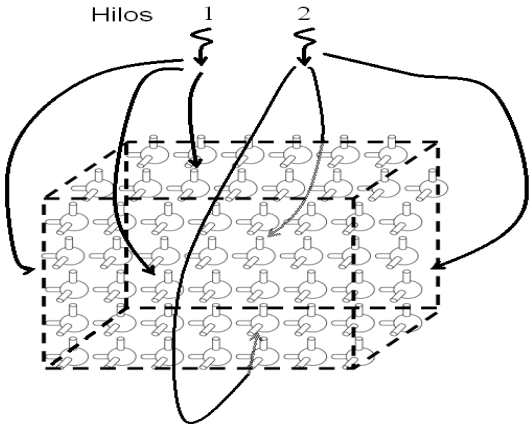


Figura 5.3: Dos hilos realizando los procesos de intrusión/retracción, un hilo por cada tres caras.

Cuando se utilizan dos hilos, un hilo se encarga de intruir/retraer tres caras de la red y el otro hilo se encarga de las tres caras restantes. Ambos hilos trabajan en paralelo para realizar estos procesos como se ve en la Figura 5.3. En este caso los hilos trabajan simultáneamente

sobre una de las tres caras asignadas, y no es hasta que terminan su tarea en esa cara que comienzan con la siguiente.

Para el caso de la configuración con tres hilos, estos se encargan de intruir/retraer desde dos de las caras de la red de manera paralela como se ve en la Figura 5.4.

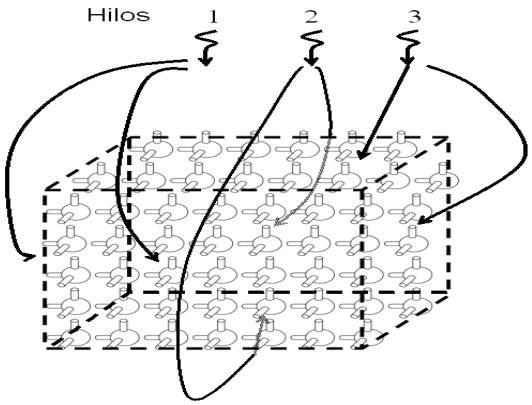


Figura 5.4: Tres hilos realizando los procesos de intrusión/retracción, un hilo por cada dos caras.

Cuando se usan cuatro o cinco hilos, no se tiene un divisor para que se pueda asignar el el mismo número de caras por hilo. Para el caso de cuatro hilos, se les asigna a dos hilos dos caras por hilo, teniendo así 4 caras cubiertas luego se asignan a los otros dos hilos restantes una sola cara como se ve en la Figura 5.5. Cada uno de los hilos trabaja de manera paralela compartiendo la red porosa modelada. En el caso de cinco hilos, se asignan a cuatro hilos una cara por hilo, y el hilo restante trabaja sobre las dos caras restantes como se ve en la Figura 5.6. Al igual que los otros casos los hilos trabajan de manera paralela, y se intruye o retrae por las caras de la red.

Cuando se usan seis hilos, ahora se tiene la posibilidad de repartir de una manera más sencilla el número de hilos que corresponda en cada cara . En este caso cada hilo se encarga de los procesos de intrusión/retracción desde una cara diferente de la red de manera paralela, como se ve en la Figura 5.7. El trabajo lo realizan al mismo tiempo y cuando terminan los

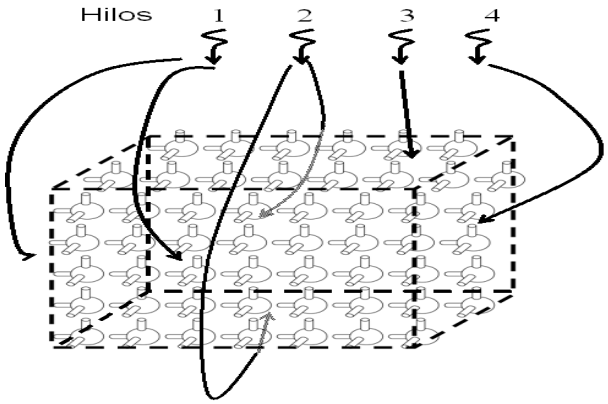


Figura 5.5: Cuatro hilos realizando los procesos de intrusión/retracción, cuatro caras ocupadas por dos hilos y dos caras por dos hilos.

procesos de intrusión o retracción sobre todas las caras asignada acaba la simulación.

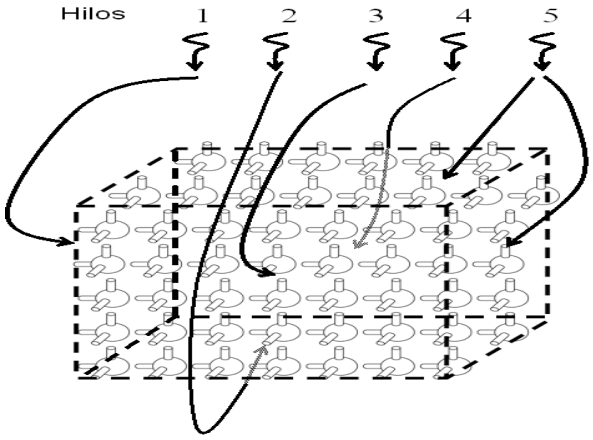


Figura 5.6: Cinco hilos realizando los procesos de intrusión/retracción, cuatro caras ocupadas por un hilo distinto y un hilo ocupando dos caras.

En esta primera parte se usan a lo más 6 hilos. La siguiente parte consiste en diseñar el simulador con más de seis hilos en ejecución.

Para más de seis hilos, las caras externas son divididas para ser procesadas por más de un hilo a la vez. Por ejemplo, si se tienen siete hilos, 5 hilos se encargan de cinco caras de la

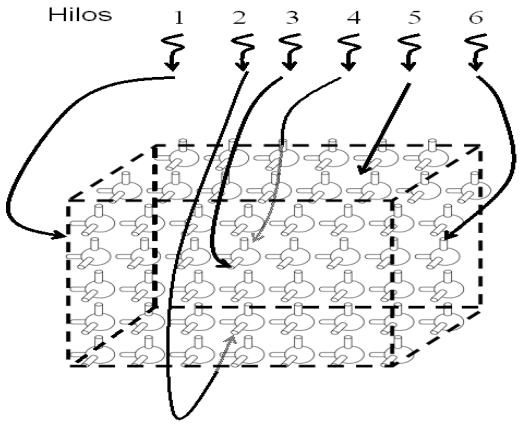


Figura 5.7: Seis hilos realizando los procesos de intrusión/retracción, un hilo por cara.

red y los dos hilos restantes se encargan de una mitad de la cara que falta. Para el caso de doce hilos la manera en la que se distribuyen los hilos es usando dos hilos por cara como se ve en la Figura 5.8.

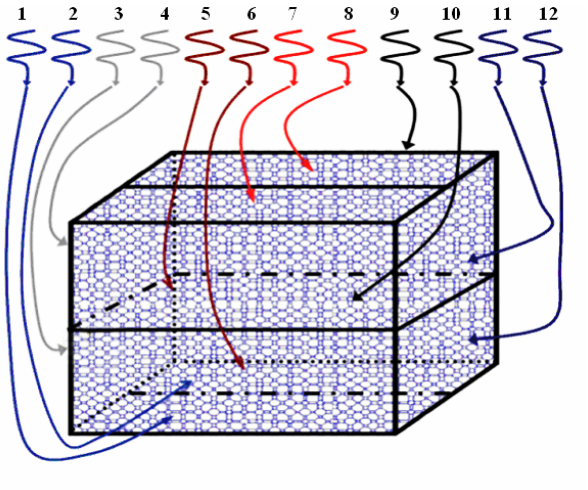


Figura 5.8: Doce hilos realizando los procesos de intrusión/retracción, asignando dos hilos por cara.

Cuando se usan N hilos para llevar acabo los procesos de intrusión/retracción la manera en la que se distribuyen los hilos es usando N/6 hilos por cara.

En esta primera versión se explotan los recursos existentes en un solo nodo, en la siguiente sección se describe como se extiende el simulador para que los procesos de intrusión y de retracción de mercurio se realicen con una arquitectura multicomputador. Como ya se mencionó, las arquitecturas multicomputador se apoyan en los esquemas de paso de mensajes, para distribuir el trabajo en varias máquinas, y ejecutarlas de manera paralela.

En este esquema se hizo una modificación en cuanto a la manera en la que se reparte el trabajo entre los hilos sobre la red, que son lanzados para ejecutar los procesos de intrusión y de retracción. Esta modificación se hace para que sea mas equitativa la repartición de hilos y buscar un mejor tiempo al usar 6 hilos en un nodo de 4 procesadores, por lo que en esta versión equitativa se reparte más adecuadamente el trabajo entre los hilos. Cuando se usan 4 hilos se asigna $\frac{6}{4}$ (Figura 5.9) caras a cada hilo.

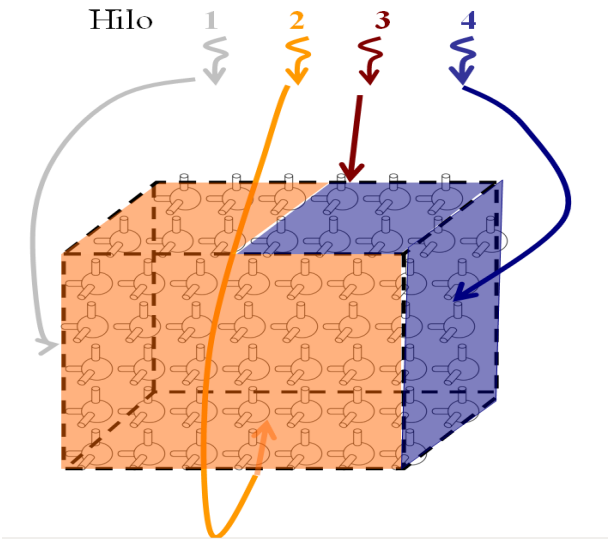


Figura 5.9: Cuatro hilos realizando los procesos de intrusión/retracción, asignando una cara y media por hilo.

cuando se ocupan 5 hilos se asignan $\frac{6}{5}$ (Figura 5.10) caras a cada hilo, de esta manera los hilos trabajan más equitativamente que en la versión anterior.

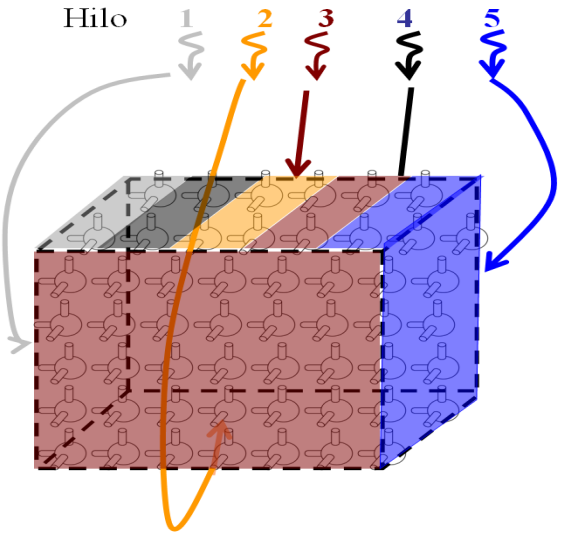


Figura 5.10: Cinco hilos realizando los procesos de intrusión/retracción, asignando una cara y $\frac{1}{5}$ por hilo.

5.2. Particionamiento

El simulador basado en el modelo de memoria compartida fue mejorado mediante la adición de uso de una arquitectura multicomputador. Con esta arquitectura se tiene la ventaja de distribuir tareas en múltiples procesadores que trabajan conjuntamente sobre un sistema.

En este caso la red cúbica es particionada en varias secciones (sub-redes), para ser distribuidas en los diferentes procesadores que se tengan disponibles, como se ve en la Figura 5.11.

Al realizar el particionamiento, es importante mencionar que las sub-redes generadas presentan 2 tipos de caras. Existen caras llamadas externas que son las caras que se encuentran en las fronteras de la red cúbica, y existen caras internas que son las caras que se encuentran en el interior de la red cúbica generadas durante la partición como se ve en la Figura 5.12.

Técnicas de particionamiento

Apoyándonos en la representación de la red porosa cúbica, se tiene a la red en tres dimensiones como se muestra en la Figura 5.13. Con ayuda de los ejes X, Y y Z la manera en la que se elaboró el particionamiento de la red es la siguiente:

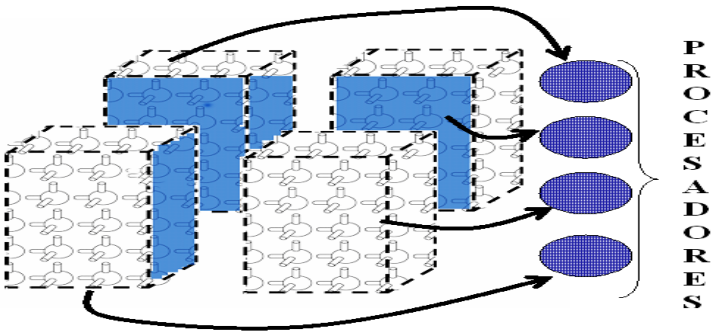


Figura 5.11: Distribución de las sub-redes en los procesadores.

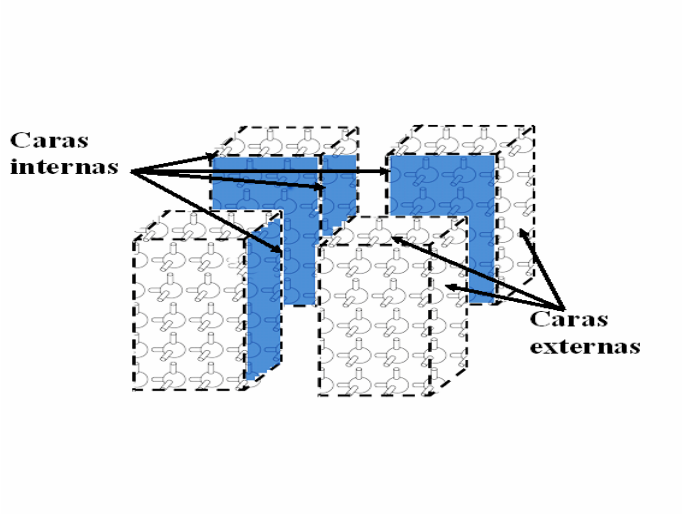


Figura 5.12: Tipo de cara en las sub-redes.

Como primer paso se divide la red porosa en dos partes (Figura 5.11 A), aquí se realiza una partición en el eje Z solamente. En este caso se distrribuyen las dos sub-redes resultantes en dos nodos de un cluster. Para cuatro particiones se divide en el eje Z y en el eje X (como se ve en la Figura 5.12 B) de igual manera las sub-redes se distribuyen en 4 nodos.

Para 8 particiones se divide el eje X, el Y y el Z (como se ve en la Figura 5.12 C). En este caso se distribuyen las 8 sub-redes en 8 nodos. Para más de 8 particiones las subsecuentes divisiones solo pueden ser generadas en el eje Y para que exista almenos una cara externa y se cumplan las condiciones para que puedan ejecutarce los procesos de intrusión y retracción

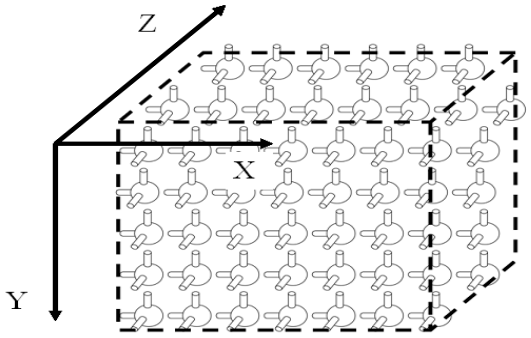


Figura 5.13: Ejes coordenados en la red.

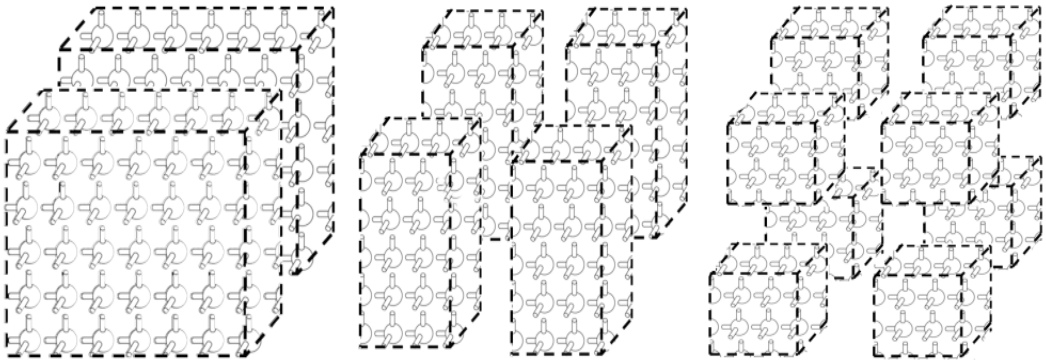


Figura 5.14: Particionamiento de la red: A) 2, B) 4, C) 8 particiones.

(Figura 5.15). De esta manera, cada sub-red pose al menos dos caras externas.

Esto último permite que todas las particiones tengan al menos 2 caras con enlaces externos, y sea posible llevar acabo los procesos de intrusión/retracción de mercurio desde el principio. Las particiones que se generan solo son en potencias de 2 es decir en algún valor de 2^n. En la particiones generadas existen enlaces que se encontraban unidos en el interior de la red, y cada partición separa estos enlaces como se ve en la Figura 5.16.

Como al simular los procesos de intrusión y retracción se debe tener el mismo comportamiento que en una red sin particionar, se busca una manera de que cada uno de los enlaces separados se comporten de la misma manera que en la red original. Los enlaces que fueron

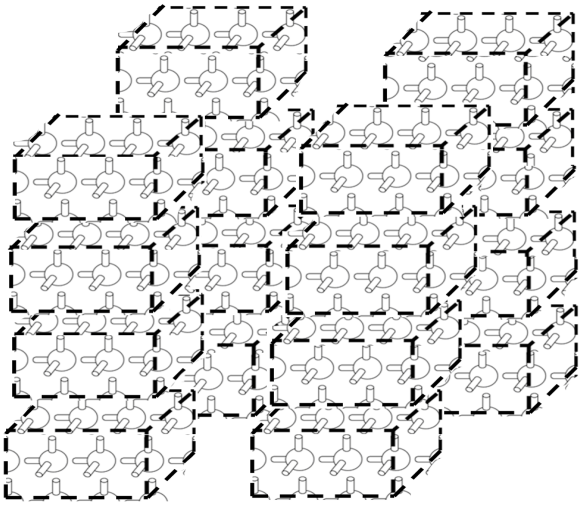


Figura 5.15: Particionamiento de la red en 16 particiones.

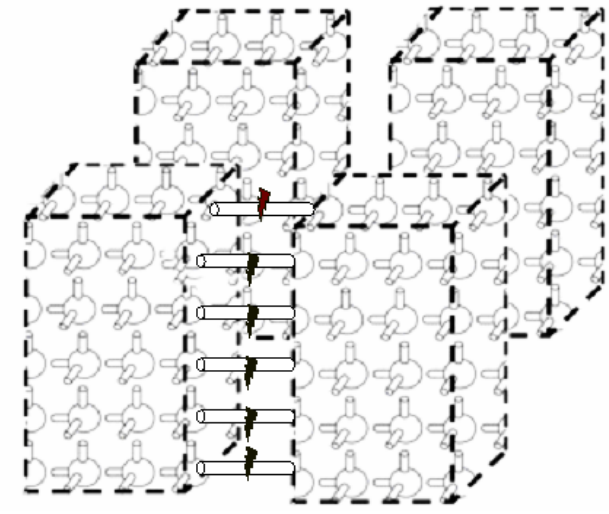


Figura 5.16: Enlaces separados por la partición.

separados deben de ser copiados para que al simular los procesos se tenga el mismo comportamiento. Esto se logra con copias de los enlaces en cada cara de las sub-redes llamadas ***interfaces***. El uso de interfaces y su descripción entre las sub-redes, se presenta en la siguiente sección.

5.3. Creación de interfaces

Una vez que se realizan n particiones, se generan 2^n sub-redes, estas sub-redes son distribuidas entre cada uno de los nodos. Cada una de las sub-redes son procesadas de manera independiente y sobre ellos se realizan los procesos de intrusión y de retracción de manera paralela utilizando el esquema de memoria compartida descrito en la sección 5.1. Por medio de paso de mensajes es como se lleva acabo la sincronización y la comunicación entre procesadores quienes procesan las sub-redes. Posteriormente se deben identificar el tipo de cara que presenta cada una de las sub-redes.

Una vez identificadas las caras (internas y externas) se debe tener control de la comunicación que existe en las sub-redes distribuidas en los nodos, para los procesos de intrusión y de retracción. Por este motivo se crean 6 interfaces para cada una de las sub-redes.

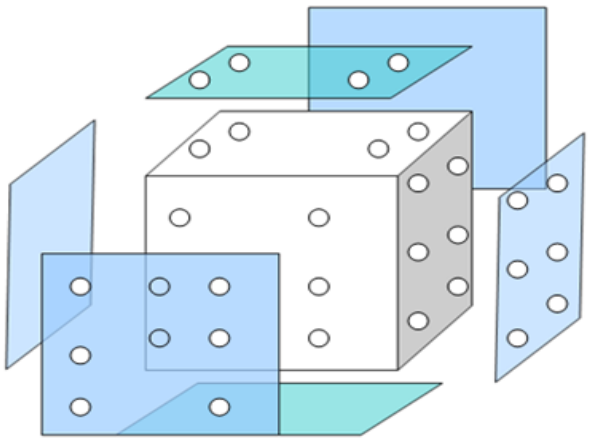


Figura 5.17: Interfaz para las sub-redes de la red porosa cúbica.

Las interfaces son un conjunto de 6 planos (Figura 5.17), cada uno contiene una copia de

los enlaces presentes en los extremos de las caras de la sub-red. Esta interfaz da información de la cantidad de enlaces que se vieron afectados por una intrusión (se llenaron de mercurio) o retracción (se vaciaron) a una presión dada. Después del procesamiento local (entre cada una de las sub-redes) las interfaces son intercambiadas entre las sub-redes adyacentes, para comparar el comportamiento de los enlaces. A una presión dada al simular los procesos de intrusión o retracción de mercurio se deben conocer cada uno de los enlaces intruídos o retraídos locamente, para que de esta manera se construyan y se intercambien las interfaces y se decida si se continúa o no con el proceso, partiendo de enlaces internos que se afectaron por una sub-red adyacente.

5.4. Criterio de terminación

Cuando se crean las interfaces y se intercambian entre los vecinos de cada sub-red, se verifica si los enlaces compartidos intruídos o retraídos son los mismos. Si no coinciden estos enlaces es porque falta por intruir o retraer algún sitio con su enlace de esa interfaz, si esto ocurre se repiten la intrusión o la retracción a la misma presión para todos los enlaces. Se repite el proceso en todas las particiones porque si no tienen los mismos enlaces intruídos o retraídos los procesos no se están comportando como en la red sin particionar (pues los enlaces estaban unidos). En la Figura 5.18 se ven dos caras de enlaces compartidos con diferentes enlaces intruídos o retraídos, los puntos negros son los enlaces que si están intruídos y los blancos son los vacíos o retraídos, en esta figura no se tienen los mismos enlaces por lo que se continúa con los procesos.

La simulación a una presión dada termina cuando se tiene en cada una de las interfaces los mismos enlaces intruídos o retraídos para todas las sub-redes.

Cuando se tienen los mismos enlaces intruídos o retraídos en las interfaces que fueron intercambiadas se continúa con la simulación, trabajando a una nueva presión. En la Figura 5.19 se observan los mismos enlaces intruídos o retraídos que son los puntos de color negro y blanco, como se tienen los mismos enlaces se terminan los procesos a esa presión. Cada vez que ocurre lo anterior el valor de presión se aumenta para la intrusión o se disminuye para

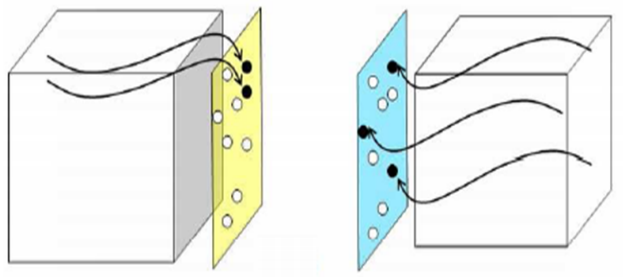


Figura 5.18: Sub-redes con diferentes enlaces intruídos o retraídos en la simulación.

la retracción.

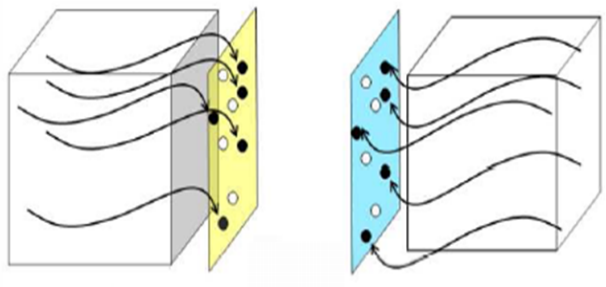


Figura 5.19: Sub-redes con los mismos enlaces intruídos o retraídos en su interfaz.

El pseudocódigo general del simulador paralelo con el esquema de paso de mensajes es el siguiente:

Proceso Coordinador

```
1. {
2.   particiona_y_distribuye_matriz();
3.   envia_caras_compartidas();
4.   seguimos = 1;
5.   Bcast(presión, coordinador); //Informa el valor de la presión a la que se intruye
6.   mientras(presión ≠ presión_final)
7.   {
8.       mientras(seguimos ≠ 0)
9.        {
10.         Reduce(&seguimos, SUMA, coordinador);
11.         Bcast(seguimos, coordinador);// informa si se continua con los procesos
12.        }
13.       Reduce(&num_sitios, SUMA, coordinador);
14.       Reduce(volumen, SUMA, coordinador);
15.       si(num_sitios ≠ L^3 & vol ≠ vol_tot)//termina intrusión/retracción
16.         modifica_presiones();//disminuye o aumenta la presión
17.       Bcast(presión, coordinador);// Se informa la presión que se esta manejando
18.   }
19. }
```

Figura 5.20: Algoritmo para el simulador paralelo de los procesos de intrusión y retracción (código del proceso coordinador).

Procesos Trabajadores

```
1. {
2.   Red = recibe_matriz();//recibe la sub-red en la que realizará la intrusión/retracción
3.   recibe_compartidas();
4.   construye_caras_interfaz();
5.   seguimos = 1;
6.   Bcast(presión, coordinador);
7.   mientras(presión ≠ presión_final)
8.   {
9.       Intru_externa();/ Retra_externa();
10.       Intercambia_interfaz(caras_locales, caras_vecinas);
11.       cuantos = verifica_enlaces_diferentes(caras_locales, caras_vecinas);
12.       Reduce(cuantos, SUMA, coordinador);
13.       mientras(cuantos ≻ 0)
14.       {
15.         Intru_interna();/ Retra_interna();
16.         Reduce(vol_acumulado, SUMA, coordinador);
17.         Reduce(num_sitios, SUMA, coordinador);
18.         Bcast(cuantos, coordinador);
19.       }
20.       Bcast(presión, coordinador);
21.   }
22. }
```

Figura 5.21: Algoritmo para el simulador paralelo de los procesos de intrusión y retracción (código del procesos trabajadores).

En el pseudocódigo anterior se muestra como funciona el simulador paralelo de los procesos de intrusión y de retracción de mercurio. Se muestra en dos partes, el primer pseudocódigo (Figura 5.18) el proceso coordinador lleva a cabo sus tareas y el segundo (Figura 5.19) es para los procesos trabajadores. El proceso coordinador es el que se encarga de controlar los procesos de intrusión y de retracción paralela que ocurren en las sub-redes de la matriz de la siguiente manera:

La función de la línea 2 en el pseudocódigo uno (figura 5.17) se encarga de realizar la partición de la matriz como se describe en la Sección 5.2 de particionamiento. Cuando es realizada la partición esta función también se encarga de que cada una de las sub-redes sea distribuida en los procesadores. La función de la línea 3 informa cuales son las caras que compartían enlaces dentro de la red original una vez que son divididas en varias sub-redes (también descrita en la Sección 5.2). Esta información es de gran utilidad pues para cada una de las caras compartidas posteriormente se crean interfaces de enlaces con esa información. De la linea 8 a la 12 el proceso coordinador avisa a los otros procesos si se continúan con los procesos de intrusión y de retracción, esto lo realiza cuando las interfaces no tienen los mismos enlaces intruídos o retraídos. En las líneas 13 y 14 se actualizan los valores de número de sitios y del volumen que se tiene en los procesos de intrusión o de retracción. El número de sitios y el volumen nos sirven para saber hasta cuando se detienen los procesos, en la línea 15 se verifica si no se han alcanzado el total de sitios y el volumen máximo. Si no se han alcanzado el número de sitios y el volumen máximo se actualizan las presiones para intruir o retraer a una nueva presión, (se hace en la línea 16) si se actualiza la presión o no se informa a los otros procesos el valor de la presión (línea 17).

El pseudocódigo de los procesos trabajadores se encargan de ejecutar los procesos de intrusión y retracción en las sub-redes generadas. El proceso trabajador, recibe su sub-red (línea 2) y la información de las caras de esa sub-red que compartían enlaces (línea 3). Posteriormente se crea una interfaz (línea 4) para cada una de las caras de las sub-redes

(descrita en la Sección 5.3). Una vez que los procesos trabajadores reciben su sub-red se len informa cual es la presión a la que se intruye o se retrae (linea 6). A partir de ahí, los procesos llevan acabo la intrusión o la retracción en las caras externas de la red (línea 9). En la línea 10 se intercambian las interfaces de las caras vecinas en las sub-redes, como se describe en la Sección 5.4, para posteriormente verificar si se tienen los mismos enlaces intruídos o retraídos (línea 11). Si no se tienen los mismos enlaces intruídos o retraídos es necesario que se continúen los procesos de intrusión/retracción sobre las sub-redes, esto se realiza en de la línea 13 a la 18. Si se tienen los mismos enlaces intruídos o retraídos se presenta el criterio de terminación descrito en la Sección 5.4.

En el siguiente capítulo se presentan la evaluación de las pruebas realizadas para este trabajo de tesis.

Capítulo 6

Resultados y Evaluación

En este capítulo se muestra la evaluación del simulador paralelo así como los resultados obtenidos en las pruebas realizadas.

El simulador paralelo es probado en diferentes sistemas de cómputo para poder determinar su desempeño. La meta principal es verificar que los tiempos obtenidos se reduzcan, en comparación con los tiempos secuenciales, si es así se considera con buen desempeño. Para realizar estas pruebas se propone la siguiente plataforma experimental.

6.1. Plataforma de experimentación

6.1.1. Clusters utilizados

El simulador fue ejecutado en tres clusters, los cuales presentan las siguientes características:

Cluster 1. Pacífico

- Arquitectura heterogéneo
- 10 nodos (40 unidades de procesamiento)
    - 4 nodos, c/u con 2 Intel Xeon Dual_core 3 GHz y 2 Gb de RAM
    - 6 nodos, c/u con Intel Quad_core 2.4 GHz y 4 GB de RAM

- Comunicación - Gigabit Ethernet
- Sistema Operativo Linux Centos 5
- Ubicación - Laboratorio de Sistemas Distribuidos UAM-Iztapalapa (México)

Cluster 3. Aitzaloa

- Arquitectura homogénea
- 270 nodos (2160 unidades de procesamiento)
- 2 procesadores por nodo (Intel Quad_core Xeon 3Ghz, 16 GB RAM)
- Comunicación Infiniband bus de alta velocidad.
- Sistema Operativo Linux Centos 5.2
- Ubicación - Laboratorio de supercómputo y visualización en paralelo UAM-Iztapalapa (México)

6.1.2. Pruebas realizadas

El simulador paralelo fue probado con redes porosas de diferente tamaño. Al usar redes de diferente tamaño se determina cual es el desempeño del simulador paralelo, pues el número de poros (enlaces y sitios) descrito en la ecuación 3.2, se incrementa con redes de mayor tamaño. Cuando es incrementado el número de poros, los procesos de intrusión y de retracción consumen una mayor cantidad de tiempo en su ejecución, dado que la cantidad de datos a procesar aumenta como se mencionó en la ecuación 3.4.

Con redes de tamaño 25, 50, 75 y 100 sitios por lado se realizaron las pruebas sobre las versión de memoria compartida con hilos y la de memoria compartida distribuida del simulador paralelo. La primera prueba del simulador paralelo consiste en determinar la eficiencia en sistemas multiprocesador de memoria compartida. Para esta prueba se obtuvo el número de hilos, con los cuales se obtiene un menor tiempo de ejecución sobre los procesos de intrusión y de retracción. Los hilos son aumentados siguiendo el esquema descrito en el Capítulo 5, y con los tiempos obtenidos se encuentra la mejor configuración de hilos.

Para esta primera prueba determina el número de hilos con el que se obtiene el menor tiempo para la intrusión y la retracción, probando las dos versiones la equitativa y la no equitativa. Estas pruebas se realizan en un nodo de 4 procesadores del Cluster 1 y en un nodo de 8 procesadores del Cluster 3. Una vez que se determina este número se utiliza esta configuración para continuar con la siguiente prueba.

En la segunda prueba se particiona la red y cada partición creada se asigna a un procesador el cual sigue utilizando memoria compartida con OpenMP (la mejor configuración de la primera prueba) pero también aplica un protocolo de paso de mensajes con MPI para llevar acabo los procesos de intrusión y retracción en paralelo. La manera en la que se particiona la red y se distribuye cada una de las sub-redes se hace como se describe en el Capítulo 5.

En esta segunda prueba se ejecutan las tareas de intrusión y retracción, sobre las sub-redes generadas cuando se particiona la red. Cada partición de la red es distribuida en los nodos del Cluster 3, para estas pruebas se determina el número de procesos a los cuales se tiene el menor tiempo durante la ejecución de los procesos de intrusión y retracción.

Cuando se generan las particiones sobre una red es posible que existan problemas en cuanto al número máximo posible de particiones por red. En una red porosa el número de particiones máximas está determinado por el número L de enlaces que presente por cada una de las caras. Como el máximo de las divisiones posibles en una red son obtenidas una sobre el eje X, otra sobre el eje Z y L sobre el eje Y, se tienen entonces $2 * 2 * L$ divisiones en total. Este número de particiones nos va a limitar en cuanto al número de procesos que se pueden lanzar para ejecutar los procesos de intrusión y de retracción ya que las particiones

son los procesos que se lanzan. Este esquema se apoya del esquema de memoria compartida lanzando 8 hilos por procesador (la mejor configuración de la primera versión). Entonces se están utilizando un total de 8 hilos por procesador por lo que se están lanzando 64 hilos por nodo. La tercera prueba consiste en reducir el número de hilos que se lanzan por nodo, por lo que el esquema anterior se modifica para que solo se lance un proceso por procesador. Al lanzar un proceso por procesador este procesador tiene solo 8 hilos (la mejor configuración del esquema de memoria compartida). De la misma forma que la prueba anterior se determina el número de procesos a los cuales se obtiene el mejor tiempo en realizar los procesos de intrusión y de retracción.

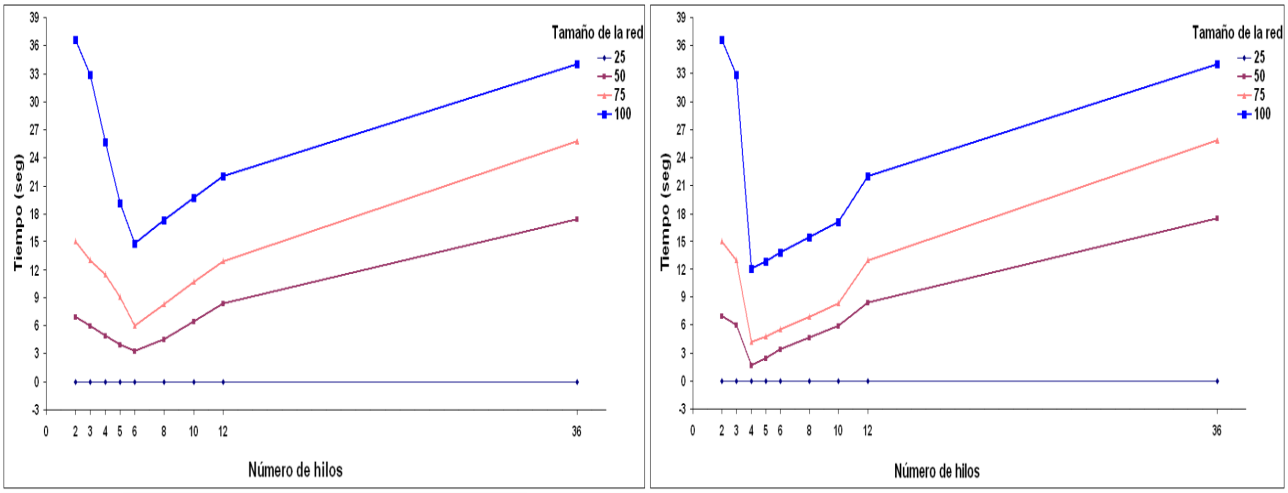
6.2. Resultados

En primer lugar, se muestran los resultados al utilizar un solo nodo en un sistema multiprocesador con memoria compartida. Después, se muestra las ventajas obtenidas cuando se mejora el simulador usando un cluster que combinan los modelos de memoria compartida y paso de mensajes.

6.2.1. Esquema de sistema multiprocesador

El rendimiento del simulador al ser ejecutado en un nodo depende del número de hilos creados para procesar una red porosa. Para determinar qué configuración alcanza los mejores resultados, se realizaron varios experimentos mediante el uso de redes de diferentes tamaños (L = 25, 50, 75, 100). La primera versión del simulador paralelo (descrita en el Capítulo 5) es probada en 2 nodos; un nodo con 4 procesadores del Cluster 1 y un nodo de 8 procesadores del Cluster 3 (descritos en la plataforma experimental). La manera en la que se probó fue aumentando el número de hilos; los hilos empleados fueron 2, 3, 4, 5, 6, 8, 10, 12 y 36.

En la Figura 6.1a se muestra el proceso de intrusión con la versión de hilos no equitativo en un nodo de 4 procesadores (Pacífico), se observan los tiempos de ejecución en donde el

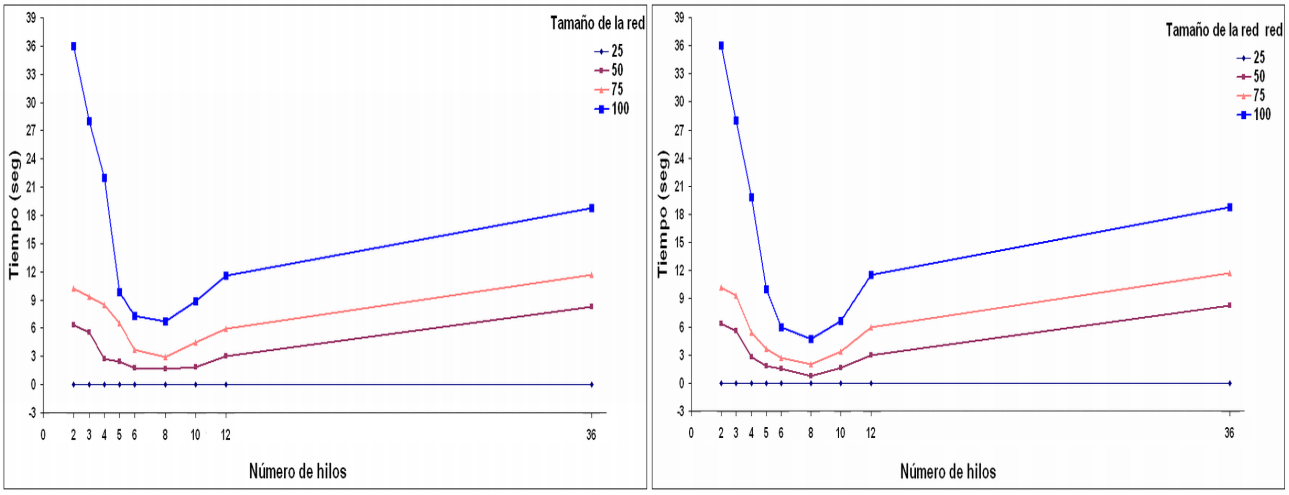


(a) Esquema no equitativo (b)Esquema equitativo

Figura 6.1: Tiempos de respuesta del proceso de intrusión de mercurio en el esquema de memoria compartida, con el esquema no equitativo (a) y equitativo (b) en un nodo de 4 procesadores.

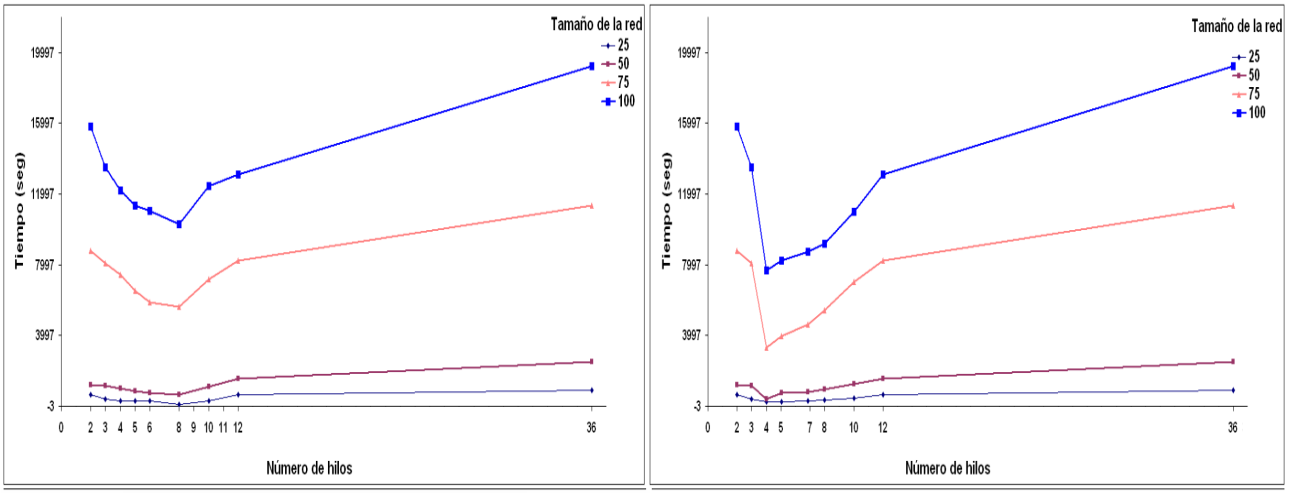
mejor tiempo es utilizando 6 hilos. En la figura 6.1b, se muestra el proceso de intrusión con la versión de hilos equitativa en un nodo de 4 procesadores. En la versión equitativa se reparten de mejor manera los hilos entre los procesadores, y al tener 4 procesadores el mejor tiempo fue al usar 4 hilos. En la Figura 6.2a se muestra el proceso de intrusión ahora probado en un nodo de 8 procesadores del Cluster 3 (Aitzaloa), con la versión de hilos no equitativa, para este caso el mejor tiempo se obtiene cuando son usados de 6 hasta 8 hilos. En la Figura 6.2b se muestra el proceso de intrusión probado en el Cluster 3 (Aitzaloa), con la versión de hilos equitativa, en donde el mejor tiempo se obtiene cuando son usados 8 hilos.

En la Figura 6.3a se muestra el proceso de retracción con la versión de hilos no equitativo en un nodo de 4 procesadores. De la figura se observa que el mejor tiempo de ejecución se obtiene utilizando 6 hilos (de la misma forma que sucede en el proceso de intrusión). En la figura 6.3b, se muestra el proceso de retracción con la versión de hilos equitativa en un nodo de 4 procesadores, en la figura se observa que el mejor tiempo se obtiene con 4 hilos en ejecución. Estas versiones fueron probadas en el Cluster 2 de la Universidad de Münich.



(a) Esquema no equitativo (b)Esquema equitativo

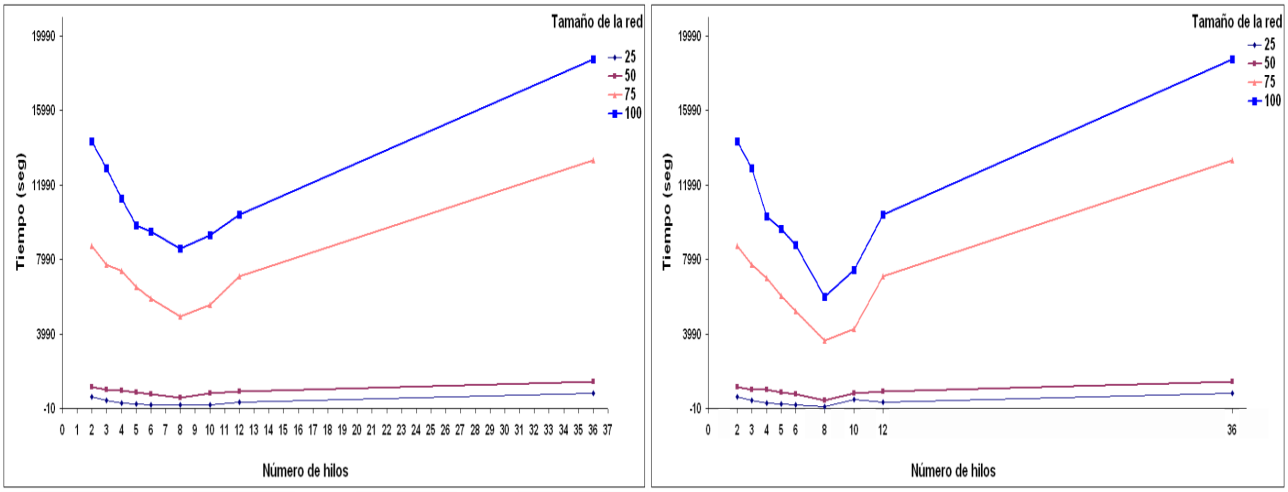
Figura 6.2: Tiempos de respuesta del proceso de intrusión de mercurio en el esquema de memoria compartida, con el esquema no equitativo (a) y equitativo (b) en un nodo de 8 procesadores.



(a) Esquema no equitativo (b)Esquema equitativo

Figura 6.3: Tiempos de respuesta del proceso de retracción de mercurio en el esquema de memoria compartida, con el esquema no equitativo (a) y equitativo (b) en un nodo de 4 procesadores.

En la Figura 6.4a se muestra la simulación del proceso de retracción ejecutada en un nodo de 8 procesadores con la versión no equitativa, en este caso el mejor tiempo se obtuvo con 6 a 8 hilos. En la Figura 6.4b se muestran, la simulación del proceso de retracción con la versión equitativa, en este caso el mejor tiempo se obtuvo con 8 hilos. Con la versión de hilos equitativa se reduce aun más el tiempo que la versión no equitativa como en el proceso de intrusión.



(a) Esquema no equitativo (b)Esquema equitativo

Figura 6.4: Tiempos de respuesta del proceso de retracción de mercurio en el esquema de memoria compartida, con el esquema no equitativo (a) y equitativo (b) en un nodo de 8 procesadores.

6.2.2. Análisis de resultados con el esquema de sistema multiprocesador

Primero se analizan los resultados para el sistema multiprocesador en donde se determina qué número de hilos es el adecuado. En este caso consideramos que el mejor tiempo de ejecución tanto para el proceso de de intrusión como en el de retracción obtenido en la versión no equitativa es con 6 hilos. Al realizar la ejecución de estos procesos de porosimetría de mercurio, se comienza el procesamiento a partir de las 6 caras externas de la red y se

puede usar cualquier número de hilos en el nodo. Si el nodo tiene 4 procesadores y sólo 2 hilos se utilizan, el paralelismo no se ha explotado al máximo ya que 2 procesadores se encuentran ociosos. Cuando usamos 4 hilos, en una primera etapa los hilos trabajan de manera simultánea en 4 diferentes caras externas, por lo que quedan 2 caras esperando ser intruídas o retraídas según sea el caso. Una vez que se termina el proceso de porosimetría de mercurio, en una segunda fase, dos de los 4 hilos continúan trabajando con las 2 caras restantes. Cuando se utilizan 6 hilos en un nodo de 4 procesadores, llamemos H_1 a H_6 a los hilos, cada uno trabaja con un hilo por diferente cara externa. A pesar de que al principio 2 pares de hilos comparten el mismo procesador, por ejemplo, (H_1, H_5) y (H_2, H_6). Aún mejor, cuando H_3 y H_4 terminan su trabajo y liberan su procesador, entonces los hilos (H_5 o H_6) pueden ser ejecutados por un procesador, favoreciendo así un buen equilibrio de carga y logrando un mejor tiempo de ejecución . Al utilizar más de 6 hilos de una cara externa se puede compartir por dos o más hilos. En este caso aumenta el tiempo de ejecución debido al costo de los cambios de contexto de los hilos en cada procesador. Cuando se utilizaron más de 6 hilos se observa que algunos de los procesadores se encuentran trabajando a menos del 100 % de su capacidad. Estos resultados fueron probados en un nodo del Cluster 1 Pacífico y en el Cluster 2 Infinicluster en un nodo de 4 procesadores.

En la versión equitativa, la simulación de los procesos de intrusión y de retracción obtuvo el mejor tiempo, cuando se ocuparon 4 hilos la razón de esta reducción es la siguiente:

De igual manera que en la anterior cuando se lanzan de 1 a 3 hilos el paralelismo no es adecuadamente explotado, ya que algunos procesadores se encuentran inactivos. Si tenemos 2 hilos estos se encargan de realizar los procesos de porosimetría, cada uno sobre tres caras de la red, pero 2 procesadores se encuentran inactivos. Cuando se utilizan 3 hilos se asigna a dos caras por hilo, pero 1 procesador se encuentra inactivo. Cuando se utilizan 4 hilos ahora se van a asignar $\frac{6}{4}$ caras a cada hilo y cuando se ocupan 5 se van a asignar $\frac{6}{5}$ caras a cada hilo de esta manera se distribuye más equitativamente el trabajo.

Al asignar $\frac{6}{4}$ caras a cada hilo, para el caso de 4 hilos, el consumo del procesamiento se mantiene al 100 % para cada uno de los procesadores presentes. De la misma forma que con

4 hilos, al usar 5 hilos se mantiene el procesamiento al 100 % ocupado por los 4 procesadores pero se tiene un hilo más en ejecución el cual comparte un procesador con otro hilo y los cambion de contexto hacen que el tiempo aumente.

Al usar 6 hilos en las pruebas realizadas se observa que el procesamiento en los 4 procesadores no se mantiene al 100 %, debido a que algunos hilos terminaban antes sus tareas.

Al usar más de 6 hilos el tiempo se incrementa debido a que los procesadores utilizan más de 1 hilo por cara en la red. De la misma forma que en la versión no equitativa el costo de los cambios de contexto de los hilos en cada procesador es mayor por ocupar 2 o más hilos por cara.

El siguiente resultado obtenido surge al probar la versión equitativa utilizando un nodo de 8 procesadores. Para este caso el mejor tiempo obtenido para los procesos de intrusión y de retracción se obtiene al utilizar 8 hilos. El mejor tiempo obtenido con 8 hilos es debido a que ahora se tiene una mayor cantidad de procesadores y los hilos se distribuyen mejor en las 6 caras de la red porosa.

Cuando el número de hilos lanzados es menor a 8 no se utilizan todos los 8 procesadores, por ese motivo el tiempo no es el mejor.

Si tenemos 4 hilos lanzados en 8 procesadores el paralelismo no es adecuado porque 4 procesadores están sin tareas al igual que con 5 hilos.

Si se tienen ahora 6 hilos ocurre lo mismo que con 4, no se explota al máximo el paralelismo ya que 2 procesadores se encuentran ociosos sin tareas. Cuando son lanzados 8 hilos, las 6 caras de la red están siendo intruídas y retraídas por al menos 1 hilo y los 2 hilos restantes son repartidos en dos caras con la versión equitativa, mejorando así el tiempo de ejecución. Cuando son usados 8 hilos los procesadores se encuentran trabajando a un 100 % durante el tiempo de ejecución.

Al momento de utilizar más de 8 hilos para ejecutar las tareas de intrusión o retracción, los hilos que no están siendo ocupados por los 8 procesadores, solo se reparten entre algunos procesadores. El uso de más hilos por procesador hace que aumente el tiempo debido a los cambios de contexto entre los hilos. Para redes de tamaño mayor se nota claramente como

aumenta el tiempo de ejecución cuando se usan más de 8 hilos, con redes pequeñas no se nota el aumento de tiempo. Cabe mencionar que el proceso de intrusión se realiza en menos tiempo que el proceso de retracción porque la intrusión no presenta algunos fenómenos como los de la retracción (mencionados en el Capítulo 3).

6.2.3. Esquema de sistema de memoria compartida y distribuida

La segunda versión del simulador paralelo utiliza recursos de sistemas multiprocesador y multicomputador. En esta versión se tiene un cluster de nodos multiprocesador para las pruebas y se busca determinar el mejor número de procesos para la obtención del menor tiempo en ejecución.

En esta versión el número de procesos está relacionado con el número de particiones en la red, entre más particiones se necesitan más procesos.

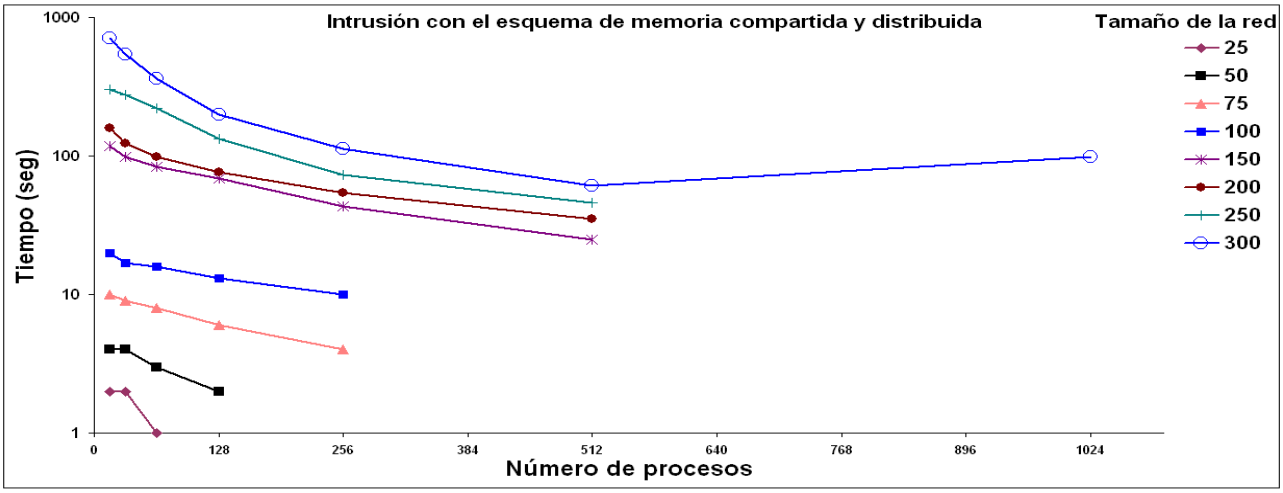


Figura 6.5: Mejor número de procesos para la intrusión.

Las particiones se definen en nuestras pruebas teniendo un número de 2, 4, 8, 16 ... particiones, es decir en potencias de $2^n = P$ donde n = número de particiones y P = número de procesos. Sin embargo, el número de particiones mencionado no se ocupa en todas las redes probadas. Se generan particiones de tal manera que el máximo número de particiones que se puede generar en una red de tamaño L, es de $2 * 2 * L$ particiones en total este número es el

máximo de particiones posibles. Por lo tanto el número de particiones nos indica el número de procesos que pueden ser lanzados, para una red de tamaño 300 se usan 1024 procesos, para redes de 150, 200 y 250 se pueden ocupar 512 procesos, y para redes de tamaño 100 y 75 se pueden lanzar 256 procesos.

En este esquema se ocupa la mejor configuración obtenida con la prueba anterior, la cual es usando la versión equitativa cuando son lanzados 8 hilos para llevar acabo los procesos de intrusión y retracción en cada partición. La ejecución se realiza en el Cluster 3 Aitzaloa.

En las Figuras 6.5 y 6.6, se muestran los tiempos obtenidos por el simulador paralelo del sistema multiprocesador y multicomputador de los procesos de intrusión y retracción de mercurio.

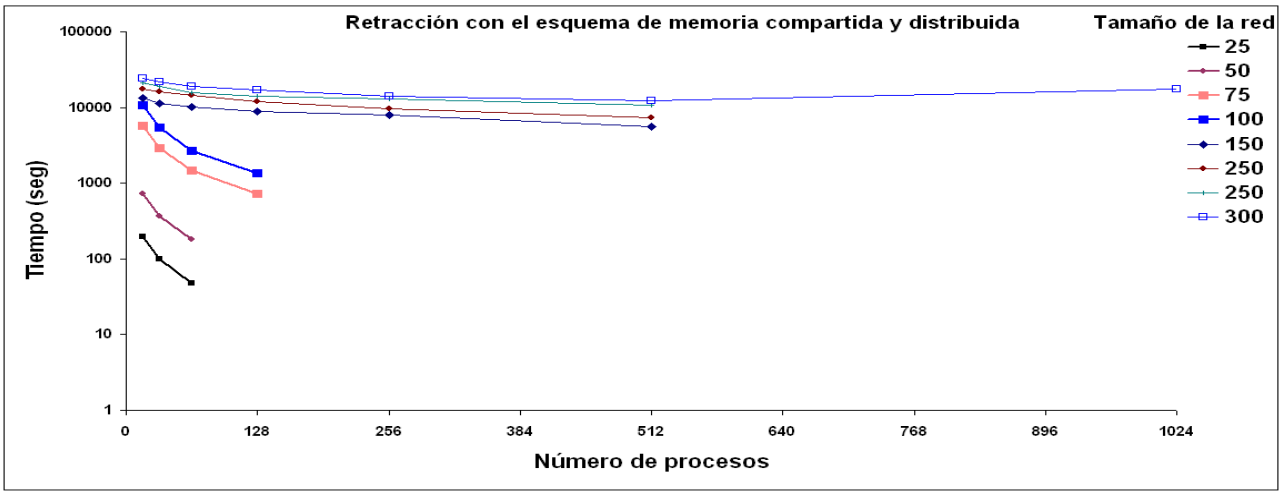


Figura 6.6: Mejor número de procesos para la retracción.

La mejor configuración se obtuvo al ejecutar los procesos de intrusión y retracción aumentando el número de procesos lanzados cuando se utilizaron 512 procesos. En las Figuras 6.7 y 6.8, se muestran los tiempos obtenidos por el simulador paralelo de los procesos de intrusión y retracción de mercurio, en el sistema multiprocesador y multicomputador con la modificación en donde se ejecuta un proceso por nodo.

La ejecución se realiza en el Cluster 3 Aitzaloa, en este caso se tienen 256 nodos, con los cuales se obtienen los mejores tiempos.

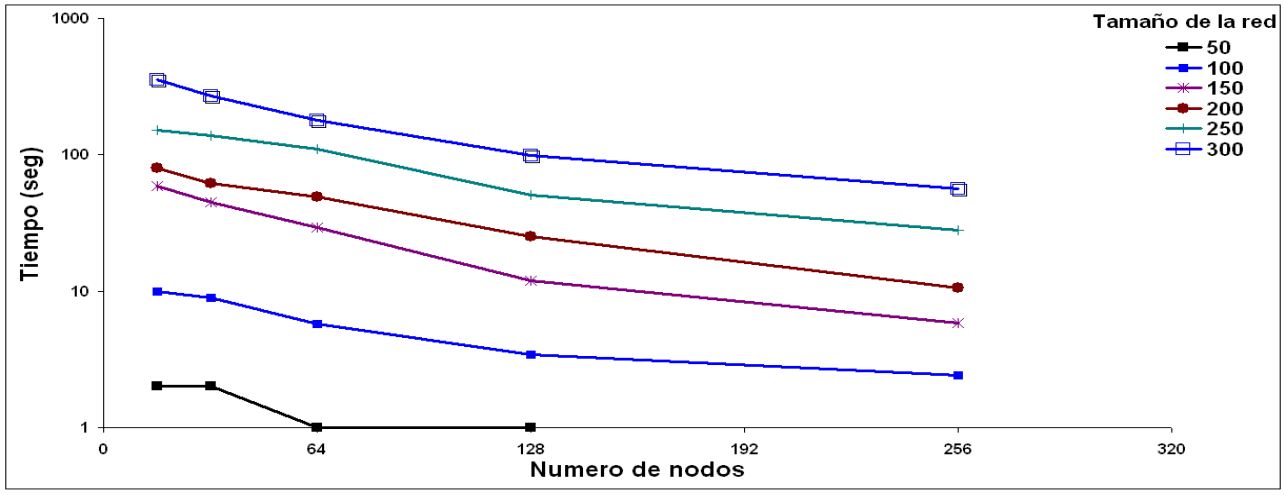


Figura 6.7: Mejor número de procesos para la intrusión en un proceso por nodo.

6.2.4. Análisis de resultados para el esquema de memoria compartida y distribuida

En el proceso de intrusión y retracción de mercurio para el esquema de memora compartida y distribuida, el mejor tiempo se obtuvo al usar 512 procesadores. Se ocuparon 256 procesadores para determinar el número de procesos con los que se obtiene la mejor configuración durante la simulación de los procesos de intrusión y de retracción.

Al generar sub-redes estas tienen diferente tiempo de procesamiento, debido a las condiciones mencionadas en la ecuación 3.4 tomarán distintos tiempos en concluir sus tareas. En una determinada sub-red, las condiciones de sus poros hacen que los procesos de intrusión y retracción puedan ser más difíciles de realizar, mientras que en otras sub-redes del mismo tamaño el procesamiento de los procesos podría ocurrir más rápido. El procesamiento ocurre dependiendo de cuánto se tarden en cumplir las condiciones mencionadas en el Capítulo 3. De esta manera los procesadores terminan sus procesos a diferentes tiempos por lo que algunos de ellos se encuentran inactivos en algún momento sin procesar alguna tarea.

Cuando se ocupa un número muy pequeño de particiones como 4, 8, y 32 el tiempo en que se llevan acabo los procesos de intrusión y de retracción es mejor que el obtenido por el esquema probado de un nodo multiprocesador (de hilos en OpenMP). El tiempo disminuye

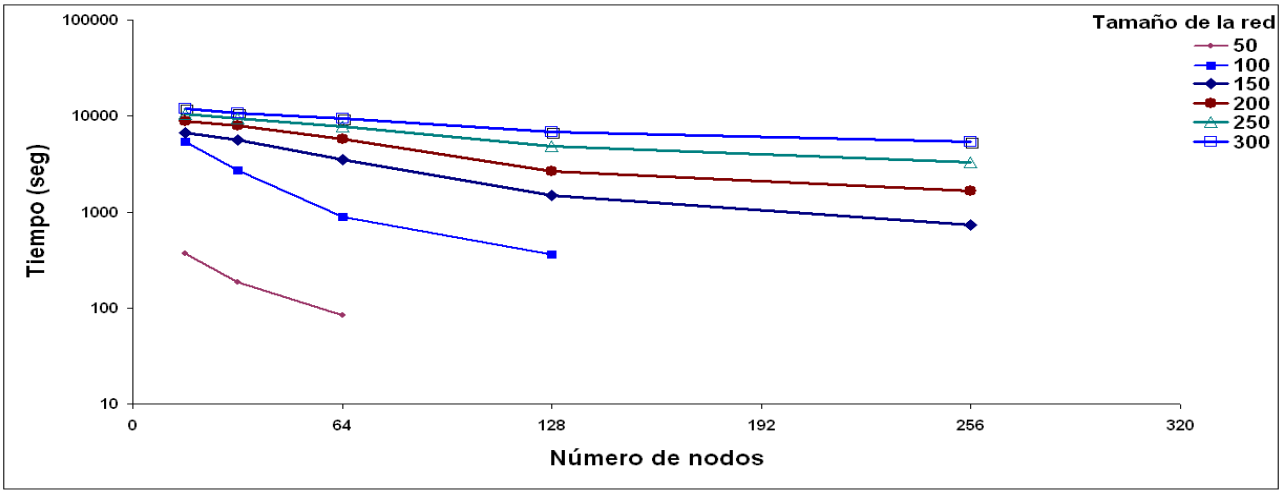


Figura 6.8: Mejor número de procesos para la retracción en un proceso por nodo.

debido a que al repartir las sub-redes las tareas pueden concluirse en un menor tiempo. Sin embargo, conforme se van aumentando el número de particiones el tiempo de ejecución sigue disminuyendo, por lo tanto se siguen probando para buscar la mejor configuración. Cuando se generan 64, 128 y 256 particiones el tiempo disminuye más rápidamente en las redes de tamaño de 100 hasta 250, debido a que la distribución de carga entre los procesadores disminuye pues se están ocupando 256 unidades de procesamiento. No obstante cuando se generan 512 particiones el tiempo disminuye aun más que en las particiones anteriores. Esto se debe a que entre más procesos estén trabajando en las particiones (sub-redes), el procesamiento también es distribuido entre los procesadores y si existen sub-redes difíciles de procesar por lo mencionado anteriormente con mas procesadores usados se lleva acabo el proceso más rápidamente.

Al utilizar más procesadores la sub-red es procesada más rápidamente debido a que se mejora la distribución de la carga entre ellos. Sin embargo, cuando se ocupan un total de de 1024 procesadores el tiempo aumenta al llevar acabo los procesos de intrusión y retracción. El tiempo aumenta debido a que la carga entre los procesadores es mayor porque se están usando más comunicaciones, y además se generan más procesos en los procesadores totales del cluster.

Para las pruebas realizadas al ocupar solo un proceso por nodo se observa que el tiempo en que se realizan los procesos de intrusión y retracción disminuye aun más en redes de menor tamaño. Para redes de tamaño 50, 100, 150 y 200 la disminución del tiempo es mayor que en las redes de 250 y 300. Esto se debe porque la cantidad de poros en redes de tamaño 250 y 300 es de 46875000 y 81000000 poros respectivamente, y al realizar las operaciones de los procesos de intrusión y retracción consumen una mayor cantidad de memoria y procesamiento. Aunque el nodo presente una mayor cantidad de memoria y de procesadores en redes de tamaños de 250 y 300 se consumen más rápidamente estos recursos por la cantidad de operaciones que se realizan sobre los poros de la red, y hace que el tiempo aumente. No obstante en las redes de tamaño pequeño los recursos de memoria y procesamiento no se consumen tan rápidamente haciendo que el tiempo en que se llevan a cabo los procesos de intrusión y retracción sea menor que en las redes grandes.

Con los resultados obtenidos al comparar el tiempo secuencial (sección 3.4) con el las dos versiones en paralelo, se observa que se obtiene una ganancia para las dos versiones. En la Figura 6.9 se muestran los tiempos secuencial para redes de tamaño 150, 200, 250 y 300 que son comparados con los dos esquemas para el proceso de retracción (que consume mayor tiempo de ejecución). Se observa que para redes más grandes el tiempo disminuye hasta en 100 veces lo que nos indica que se obtiene una ganancia para las dos versiones del simulador, y con la segunda versión se mejora el tiempo aun más.

La memoria en RAM que se usa al realizar los procesos de intrusión y retracción de forma secuencial es mayor que cuando se realiza en la segunda versión del simulador. La razón por la que ocurre este comportamiento se debe a que en la versión de hilos la memoria es compartida por los diferentes hilos lanzados y esto hace que el proceso tome menos tiempo, en comparación con la versión secuencial en donde toma mas tiempo y el uso de memoria aumenta conforme aumenta el tiempo. Cuando se utiliza la segunda versión del simulador la memoria es distribuida entre cada uno de los nodos del cluster, por lo que la memoria también es distribuida y el uso de la memoria RAM disminuye en cada nodo. En la Tabla 2 se observa el uso de memoria RAM utilizada en las segunda versión del simulador comparadas con la

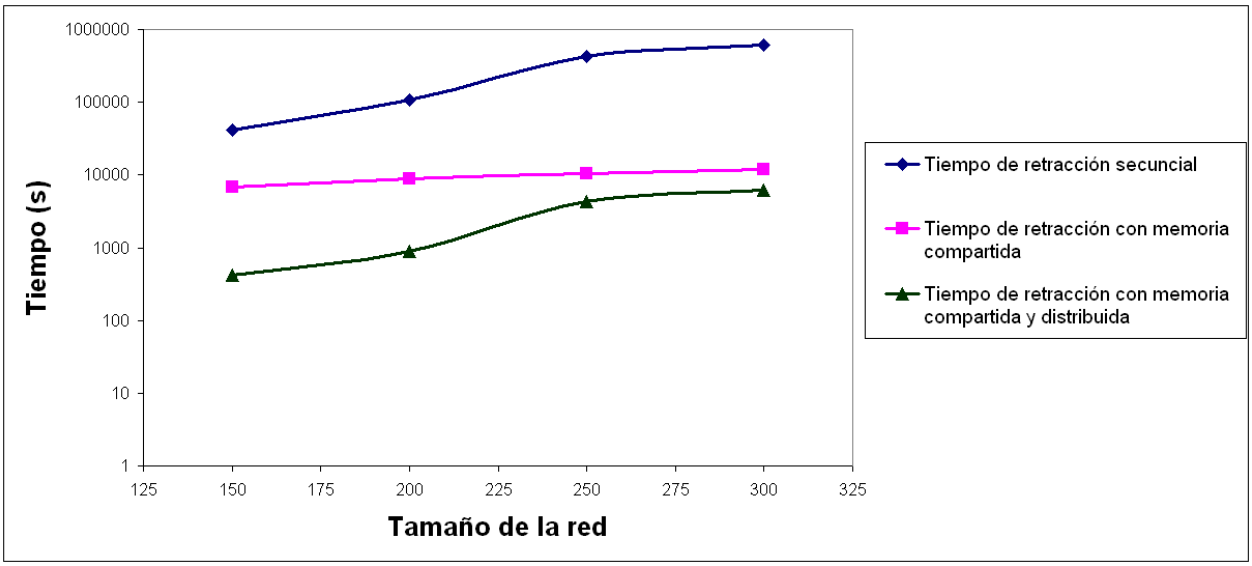


Figura 6.9: Tiempos en las tres versiones para la retracción

versión secuencial. En la versión secuencial es mayor el uso de memoria RAM para redes de tamaño mayor, para la red de 300 en el proceso de retracción disminuye hasta en 11 veces de 6 Gb a 460 Mb.

Red	*Uso de memoria*	*Uso de RAM secuencial*	*Uso de RAM paralela*
150	*54 Mb*	*500 Mb*	*64 Mb*
200	*128 Mb*	*3 Gb*	*100 Mb*
250	*250 Mb*	*4 Gb*	*240 Mb*
300	*500 Mb*	*6 Gb*	*460 Mb*

Tabla 2. Uso de memoria RAM para la simulación

En el siguiente Capítulo se muestran las discusiones así como las conclusiones del proyecto realizado, así como el trabajo futuro.

Capítulo 7

Conclusiones y recomendaciones para el trabajo futuro

En esta tesis se ha presentado la elaboración de un simulador paralelo para dos procesos de la porosimetría de mercurio, el proceso de intrusión y el proceso de retracción. Este simulador paralelo permite ejecutar los procesos de porosimetría de mercurio en una red modelada por computadora que representa un medio poroso. La manera en la que se representa la red porosa es utilizando el DSBM que es un modelo diseñado en la UAM-I, y que permite que las redes porosas se asemejen mucho a la realidad. En cuanto al simulador que proponemos, este fue diseñado siguiendo el modelo de memoria compartida y distribuida y ambos implementados con OpenMP y MPI respectivamente.

La versión inicial se realizó para una red que se procesa en un modelo multiprocesador con esquemas de memoria compartida. Esta versión se probó en diferentes nodos con diferente número de procesadores, y se noto que la tendencia hacia el mejor rendimiento se obtiene al usar el mismo número de hilos que procesadores. Se probó que en un nodo de 4 procesadores el número ideal de hilos fue 4, y al probarlo en un nodo de 8 procesadores la mejor configuración se obtiene al usar 8 hilos. Este resultado se obtuvo con una versión del simulador en donde se reparte equitativamente las caras de la red entre los hilos para realizar los procesos de intrusión y de retracción. Esto nos indica que entre mejor se reparten (de manera equitativa) los datos entre los hilos se obtienen los mejores resultados. De igual manera con el esquema equitativo se observa que los procesadores se encuentran trabajando en un 100 %, indicando

que se explota al máximo los recursos paralelos del esquema multiprocesador. Con la versión de hilos no equitativa el tiempo es mayor en comparación con la versión de hilos equitativa.

Considerando los resultados de la primera versión se generó una segunda versión del simulador, en la que se particiona y distribuye la red porosa cúbica. Estas particiones de la red se hacen para crear sub-redes asignándolas a diferentes procesadores para así distribuir el uso de memoria y los requerimientos de procesamiento entre todo el conjunto de procesadores. En esta versión los recursos de memoria compartida en cada nodo son explotados, pero es agregado el protocolo de paso de mensajes para permitir la comunicación entre los procesadores. En esta versión se buscó la mejor configuración en cuanto a procesos utilizados en diferentes sistemas de cómputo. En un cluster de 32 nodos (128 procesadores) la mejor configuración se obtuvo al usar 256 procesos, dos procesos por procesador.

En el cluster Aitzaloa se probaron 256 procesadores y en esta configuración se obtuvo el mejor tiempo con 512 procesos lanzados. Cuando se ocupan de la mejor manera posible los recursos del cluster el tiempo disminuye aún mas pues la distribución de carga se realiza de una manera más adecuada, esto se ve cuando ejecutamos el simulador paralelo usando un proceso por nodo. Con estos resultados se muestra una buena escalabilidad cuando se aumenta el número de procesadores a utilizar.

Otra conclusión que se determina al utilizar el simulador paralelo, es que cuando son ocupadas redes de mayor tamaño, la simulación de los dos procesos de porosimetría de mercurio toma más tiempo, debido a la gran cantidad de datos que se manejan y las operaciones que realizan. Ademas en redes de tamaño muy grande el uso de memoria aumenta, debido a que las redes son cargadas en memoria para su ejecución. Cuando se utiliza el esquema de memoria compartida y distribuida el uso de memoria se distribuye en los diferentes nodos del cluster utilizado, reduciendo así el uso de memoria por nodo. En el simulador, al distribuir la red en los diferentes nodos se aprovechan más los recursos de memoria y de procesamiento distribuidos en las máquinas utilizadas, haciendo que el tiempo que toma ejecutar los procesos de intrusión y retracción disminuya.

Al comparar los tiempos obtenidos para los procesos de intrusión y retracción de mercurio

con los algoritmos secuenciales mencionados en el Capítulo 3 se obtienen mejores resultados en cuanto a menor uso de memoria y procesamiento al distribuir las tareas en varias maquinas.

El simulador paralelo mejoró los tiempos secuenciales obtenidos para los procesos de intrusión y de retracción de mercurio hasta en 100 veces en redes de mayor tamaño lo que nos indica que se mejora con este simulador la ejecución de estos procesos.

Trabajo futuro

Hasta ahora, el simulador paralelo ha sido probado en sistemas multiprocesador / multicomputador en diferentes sistemas de clusters mostrando buena escalabilidad al aumentar el número de procesadores. Sin embargo, conforme se aumenta el tamaño del problema, se ocupa tanta memoria que los tiempos ya no se pueden reducir. Una alternativa es migrar el simulador paralelo a sistemas de Grids. En estos sistemas se tiene un conjunto de clusters, con los cuales se pueden ejecutar los procesos de intrusión y retracción. Con este tipo de sistemas el procesamiento se puede distribuir en una mayor cantidad de nodos contando con una gran cantidad de recursos de memoria y procesamiento. En el simulador paralelo se generan las particiones de tal manera que el máximo número de particiones en una red de tamaño L es de $2 * 2 * L$ particiones en total y este número es el máximo de procesos que ejecuten. Por esta razón se puede buscar una nueva manera de particionar la red para que se pueda tener un número mayor de particiones en la red y se pueda probar un número mayor de procesos para la red.

También se puede extender el simulador de tal manera que se puedan generar particiones de manera automática en la red. Es decir, para un sistema de cluster de determinadas características se puede determinar el número ideal de particiones (procesadores) para que las redes porosas sean procesadas eficientemente. Esto se puede lograr con experimentos en diferentes sistemas de cluster, homogéneos, heterogéneos, etc.
En este simulador se puede particionar la red de como se describio anteriormente, y se tiene un tamaño máximo de particiones dependiendo del tamaño de la red, no obstante se podría crear una nueva manera en la que se realicen las particiones para que el número de

particiones sea el mismo en todas las redes. Sin embargo, al buscar una nueva manera de particionar la red se debe tomar en cuenta los principios de la intrusión o retracción para que se respeten todas las condiciones de la mejor manera posible. Finalmente, también se pueden diseñar otras versiones del simulador paralelo con diferentes herramientas de programación paralela para determinar su eficiencia, al comparar con las versiones actuales del simulador. De esta manera, se puede evaluar con este simulador el desempeño de herramientas de programación paralela comparando los tiempos y los resultados obtenidos entre cada una de las simulaciones.

Apéndice 1

En este apéndice se muestra el pseudocódigo de los algoritmos recursivos diseñados para la intrusión y de la retracción. En la primera parte se muestra el pseudocódigo de la función de llamadas recursivas para la intrusión, y el segundo muestra el pseudocódigo del algoritmo de intrusión de mercurio.

Rntru_rec

```
1. {
2.   si(cumple_cond_llenado_enlace())
3.  {
4.     llena_enlace( );//llena el enlace
5.       si(cumple_condición_llenado_sitio( ))
6.      {
7.         intru_rec(red, ARRIBA);//intruye por el enlace de arriba
8.         intru_rec(red, ABAJO);// intruye por el enlace abajo
9.         intru_rec(red, IZQUIERDA);// intruye por el enlace de izquierda
10.         intru_rec(red, DERECHA);// intruye por el enlace de derecha
11.         intru_rec(red, FRENTE);// intruye por el enlace de frente
12.         intru_rec(red, FONDO);// intruye por el enlace de fondo
13.      }// llamadas recursivas a las 6 posiciones
14.  }
15. }
```

Intrusión

```
1. {
2.   Mientras (presión ≤ presion_final))
3.  {
4.     para (i=0;i<L;i++)
5.       para (j=0;j<L;j++)
6.      {
7.         intru_rec(FRENTE);//cara de enfrente
8.         intru_rec(FONDO);//cara fondo
9.      }
10.     para (j=0;j<L;j++)
11.       para (k=0;k<L;k++)
12.      {
13.         intru_rec(IZQUIERDA);//cara izquierda
14.         intru_rec(DERECHA); //cara derecha
15.      }
16.     para (i=0;i<L;i++)
17.       para (k=0;k<L;k++)
18.      {
19.         intru_rec(ARRIBA);//cara arriba
20.         intru_rec( ABAJO);//cara abajo
21.      }
22.       si (sitios_intruidos = sitios_totales y vol_acum == vol_esperado)
23.         Aumenta presión( );
24.    }
25. }
```

En la primera parte se muestra el pseudocódigo de la función de llamadas recursivas para la retracción, y el segundo muestra el pseudocódigo del algoritmo de retracción de mercurio.

Retra_rec

```
1. {
2.   si(cumple_cond_vaciado_enlace())
3.  {
4.     Vacia_enlace( );//llena el enlace
5.       si(cumple_condición_vaciado_sitio( ))
6.      {
7.         Retra_rec(red, ARRIBA);//intruye por el enlace de arriba
8.         Retra_rec(red, ABAJO);// intruye por el enlace abajo
9.         Retra_rec(red, IZQUIERDA);// intruye por el enlace de izquierda
10.         Retra_rec(red, DERECHA);// intruye por el enlace de derecha
11.         Retra_rec(red, FRENTE);// intruye por el enlace de frente
12.         Retra_rec(red, FONDO);// intruye por el enlace de fondo
13.      }// llamadas recursivas a las 6 posiciones
14.  }
15. }
```

Retracción

```
1. {
2.   Mientras (presión ≤ presion_final ))
3.  {
4.     para (i=0;i<L;i++)
5.      para (j=0;j<L;j++)
6.     {
7.       si ((Enlace_no_Vacio(FRENTE))) //
8.            retrac_rec(FRENTE);//cara de enfrente
9.       si ((Enlace_no_Vacio(FONDO)))//
10.             retrac_rec(FONDO);//cara de enfrente
11.     para (j=0;j<L;j++)
12.      para (k=0;k<L;k++)
13.     {
14.       si ((Enlace_no_Vacio(IZQUIERDA))) //
15.            retrac_rec(IZQUIERDA);//cara de enfrente
16.       si ((Enlace_no_Vacio(DERECHA)))//
17.             retrac_rec(DERECHA);//cara de enfrente
18.     para (i=0;i<L;i++)
19.      para (k=0;k<L;k++)
20.     {
21.       si ((Enlace_no_Vacio(ARRIBA))) //
22.            retrac_rec(ARRIBA);//cara de enfrente
23.       si ((Enlace_no_Vacio(ABAJO)))//
24.             retrac_rec(ABAJO);//cara de enfrente
25.       si (sitios_retraidos = sitios_totales y vol_acum == vol_esperado)
26.         Disminuye_presión( );
27.    }
28. }
```

Referencias

[1] V. Mayagoitia, F. Rojas, I. Kornhauser, G. Zgrablich, J. Riccardo; *Fluid-Phase Morphologies Induced by Capillary Processes in Porous Media*; Characterization of Porous Solids III. Studies in Surface Science and Catalysis; J. Rouquerol, F. Rodríguez- Reinoso, K. S. W. Sing, K. K. Unger, Eds.; Elsevier, 87 (1994) 141-150.

[2] V. Mayagoitia, F. Rojas, I. Kornhauser; *Twofold Description of Porous Media and Surface Structures: A Unified Approach to Understand Heterogeneity Effects in Adsorption and Catalysis*; Langmuir; 9, 10 (1993) 2748-2754.

[3] S. Cordero, F. Rojas, J. L. Riccardo; *Simulation of Three-Dimensional Porous Networks*; Colloids and Surfaces A: Physicochemical and Engineering Aspects; 187-188 (2001) 425-438.

[4] B. Zhang; *Relationship Between Pore Structure and Mechanical Properties of Ordinary Concrete Under Bending Fatigue*; Cement and Concrete Research; 28, 5 (1998) 699-711.

[5] Mayagoitia, M. J. Cruz, F. Rojas, I. Kornhauser, G. Zgrablich, V. Pereyra; *Simulation of Heterogeneous Surfaces of Adsorbents by Monte Carlo Methods*; Gas Separation & Purification; 6, 1 (1992) 35-41.

[6] A. J. Ramirez-Cuesta, S. Cordero, F. Rojas, R. J. Faccio, J. L. Riccardo; *On Modeling, Simulation and Statistical Properties of Realistic Three Dimensional Porous Networks*; Journal of Porous Materials; 8 (2001) 61-76.

[7] V. Mayagoitia, F. Rojas, I. Kornhauser, H. Pérez-Aguilar; *Modeling of Porous Media and Surfaces Structures: Their True Essence as Networks*; Langmuir; 13, 5 (1997) 1327-1331.

[8] G. B. Kuznetsova, V. Mayagoitia, I. Kornhauser; *Twofold Description of the Morphology of Polymers*; Intern. J. Polymeric Mater.; 19 (1993) 19-28.

[9] V. Mayagoitia, I. Kornhauser; *Capillary Processes in Porous Networks: 1. Models of Porous Structures*; En Principles and Applications of Pore StructuralCharacterization; J. M. Haynes, P. Rossi-Doria, Eds.; J. W. Arrowsmith, Bristol, (1985) 15-26.

[10] Z. Lu, X. Zhou; *The Waterproofing Characteristics of Polymer Sodium Carboxymethyl-Cellulose*; Cement and Concrete Research; 30 (2000) 227-231.

[11] G. Zgrablich, S. Mendioroz, L. Daza, J. Pajares, V. Mayagoitia, F. Rojas, W. C. Conner; *Effect of Porous Structure on the Determination of Pore Size Distribution by Mercury Porosimetry and Nitrogen Sorption*; Langmuir; 7, 4 (1991) 779-785.

[12] M. A. Ioannidis, I. Chatzis, F. A. L. Dullien; *Macroscopic Percolation Model of Immiscible Displacement: Effects of Buoyancy and Spatial Structure*; Water Resourses Research; 23 (1996) 3297-3310.

[13] C. D. Tsakiroglou, A. C. Payatakes;*Mercury Intrusion and Retraction in Model Porous Media*; Advances in Colloid and Interface Science; 75 (1998) 215-253.

[14] S. Cordero, F. Rojas, J. L. Riccardo; *Simulation of Three-Dimensional Porous Networks*; Colloids and Surfaces A: Physicochemical and Engineering Aspects; 187-188 (2001) 425-438.

[15] Ö. Z. Cebeci; *The Intrusion of Conical and Spherical Pores in Mercury Intrusion Porosimetry*; Journal of Colloid and Interface Science; 78, 2 (1980) 383-388.

[16] C. D. Tsakiroglou, G. B. Kolonis, T. C. Roumeliotis, A. C. Payatakes; *Mercury Penetration and Snap-off in Lenticular Pores*; Journal of Colloid and Interface Science; 193, (1997) 259-272.

[17] S. P. Rigby, R. S. Fletcher, S. N. Riley; *Determination of Cause of Mercury Entrapment during Porosimetry Experiments on Sol-Gel Silica Catalyst Supports*; Applied Catalysis A: General; 247 (2003) 27-39.

[18] I. Foster, J. Geisler, B. Nickless, W. Smith, S. Tuecke, *Software Infrastructure for High-Performance Distributed Computing*, to appear in Proc. HPDC-5, 1996.

[19] Michael Flynn. *Some computer organizations and their effectiveness.* IEEE Transactions on Computers, C-21 (9):948-960, September 1972.

[20] Tobias Wittwer *An Introducction to Parallel Programing.*, VSSD First edition 2006 ISBN-10 90-71301-78-8

[21] George Em Karniadakis and Robert M. Kirby II *Parallel Scientific Computing in C++ and MPI. A seamless approach to parallel algorithms and their implementation* Cambridge University Press, New York, 2003 ISBN 0-521-52080-0

[22] Rohit Chandra, Ramesh Menon, Leo Dagum, David Kohr, Dror Maydan, and Jeff McDonald. *Parallel Programming in OpenMP.* Morgan Kaufmann, October 2000.

[23] Leonardo Dagum and Ramesh Menon. *OpenMP: An industry-standard API for shared-memory programming.* IEEE Computational Science and Engineering, 46-55, January-March 1998.

[24] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM 3 user´s guide and reference manual.* Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, September 1994.

[25] G. A. Geist and V. S. Sunderam, *Network Based Concurrent Computing on the PVM System* Journal of Concurrency, Practice and Experience. 4, 293-311.

[26] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine, A User´s Guide and Tutorial for Networked Parallel Computing.* Scientific and Engineering Computation. MIT Press, November 1994.

[27] *Message Passing Interface Forum. MPI: A message passing interface standard.* International Journal of Supercomputing Applications, 8: 169-416, 1994.

[28] Marc Snir, Steve Otto, Steve Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference, volume 1.*Scientific and Engineering Computation. MIT Press, 2nd edition, September 1998.

[29] Brian W. Kernighan and Dennis M Ritchie. *The C Programming Language..* Pearson Education, second edition, August 2001.

[30] L. Lamport and N. Lynch. *Distributed computing: Models and methods.* In J. Van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B, chapter 188. Elsevier and MIT Press 1990.