ANÁLISIS DE DESEMPEÑO EN DOS ALGORITMOS PARALELOS PARA LA RECONSTRUCCIÓN DE IMÁGENES ANGIOGRÁFICAS *

Robinson Rivas Suárez[1,2], María Blanca Ibáñez[2]

[1] Facultad de Ciencias, Escuela de Computación, Centro de Computación Paralela y Distribuida.
Universidad Central de Venezuela. Apartado 47002, Caracas 1041-A, Venezuela
rrivas@strix.ciens.ucv.ve

[2] Departamento de Computación y Tecnología de la Información.
Grupo de Paralelismo y Sistemas Distribuidos.
Universidad Simón Bolívar, Apartado 89000, Caracas 1080-A, Venezuela
{rivas.suarez,ibanez}@ldc.usb.ve

Resumen

El diagnóstico adecuado de ciertas dolencias cardiovasculares es posible mediante la obtención de imágenes tridimensionales (3D) de las arterias coronarias. Estas imágenes 3D se obtienen a partir de un espacio de soluciones generado por la proyección geométrica de dos imágenes angiográficas bidimensionales. Un conjunto de condiciones fijadas *a priori* por bioingenieros permiten discriminar entre soluciones válidas e inválidas.

Un algoritmo secuencial basado en la combinación de segmentos arteriales prolongados *(ramificaciones unitarias)* permite generar el conjunto de todas las soluciones permitidas por las condiciones con un alto tiempo de cómputo. Dicho algoritmo es paralelizado para reducir el tiempo de respuesta. En este trabajo se presentan y discuten dos alternativas de paralelización y se analizan su desempeño y balance de carga.

Palabras clave: Computación Paralela. Aplicación Médica. Análisis de Desempeño. Balance de Carga.

* Este trabajo fue financiado por el proyecto ITDC 139-82158 A2IM de la CEE. Las pruebas se desarrollaron en el Centro de Computación Paralela y Distribuída de la Universidad Central de Venezuela.

1.- Introducción.

El problema de reconstrucción tridimensional de imágenes angiográficas es un problema de optimización discreta donde se tiene un conjunto finito pero extenso de soluciones que satisfacen ciertas condiciones (filtros). Las soluciones representan todas las posibles representaciones en el espacio del árbol coronario que pueden construirse a partir de dos imágenes angiográficas planas tomadas simultáneamente desde dos ángulos distintos. Tal espacio de soluciones tiene unas dimensiones que imposibilitan su tratamiento de manera convencional mediante algoritmos de búsqueda [WB&C95].

La implantación de algoritmos de búsqueda para resolver este tipo de problemas resulta en sistemas con un elevado tiempo de cómputo. Debido a esto se ha estudiado el impacto de los filtros para reducir el espacio de soluciones [WLB&C95], al mismo tiempo que se han propuesto diversos algoritmos y alternativas de paralelización.

Una primera solución al problema [C&R97], [RC&I97] consistió en la comparación de dos tipos de paralelizaciones: funcional [1] y por datos[2] sobre un algoritmo secuencial desarrollado por P. Windyga [Win94]. De esta experiencia se concluyó que para las condiciones de discriminación dadas, el mejor método es el de paralelización por datos. La experimentación realizada con el algoritmo hallado, permitió mejorar los filtros y desarrollar un nuevo algoritmo secuencial susceptible a una mejor paralelización. Éste último [RI&C99] se basó en la técnica de *ramificaciones unitarias*. Usando esta técnica, es posible estimar *a priori* espacios de soluciones disjuntos pero, debido al gran volumen de datos involucrados, es necesario recalcular subpartes de posibles soluciones antes que comunicar información entre procesos.

En este trabajo se presenta una nueva versión del algoritmo basado en *ramificaciones unitarias* que, gracias a un mecanismo de empaquetamiento de datos, evita la repetición sistemática de trabajo a cambio de establecer intercambio de datos compactados entre procesos. La descripción y análisis de la técnica de empaquetamiento no forman parte de este estudio, aquí presentamos las paralelizaciones logradas y su comportamiento.

El trabajo se desarrolló implementando los programas en lenguaje C y utilizando la librería de comunicaciones nativa del Sistema de Operación PARIX, que opera en modo asíncrono. Nuestra plataforma de trabajo consistió en una máquina Parsytec MC-3DE con 4 procesadores, conectada a un host SUN SparcStation, del Centro de Computación Paralela y Distribuida de la Universidad Central de Venezuela.

El artículo está organizado de la siguiente manera. En la sección 2 se describe en detalle el algoritmo secuencial y se estudia el problema del espacio de almacenamiento como aspecto resaltante del algoritmo. En las secciones 3 y 4 se describen dos técnicas paralelas aplicadas al algoritmo y se analizan los resultados obtenidos, particularmente lo relacionado con el balance de carga y el desempeño. Finalmente en la sección 5 se presentan las conclusiones y el trabajo futuro de esta investigación.

[1] Los procesadores ejecutan tareas diferentes sobre el mismo conjunto de datos
[2] Los procesadores realizan tareas similares sobre diferentes datos

2.- Descripción del Algoritmo Secuencial

El algoritmo recibe como entrada un grafo que contiene todas las representaciones tridimensionales posibles del árbol coronario (*Grafo de Soluciones Totales*: GST) y produce como salida un conjunto de soluciones válidas (*Conjunto de Soluciones Parciales*: CSP). Cada solución posible del CSP (*Árbol de Soluciones Parciales*: ASP) es válida sí y sólo si satisface el conjunto de filtros que se describe a continuación

- **Condición de Ramificación (filtroCR)**
  En el tronco arterial (*tronco común*) puede haber bifurcaciones y trifurcaciones de arterias. A partir de ese punto la única ramificación permitida es la de bifurcación. Tampoco puede haber ciclos (dos arterias diferentes que converjan en el mismo punto).
- **Superposición Aceptable (filtroSA)**
  Las arterias pertenecientes a una solución no deben tener una superposición visual una sobre otra que supere un umbral preestablecido.
- **Verticalidad Aceptable (filtroVA)**
  Las arterias deben tener una orientación vertical con respecto a la línea central del corazón, en sentido descendente a partir de la arteria principal.
- **Dirección Aceptable (filtroDA)**
  Debe verificarse que ninguna arteria vaya en sentido contrario al del flujo sanguíneo ni tampoco en dirección tal que atraviese el músculo cardíaco.
- **Condición de Completitud (filtroCC)**
  Cada arteria de una proyección 2D debe tener su imagen en las soluciones obtenidas.

El algoritmo secuencial (ver **Algoritmo 1**) se divide en dos partes fundamentales: el *Preprocesamiento* y la *Combinación de Unitarias*. El preprocesamiento permite reducir el espacio de búsqueda antes de comenzar el proceso de obtención de soluciones, mediante la conversión del grafo de entrada (**GST**) en una estructura arborescente (paso **1.1** del algoritmo). Durante esta conversión se aplica el filtro **filtroVA**. Luego se generan ramas arteriales extendidas (paso **1.2**) donde se producen las *Ramificaciones Unitarias*, aplicándose el filtro **filtroDA**.

Es en la etapa 2 del algoritmo donde se construye iterativamente el conjunto de soluciones. En el paso **2.1** se construyen todas las soluciones posibles que constan de dos unitarias, conjunto que sirve de base para el proceso siguiente. En el paso **2.2** se construyen, a partir de un conjunto de soluciones de **n** unitarias, las soluciones que puedan tener **n+1** unitarias mediante la adición selectiva de nuevas unitarias a las soluciones previas. En cada uno de estos pasos se debe verificar que las soluciones obtenidas cumplan los filtros **filtroSA** y **filtroCR**, usándose la función **válida()**.

Una de las características más importantes de este algoritmo, es que necesita un gran espacio de almacenamiento para las soluciones parciales, debido a que entre una iteración y otra se debe preservar todo el conjunto de soluciones parciales válidas. En el paso **2.1** puede verse que el conjunto ***Soluciones*** se crea con todas las combinaciones posibles que puedan tener dos unitarias. Hasta este punto no se considera la ***Condición de Completitud***. Con datos de prueba experimentales y valores estándar para los filtros establecidos por los bioingenieros, el conjunto ***Soluciones*** para dos unitarias tiene decenas de

miles de posibles soluciones válidas. Más aún, para lograr la efectiva combinación y construcción de los resultados, es necesario además conservar el conjunto ***Soluciones-Parciales*** lo que incrementa el costo en espacio.

1: Preprocesamiento y cálculo de Ramificaciones Unitarias

1.1: Preprocesamiento

Se toma el GST y se convierte a una representación de grafo dirigido arborescente (*Árbol de Soluciones Totales*: AST). Durante ese proceso, se aplica a cada par de segmentos contiguos el filtro de *Verticalidad Aceptable*.

1.2: Cálculo de Ramificaciones Unitarias:

Se construyen todos los caminos posibles desde la raíz del AST hasta sus hojas, aplicando a cada par de segmentos contiguos el filtro de *Dirección Aceptable*. Cada camino válido desde la raíz hasta una hoja del AST lo llamaremos *Ramificación Unitaria*. Llamemos *UNITARIAS* el conjunto de todas las ramificaciones halladas.

2: Hacer Combinaciones

2.1: Generación de soluciones parciales de dos (2) unitarias

Soluciones ← { }

∀ par=(*unitaria*$_1$, *unitaria*$_2$) ∈ *UNITARIAS* :

if (válida(*unitaria*$_1$,*unitaria*$_2$, filtroSA,filtroCR))

then *Soluciones* ← *Soluciones* U { <*unitaria*$_1$, *unitaria*$_2$>}

2.2: Generar nuevas soluciones a partir de un conjunto previo

loop

2.2.1: Creación de un nuevo conjunto *solución*

Sol-Parciales ← { }

∀ *sol-parcial*$_i$ ∈ *Soluciones* :

sea *sol-parcial*$_i$ = <*unitaria*$_1$...*unitaria*$_n$>

∀ *unitaria*$_i$ ∈ *UNITARIAS* / *unitaria*$_i$ no está en <*unitaria*$_1$...*unitaria*$_n$>:

if (∀ *unitaria*$_j$ ∈ *UNITARIAS* :

(*unitaria*$_j$ está en <*unitaria*$_1$...*unitaria*$_n$>) and

(válida(*unitaria*$_i$,*unitaria*$_j$,filtroSA,filtroCR)))

then *Sol-Parciales* ← *Sol-Parciales*

U

{<*unitaria*$_1$...*unitaria*$_n$, *unitaria*$_i$>}

if (Soluciones-Parciales = { })

then break loop

else Soluciones ← Soluciones-Parciales

end loop

Algoritmo 1: Algoritmo Secuencial

El problema de espacio, trae como consecuencia un costo adicional u *overhead* para este algoritmo, pues no es posible almacenar todas las combinaciones utilizando estructuras de datos convencionales sin que se presenten problemas de espacio. Aun si se utilizaran estructuras de datos sencillas para el almacenamiento de los conjuntos, eso traería como consecuencia que las versiones paralelas en las cuales es necesario comunicar datos entre procesadores, tardaría demasiado tiempo en tales comunicaciones, lo que afectaría considerablemente el desempeño.

Sin embargo, poder almacenar todas las soluciones parciales trae como ventaja el que no se necesite volver a probar una solución parcial que ya haya sido descartada. Si el tiempo de cómputo de generar las soluciones es mucho mayor que el tiempo requerido para manipular la estructura de datos y enviar mensajes entre procesadores, ésta técnica resultará más adecuada para los fines de reducción del tiempo total de cómputo. En este trabajo se utiliza una técnica de compactación que permite una manipulación adecuada de conjuntos de datos grandes a expensas de un mayor *overhead* para la compresión-descompresión de los datos.

Desempeño del algoritmo secuencial

Para probar el algoritmo secuencial se utilizó un juego de datos de imágenes angiográficas que genera 1800 unitarias una vez terminado el paso 1 del algoritmo. De las 1800 unitarias se tomaron como conjunto *UNITARIAS* solamente las primeras 100.

Los tiempos del algoritmo secuencial se muestran en la **Tabla 1**, y corresponden a una única ejecución del algoritmo secuencial. El resto de los tiempos mostrados en las otras tablas son promedios de 4 ejecuciones del algoritmo. El módulo ***Preprocesamiento y Cálculo de unitarias*** se refiere al paso **1** del algoritmo secuencial. El módulo ***Soluciones de dos unitarias*** equivale al paso **2.1**, mientras el módulo ***Generar nuevas soluciones*** equivale al paso **2.2** del algoritmo.

Módulo	Tiempo (seg.)
Preprocesamiento y cálculo de unitarias	5,27
Soluciones de dos unitarias	7,70
Generar nuevas soluciones	401,92
Total	414,89

Tabla 1: Tiempos del programa secuencial

Se puede observar que la mayor parte del tiempo de cómputo se utiliza en generar nuevas combinaciones a partir de las anteriores. El preprocesamiento y cálculo de las unitarias se realiza en tiempo despreciable.

3.- Paralelización por distribución estática de los datos.

Como se mencionó anteriormente, el esquema de paralelización que ha mostrado mejores resultados es la paralelización por datos, éste consiste en distribuir la carga de trabajo de la manera más apropiada entre los diferentes procesadores disponibles. Eso no solo implica una repartición equitativa de los datos (y por ende del total de trabajo) sino que además se debe tomar en consideración aspectos como las comunicaciones entre los procesadores, ya que debido a las dimensiones de los datos a

procesar, la transferencia de información puede consumir tiempos de comunicación muy altos en comparación al proceso.

La primera alternativa a estudiar es la *Distribución Estática de los Datos*. Tomando un procesador como *Procesador Maestro* y el resto como *Procesadores Esclavos*, esta técnica consiste en la distribución de espacios de soluciones disjuntos entre todos los procesadores, los cuales resuelven su porción de trabajo y envían de vuelta los resultados obtenidos.

La descripción del algoritmo paralelo de *Distribución Estática de Datos* está en el recuadro **Algoritmo 2.**

PROCESADOR MAESTRO :
1.- Hacer Preprocesamiento (paso 1 Algoritmo Secuencial)
2.- Obtener Soluciones Parciales de 2 unitarias (paso 2.1 Algoritmo Secuencial)
3.- Sea N el número de procesadores. Dividir el conjunto *Soluciones* en N subconjuntos disjuntos del mismo tamaño. Sea $Soluciones_i$ cada uno de esos subconjuntos.
4.- ∀ procesador esclavo $PEsclavo_i$: ENVIAR($Pesclavo_i$,$Soluciones_i$).
5.- Generar nuevas soluciones parciales (paso 2.2 Algoritmo Secuencial). Salida: *Soluciones*.
6.- ∀ procesador esclavo $PEsclavo_i$: RECIBIR($Pesclavo_i$,$Soluciones_i$).
7.- ∀ $Soluciones_i$: *Soluciones* ← *Soluciones* U $Soluciones_i$

PROCESADORES ESCLAVOS:
1.- Hacer Preprocesamiento (paso 1 algoritmo secuencial)
2.- RECIBIR (*Maestro,Soluciones*)
3.- Generar nuevas soluciones parciales (paso 2.2 algoritmo secuencial)
4.- ENVIAR(*Maestro, Soluciones*)

Algoritmo 2: Algoritmo Paralelo de Distribución Estática de Datos.

Como puede observarse hay sólo un paso de comunicación al inicio del algoritmo (para enviar los datos a todos los procesadores esclavos) y un solo proceso de comunicación al final (para que el procesador *Maestro* reciba y una los datos). La principal ventaja de la repartición estática de datos es precisamente la ausencia de comunicaciones. En problemas de búsqueda de tipo combinatorio con restricciones, sin embargo, la principal desventaja es que si no se conoce *a priori* el comportamiento de la búsqueda, se pueden dar anomalías en el desbalance de carga [Kum94].

Desempeño del algoritmo Paralelo Estático

Como se aprecia en la **Tabla 2**, el tiempo de respuesta de la versión paralela es mejor que el de la versión secuencial. En condiciones ideales, el tiempo de respuesta de un algoritmo paralelo con respecto al secuencial debería estar en relación **1/n** para **N** procesadores, lo que se conoce como *aceleración óptima*. Sin embargo, en nuestro experimento dicho tiempo de respuesta no está en relación directa con el número de procesadores que se están utilizando. Las pruebas muestran que la relación obtenida acá es de 1/1.5 en lugar de 1/4. Esto significa que al algoritmo paralelo estático se ejecuta en

0.66 veces el tiempo del algoritmo secuencial, en lugar de hacerlo en 0.25 veces ese tiempo que es la medida teórica óptima para 4 procesadores.

Versión	Tiempo de cómputo (seg.)
Secuencial	414,89
Partición Estática	272,46

Tabla 2: Tiempos del programa secuencial vs. Paralelo estático

Una de las razones principales de este desequilibrio es el desbalance de carga. El monto de trabajo de los diferentes procesadores no está equilibrado, lo que significa que unos procesadores trabajen más que los otros. En la **Tabla 3** se muestra la relación de tiempos entre los diversos procesadores (incluyendo al procesador maestro).

Como puede observarse hay diferencias del tiempo total de mas del 60% (*Maestro vs. Esclavo1*). Esto se debe a que el proceso de búsqueda de las soluciones no es uniforme entre las diferentes unitarias, ya que dependen de la validez de filtros, y ésta validez no se puede determinar *a priori*. Los procesadores ***Maestro*** y ***Esclavo 3*** trabajan mucho más ya que las soluciones iniciales de pares de unitarias que les corresponden permiten generar más soluciones que sus contrapartes en los procesadores 1 y 2, por lo que éstos terminan mucho más rápido el proceso de combinaciones y el resto del tiempo están ociosos, en espera que el *Maestro* termine su ejecución para enviar los resultados.

Procesador	Tiempo de cómputo (seg.)	% del tiempo total
Maestro	268,85	65.50
Esclavo 1	1,31	0.31
Esclavo 2	20,50	4.99
Esclavo 3	119,70	29.10

Tabla 3: Tiempos de los procesadores del programa paralelo estático

Puede observarse que la sumatoria de los tiempos de los procesadores en la versión paralela es casi la misma que el tiempo total del algoritmo secuencial (410,36 seg. el tiempo paralelo vs. 414 seg. el tiempo secuencial). El tiempo total de envío y recepción de mensajes es de 2,53 segundos, lo que no introduce una penalización en tiempo ni afecta los resultados.

4.- Paralelización por distribución dinámica de los datos.

Para solventar el inconveniente del desbalance de carga, se desarrolló otra versión paralela del algoritmo secuencial llamada *Distribución Dinámica de los Datos*. En esta versión, en cada iteración del ciclo de combinación de las unitarias las soluciones parciales obtenidas se envían al procesador *Maestro*, quien agrupa todas las soluciones en un solo conjunto de *Soluciones* y luego las redistribuye equitativamente.

El algoritmo se describe en el siguiente recuadro:

```
PROCESADOR MAESTRO :
1.- Hacer Preprocesamiento (paso 1 Algoritmo Secuencial)
2.- Obtener Soluciones Parciales de 2 unitarias (paso 2.1 Algoritmo Secuencial)
3.- loop
      Sea N el número de procesadores. Dividir el conjunto Soluciones en N subconjuntos
        disjuntos del mismo tamaño. Sea Soluciones_i cada uno de esos subconjuntos.
      ∀ procesador esclavo PEsclavo_i:
          ENVIAR(PEsclavo_i,Soluciones_i).
      Generar nuevas soluciones parciales (paso 2.2 Algoritmo Secuencial). Salida: Soluciones-Parciales.
      ∀ procesador esclavo PEsclavo_i:
          RECIBIR(PEsclavo_i,Soluciones_i).
          Soluciones-Parciales ← Soluciones-Parciales U Soluciones_i
      if Soluciones-Parciales = { }
        then break loop
        else Soluciones <- Soluciones-Parciales
   end loop
4.- ∀ procesador esclavo PEsclavo_i, enviar(PEsclavo_i,{})


PROCESADORES ESCLAVOS :
1.- Hacer Preprocesamiento (paso 1 Algoritmo Secuencial)
2.- loop
      RECIBIR(Maestro,Soluciones)
      if (Soluciones = {})
        then break loop
      Crear un nuevo conjunto Soluciones (paso 2.2.1 Algoritmo Secuencial) Salida: Soluciones
      ENVIAR(Maestro,Soluciones)
   end loop
```

Algoritmo 3: Algoritmo Paralelo de Distribución Dinámica de Datos.

La distribución dinámica produce resultados buenos siempre y cuando el tiempo de comunicaciones entre el procesador *Maestro* y los procesadores *Esclavos* sea bajo. Eso depende tanto de la librería de comunicaciones que se utilice como del tamaño de los datos. En el caso estático, se observa que los tiempos de comunicación son muy bajos debido a que sólo se hacen dos veces, y además la mayor carga de trabajo recae sobre el procesador *Maestro*, lo que hace que la data más voluminosa se encuentre en ese procesador y que no sea necesario transmitirla.

Para este experimento se obtuvieron los resultados que se muestran en la **Tabla 4**

Como se observa, el tiempo de ejecución mostrado en **Tabla 4** es mejor que el de las versiones anteriores. Sin embargo, aún no se alcanza la cota de 1/4 que es de desear para el número de procesadores utilizado. En este caso, el tiempo del algoritmo paralelo es 39% del tiempo secuencial (óptimo : 25%).

Versión	Tiempo de cómputo (seg.)
Secuencial	414,89
Partición Estática	272,46
Dinámica	162,67

Tabla 4: Tiempo de ejecución del programa paralelo dinámico

En el caso de la distribución estática la anomalía se debía al problema de desbalance de carga. Sin embargo, como se observa en la **Tabla 5**, el balance de carga es mejor que en la primera versión, lo que significa que los procesadores tienen poco tiempo ocioso. En esa tabla sólo se contabiliza el tiempo de cómputo, no el necesario para las comunicaciones. Si solamente se tiene en cuenta el tiempo de cómputo, vemos que la mejora es del 31%, así que se debe tomar en cuenta el tiempo de comunicaciones para un mejor análisis del problema.

Procesador	Tiempo de cómputo (seg.)	% del tiempo total
Maestro	129,78	31.5
Esclavo 1	92,45	22.4
Esclavo 2	84,91	20.6
Esclavo 3	104,37	25.3

Tabla 5: Tiempos de los procesadores del programa paralelo dinámico

Al estudiar el algoritmo en detalle, vemos que si hay N procesadores el procesador *Maestro* hará 2x(N-1) veces tantas comunicaciones como iteraciones haya del ciclo **loop** (paso 3 del procesador *Maestro*), **(N-1)** comunicaciones en dirección *Maestro-Esclavo* y **(N-1)** en dirección *Esclavo-Maestro* para unir nuevamente la data. El elevado tiempo de comunicaciones puede atribuirse al tamaño de los datos que deben ir en dos direcciones.

La **Tabla 6** muestra en detalle los tiempos utilizados para las comunicaciones.

Procesador	Tiempo de envío de datos (seg.)	Tiempo de recepción de datos (seg.)	Tiempo total de comunicaciones (seg.)
Maestro	12,08	15,02	27,10
Esclavo 1	48,80	14,48	63,28
Esclavo 2	55,28	15,52	70,80
Esclavo 3	37,68	13,76	51,44

Tabla 6: Tiempos de comunicación de los procesadores del programa paralelo dinámico

La aparente anomalía entre los tiempos de envío y recepción de los procesadores se debe a que se está utilizando comunicación asíncrona. Esto en la práctica significa que los procesadores envían los mensajes de una manera más "rápida" de lo que los reciben, pues no deben esperar asentimiento del otro procesador, mientras que cuando reciben mensajes en muchos casos deben esperar a que el procesador origen del mensaje termine un proceso previo y luego envíe los datos. A pesar de esta anomalía, vemos que la sumatoria de tiempos de comunicación más los tiempos de procesamiento equivalen al tiempo total consumido por el sistema (**Tabla 7**).

Procesador	Tiempo de cómputo	Tiempo de comunicación	Tiempo total del procesador
Maestro	129,78	27,10	156,88
Esclavo 1	92,45	63,28	155,73
Esclavo 2	84,91	70,80	155,71
Esclavo 3	104,37	51,44	155,81

Tabla 7: Tiempos de comunicación y tiempos de procesamiento de los procesadores del programa paralelo dinámico

La sumatoria de los tiempos de cómputo es de 411,51 segundos, mientras la sumatoria de los tiempos de comunicación es de 212,62 segundos. Esto significa que el tiempo de comunicaciones equivale al 34,1% del tiempo total, mientras el tiempo de cálculo representa el 65,9%. Esta relación es peor que para el caso estático y se debe a la cantidad de comunicaciones de una data de gran tamaño y al *overhead* creado por la manipulación de tales datos.

5.- Conclusiones y trabajo futuro

Las pruebas experimentales demuestran que los tiempos de los algoritmos paralelos son mejores que los del algoritmo secuencial. Para el caso de la partición estática de los datos se llega a un factor de 0.6, mientras que en el caso dinámico tal relación llega al 0.39. La mejora de tiempo del algoritmo dinámico con respecto al estático se explica por el mejor balance de carga que tiene tal algoritmo. En el peor de los casos, la diferencia entre el procesador más ocupado y el más ocioso varía del 65,19% del tiempo total (caso estático) al 10.9% en el caso dinámico.

Para solucionar el problema de la mala distribución de trabajo en el proceso estático se estudia actualmente una distribución mediante función aleatoria, de manera que se distribuyan las soluciones de dos unitarias de manera aleatoria entre los procesadores, como una forma de incrementar el porcentaje de uso de los procesadores más ociosos y al mismo tiempo liberar de algo de trabajo a los procesadores más ocupados.

El particionamiento de datos entre los procesadores de manera dinámica mejora considerablemente el desempeño del problema. Sin embargo, debe tenerse en cuenta que para desarrollar esta técnica debe contarse con alguna forma de manipular datos de gran tamaño, al mismo tiempo que se debe mejorar la forma de comunicar tales datos entre los procesadores. De esta manera se espera reducir el porcentaje de tiempo consumido en comunicaciones.

Otras alternativas de paralelización pueden ser aplicadas a este problema, de manera que se logre un compromiso óptimo entre el tiempo de cómputo (que no dejará de ser alto) y el número de comunicaciones necesarias, así como sobre el tamaño y características del conjunto de datos a ser transmitidos.

En esta investigación se ha utilizado una técnica de compresión de los datos que ayuda en el proceso de transmisión al hacer más pequeños los mensajes, pero incrementa el *overhead* necesario para la compresión y descompresión de los mismos. Se sigue trabajando en este sentido para lograr tiempos menores.

6.- Referencias bibliográficas.

[C&R97] Cardinale Y., Cabrera C., Cabrera A., Rivas R., "Paralelización por dominio del proceso de reconstrucción tridimensional del árbol coronario", Memorias del III Simposium Internacional de Computación, Ciudad de México, noviembre 1997.

[Chau92] Chauduri P., "Parallel algorithms design and analysis", Advances in Computer Science Series, Prentice Hall, Boston, 1992.

[Fost95] Foster I., "Designing and building parallel programs. Concepts and tools for Parallel Software Engineering", Addison Wesley International, Boston, 1995.

[Kum94] Kumar A., Grama A., Gupta A., Karypis G., "Introduction to parallel computing. Design and analysis of algorithms", The Benjamin/Cummings, Boston, 1994.

[LC&M95] La Cruz Alexandra y Morinelli G., "Herramienta para el análisis de Imágenes Angiográficas : ANIA", Informe final de cursos de cooperación, Universidad Simón Bolívar, Caracas, 1995.

[Lai84] Lai T., Sahni S., "Anomalies in parallel branch and bound algorithms", Communications of the ACM, pp. 594-602, febrero 1992.

[P&M95] Passariello, G. y Mora, F. "Imagenología Médica". Editorial Equinoccio, Universidad Simón Bolívar, Caracas, 1995.

[RC&I97] Rivas R., Cardinale Y., Ibáñez M.B., Windyga P., "Enfoques de paralelización de la reconstrucción 3D de imágenes del árbol coronario", Memorias de la XXIII Conferencia Latinoamericana de Informática, PANEL 97, Valparaíso, Chile, pp. 985-996, noviembre 1997.

[RI&C99] Rivas R, Ibáñez M.B., Cardinale Y y Windiga P, "A Parallel algorithm for 3D reconstruction of angiographic images", High-Performance Computing and Networking. 7th International Conference, HPCN Europe 1999. Sloot, P. and Bubak, M and Hoekstra, A. and Hertzberger, B. editores, Springer. pp. 168-177. 1999.

[Win94] Windiga P. "Evaluation et modelisation de connaissances pour la reconstruction tridimensionnelle du reseau vasculaire cardiaque en angiographie biplan", PhD Thesis, Université de Rennes I, marzo 1994.

[WLB&C95] Windyga P., López I., Bevilacqua G., Garreau M., Coatrieux J.L., "Utility of 2D properties in the reconstruction of coronary arteries from biplane angiographic images". XVIIth IEEE-EMBC conference, Montreal, Canadá, 1995.