Un Modelo Analítico para la Predicción del Rendimiento en Reconstrucción Tomográfica. ⋆

Paula Cecilia Fritzsche[1], José-Jesús Fernández[2], Ana Ripoll[1],
Inmaculada García[2], Emilio Luque[1]

[1] *Departamento de Arquitectura de Computadores y Sistemas Operativos.*
Universidad Autónoma de Barcelona. España.
[2] *Departamento de Arquitectura de Computadores y Electrónica.*
Universidad de Almería. España.

Resumen. Los estudios tridimensionales (3D) de especímenes biológicos a niveles subcelulares han sido posible gracias a la tomografía electrónica, al procesamiento de imágenes y a las técnicas de reconstrucción 3D. Para lograr la demanda de los requerimientos computacionales de grandes volúmenes, se han aplicado estrategias de paralelización con descomposición de dominio. Aunque esta combinación ya ha probado ser útil para tomografía electrónica de especímenes biológicos, un modelo de predicción de rendimiento aún no ha sido descrito. Tal modelo debería permitir conocer la aplicación paralela y predecir su funcionamiento bajo diferentes parámetros o plataformas hardware. Este artículo describe un modelo analítico de predicción de rendimiento para BPTomo, una aplicación paralela para reconstrucción tomográfica. El funcionamiento de la aplicación es analizado paso a paso para crear una formulación analítica del problema. El modelo es validado comparando los tiempos estimados para conjuntos de datos representativos con tiempos medidos en un cluster tipo beowulf.

Palabras claves

Tomografía electrónica, Computación paralela, Predicción de rendimiento, Modelos analíticos

1 Introducción

Los modelos analíticos de predicción de rendimiento permiten tener un conocimiento profundo de las aplicaciones paralelas, predecir su funcionamiento bajo diferentes parámetros o plataformas de hardware y mejorar la explotación de la potencia computacional de los sistemas paralelos. Estos modelos pueden proporcionar respuestas a preguntas tales como "qué pasa con el rendimiento si el tamaño de los datos y el número de máquina son modificados?", "cuántas máquinas son necesarias para finalizar el trabajo en un tiempo dado?", "qué mejoras se pueden obtener cambiando los parámetros?". La metodología basada en modelos de predicción posibilita la evaluación del rendimiento de la aplicación sin la necesidad de su implementación.

La microscopía electrónica, el procesamiento de imágenes y las técnicas de reconstrucción tridimensional (3D) permiten el análisis estructural de especímenes biológicos complejos a niveles subcelulares [1, 2, 3]. Esta información estructural es crítica para entender la función biológica de los especímenes [7]. La reconstrucción de imágenes determina estructuras 3D a partir de una serie de imágenes bidimensionales (2D). Estas imágenes 2D son adquiridas rotando el espécimen sobre uno o más ejes, mediante tomografía de transmisión (electrónica o de rayos X). Es importante remarcar que la metodología de análisis tomográfico puede ser usada en diversas disciplinas que van desde ingenierías hasta ciencias del sistema de la tierra [8].

Las técnicas de reconstrucción algebraicas (Algebraic Reconstruction Techniques - ART) introducidas en los años 70, se descartaron en aplicaciones prácticas en un primer momento debido al alto requerimiento computacional. Los ficheros generados por la tomografía de transmisión generalmente son grandes y, en consecuencia, el tiempo de procesamiento que se necesita es considerable. En la actualidad, las estrategias de paralelización conjuntamente con la distribución de los datos, prestan diversas soluciones a los problemas

⋆ Este trabajo fue financiado por el MCyT bajo contratos 2001-2592 y 2002-00228 y patrocinado por la Generalitat de Catalunya (Grup de Recerca Consolidat 2001SGR-00218).

anteriormente planteados. Aunque existe un método estándar de reconstrucción en tomografía electrónica llamado retro proyección ponderada (Weighted Back Projection - WBP), hoy en día los investigadores le brindan mayor atención a los métodos de expansión en serie (también llamados técnicas de reconstrucción iterativas). Esto se debe a que éstos últimos métodos mejoran la reconstrucción por medio del uso de otra función base. Además, no presentan problemas relacionados con la cantidad de ángulos de entrada o el tratamiento de las condiciones de ruido.

Las cuatro imágenes de la Fig. 1 ilustran las distintas calidades de una reconstrucción según la elección de los parámetros.

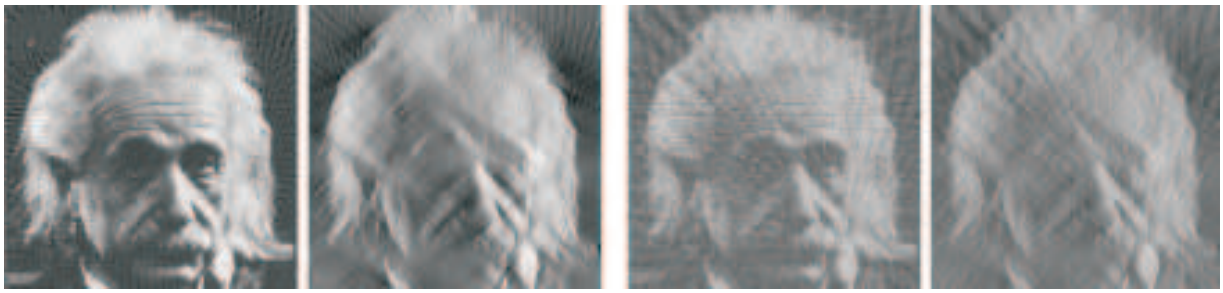


Fig. 1. La primera imagen muestra una reconstrucción de una imagen 2D casi perfecta. El rango de inclinación es +/- 90 con un incremento de 2. En la segunda imagen el rango de inclinación es +/- 60. En la tercera y cuarta, el rango de inclinación es +/- 90 y +/- 60 respectivamente con un incremento de 5.

BPTomo es una aplicación paralela para reconstrucción tomográfica en biología estructural que sigue el paradigma SPMD (Single Program Multiple Data). Trabaja con los métodos de expansión en serie usando blobs como función base. Existen trabajos previos sobre paralelización en reconstrucción de imágenes, pero muchos de éstos trabajan con imágenes de reconstrucción 2D [4] o tratan con reconstrucciones 3D usando voxels como función base [8].

Este trabajo presenta un modelo analítico de rendimiento para BPTomo. Debido a su naturaleza y regularidad, una predicción analítica es más apropiada que la utilización de técnicas de simulación. Así, de esta manera el tiempo necesitado para evaluar el rendimiento es reducido. El funcionamiento de la aplicación es analizado paso por paso para crear una formulación analítica del problema. El modelo es validado comparando los tiempos estimados con los tiempos medidos en un cluster de PCs.

El resto de este artículo está organizado como sigue. La sección 2 analiza la aplicación BPTomo. La sección 3 describe el modelo analítico para la aplicación. La sección 4 presenta la evaluación del modelo y la validación experimental. La sección 5 resume las conclusiones del trabajo.

2 Aplicación BPTomo

BPtomo es una aplicación paralela de reconstrucción tomográfica que trabaja con elementos de volumen simétrico esférico (blobs) [6]. Los voxels y los blobs son funciones bases usadas en el campo de la reconstrucción de imágenes. Los blobs tienen una forma esférica suave mientras que los voxels tienen una forma cúbica.

2.1 Funcionamiento

La Fig. 2 muestra el proceso completo de la tomografía electrónica para la reconstrucción de un objeto 3D utilizando cómputo paralelo. Para la adquisición de datos se requiere el ingreso del ángulo de giro y de la cantidad de tomas a realizar en el ordenador conectado al HVEM y, luego, la sintonización de otros parámetros, ver Fig. 2(b). El principal objetivo de los HVEMs es registrar en forma automática las imágenes 2D teniendo en cuenta las especificaciones realizadas por el usuario, ver Fig. 2(a). Una vez registradas estas imágenes se las almacena, en forma transitoria, en el ordenador conectado al HVEM. Las etapas mencionadas pueden repetirse hasta que las imágenes tengan la calidad deseada. Una vez que

las imágenes se consideran idóneas, se retransmiten a un cluster de PCs. Seguidamente, en el cluster se procesan las imágenes usando alguna técnica de reconstrucción iterativa con el fin de obtener el objeto 3D, Fig. 2(c).

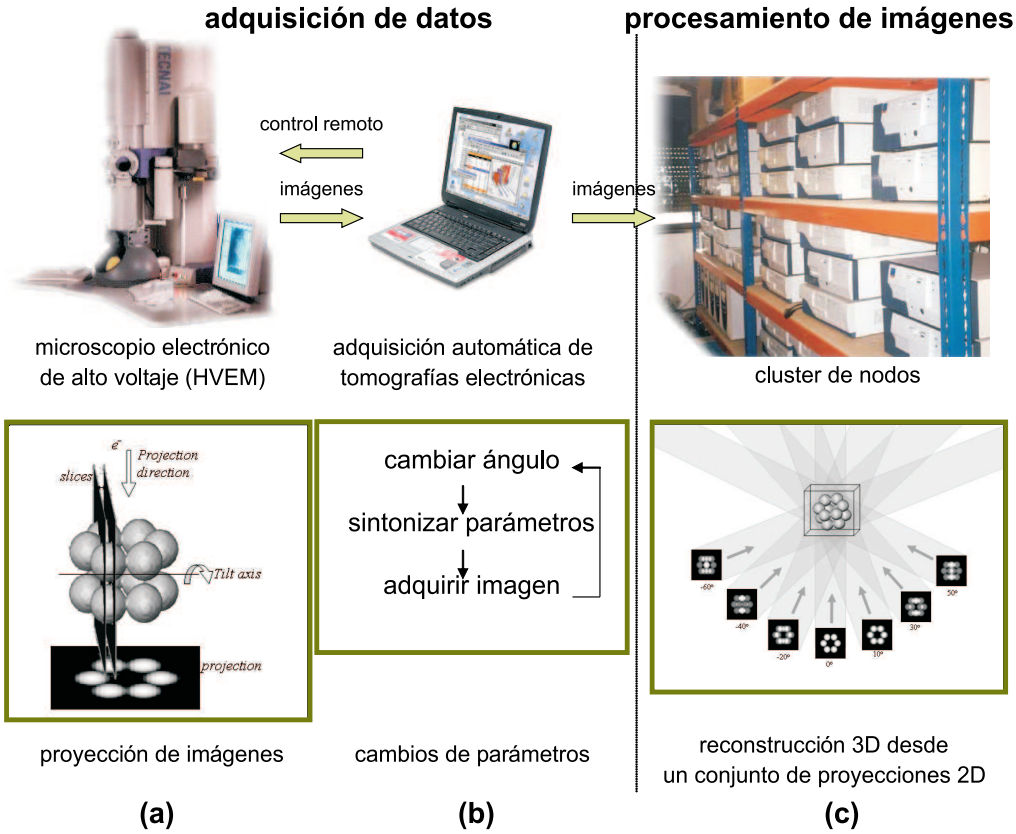


Fig. 2. Proceso BPTomo.

BPTomo implementa una Técnica de Reconstrucción Iterativa (IRT) que considera que el objeto 3D o función f que debe ser reconstruido puede aproximarse por una combinación lineal de un conjunto finito J de funciones bases conocidas y fijas b_j del siguiente modo:

$$f(r, \phi_1, \phi_2) \approx \sum_{j=1}^{J} x_j b_j(r, \phi_1, \phi_2) \tag{1}$$

donde (r, ϕ_1, ϕ_2) son coordenadas esféricas y el principal objetivo es estimar el desconocido vector x_j. IRT está basado en un modelo de formación de imagen donde las medidas dependen linealmente del objeto a reconstruir. Entonces la i-ésima medida de f (denotada y_i) es aproximadamente

$$y_i \approx \sum_{j=1}^{J} l_{i,j} x_j \tag{2}$$

donde $l_{i,j}$ define el valor de la i-ésima medida de la j-ésima función base. Bajo estos supuestos, el problema de reconstrucción de imágenes puede modelarse como el problema inverso de estimar los x_j's desde los y_i's resolviendo el sistema de ecuaciones lineales dado por la Eq. (2). Este puede dividirse en B bloques cada uno de tamaño S y, en consecuencia, el proceso iterativo puede describirse por sus propios pasos iterativos desde el k estimado al $k+1$ estimado utilizando la siguiente expresión:

$$x^0 \quad \textit{vector } J \textit{ dimensional de valores constantes}$$

$$x_j^{k+1} = x_j^k + \lambda_k \sum_{s=1}^{S} \underbrace{\frac{y_i - \sum_{v=1}^{J} l_{i,v} x_v^k}{\sum_{v=1}^{J} s_v^b (l_{i,v})^2} l_{i,j}}_{Término(1)} \tag{3}$$

donde λ_k es el parámetro de relajación, $b = (k \bmod B)$ es el índice del bloque, $i = bS + s$ es la ecuación i de todo el sistema y s_v^b es el factor de peso.

El *Término (1)* puede descomponerse en tres etapas: el cálculo de la proyección del objeto ($FProj$), el cálculo del error entre las proyecciones experimentales y las proyecciones calculadas ($EComp$) y el refinamiento del objeto por medio de la retro proyección del error ($BProj$).

$$FProj = \sum_{v=1}^{J} l_{i,v} x_v^k \tag{4}$$

$$EComp = y_i - FProj \tag{5}$$

$$BProj = \frac{EComp}{\sum_{v=1}^{J} s_v^b (l_{i,v})^2} l_{i,j} \tag{6}$$

Luego la Eq. (3) puede reescribirse como la Eq. (7). El proceso pasa a través de estas etapas para cada bloque de ecuaciones y para cada iteración.

$$x_j^{k+1} = x_j^k + \lambda_k \sum_{s=1}^{S} BProj \tag{7}$$

2.2 Modelo computacional SPMD

La aplicación ha sido diseñada siguiendo el paradigma SPMD. Debido a que los blobs se expanden más allá de un único slice, los slices adyacentes son interdependientes. Al distribuir los datos, los slices se reparten entre tantos slabs como el número de nodos, según se exhibe en la Fig. 3(a). Cada slab se compone de su subdominio de slices que incluyen además slices redundantes de los nodos vecinos. De este modo, cada nodo ejecuta el mismo programa sobre su propio slab.

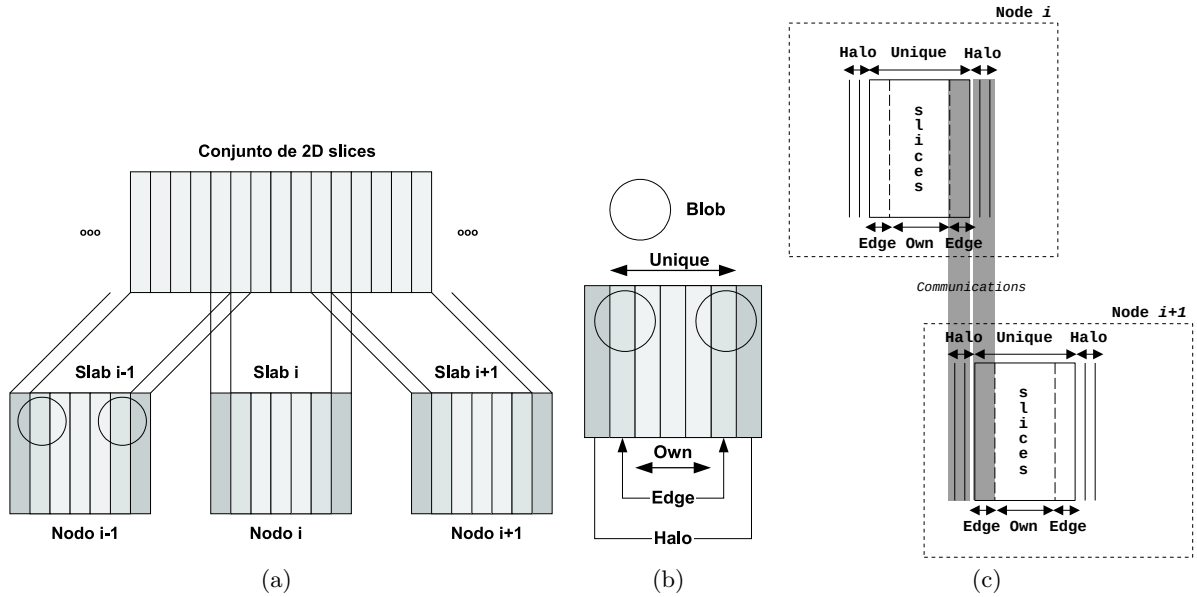
Los slices contenidos en cada slab se dividen en las siguientes categorías, ver Fig. 3(b):

Halo Estos slices son necesarios sólo para que el nodo pueda reconstruir algunos de sus propios slices. Son slices redundantes y vienen de los nodos vecinos donde se reconstruyen.

Unique Estos slices se reconstruyen en el nodo. Se subdividen en:

- **Edge** Slices que requieren información de los slices halo provenientes de los nodos vecinos.
- **Own** Slices que no requieren información de los slices halo provenientes de los nodos vecinos. Estos slices son independientes.

La interdependencia entre slices vecinos implica que, para calcular la proyección Eq. (4) o la retro proyección del error Eq. (6) para un slab dado, debe haber un intercambio adecuado de información entre nodos vecinos, Fig. 3(c).



(a) (b) (c)

Fig. 3. (a) Distribución de los slices en slabs. (b) Clasificación de los slices en un slab. (c) Modificación de los halo slices con edge slices del vecino.

En consecuencia, dos nuevas etapas aparecen en el proceso IRT como se muestra en la Fig. 4. La primera corresponde a la comunicación de diferencias entre las proyecciones experimentales y las proyecciones calculadas y la segunda a la comunicación de slices reconstruidos que vienen de la etapa de retro proyección del error. Estas nuevas etapas implican sincronización donde los nodos esperan por sus vecinos.

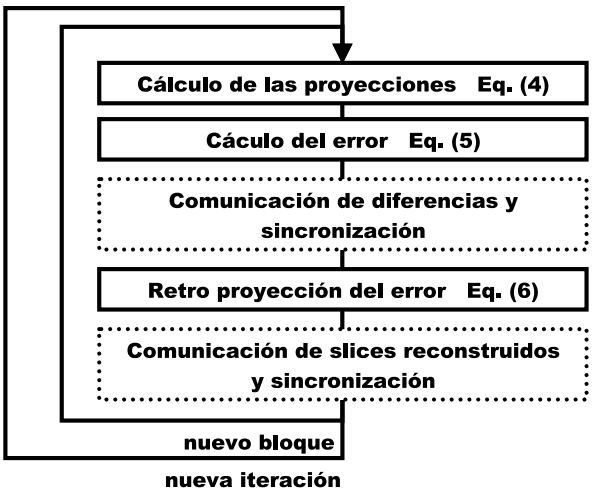


Fig. 4. Etapas de las técnicas de reconstrucción iterativas en la versión paralela siguiendo el paradigma SPMD.

2.3 Implementación BPTomo

En BPTomo, se conjugan dos entornos dominantes de hoy en día, C y MPI. C es un lenguaje de programación de propósito general y si bien se lo asocia al sistema UNIX, no está ligado a ningún sistema operativo ni a ninguna máquina. C proporciona las instrucciones de control de flujo lineal que se necesitan en los programas estructurados: condiciones, ciclos, agrupamientos y subprogramas, pero no multiprogramación,

operaciones paralelas, ni sincronización. Todo esto último es solucionado con la interfaz MPI, la cual permite comunicaciones punto a punto, comunicaciones colectivas, sincronización, creación de procesos, entre otras cosas [9].

Al igual que cualquier otra aplicación que trabaja con MPI, BPTomo posibilita las llamadas a funciones MPI realizando una inicialización del entorno mediante la primitiva $MPI\_Init()$. De modo similar, después de todas las llamadas a funciones MPI realiza una terminación mediante la primitiva $MPI\_Finalize()$.

A continuación, BPTomo define un comunicador por defecto llamado MPI_COMM_WORLD que incluye todos los procesos envueltos en la ejecución paralela. Los comunicadores de MPI se usan para las comunicaciones de paso de mensajes punto a punto o colectivas. Un comunicador es un dominio de comunicación que define un conjunto de procesos que se pueden comunicar entre sí. Un proceso tiene un único identificador *id* dentro de un comunicador (0 a $n-1$), donde n es el número de procesos). En BPTomo, los procesos cuyos identificadores van de 1 a $n-2$ envían y reciben datos de los procesos $id-1$ e $id+1$. El proceso 0 y el $n-1$ sólo envían y reciben datos del proceso 1 y del proceso $n-2$ respectivamente.

Cada declaración de variable global se duplica en cada proceso debido a que la aplicación sigue el paradigma SPMD. Las variables que no necesitan duplicarse se declaran localmente dentro del código del proceso. La distribución de parámetros y datos se hace mediante la primitiva $MPI\_Bcast()$.

Para el paso de mensajes, BPTomo usa las primitivas de envío y de recepción no bloqueantes con el fin de solapar el cómputo y la comunicación y así lograr una mayor eficiencia. Las rutinas no bloqueantes continúan inmediatamente con la próxima instrucción a ejecutar. Es decir, la instrucción de envío no bloqueante, $MPI\_Isend()$ (I se refiere a inmediato), retorna aún antes que la dirección del buffer donde se encuentran los datos a enviar esté segura de modificación. En cuanto a la instrucción de recepción no bloqueante, $MPI\_Irecv()$, retorna aún si no hay mensaje de aceptación [10].

El formato de la primitiva de envío no bloqueante es:

$MPI\_Isend($ *void* $*buf$, *int count*, $MPI\_Datatype$ *datatype*, *int dest*, *int tag*,
MPI_Comm *comm*, $MPI\_Request$ $*request$)

donde buf es la dirección del buffer donde se encuentran los datos a enviar, *count* es el tamaño de los elementos a enviar, *datatype* es el tipo de dato que se envía, *dest* es el nodo destino, *tag* es la marca del mensaje, *comm* es el comunicador y *request* es el que pide la comunicación.

El formato de la primitiva de recepción no bloqueante es:

$MPI\_Irecv($ *void* $*buf$, *int count*, $MPI\_Datatype$ *datatype*, *int source*, *int tag*,
MPI_Comm *comm*, $MPI\_Request$ $*request$)

donde buf es la dirección del buffer donde se reciben los datos enviados, *count* es el tamaño de los elementos a recibir, *datatype* es el tipo de dato que se recibe, *source* es el nodo fuente, *tag* es la marca del mensaje, *comm* es el comunicador y *request* es el que pide la comunicación.

La completitud de las instrucciones de envío-recepción mencionadas anteriormente, es detectada por medio de la primitiva $MPI\_Wait()$ en BPTomo. Esta instrucción espera hasta que se termine una determinada operación. El formato de ésta primitiva es:

$MPI\_Wait($ $MPI\_Request$ $*request$, MPI_Status $*status$)

donde *request* es el que ha pedido una comunicación y espera por su completitud y *status* es el estado del objeto.

La Fig. 5 exhibe un seudocódigo de la aplicación BPTomo conteniendo las primitivas mencionadas a lo largo de esta sección.

```
#include "mpi.h"
#include "mpicomm.h"
extern MPI_Comm MPI_COMM_BPTOMO       // Comunicador global para BPTOMO

main(int argc, char *argv[])          // Programa principal
{
  Inicializar_tiempos
  MPI_Init(&argc, &argv)              // Inicializar el ambiente MPI
  MPI_Comm_size(MPI_COMM_WORLD,&nodes)
  MPI_Comm_rank(MPI_COMM_WORLD,&myid) // Encontrar id del proceso
  .
  if (&myid == MASTER) {
    Analizar_parametros_ingresados    // Inic. realizadas por el proceso master
  }
  .
  MPI_Bcast(...)                      // Distribuir de datos y parametros
  .
  Leer_datos                          // Leer sinogramas que corresponden slab
  Proceso_IRT                         // Proceso principal
  .
  Escribir_volumen                    // Escribir volumen reconstruido
  Mostrar_estadisticas_tiempos
  MPI_Barrier (MPI_COMM_WORLD)
  MPI_Finalize ()                     // Finalizar el ambiente MPI
}

Proceso_IRT(...)
{
  while(iter <= Niter) {
    for(k=0; k<NBlocks; k++) {

      Forward_Projection_left_edge    // Calculo proyecciones left-edge slices
      Comp_difference_left_edge       // Calculo error left-edge slices
      Init_Comm                       // Comunicar diferencias

      Forward_Projection_right_edge   // Calculo proyecciones right-edge slices
      Comp_difference_right_edge      // Calculo error right-edge slices
      Init_Comm                       // Comunicar diferencias

      Forward_Projection_own          // Calculo proyecciones own slices
      Comp_difference_own             // Calculo error own slices

      MPI_Wait(&LeftSend_hndl, ...)   // Esperar finalice comunicacion
      MPI_Wait(&LeftRecv_hndl, ...)
      MPI_Wait(&RightSend_hndl, ...)
      MPI_Wait(&RightRecv_hndl, ...)

      Acumular_tiempos

      Diff_BackProjection_left_edge   // Calc. retroproyeccion left-edge slices
      Init_Comm                       // Comunicar slices reconstruidos

      Diff_BackProjection_right_edge  // Calc retroproyeccion right-edge slices
      Init_Comm                       // Comunicar slices reconstruidos

      Diff_BackProjection_own         // Calc retroproyeccion own slices

      MPI_Wait(&LeftSend_hndl, ...)   // Esperar finalice comunicacion
      MPI_Wait(&LeftRecv_hndl, ...)
      MPI_Wait(&RightSend_hndl, ...)
      MPI_Wait(&RightRecv_hndl, ...)

      Acumular_tiempos
    }
    Mostrar_tiempos
  }
  iter++
}

Init_Comm(tosend, torecv, ...)
{
  if(tag == SEND_TO_LEFT) {
     To = myleft; From = myright;
  }
  else
    if(tag == SEND_TO_RIGHT) {
      To = myright; From = myleft;
    }

  MPI_Isend(tosend, ...,To, tag, MPI_COMM_BPTOMO, ...);   // Envio de datos
  MPI_Irecv(torecv, ...,From, tag, MPI_COMM_BPTOMO, ...); // Recepcion de datos
}
```

Fig. 5. Seudocódigo de la aplicación BPTomo.

3 Modelo analítico para BPTomo

El objetivo del modelo analítico es proporcionar una estimación de la cantidad de tiempo de cómputo y de comunicación por nodo, requerido para alcanzar la solución de una aplicación que es ejecutada en un sistema paralelo. Este asume que el sistema paralelo es homogéneo para los elementos hardware, la red y el procesamiento.

El tiempo de ejecución total estimado (T_{Est}) puede aproximarse por el tiempo IRT para un nodo (T_{IRT}) despreciándose los tiempos de inicialización, distribución de datos, lectura y escritura por ser poco significativos ($T_{Est} \approx T_{IRT}$).

Luego, de acuerdo con la Fig. 4, el T_{IRT} para un nodo puede estimarse por:

$$T_{IRT} = T_{FProj} + T_{EComp} + T_{CommDiff} + T_{BProj} + T_{CommRSlices} \tag{8}$$

donde T_{FProj} es el tiempo para calcular la proyección del objeto, T_{EComp} es el tiempo para calcular el error entre las proyecciones experimentales y las proyecciones calculadas, $T_{CommDiff}$ es el tiempo de comunicar las diferencias, T_{BProj} es el tiempo de refinar el objeto por medio de la retro proyección del error y $T_{CommRSlices}$ es el tiempo de comunicar los slices reconstruidos. T_{FProj}, T_{EComp} y T_{BProj} son valores estimados de tiempos de cómputo, mientras $T_{CommDiff}$ y $T_{CommRSlices}$ son valores estimados de tiempos de comunicación.

En la Tabla 1 se da una descripción de las variables usadas tanto para estimar los tiempos de cómputo como de comunicación. Algunas de estas variables se delinean en la Fig. 6.

Tabla 1. Notación usada en el modelo analítico y definiciones.

Var	Significado
$AbyB$	Número de imágenes de proyección por bloque. En Eq. (3) denotada por S.
$Blocks$	Número de bloques en que las proyecciones se subdivide.
$Height$	Altura de un slice.
$Iter$	Número de iteraciones. Relacionado con la calidad de la reconstrucción.
$Nodes$	Número de nodos o procesadores.
N_{Slices}	Número total de slices a ser procesados.
S_{Edge}	Número de slices edge a ser comunicados.
S_{Height}	Altura de una imagen de proyección.
S_{Width}	Ancho de una imagen de proyección.
T_{Comm}	Tiempo requerido para transmitir un pixel.
$Width$	Ancho de un slice.

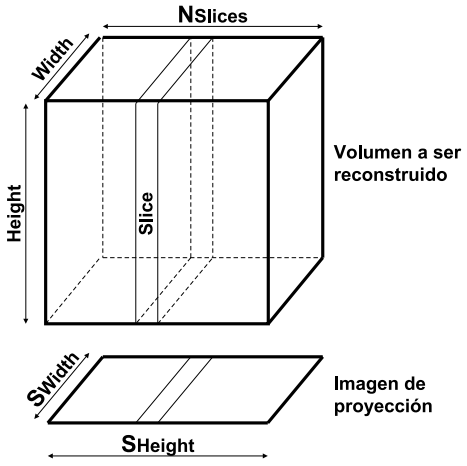


Fig. 6. Dimensiones del volumen a reconstruir por BPTomo y de la imagen de proyección.

3.1 Tiempo de cómputo estimado

Con la notación de la Tabla 1 y teniendo en cuenta las Eq. (4)-Eq. (7), en esta sección se analiza la estimación del tiempo de cálculo requerido por nodo para evaluar la proyección (T_{FProj}), el cálculo del error (T_{EComp}) y la retro proyección del error (T_{BProj}).

El tamaño de cada slice es $Height \cdot Width$ y cada slice se proyecta $AbyB$ veces por bloque. El tamaño de la imagen de proyección es $S_{Height} \cdot S_{Width}$. En la práctica, $Height = Width = N_{Slices} = S_{Height} = S_{Width}$. Hay tantos $Nodos$ como slabs y $N_{Slices}/Nodes$ slices para cada slab. De acuerdo con la Eq. (3) las expresiones son procesadas para cada bloque de ecuaciones ($Blocks$) y para cada paso iterativo ($Iter$). En consecuencia, el tiempo para calcular la proyección del objeto para un nodo (T_{FProj}), dado por Eq. (4), puede estimarse como:

$$T_{FProj} = Height \cdot Width \cdot AbyB \cdot (N_{Slices}/Nodes) \cdot Blocks \cdot Iter \cdot T_{PFProj} \tag{9}$$

donde T_{PFProj} es el tiempo requerido para proyectar un pixel de un slice en la dirección de un ángulo.

El tiempo necesario de un nodo para calcular el error (T_{EComp}), Eq. (5), debe tener en cuenta que el número de pixels de una proyección (un ángulo) por nodo es $S_{Width} \cdot N_{Slices}/Nodes$ y que hay $AbyB$ proyecciones por bloque. Nuevamente, esto se realiza para cada bloque de ecuaciones y para cada paso iterativo. Entonces, el tiempo para calcular el error para un nodo, puede ser estimado por:

$$T_{EComp} = S_{Width} \cdot AbyB \cdot (N_{Slices}/Nodes) \cdot Blocks \cdot Iter \cdot T_{PEComp} \tag{10}$$

donde T_{PEComp} es el tiempo requerido para calcular el error de una proyección de un pixel de un slice.

El error de retro proyección Eq. (6) exhibe una complejidad computacional igual a la proyección; en consecuencia, una estimación del tiempo para evaluar el refinamiento del objeto por medio de la retro proyección del error puede expresarse por medio de la siguiente ecuación:

$$T_{BProj} = Height \cdot Width \cdot AbyB \cdot (N_{Slices}/Nodes) \cdot Blocks \cdot Iter \cdot T_{PBProj} \tag{11}$$

donde T_{PBProj} es el tiempo requerido para retro proyectar el error en una proyección de un pixel de un slice.

Los tiempos T_{PFProj}, T_{PEComp}, y T_{PBProj} tienen que evaluarse exactamente. Estos pueden ser medidos ejecutando la aplicación una vez o pueden ser ponderados de acuerdo con estadísticas previas obtenidas en otros artículos [6]. Los mismos son dependientes de la plataforma, de modo que para otro ambiente con diferentes máquinas y red, estos valores deben ser ajustados.

3.2 Tiempo de comunicación estimado

Antes de proporcionar las expresiones para estimar los tiempos de comunicación, es esencial investigar cómo trabaja IRT cuando comunica. Para mejorar el rendimiento, las etapas de comunicación son solapadas con las etapas de cómputo. En consecuencia, las comunicaciones son ocultadas de una forma total o parcial debido al orden en que los slices son analizados. Como muestra la Fig. 7, en una primera etapa se procesan los left-edge slices y se envían inmediatamente al nodo vecino. La comunicación de los left-edge slices se solapa con el procesamiento de los right-edge slices. De manera similar, la comunicación de los right-edge slices se solapa con el procesamiento de los own slices. Esta estrategia es aplicada en ambos puntos de comunicación; antes y después de la etapa de retro proyección del error [6].

La transmisión de las diferencias entre la proyección experimental y la proyección calculada de los left-edge slices, para un nodo, implica la siguiente cantidad de datos:

$$Data_{DiffLeft} = S_{Width} \cdot AbyB \cdot S_{Edge} \cdot Blocks \cdot Iter \tag{12}$$

De igual forma, la transmisión de las diferencias de los right-edge slices ($Data_{DiffRight}$) para un nodo, implica la misma cantidad de datos de la Eq. (12).

Mientras se realiza la comunicación de las diferencias de los left-edge slices, el procesamiento de las etapas de proyección y cálculo de error de los right-edge y own slices tiene lugar. De manera similar, la comunicación de las diferencias de los right-edge slices se realiza mientras tiene lugar el procesamiento de

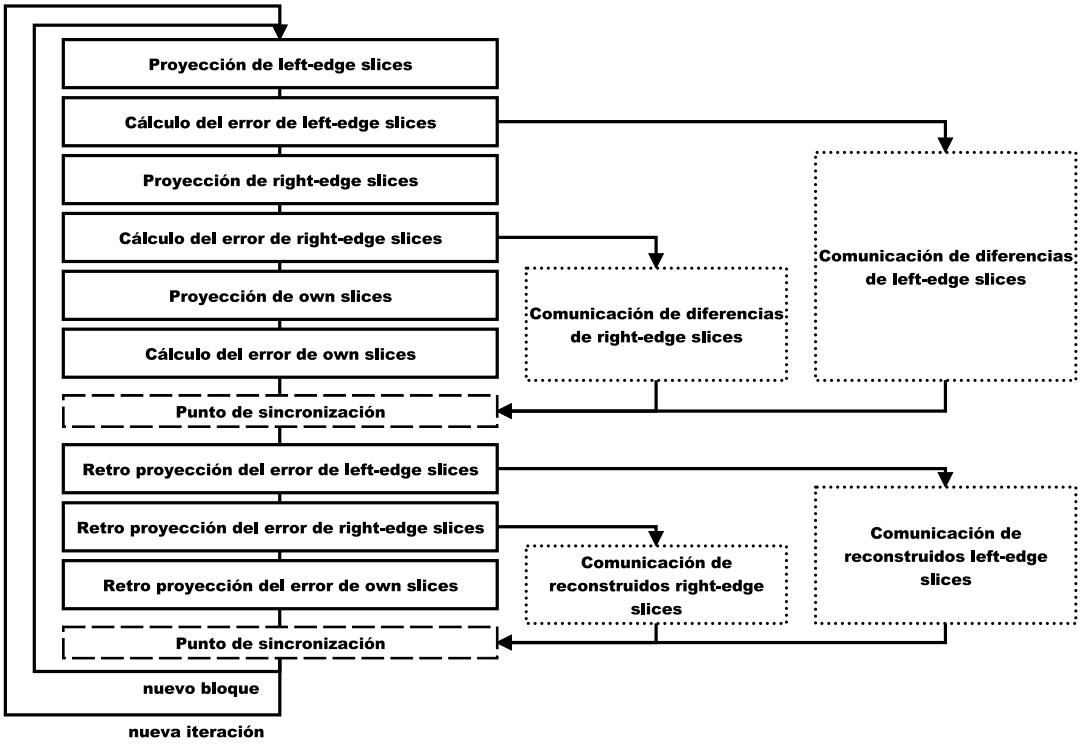


Fig. 7. Solapamiento entre comunicación y cómputo. Las cajas continuas representan etapas del algoritmo iterativo, las cajas punteadas denotan la transmisión de los datos ya procesados y las cajas con líneas cortadas son las barreras de sincronización.

las etapas de proyección y cálculo del error de los own slices. En consecuencia, el tiempo estimado es el que sigue:

$$T_{CommDiff} = \begin{cases} T_{ExpCommDiff} & \text{si } T_{ExpCommDiff} \geq 0 \\ 0 & \text{si } T_{ExpCommDiff} < 0 \end{cases}$$

donde

$$T_{ExpCommDiff} = (Data_{DiffLeft} + Data_{DiffRight}) \cdot T_{Comm} - \\ -Height \cdot Width \cdot AbyB \cdot (N_{Slices}/Nodes - S_{Edge}) \cdot Blocks \cdot Iter \cdot T_{PFProj} \\ -S_{Width} \cdot AbyB \cdot (N_{Slices}/Nodes - S_{Edge}) \cdot Blocks \cdot Iter \cdot T_{PEComp}$$

y T_{Comm} es el tiempo requerido para comunicar un valor a través de la red.

La transmisión de los slices reconstruidos de los left-edge slices, para un nodo, implica la siguiente cantidad de datos:

$$Data_{RSlicesLeft} = Height \cdot Width \cdot S_{Edge} \cdot Blocks \cdot Iter \tag{13}$$

De igual manera, la transmisión de los slices reconstruidos de los right-edge slices implica la misma cantidad de datos de la Eq. (13), ($Data_{RSlicesLeft} = Data_{RSlicesRight}$).

Mientras la comunicación de los slices reconstruidos de los left-edge slices es realizada, el procesamiento de la retro proyección del error de los right-edge y own slices tiene lugar. De modo similar, la comunicación de los slices reconstruidos de los right-edge slices es realizada mientras el procesamiento de la retro proyección del error de los own slices tiene lugar. En consecuencia, la expresión del tiempo estimado es el que sigue:

$$T_{CommRSlices} = \begin{cases} T_{ExpCommRSlices} & \text{si } T_{ExpCommRSlices} \geq 0 \\ 0 & \text{si } T_{ExpCommRSlices} < 0 \end{cases}$$

donde

$$T_{ExpCommRSlices} = (Data_{RSlicesLeft} + Data_{RSlicesRight}) \cdot T_{Comm} - \\ -Height \cdot AbyB \cdot Width \cdot (N_{Slices}/Nodes - S_{Edge}) \cdot Blocks \cdot Iter \cdot T_{PBProj}$$

4 Evaluación del modelo

Para validar el modelo analítico obtenido en la sección previa, BPTomo ha sido ejecutado para dos conjuntos de datos con diferentes tamaños. Los tamaños son representativos de los estudios biológicos estructurales actuales para tomografía de electrones. Posteriormente, los tiempos de ejecución obtenidos han sido comparados con aquellos tiempos estimados por el modelo. El porcentaje de desviación ha sido usado para mostrar la precisión. Las tablas y los gráficos ayudan a concluir.

4.1 Conjunto de datos 1

Para validar el modelo analítico se ha utilizado, un primer conjunto de datos que consiste de 70 imágenes de proyección de 64×64 pixels tomadas del objeto girado en el rango [-70º, +68º] a intervalos de 2º. Con el objeto de obtener los tiempos, es esencial dar valores a ciertos parámetros importantes. Se eligió trabajar con 10 iteraciones, 1 bloque con 70 ángulos por bloque y 1 o 2 slices edge para ser comunicados entre nodos vecinos (denotado por $S_{Edge}1$ o $S_{Edge}2$ respectivamente) para IRT.

Para obtener los tiempos T_{PFProj}, T_{PEComp}, T_{PBProj} y T_{Comm} se ha ejecutado la aplicación en un nodo y se han medido aquellos tiempos, obteniéndose 8.39E-07 sec., 3.95E-08 sec., 8.48E-07 sec. y 3.05E-05 sec., respectivamente.

La Tabla 2 muestra la escalabilidad hardware de la aplicación usando diferentes números de nodos (primera columna): 1, 5, 10, 15, 20 y 25. La segunda, tercera y quinta columnas representan los tiempos de cálculo estimados (T_{FProj}, T_{EComp} y T_{BProj}) mientras que la cuarta y sexta columnas representan los tiempos de comunicación estimados ($T_{CommDiff}$ y $T_{CommRSlices}$). Para los tiempos de comunicación estimados, hay dos subcolumnas que corresponden al número de edge slices (S_{Edge}) transmitidos entre nodos adyacentes. La última columna en la tabla, es el tiempo de ejecución total estimado (T_{IRT}); suma de los valores de las columnas que van de la segunda a la sexta (T_{FProj}+T_{EComp}+$T_{CommDiff}$+T_{BProj}+$T_{CommRSlices}$). Nuevamente, se tienen dos subcolumnas que se refieren al número de edge slices transmitidos entre nodos adyacentes.

La principal observación de la Tabla 2 es que los tiempos de cálculo estimados disminuyen a medida que el número de nodos crece. Cuando el número de slices edge es 1, los tiempos de comunicación estimados son 0, significando que la comunicación se solapa (oculta) completamente con el cálculo. En cambio, si el número de slices edge es 2, en algunos casos los tiempos de comunicación estimados son mayores a 0. En consecuencia, la comunicación no se oculta totalmente por el cómputo. Como el tiempo de ejecución total estimado es la suma de los tiempos de cómputo y comunicación, cuando el número de slices edge es 1, los valores de T_{IRT} decrecen. Con slices edge igual a 2, el T_{IRT} deja de decrecer en un punto. Esto es alrededor del nodo 15, donde la comunicación juega un papel importante. En consecuencia, el rendimiento deja de mejorar.

Tabla 2. Tiempos de ejecución estimados para el conjunto de datos 1 (en seg.).

$Nodes$	T_{FProj}	T_{EComp}	$T_{CommDiff}$		T_{BProj}	$T_{CommRSlices}$		T_{IRT}	
			$S_{Edge}1$	$S_{Edge}2$		$S_{Edge}1$	$S_{Edge}2$	$S_{Edge}1$	$S_{Edge}2$
1	154.00	0.11	0.00	0.00	155.51	0.00	0.00	309.62	309.62
5	30.80	0.02	0.00	0.00	31.10	0.00	0.00	61.92	61.92
10	15.40	0.01	0.00	0.00	15.55	0.00	0.00	30.96	30.96
15	10.27	0.01	0.00	0.10	10.37	0.00	0.00	20.64	20.66
20	7.70	0.01	0.00	2.58	7.78	0.00	2.08	15.48	20.15
25	6.16	0.00	0.00	4.12	6.22	0.00	3.64	12.38	20.15

4.2 Conjunto de datos 2

Para continuar la validación del modelo analítico, se ha utilizado un segundo conjunto que consiste de 70 imágenes de proyección de 128×128 pixels tomadas del objeto girado en el rango [-70º, +68º] a intervalos de 2º. Los parámetros y tiempos para T_{PFProj}, T_{PEComp}, T_{PBProj} y T_{Comm} son los mismos que para el conjunto de datos 1, con excepción del número de iteraciones usado en las distintas comprobaciones.

Tabla 3. Tiempos de ejecución estimados para el conjunto de datos 2 (en seg.).

$Nodes$	T_{FProj}	T_{EComp}	$T_{CommDiff}$	T_{BProj}	$T_{CommRSlices}$	$T_{IRT}10$	$T_{IRT}20$	$T_{IRT}30$
1	1231.95	0.45	0.00	1244.14	0.00	2476.54	4953.08	7429.62
5	246.39	0.09	0.00	248.83	0.00	495.31	990.62	1485.93
10	123.20	0.05	0.00	124.41	0.00	247.65	495.30	742.95
15	82.13	0.03	0.00	82.94	0.00	165.10	330.20	495.30
20	61.60	0.02	0.00	62.21	0.00	123.83	247.66	371.49
25	49.28	0.02	0.00	49.77	0.00	99.06	198.12	297.18

El objetivo de usar este nuevo conjunto de datos es cuantificar la influencia del tamaño de las imágenes de entrada y la influencia del número de iteraciones. En este conjunto de datos, los tiempos de comunicación estimados son 0 tanto eligiendo 1 o 2 como cantidad de slices edge a comunicar. La comunicación se oculta totalmente. Esto se debe a que el tamaño de entrada del conjunto de datos 2 es mayor que el conjunto de datos 1 y por ende se necesita más tiempo de cálculo, siendo este tiempo suficiente para ocultar completamente las comunicaciones (Tabla 3).

4.3 Validación experimental

El modelo analítico ha sido validado con un cluster tipo beowulf de 25 nodos homogéneos (Pentium IV 1.8GHz., 512Mb DDR266, fast Ethernet). El cluster dispone de una red de interconexión segmentada que permite concurrencia de comunicaciones entre nodos diferentes.

La Fig. 8 muestra los tiempos estimados de la Tabla 2 y los tiempos de ejecución medidos en BPTomo para el conjunto de datos 1 usando los parámetros previamente descritos. De forma similar, la Fig. 9 muestra los tiempos estimados de la Tabla 3 y los tiempos de ejecución medidos en BPTomo para el conjunto de datos 2 usando los parámetros ya mencionados pero considerando 10, 20 y 30 iteraciones.

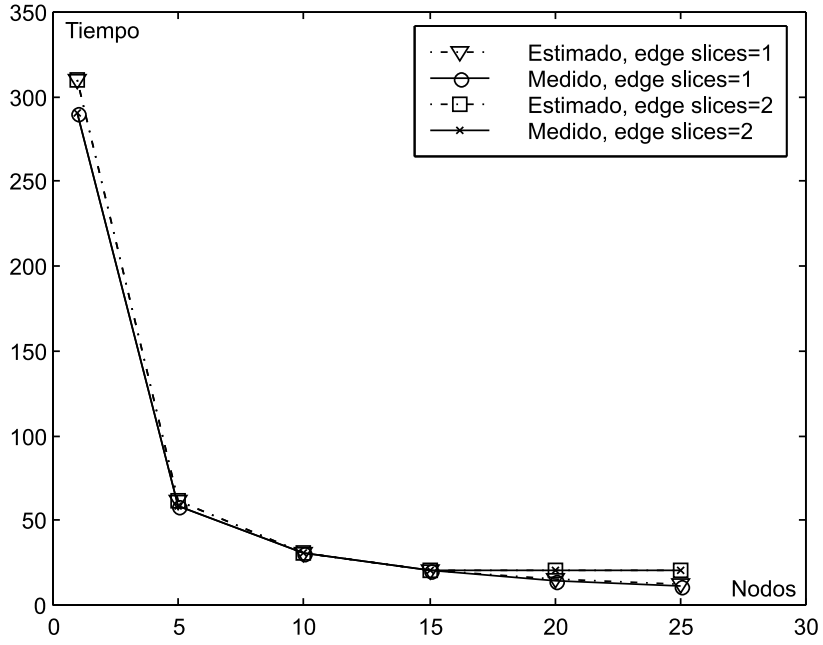


Fig. 8. Comparación entre los tiempos estimados y medidos (seg.), conj. de datos 1.

Básicamente, el tiempo total de ejecución depende del número de procesadores en el cluster. Es evidente que el tiempo de ejecución disminuye a medida que el número de procesadores aumenta. También, las comunicaciones y la sobrecarga de la red crecen siendo inútil más procesadores activos en el cluster a partir de un número dado. Encontrar este punto de inflexión es uno de los desafíos más importantes para los usuarios de aplicaciones paralelas. Es claro que para nuestras pruebas dicho punto está alrededor del nodo 15 para el conjunto de datos 1. En consecuencia, la relación entre el costo y el beneficio de tener más nodos no se justifica. Para el conjunto de datos 2, no se ha encontrado el punto de inflexión para el rango de nodos que ha sido estudiado en este trabajo.

Como se puede observar en la Fig. 8, los tiempos totales de ejecución medidos, están cerca de los estimados. La desviación es inferior al 10% tanto para el envío de 1 o 2 slices edge. Nuevamente, en la Fig. 9, los tiempos de ejecución estimados y medidos son bastante exactos. La desviación es inferior al 10% para los distintos números de iteraciones.

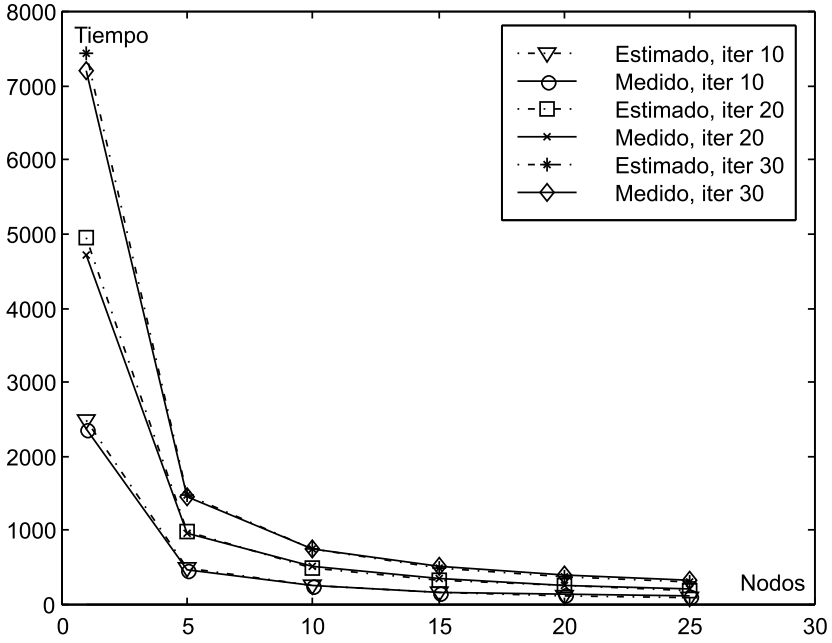


Fig. 9. Comparación entre los tiempos estimados y medidos (seg.), conj. de datos 2.

5 Conclusiones

Obtener un buen rendimiento en aplicaciones paralelas distribuidas es definitivamente una tarea difícil. Por ejemplo, muchas aplicaciones no escalan linealmente en el número de nodos y, en consecuencia, una configuración de cluster más potente (por ej. más nodos) puede no mejorar el rendimiento de la aplicación general. Además, la ejecución de sesiones de medición puede ser compleja y consumir bastante tiempo. Considerando lo antes expresado la habilidad de predecir el rendimiento es un trabajo útil.

En este artículo se ha descrito un modelo analítico de predicción de rendimiento para una aplicación paralela distribuida de reconstrucción tomográfica, llamada BPTomo. Las características y la regularidad de la aplicación permiten el uso de métodos de predicción analíticos en lugar de simulación; simultáneamente de esta manera el tiempo necesitado para evaluar el rendimiento se reduce.

La adopción de expresiones analíticas permite a los usuarios hacer análisis complejos para diferentes tamaños de problemas y número de procesadores en sólo unos minutos. Los tiempos estimados son bastante

exactos con una desviación inferior al 10% en todos los casos. Esto hace posible, por ejemplo, seleccionar el mejor número de procesadores para una dimensión de entrada dada. Además, la ejecución de ciertos casos puede requerir horas o incluso días. Por esta razón, es esencial tener una idea del tiempo estimado para diferentes parámetros con el fin de elegir la mejor opción.

Referencias

[1] J. Frank, *Electron Tomography. Three-Dimensional Imaging with the Transmission Electron Microscope*, Plenum Press, 1992.

[2] A.J. Koster, R. Grimm, D. Typke, R. Hegerl, A. Stoschek, J. Walz y W. Baumeister, *Perspectives of molecular and cellular electron tomography*, J. Struct. Biol., Vol. 120, 276-308, 1997.

[3] B.F. McEwen y M. Marko, *The emergence of electron tomography as an important tool for investigating cellular ultrastructure*, J. Histochem. Cytochem., Vol. 49, 553-564, 2001.

[4] Y. Censor, D. Gordon y R. Gordon, *Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems*, Parallel Computing, Vol. 27, 777-808, 2001.

[5] J.J. Fernández, A.F. Lawrence, J. Roca, I. García, M.H. Ellisman y J.M. Carazo, *High Performance Computing in Electron Microscope Tomography of Complex Biological Structures*, Lecture Notes in Computer Science: VECPAR 2002, Vol. 2565, 166-180, 2003.

[6] J.J. Fernández, J. M. Carazo y I. García, *Three-Dimensional Reconstruction of Cellular Structures by Electron Microscope Tomography and Parallel Computing*, J. Paral. Distr. Comp., Vol. 64, 285-300, 2004.

[7] A. Sali, R. Glaeser, T. Earnest y W. Baumeister, *From words to literature in structural proteomics*, Nature, Vol. 422, 216-225, 2003.

[8] S.T. Peltier, A.W. Lin, D. Lee, S. Mock, S. Lamont, T. Molina, M. Wong, L. Dai, M. E. Martone y M. H. Ellisman, *The Telescience portal for tomography applications*, J. Paral. Distr. Comp., Vol. 63, 539-550, 2003.

[9] B.W. Kerninghan y D.M. Ritchie, *El lenguaje de programación C*, Prentice Hall Hispanoamérica, 1991.

[10] B. Wilkinson y M. Allen, *Parallel Programming*, Prentice Hall, 1999.