# Estudio y evaluación del encaminamiento multidestino en arquitecturas cc-NUMA con directorios comprimidos

Alejandro Orozco, Manuel E. Acacio y José M. García
Departamento de Ingeniería y Tecnología de Computadores
Universidad de Murcia
30080 Murcia (España)
e-mail: alejandroorozco@gmail.com {meacacio,jmgarcia}@ditec.um.es

## Resumen

Un importante problema de los multiprocesadores con memoria compartida-distribuida y esquemas de coherencia basados en una estructura de directorio comprimido es la enorme sobrecarga en tiempo que supone la creación y el envío de mensajes de coherencia innecesarios. En este artículo se evalúan dos técnicas de enrutamiento multidestino, llamadas *multi-path* y *arbóreo vertical*, que reducen el tráfico de la red y la sobrecarga que supone al directorio la creación de excesivos mensajes de coherencia.

## 1. Introducción

La tendencia actual en multiprocesadores de memoria compartida escalables son las arquitecturas cc-NUMA, que se basan en una red de interconexión escalable punto a punto [3], donde la memoria principal está físicamente distribuida para garantizar que el ancho de banda de memoria escala. Sin embargo, estos multiprocesadores se tienen que enfrentar al problema de la sobrecarga de memoria que introduce la estructura de directorio, necesaria para garantizar la coherencia de las caches. Si queremos un sistema escalable será necesaria la introducción de directorios comprimidos, con los que conseguimos reducir el tamaño de la estructura de directorio a costa de perder información acerca de los compartidores de cada línea de memoria. Esta pérdida de información provocará una sobrecarga en términos de tiempo de ejecución, tanto en el directorio (a la hora de crear los mensajes de coherencia), como en la red (que tendrá que soportar todo el tráfico extra). En la figura 1 podemos ver el efecto que tiene la utilización de dos códigos de compartición comprimidos como *binary tree* y *binary tree with subtrees* [2] sobre el tiempo de ejecución. Con el fin de reducir la penalización provocada por el uso de esquemas de directorio comprimidos, hemos diseñado e implementado dos técnicas de enrutamiento multidestino. La primera está basada en el concepto de camino, y se conoce como enrutamiento *multi-path* [5]. A la otra técnica la hemos llamado enrutamiento *arbóreo vertical* y es una idea original de este trabajo que se basa en el concepto de enrutamiento arbóreo.

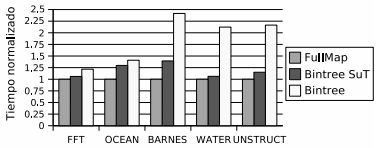


Figura 1: Fullmap frente a dir comprimidos.

## 2. Trabajo relacionado

El trabajo relacionado lo dividimos en dos partes, lo relativo a los códigos de compartición comprimidos, y lo relativo a las técnicas de encaminamiento multidestino.

### 2.1. Códigos de compartición comprimidos

Para reducir el tamaño del directorio se usa una representación comprimida de los compar-

**Actas de las XVI Jornadas de Paralelismo, JP2005, pp.101-108**
**ISBN: 84-9732-430-7** **© 2005 Los autores, Thomson**

tidores de cada línea. Algunos ejemplos son *coarse vector*, *gray-tristate* o *home*. Una técnica muy reciente ha demostrado mejorar a las anteriores y se basa en el concepto de agrupamiento multicapa en árboles binarios [2]. En esta codificación tenemos que distinguir entre sistema físico, que puede ser cualquier topología de red, y sistema lógico, que tendrá una estructura de árbol en el que los nodos se encuentran situados en las hojas. Los nodos son empaquetados en *clusters* de igual tamaño, y estos clusters situados en el primer nivel serán empaquetados en clusters de segundo nivel y así hasta que todos los nodos se encuentren agrupados en un único cluster. De esta organización lógica vamos a ver dos códigos de compartición:

- *Binary tree*: es un código de compartición muy compacto, que tiene como inconveniente la pérdida de precisión que introduce.
- *Binary tree with subtrees*: soluciona las deficiencias de *binary tree* incrementado ligeramente el tamaño del directorio.

### 2.2. Técnicas Multidestino

Pueden dividirse en dos grupos [3]:

- *Enrutamiento arbóreo (Tree-Based multicast Routing)*: se trata de mandar el mensaje multidestino a través de un camino lo más lejos posible, para después replicar el mensaje y reenviar cada copia por un canal de salida diferente.
- *Enrutamiento basado en caminos (Path-Based multidestino Routing)*: se envía desde el origen uno o varios mensajes que irán recorriendo el conjunto de destinos. Un ejemplo es *Multi-Path* [5].

## 3. Multidestino con un protocolo de coherencia basado en directorios

Las técnicas de encaminamiento multidestino nos permitirán crear un reducido conjunto de mensajes, que partiendo del nodo *home* recorran todos los nodos involucrados en un fallo de L2. Podemos dividir la técnica multidestino en dos fases. La primera la realizará el directorio al crear los mensajes multidestino, y la segunda el router para encaminar los mensajes.

### 3.1. Creación de los mensajes multidestino

A la hora de crear los mensajes multidestino el directorio tiene que dividir como máximo en cuatro subconjuntos el conjunto total de destinos. Esta distribución de los destinos se hace en base a cuatro áreas en las que queda dividida la red tomando como referencia el nodo origen. Dependiendo de la técnica multidestino esta división en áreas variará sensiblemente, como se puede ver en la figura 2.

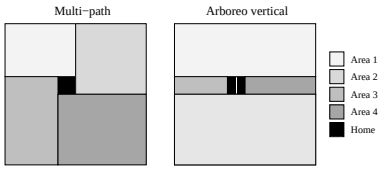


Figura 2: Situación de las áreas.

En la figura 3 se muestra el algoritmo que ha de seguir el directorio para crear los mensajes para el enrutamiento *multi-path*. Los conjuntos $D_H$ y $D_L$ contienen los nodos con etiquetas mayor y menor que la del nodo origen respectivamente. A su vez, estos dos conjuntos son divididos en una parte izquierda ($D_{H1}$ y $D_{L1}$) y una parte derecha ($D_{H2}$ y $D_{L2}$). Finalmente se ordenan los destinos de cada mensaje para facilitar posteriormente la labor del router, y se envían en dirección al primer destino.

El algoritmo para la creación de los mensajes para el enrutamiento *arbóreo vertical* lo podemos ver en la figura 4. Los conjuntos $D_H$ y $D_L$ contienen los nodos con $Y$ mayor y menor que la del nodo origen respectivamente. Los mensajes correspondientes a $D_{E1}$ y $D_{E2}$ contienen los destinos que se encuentran justo en la misma fila que el origen. Una ordenación simple de los destinos no será útil, ya que el mensaje podrá ser replicado dividiéndose el conjunto de destinos. Una posible solución consiste en dividir los destinos en subpartes, que posteriormente serán introducidas

**Algoritmo: Creación de los mensajes para el enrutamiento *multi-path*.**
**Entrada:** Conjunto total de destinos $D$, dirección del nodo origen $u_0 = (x_0, y_0)$, y su etiqueta $l(u_0)$.
**Salida:** 4 conjuntos ordenados de destinos $D_{H1}$, $D_{H2}$, $D_{L1}$, $D_{L2}$.
**Procedimiento:**
1. Dividir $D$ en $D_H$ (conteniendo los nodos con etiqueta $l$ mayor que $l(u_0)$) y $D_L$ (los nodos con $l$ menor que $l(u_0)$).
2. Dividir $D_H$ en dos subconjuntos $D_{H1}$ y $D_{H2}$:
   $D_{H1} = \{(x, y) \mid x < x_0\}$
   $D_{H2} = \{(x, y) \mid x \geq x_0\}$
3. Obtener $D_{L1}$ y $D_{L2}$ de forma equivalente a $D_{H1}$ y $D_{H2}$.
4. Ordenar las cuatro listas en $X$, siguiendo un orden creciente respecto a la distancia a $u_0$. Si existen puntos con coordenada $X$ igual, seleccionar primero aquel nodo más próximo al último nodo de la columna anterior, para continuar con un orden lineal con el resto de destinos de esa columna.
5. Construir los mensajes que contengan las listas como parte de la cabecera, y enviarlos al nodo situado en la primera posición de la lista.

Figura 3: Algoritmo de creación de mensajes para el enrutamiento *multi-path*.

**Algoritmo: Creación de los mensajes para el enrutamiento *arbóreo vertical*.**
**Entrada:** Conjunto total de destinos $D$, dirección del nodo origen $u_0 = (x_0, y_0)$.
**Salida:** 4 conjuntos ordenados de destinos $D_H$, $D_L$, $D_{E1}$, $D_{E2}$.
**Procedimiento:**
1. Dividir el conjunto $D$ en tres subconjuntos $D_H$, $D_L$ y $D_E$ de la siguiente forma:
   $D_H = \{(x, y) \mid y > y_0\}$
   $D_L = \{(x, y) \mid y < y_0\}$
   $D_E = \{(x, y) \mid y = y_0\}$
2. Dividir $D_E$ en dos subconjuntos $D_{E1}$ y $D_{E2}$:
   $D_{E1} = \{(x, y) \mid x < x_e, \forall (x_e, y_e) \in D_E\}$
   $D_{E2} = \{(x, y) \mid x \geq x_e, \forall (x_e, y_e) \in D_E\}$
3. Construir el mensaje que contenga $D_H$ como parte de la cabecera, y enviarlo por el canal de salida superior.
4. Construir el mensaje que contenga $D_L$ como parte de la cabecera, y enviarlo por el canal de salida inferior.
5. Construir el mensaje que contenga $D_{E1}$ como parte de la cabecera, y enviarlo por el canal de salida izquierdo.
6. Construir el mensaje que contenga $D_{E2}$ como parte de la cabecera, y enviarlo por el canal de salida derecho.

Figura 4: Algoritmo de creación de mensajes para el enrutamiento *arbóreo vertical*.

**Algoritmo: Procedimiento de enrutamiento *multi-path*.**
**Entrada:** Mensaje con una lista de destinos ordenada $D_M = \{d_1, d_2, \ldots, d_k\}$, y una dirección local $w$.
**Procedimiento:**
1. Si $w = d_1$ entonces $D'_M = D_M - \{d_1\}$, enviándose copia del mensaje a $w$.
   Si no $D'_M = D_M$.
2. Si $D'_M = \emptyset$ entonces fin de transmisión del mensaje.
3. Sea $d$ el primer nodo de $D'_M$ y $w' = R(w, d)$.
4. El mensaje es enviado a $w'$ con la lista de destinos $D'_M$ en su cabecera.

Figura 5: Algoritmo de enrutamiento *multi-path*.

**Algoritmo: Procedimiento de enrutamiento *arbóreo vertical*.**
**Entrada:** Mensaje con una lista de destinos $D_M = \{d_1, d_2, \ldots, d_k\}$, y una dirección local $w$.
**Procedimiento:**
1. Si $\exists d_i \in D_M \ tq \ w = d_i$ entonces $D_{M1} = D_M - \{d_i\}$, enviándose copia del mensaje a $w$.
   Si no $D_{M1} = D_M$.
2. $\forall d_i \in D_{M1} \ tq \ y(d_i) = y(w) \wedge x(d_i) < x(w)$ entonces $D_{M1} = D_{M1} - \{d_i\}$, y $D_{M2} = D_{M2} + \{d_i\}$.
3. $\forall d_i \in D_{M1} \ tq \ y(d_i) = y(w) \wedge x(d_i) > x(w)$ entonces $D_{M1} = D_{M1} - \{d_i\}$, y $D_{M3} = D_{M3} + \{d_i\}$.
4. Si $D_{MX} = \emptyset$ para $X = 1, 2, 3$, entonces fin de transmisión del mensaje $D_{MX}$.
5. Enviar el mensaje con la lista $D_{M1}$ en su cabecera por el canal de salida que sigue la dirección y sentido del mensaje $D_M$.
6. Enviar el mensaje con la lista $D_{M2}$ en su cabecera por el canal de salida izquierdo.
7. Enviar el mensaje con la lista $D_{M3}$ en su cabecera por el canal de salida derecho.

Figura 6: Algoritmo de enrutamiento *arbóreo vertical*.

íntegramente en copias del mensaje original, pero es claramente no escalable. Otra solución consiste en no ordenar el conjunto de destinos, y dejar al router que se encargue de todo el trabajo. Esta es la solución más interesante, ya que realmente no supone tanto sobreesfuerzo para el router como se verá más adelante.

### 3.2. Enrutamiento de un mensaje multidestino

La función de los routers consiste en encaminar correctamente los mensajes desde un nodo origen hasta otro destino. En el caso de los mensajes multidestino la funcionalidad de los routers se ve aumentada. Cuando un mensaje multidestino llega a un destino, es posible que el router tenga que enviar una copia del mensaje hacia otro destino. En la figura 5 podemos ver el algoritmo que usará el router para encaminar correctamente los mensajes *multi-path*.

En el caso del enrutamiento *arbóreo vertical* el trabajo del router se complicará al no estar la lista de destinos ordenada y tener que crear varias copias de un mensaje bajo ciertas circunstancias. Además, habrá que distinguir entre aquellos mensajes que pueden ser replicados por diferentes canales de salida y aquellos para los que no se podrá dar esta circunstancia. Esto lo podemos saber simplemente viendo el canal por el que entra el mensaje, ya que un mensaje que se desplaza horizontalmente no podrá ser replicado por varios canales, mientras que los mensajes que se desplazan verticalmente sí podrán. Esta diferenciación de los mensajes en función del canal por el que entran, simplifica notablemente la complejidad del router, lo que implica un menor costo hardware y una mayor velocidad de enrutamiento. En la figura 6 podemos ver el algoritmo más en detalle.

## 4. Modificaciones realizadas

El simulador elegido para el desarrollo de las técnicas multidestino y su evaluación ha sido una versión ampliada de RSIM [4] que incluye varios esquemas de directorios. Sobre esta versión de RSIM se han realizado modificaciones en el código fuente para lograr que implemente las técnicas multidestino y así poder evaluarlas. Estas modificaciones las hemos clasificado:

- *Directorio*: al crear un mensaje de coherencia multidestino, primero seleccionamos aquellos destinos del conjunto total que se adapten a una de las cuatro áreas definidas por nuestros algoritmos (figura 2). A continuación reordenaremos los destinos para facilitar la labor del router.
- *Router*: cuando un mensaje llega al router se comprobará si éste es destino del mensaje, en cuyo caso enviaremos una copia hacia la cache L2. A continuación eliminamos la dirección del conjunto de destinos y se reenvía el mensaje por el canal de salida adecuado. En el caso del enrutamiento *arbóreo vertical*, es posible que haya que replicar el mensaje por varios canales de salida, diviendo el conjunto de destinos.
- *Estadísticas*: para obtener las estadísticas introducimos contadores en las peticiones de datos producidas en las caches L2, que se incrementan al avanzar por los routers y el directorio del nodo *home*.

## 5. Evaluación y resultados

### 5.1. Descripción de la arquitectura

Utilizamos una arquitectura cc-NUMA con protocolo de coherencia *MESI*, con el que las caches tendrán asociada a cada línea información sobre el estado en el que se encuentran. Estos estados son: *Inválido*, *Compartido*, *Exclusivo* y *Modificado*. Un fallo de L2 provocará una petición al directorio del nodo *home* que obtendrá tanto el estado de la línea de memoria, como la lista de compartidores. El campo llamado *estado de directorio* podrá tomar tres valores: *Uncached*, *Shared* o *Private*. Dependiendo del estado de la línea de memoria y del tipo de petición de la cache L2 (lectura o escritura), vamos a clasificar los fallos de L2 en cuatro categorías siguiendo la metodología expuesta en [1]:

- *Fallos $-to-$*: se producen cuando la línea a la que se accede se encuentra en estado

*Privado.* El directorio reenvía la petición al propietario de la línea, que a su vez envía una respuesta con la línea directamente al nodo peticionario y un mensaje de confirmación al directorio del *home*.

- *Fallos de memoria*: pueden aparecer ante una petición de lectura o escritura. El directorio accederá a memoria para enviar la línea al peticionario.

- *Fallos de invalidación*: se producen por un acceso de escritura a una línea de memoria compartida, siendo el nodo peticionario uno de los compartidores. El directorio enviará mensajes de invalidación a todos los compartidores excepto al que realizó la petición. Una vez que haya recibido todos los mensajes de reconocimiento (ack), podrá notificar al peticionario su condición de propietario.

- *Fallos invalidación + memoria*: similar a los fallos de invalidación, pero además el directorio tendrá que entregar el dato de memoria al nodo peticionario.

### 5.2. Parámetros de simulación

La evaluación de los resultados ha sido realizada simulando un conjunto de benchmarks en una máquina multiprocesador con las características que se muestran en la tabla 1. Por motivos de brevedad sólo mostraremos los resultados obtenidos para `BARNES` y `UNSTRUCT`, que consideramos representativos del conjunto total de benchmarks. Para ver los resultados para un conjunto de benchmarks mayor, ver [6]. Para obtener datos relativos a diferentes estructuras de directorio, se ha optado por los esquemas *fullmap*, *binary tree* y *binary tree with subtrees*. Para cada uno de estos esquemas, se han obtenido resultados para las técnicas de enrutamiento *unicast*, *multi-path* y *arbóreo vertical*.

### 5.3. Resultados

Los resultados se evaluan desde dos puntos de vista:

| **Parámetros del procesador ILP** | |
|---|---|
| Velocidad del procesador | 1 GHz |
| Tasa máxima *fetch/retire* | 4 |
| **Parámetros de la jerarquía de memoria** | |
| Tamaño de la línea de cache | 64 bytes |
| Cache L1 : | *write-through* |
| * Tamaño | 32KB |
| * Asociatividad | Mapeo directo |
| * Tiempo de acierto | 2 ciclos |
| Cache L2 : | *write-back* |
| * Tamaño | 512KB |
| * Asociatividad | 4-vías |
| * Tiempo de acierto | 15 ciclos (6 + 9), segmentado |
| Tiempo de acceso a memoria | 60 ciclos |
| Grado de intercalado de memoria | 4 |
| Protocolo de coherencia de cache | MESI |
| Modelo de consistencia | Secuencial |
| Ciclo de directorio | 10 ciclos |
| Tiempos de creación de mensajes de coherencia: | |
| * Primer mensaje de coherencia | 4 ciclos de dir. |
| * Siguientes mensajes | 2 ciclos de dir. |
| Velocidad del bus | 1 GHz |
| Ancho del bus | 8 bytes |
| **Parámetros de Red** | |
| Topología | Malla |
| Velocidad del *router* | 250 MHz |
| Ancho de banda del canal | 32 bits |
| Velocidad del canal | 500 MHz |
| Número de canales | 1 |
| Tamaño del *flit* | 8 bytes |
| Tamaño de mensaje sin datos | 16 bytes |
| Ancho del Router interno del bus | 64 bits |
| Retardo de arbitraje | 4 ciclos del router |

Cuadro 1: Parámetros básicos del sistema.

- Los tiempos medios necesarios para la resolución de cada uno de los tipos de fallos en la cache L2, desglosados en diferentes fases. Estos tiempos se refieren exclusivamente a los fallos de cache de tipo *$-to-$*, *invalidación* e *invalidación + memoria*. Quedan excluidos los fallos de *memoria* de estas estadísticas, ya que no producen el envío de mensajes de coherencia de ningún tipo.

- Las variaciones producidas en el tiempo de ejecución final de las aplicaciones.

#### 5.3.1. Análisis de tiempos para los diferentes tipos de fallos de L2

El proceso producido por un fallo de cache L2 lo podemos dividir en 5 fases. En primer lugar está el tiempo que pasa la petición en la red desde que abandona el nodo peticionario hasta que llega al nodo *home* y viceversa, y lo hemos

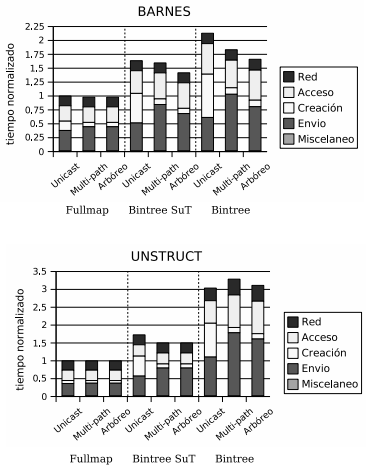


Figura 7: Tiempo empleado en las diferentes fases de un fallo de *invalidación*.

llamado tiempo de *red*. El tiempo de *acceso* es el que pasa la petición en la cola del directorio hasta que es procesada. El de *creación* es el que necesita el directorio para crear todos los mensajes de coherencia. El tiempo de *envío* es el que transcurre desde que se envía el último mensaje de coherencia hasta que se recibe la última contestación. Por último, el tiempo *misceláneo* es el tiempo que la petición pasa en los buses y caches.

Para los fallos de *invalidación* tenemos en la figura 7 el desglose de tiempos de las diferentes fases. Para un esquema de directorio *fullmap*, los tiempos se mantienen constantes con las tres técnicas de envío de mensajes de coherencia. Esto se debe a que el número de nodos que comparten una línea es muy pequeño, por lo que el uso de unicast o multidestino variará poco los resultados.

Si nos fijamos sólo en las barras correspondientes a *unicast*, vemos como la utilización de un código de compartición *binary tree with subtrees* degradará el tiempo total respecto a *fullmap* sobre un 70 %. Por su parte, *binary tree* degradará el tiempo total hasta en un 225 % respecto a *fullmap* en el caso de `UNSTRUCT`. Si analizamos detalladamente la causa por la que se produce esta degradación, vemos que la fase en la que se produce un mayor aumento de tiempo es en la de creación de los mensajes. Esto se debe al importante incremento en el número de mensajes de coherencia que implica el uso de un directorio comprimido. Si ahora ampliamos nuestros horizontes hasta las barras correspondientes a las técnicas multidestino, vemos como el tiempo de creación se mantiene prácticamente constante para todos los códigos de compartición. En este aspecto se consigue una mejora importante respecto a *unicast*. La fase que más repercute en el tiempo total es el tiempo de envío. En este caso las dos técnicas multidestino que hemos implementado se comportarán peor que unicast. Esto se puede explicar por el hecho de no encontrarse sobrecargada la red, por lo que aceptará sin problemas todo el tráfico que le introduzcamos. No consideremos el tiempo que tardan las contestaciones en volver al nodo *home*, ya que serán unicast en todos los casos y al no estar la red saturada, no influirán sobre el tiempo de envío. Con esta premisa, la diferencia de tiempo entre las tres técnicas se debe (casi siempre) al tiempo que tarda en llegar el mensaje de coherencia al nodo más alejado del origen. A la misma distancia, parece lógico que la técnica unicast sea más eficiente que el enrutamiento *multi-path* ya que el mensaje tendrá que hacer algunos rodeos, pasando por otros destinos intermedios antes de llegar al nodo más alejado. Para el enrutamiento *arbóreo vertical* evitaremos dar rodeos, pero se sigue empeorando respecto al envío unicast, ya que este algoritmo tendrá un retardo debido a la duplicación de los mensajes en los nodos intermedios. En cuanto a los tiempos de acceso y misceláneo, no parece que se vean influidos por la técnica de enrumiento utilizada. Finalmente, al sumar los tiempos de todas las fases vemos como conseguimos mejorar, en especial con el enrutamiento *arbóreo vertical* con el que conseguimos hasta un 20 % para `BARNES`.

Para los fallos de *invalidación + memoria* podemos ver en la figura 8 como se desglosan los tiempos para las diferentes fases. Se puede observar como para `UNSTRUCT` la gráfica tiene una línea muy similar a la que obtuvimos para los fallos de *invalidación*. En cambio para `BARNES` el tiempo de envío aumenta drás-

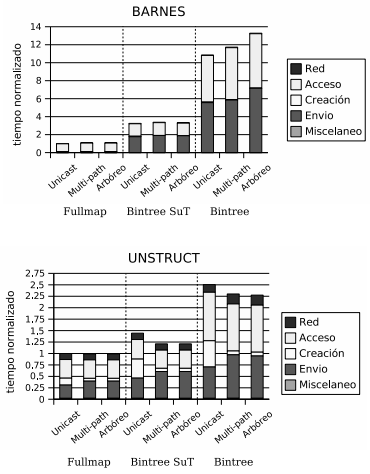


Figura 8: Tiempo empleado en las diferentes fases de un fallo de *inv + mem*.

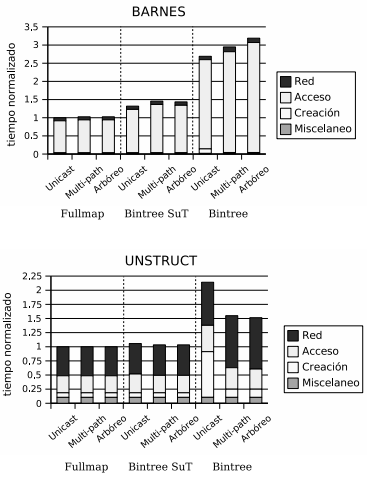


Figura 9: Tiempo empleado en las diferentes fases de un fallo de *$-to-$*.

ticamente al aumentar la compresión del código de compartición del directorio, lo que no ocurría de forma tan marcada con los fallos de *invalidación*. Este aumento desmesurado en el tiempo de envío se produce por una congestión localizada a la entrada del nodo *home* y producida por la gran afluencia de asentimientos unicast provocados por las invalidaciones. De las gráficas también se extrae un importante incremento en el tiempo de acceso al directorio, que se produce como consecuencia directa del aumento en el tiempo de envío, ya que al aumentar el tiempo de envío se acumularán más peticiones en la cola del directorio referentes a la misma línea, que tendrán que esperar a que termine la transacción actual. Esto produce que el tiempo de acceso se aproxime al tiempo de envío. Estos resultados que convierten en despreciable la ganancia en la creación de los mensajes son ligeramente negativos para las dos técnicnas multidestino.

Los fallos *$-to-$* no tendrán fase de envío ya que la respuesta se enviará directamente desde el nodo que tiene la línea de memoria en estado privado al nodo peticionario. En la figura 9 se puede observar como los tiempos para `BARNES` y `UNSTRUCT` son muy diferentes. Para el primero de ellos tenemos que el tiempo de acceso se mantiene muy elevado, debido al generalizado aumento en el tiempo de envío que vimos en los fallos de *invalidación + memoria*. De nuevo este factor perjudica a las técnicas multidestino. Para `UNSTRUCT` se puede apreciar como los tiempos obtenidos con *fullmap* y *binary tree with subtrees* son prácticamente idénticos, debido a que por un lado no aumenta el número de compartidores, y por otro el tiempo de acceso se mantiene constante. En el caso de un código de compartición *binary tree*, podemos ver como para *unicast* el tiempo de creación aumenta significativamente, cosa que evitamos con las técnicas multidestino. En consecuencia conseguimos un importante aumento de rendimiento para este tipo de fallos.

### 5.3.2. Análisis del tiempo de ejecución

Según los resultados expuestos en la figura 10 podemos decir que el enrutamiento *arbóreo vertical* se comporta mejor que el *multi-path*, en especial con un código de compartición *binary tree*, donde el número de destinos por mensaje multidestino es más elevado.

En el caso de `BARNES`, al usar una técnica de enrutamiento multidestino se produce un aumento en el tiempo necesario para llevar a cabo las peticiones de *invalidación + memoria* y de *$-to-$* debido a un incremento del tiempo

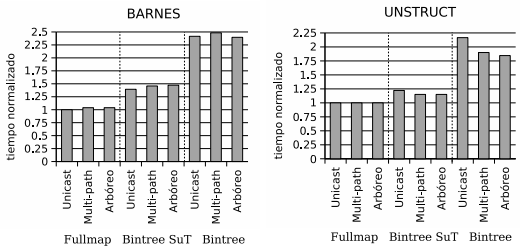


Figura 10: Tiempo total de ejecución.

de acceso. Por suerte el peso relativo de los fallos de *inv + mem* es pequeño, y la ganancia en las peticiones de *invalidación* permite que el tiempo final de ejecución mejore levemente al usar encaminamiento *arbóreo vertical*.

La aplicación `UNSTRUCT` es la que mejores resultados nos ha dado. El tiempo final de ejecución para un directorio *binary tree* ha mejorado en un 15 % al usar un enrutamiento *arbóreo vertical* respecto a los resultados obtenidos con unicast. Esto se debe a que el peso relativo de la creación de los mensajes es mucho mayor que el del envío, por lo que la ganancia es mucho mayor que la pérdida. Además para las transferencias de *$-to-$*, habrá un gran número de mensajes innecesarios, provocando una mejora en el tiempo de creación al usar técnicas multidestino.

## 6. Conclusiones y trabajos futuros

En este artículo se han presentado y evaluado dos técnicas de encaminamiento multidestino, para minimizar el costo sobre el rendimiento del multiprocesador producido por el uso de esquemas de directorio comprimidos, y así mejorar la escalabilidad de los cc-NUMA. Esta escalabilidad se logra gracias a la minimización del tiempo de creación de los mensajes de coherencia, y a la reducción de la sobrecarga de la red. Esta arquitectura permitirá diseñar multiprocesadores cc-NUMA con un número de nodos bastante mayor que los actuales. Los directorios comprimidos solucionan el problema de la sobrecarga de memoria que implica tener un sistema con miles de procesadores, pero introducen una importante degradación en el tiempo de ejecución. La utilización de técnicas de encaminamiento multidestino podría conseguir aproximar el tiempo de ejecución de un sistema con directorio comprimido al de un sistema con directorio *fullmap*.

Como trabajo futuro, resulta de gran interés implementar técnicas de enrutamiento multidestino para las contestaciones a los mensajes de coherencia, y así conseguir una mayor liberación de la carga de trabajo de la red. También sería muy interesante probar técnicas de encaminamiento multidestino adaptativo, que se acoplarían mejor a las condiciones dinámicas de la red.

## Referencias

[1] Manuel E. Acacio, J. González, J.M. García, J. Duato. An Architecture for High-Performance Scalable Shared-Memory Multiprocessors Exploting On-chip Integration. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 8, Aug. 2004.

[2] Manuel E. Acacio, J. González, J.M. García, J. Duato. A Two-Level Directory Architecture for Highly Scalable cc-NUMA Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, No. 1, Jan 2005.

[3] J. Duato, S, Yalmanchili, L.M. Ni. Interconnection Networks, an Engineering Approach. *IEEE Computer Society, Inc.*, 1997.

[4] C.J. Hughes, V.S. Pai, P. Ranganathan, and S.V. Adve. RSIM: Simulating Shared-Memory Mutiprocessors with ILP Processors. *IEEE Computer*, vol. 35, no.2, pp 40-49, Feb. 2002.

[5] X. Lin, P.K. McKinley, and L.M. Li. Deadlock-Free Multicast Wormhole Routing in 2D-Mesh Multicomputers. *Proc. Int. Conf. on Parallel Processing*, pp. III-114-III-118, Feb. 1991.

[6] Alejandro Orozco López. Estudio y evaluación del encaminamiento Multicast en cc-NUMAs con Directorios Comprimidos. *Proyecto fin de carrera en la Universidad de Murcia, Julio 2005.*