UNIVERSIDAD DE CANTABRIA

DPTO. DE ELECTRÓNICA Y COMPUTADORES.

IMPACTO DEL SUBSISTEMA DE COMUNICACIÓN EN EL RENDIMIENTO DE LOS COMPUTADORES PARALELOS: DESDE EL HARDWARE HASTA LAS APLICACIONES.

Presentada por:

Valentin Puente Varona

Dirigida por:

Ramón Beivide Palacio.

SANTANDER, OCTUBRE DE 1999

Capítulo 2

Tecnologías, Métricas y Evaluación de Rendimiento.

En este capítulo se introducirá el marco empleado en el resto del trabajo para analizar el rendimiento de una red de interconexión. Fundamentalmente describiremos el modo en el que se establecen las dependencias tecnológicas de cada propuesta y las métricas consideradas a la hora de evaluar el rendimiento de cada una de las alternativas analizadas. Para ello se establece un banco de pruebas constituido por aplicaciones reales sobre un computador paralelo.

2.1 Introducción.

En primer lugar, para los objetivos marcados en el trabajo tendremos que conocer de qué manera las restricciones que impone la implementación *hardware* influyen en el rendimiento de la red. Se propone una metodología de evaluación que permite establecer estas dependencias y explorar suficientemente el espacio de diseño. Estos condicionantes juegan un papel muy importante a la hora de evaluar comparativamente el rendimiento real de las propuestas arquitectónicas y una vez consideradas podremos medir, de forma cuantitativa, el rendimiento real. El banco de pruebas empleado va desde el análisis a partir de tráfico sintético hasta el análisis bajo condiciones de carga real. Estableceremos cuales son las aplicaciones que jugarán el papel de *benchmarks* a la hora de considerar el rendimiento de la red de interconexión. En cuanto a la arquitectura propuesta, nos centraremos en el caso particular de sistemas DSM con coherencia hardware y más concretamente en la arquitectura ccNUMA.

2.2 Análisis de los costes Hardware.

2.2.1 La Importancia de Los Costes Hardware Asociados.

Existen múltiples trabajos en los que se aborda el análisis de rendimiento de determinadas propuestas arquitectónicas desde un punto de vista estrictamente funcional. En otras palabras, no se toma en consideración las consecuencias *hardware* de las propuestas bajo estudio. Más concretamente, en el caso de los estudios correspondientes a las redes de interconexión es bastante habitual omitir este tipo de cuestiones. Sin embargo, obviar estos factores pueden dar lugar a conclusiones erróneas, en el sentido de que es posible que los costes *hardware* hagan desaparecer completamente la posible mejora introducida desde el punto de vista funcional. En la mayoría de los casos el origen del error está relacionado con que no se determina de forma precisa la influencia de las mejoras introducidas en el coste temporal de los encaminadores.

En esta línea, existen varios trabajos previos en los que se afronta esta problemática proponiendo modelos sencillos de manejar que permiten determinar de forma aproximada el coste *hardware* en función de determinadas características estructurales del encaminador [6][30]. En estos estudios se propone una arquitectura canónica *wormhole* sobre la que se pueden alterar un conjunto limitado de parámetros. Fundamentalmente, se fija el tipo de tecnología empleada (0.8μm CMOS *gate array*) y se puede determinar el camino crítico de cada uno de los elementos del encaminador en función de únicamente tres parámetros: el número de puertos del crossbar, el número de grados de libertad y el número de canales virtuales por línea física del encaminador.

A partir de unas constantes fijas, que dependen exclusivamente de la tecnología empleada en el estudio, se puede establecer el camino crítico en todos los componentes del encaminador. Esta aproximación es, sin duda, más apropiada que considerar exclusivamente las características funcionales del encaminador a la hora de evaluar su rendimiento. Sin embargo, presenta un número importante de limitaciones que restringe en muchos casos su uso. El modelo es valido exclusivamente para encaminadores *wormhole* con un espacio de almacenamiento exiguo. Además, solo es adecuado para la tecnología empleada y sirve para la extracción de las constantes típicas. Por último, estos datos son fuertemente dependientes de otros factores más sutiles como la calidad de las descripciones, el tipo de herramientas empleadas en su obtención, etc...

Estas limitaciones incitan a plantear una metodología autocontenida que permita afrontar el problema de forma más amplia. Sin duda, el sistema más fiable para alcanzar este objetivo es determinar, hasta el final, las implicaciones *hardware* de cada propuesta. En otras palabras, implementar en *hardware* y caracterizar posteriormente el coste de cada uno de los encaminadores a evaluar.

En este trabajo se ha optado por seguir esta última aproximación, logrando así determinar con un grado de precisión aceptable cuál es el coste real de todas las posibles propuestas algorítmicas o estructurales de los encaminadores pero sin limitar el espacio de diseño a estudiar. A continuación pasaremos a comentar cuales son los aspectos más importantes de la metodología empleada.

2.2.2 Tecnologías Síncrona y Asíncrona.

La metodología empleada en este trabajo se centra exclusivamente en el empleo de lógica síncrona. La razón fundamental que ha sugerido este planteamiento es la existencia de herramientas mucho más desarrolladas y fiables para este tipo de tecnología que para el caso de la lógica asíncrona. Por otro lado, actualmente prácticamente la totalidad de los sistemas reales emplean tecnología síncrona en el subsistema de comunicación [22][55][112]. La mayoría de estos encaminadores son suficientemente complejos como para que el empleo de tecnología asíncrona haga prácticamente irrealizable su desarrollo.

Queda claro que pese a que la lógica síncrona adolece de muchos inconvenientes, es actualmente la más recomendable a la hora de plantear un método de trabajo sistemático. Alguna de las herramientas que facilitan este trabajo son Synopsys [111], Cadence [24], Mentor [84], etc... Se trata de herramientas comerciales estables y que permitirán abordar el problema de diseño con un grado de dificultad aceptable.

Son claramente conocidos los inconvenientes que presenta este tipo de lógica frente a la lógica asíncrona. Entre los que más nos afectan, podemos citar el problema de la distribución del reloj, la metaestabilidad y la segmentación.

El problema de la distribución del reloj.

Este tipo de lógica esta gobernada, en el régimen de operación normal, por uno más relojes. En un estructura tan amplia como una red de interconexión y donde en la mayoría de los casos se trata de estructuras multi-chip, es prácticamente imposible distribuir el reloj del sistema de forma correcta. Es necesario tener en cuenta que el número de nodos de un sistema multiprocesador puede llegar a ser de varias decenas, cientos o incluso miles. Por lo tanto, es claro que distribuir un reloj sin ningún retraso en un sistema tan grande es prácticamente imposible. En la Figura 2-1 se puede apreciar como claramente el retraso que sufrirá la señal de reloj para llegar a un encaminador de la fila superior será mayor que el que sufrirá para llegar a uno de la fila inferior, dado que la diferencia de distancia recorrida por la señal puede ser del orden de varios centímetros o incluso metros. Esto dará lugar a problemas serios en la comunicación entre encaminadores vecinos, que pueden llegar a hacer completamente inoperante el sistema.

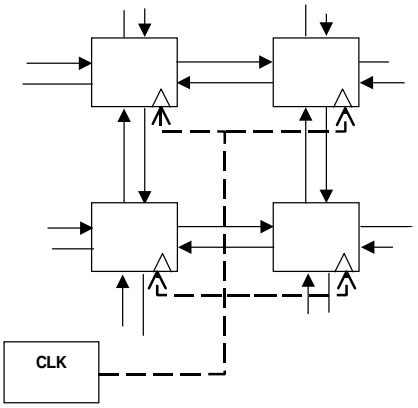

Figura 2-1. Distribución del reloj con lógica completamente síncrona.

La solución adoptada, como en gran parte de los sistemas de comunicación reales, es emplear lógica *self-timed.* La idea se centra en que cada uno de los encaminadores de la red esta gobernado por un generador de reloj propio. Todos los relojes del sistema poseen un periodo muy similar. Sin embargo, la obtención de relojes completamente idénticos no es posible, necesariamente siempre existirá una diferencia de fase entre ellos que estará relacionada con las pequeñas diferencias que tengan en su periodo y el tiempo que lleva operando el sistema. Este problema

de nuevo hace imposible la comunicación entre encaminadores de forma síncrona. Para solucionar este problema es necesario recurrir a protocolos de comunicación asíncronos.

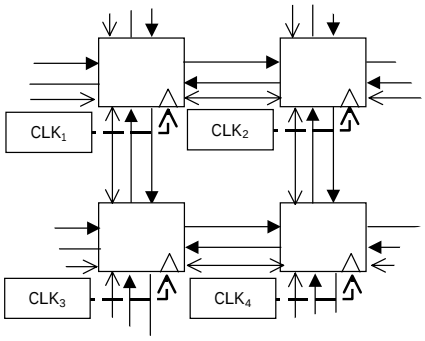

Figura 2-2. Distribución del reloj con lógica síncrona *self-timed.*

La necesidad de un modo de comunicación asíncrono entre encaminadores implica el empleo de un protocolo y mecanismos que faciliten la sincronización en la recepción de datos. Dado que internamente cada encaminador opera de forma síncrona es factible emplear una variante del protocolo asíncrono de cuatro fases. De esta forma, el dato enviado es situado en la línea de comunicación de forma simultánea con el envío de la petición. La diferencia radica en que la señal de respuesta actúa en realidad como una señal de *stop*, en el sentido de que el dato y la petición solamente serán lanzadas si la señal de repuesta (*Stop)* no esta activada. Los problemas de desbordamiento, causados por las carreras a que puede dar lugar el retraso de la señal de *stop*, se pueden solucionar fácilmente sin más que avanzar la activación de la señal de *stop* de tal forma que todos los posibles datos en vuelo sean almacenados adecuadamente. Este tipo de técnicas son aplicadas en varios sistemas comerciales como [13] para realizar *channel-pipelining*. Aunque el contexto es diferente, la técnica es muy similar, dado que hasta que la activación del *stop* llega al encaminador fuente y es tomada en consideración, es posible que se hayan enviado más datos. De esta forma podemos evitar el emplear un protocolo de 2 fases, lo que incrementará notablemente el coste de la comunicación entre encaminadores adyacentes. Es importante señalar que en este aspecto también influye el tipo de control de flujo empleado: es claro que con un control de flujo tipo *wormhole* hay que ser más cuidadosos en estos aspectos, mientras que un control de flujo VCT la comunicación será más segura.

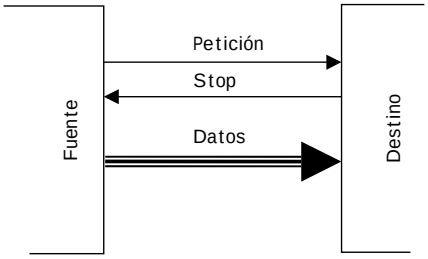

Figura 2-3. Protocolo de comunicación entre encaminadores.

Metaestabilidad

Por otro lado, es necesario incorporar un mecanismo de sincronización que actúe de interface entre los puertos de comunicación de los encaminadores y la lógica completamente síncrona del interior de cada uno. Este módulo de interface ha de ser planteado con especial cuidado dado que es el responsable de que todos los datos enviados por el encaminador fuente sean recibidos por el destino. La posible perdida de datos esta relacionada con problemas de *metaestabilidad* causados por las pequeñas diferencias de periodo entre los relojes de encaminadores adyacentes.

La aparición del fenómeno de *metaestabilidad* se produce cuando una señal de entrada no respeta los tiempos de *hold* o *setup* de la celda a la que va dirigida. Las soluciones para este problema son variadas: algunos fabricantes ofrecen *flip-flops* en sus librerías tecnológicas pensados específicamente para tolerar este fenómeno y hacer que el tiempo medio entre fallos (*MTBF*) sea prácticamente infinito. Existen soluciones independientes de la tecnología que permiten la detección del fenómeno, o evitación parcial, de tal forma que la probabilidad de que un dato se pierda es muy pequeña.

El MTBF de un *flip-flop* puede expresarse como:

$$MTBF = \frac{1}{2 \; f_{clk} \; f_{data} \; t_0 \; 10^{-(\frac{T_{clk} - t_0 - t_s}{t})}} \qquad \textbf{[2-1].}$$

donde f_{clk} es la frecuencia del reloj de referencia (frecuencia de reloj del encaminador destino), f_{data} es la frecuencia de los datos (frecuencia de reloj del encaminador fuente), T_{clk} es la inversa de f_{clk}, t_0 es el retraso intrínseco del *flip-flop* utilizado en la sincronización, t_s es su tiempo de *setup* y *t* es su constante de tiempo. Debido al carácter exponencial de la ecuación, un pequeño incremento en la frecuencia de operación del circuito puede dar lugar a una gran degradación

del tiempo promedio entre fallos. Para una tecnología submicrónica, con *flip-flops* convencionales a frecuencias superiores a 200MHz, el tiempo promedio entre fallos puede situarse por debajo de unos pocos días, lo que es inaceptable para sistemas tan críticos como los multiprocesadores.

En nuestro caso es posible enviar el reloj de la fuente a través de la señal de petición de tal forma que en el encaminador vecino, el interfaz asíncrono puede construirse a través de un sincronizador de doble etapa. Esto hace que la probabilidad de metaestabilidad del sincronizador sea el producto de las probabilidades de metaestabilidad de los dos *flip-flops* que lo forman [49]. Esta estrategia puede dar lugar a tiempos promedio entre fallos muy elevados y hacer prácticamente desaparecer el problema. Claramente, cuantas más etapas se empleen, mucho mayor será el MTBF. Sin embargo, cada una de las etapas consume un ciclo de reloj, lo que puede hacer que el tiempo de paso del encaminador sea excesivo. Otra solución frente a este problema consiste en emplear una interface asíncrona como la propuesta en [6]. Esta interface se basa en el elemento C de Muller como módulo de sincronización. Este componente consume entre 0.5 y 1.5 ciclos de reloj en lugar de 2 para la sincronización de los datos. De esta forma, al emplear este elemento, reducimos el tiempo de paso a un solo ciclo en promedio sin renunciar a un MTBF suficientemente alto. Una descripción detallada de este elemento y de su implementación como interface asíncrona de comunicación entre encaminadores *self-timed* puede encontrarse en [23].

Estructura del pipeline

El rendimiento que puede llegar a alcanzar la lógica síncrona, en lo que respecta a tiempos de paso de los encaminadores, va a ser generalmente peor que el que se puede alcanzar con lógica asíncrona. La principal causa de esta degradación de prestaciones es el modo en que esta fijada la frecuencia de operación en un circuito multiciclo. En el caso de los encaminadores, es necesario segmentar en diversas unidades funcionales que operan en ciclos consecutivos a modo de *pipeline*. El ciclo de reloj viene determinado por la etapa que tenga mayor camino crítico. Como es evidente, esto introduce una caída de rendimiento frente a una posible implementación asíncrona ya que, en el resto de unidades funcionales del pipeline, el retraso será el mismo que el de la más lenta. Por ello, es necesario prestar especial atención a la hora de segmentar el encaminador, de forma que todas las etapas del *pipeline* posean retrasos lo más similares posibles. Sin embargo, no siempre es fácil lograr balancear perfectamente el *pipeline* del encaminador ya que en muchos casos las unidades funcionales son difíciles de segmentar o trasladar parte de su cometido a las etapas adyacentes. De cualquier manera, la descompensación en las etapas del pipeline se puede aprovechar para reducir considerablemente el área de silicio de las etapas no críticas.

2.2.2.1 Lenguajes de Descripción Hardware: VHDL.

Pese a los indudables inconvenientes de la lógica síncrona, ésta posee una simplicidad que es abordable por muchas herramientas de diseño, lo que provoca que la tarea de implementación *hardware* sea *casi* automática. De hecho, existen muchas herramientas que permiten abordar el proceso de diseño desde niveles de abstracción muy elevados.

Las herramientas de síntesis de alto nivel permiten partir de descripciones funcionales de elevado nivel de abstracción. Sobre los resultados aportados por éstas herramientas se puede aplicar un proceso descendente, a través de los niveles de abstracción, para alcanzar el nivel físico, requiriendo un esfuerzo relativamente bajo comparado con las metodologías de diseño tradicionales. El punto de partida, en general, es una descripción funcional o de comportamiento del sistema a implementar. El modo de describir este comportamiento se basa en lenguajes estándar, denominados de descripción *hardware* o HDL (*Hardware Description Language)* que son interpretables, prácticamente, por la mayoría de las herramientas. Entre los lenguajes de descripción hardware más empleados podemos citar *Verilog*[21] y *VHDL*[134]. El primero, originalmente propietario de *Cadence* y posteriormente hecho publico, ha sido muy utilizado pese a que posee limitaciones importantes en cuanto a sus posibilidades y su original carácter propietario. El segundo es un estándar *IEEE* y dado el carácter abierto desde sus comienzos, ha sido adoptado por muchas más herramientas, lo que ha facilitado un mayor grado de aplicación. Además es un lenguaje más fácil de utilizar y de una flexibilidad excelente.

En este trabajo nos hemos decantado por *VHDL* a la hora de elegir el *HDL* que se empleará en el diseño de los encaminadores. Las principales ventajas de *VHDL* es que soporta la descripción de circuitos digitales desde el nivel de sistema hasta el nivel lógico, permite reutilización de módulos, creación de librerías y facilita el diseño de circuitos independiente de la tecnología a utilizar. Soporta diferentes estilos de descripción que son perfectamente combinables entre si y guarda mucha similitud con lenguajes de programación habituales como ADA.

2.2.2.2 Herramientas de Diseño Electrónico: *Synopsys y Cadence.*

La complejidad de los sistemas electrónicos digitales crece de forma exponencial a medida que la capacidad de integración evoluciona. Esto, junto con los cada vez más cortos ciclos de vida de los productos y las crecientes demandas de fiabilidad, ha forzado a los diseñadores *hardware* a incrementar considerablemente la productividad y calidad en sus diseños. Este hecho ha motivado la aparición de herramientas de diseño electrónico (*EDA*) que permiten reducir considerablemente el ciclo de diseño de los sistemas digitales, incorporando metodologías que permiten lograr los niveles de seguridad y fiabilidad demandados por el mercado. En nuestro caso emplearemos herramientas orientadas al área de los *ASICs*. Es un campo bastante restringido y

el empleo de este tipo de tecnología esta siempre dirigido a un mercado muy especifico y limitado por lo que una característica común de este tipo de herramientas es su elevado coste.

Este tipo de *software* es, por regla general, muy complejo e incorpora herramientas de diseño para la mayoría de las fases de diseño: desde el nivel más alto hasta el más bajo. Un herramienta típica que incluye todas las fases es *Design FrameWorkII* de *Cadence*. En esta *suite* EDA se incorporan desde herramientas de síntesis de alto nivel (RTL) como *Synergy* hasta herramientas de generación y análisis de layouts como *Virtuoso* o *Silicon Ensemble*. La principal baza de este entorno se encuentra en los niveles de abstracción más bajos, que es donde muestra una clara superioridad sobre muchas otras herramientas de diseño electrónico.

Sin embargo, en los niveles de abstracción superiores *Cadence*, muestra una debilidad frente a otras herramientas más evolucionadas. Un caso típico de esta clase de herramientas es la *suite* de diseño ofrecida por *Synopsys.* Incorpora herramientas que abarcan desde el nivel de sistema hasta el nivel lógico. La suite incorpora también herramientas orientadas hacia el co-diseño HW/SW como *Cosaap*, herramientas de síntesis de comportamiento como *Behavioral Compiler* o herramientas de síntesis a nivel RTL como *Design Compiler.*

En esta tesis se han empleado conjuntamente las dos herramientas. En la descripción hardware, simulación funcional de las descripciones, síntesis y extracción de las estructuras lógicas se ha empleado *Synopsys*. En las ocasiones en las que se ha descendido hasta el nivel físico se ha empleado el entorno *DFWII* de *Cadence.* Por problemas de eficacia de los simuladores *VHDL*, en determinados casos también ha sido necesario emplear el simulador *VHDL* de *Cadence* (*Leapfrog)*.

2.2.3 Tecnologías de Implementación *Hardware*.

A la hora de caracterizar correctamente el coste de las propuestas arquitectónicas, es necesario fijar la tecnología de implementación en la que va a estar definitivamente apoyado el diseño.

La mayor parte de los diseños realizados a lo largo del trabajo se basan en la librería tecnológica ECPD07. Esta librería pertenece a la *foundry* ES2/ATMEL en la que se incorporan celdas digitales CMOS con longitudes de canal de 0.7μm y soporta hasta dos niveles de metalización. Obviamente, para su empleo se cuenta con un *kit* de diseño que integra todas las celdas básicas y que es accesible desde las herramientas de diseño [49]. La longitud de canal de esta tecnología puede calificarse, en cierta medida, de obsoleta, dado el estado actual de las tecnologías de implementación. Sin embargo, en la mayor parte del trabajo desarrollado el objetivo es comparar propuestas arquitectónicas para el encaminador en el que el empleo de una tecnología más

evolucionada no es relevante. En estos casos ésta sirve perfectamente como medio para establecer los costes *hardware* y lograr así realizar comparativas justas.

No obstante, existen otras secciones en esta tesis en las que se proponen estructuras complejas para las que el empleo de tecnologías más evolucionadas se hace imprescindible. A la hora de estudiar estas propuestas se ha optado por una tecnología mucho más acorde con el estado actual del arte en este campo. Estas implementaciones se basan en la librería tecnológica MIEC3500. Se trata de una tecnología perteneciente a la *foundry* MIETEC/ALCATEL con 0.35μm de longitud de canal soporta cinco niveles de metalización y dos de polisilicio.

2.2.4 Metodología de Diseño.

El proceso de diseño completo consta de las fases representadas en la Figura 2-4. A continuación pasaremos a detallar todos los pasos seguidos en el estudio de cada uno de los encaminadores analizados.

- El punto de partida es la idea base del encaminador, estableciendo las características claves del mismo: algoritmo de encaminamiento, topología, etc. En este punto es conveniente contar con una idea aproximada del modo de operación del mismo. Esto es posible lograrlo mediante un simulador funcional convencional.
- El siguiente paso es determinar la estructura del *pipeline* y determinar los elementos más importantes del mismo. En este punto se comienza a describir en *VHDL* todos los componentes del encaminador a nivel *RTL*.

  A partir de las descripciones *RTL* de los módulos integrantes del encaminador se procede a su verificación funcional. La tarea se realiza empleando tests específicos para cada uno de los componentes analizados. Esta fase se desarrolla empleando un *front-end* de un simulador *VHDL* que facilite el acceso a todos los puntos de cada módulo para verificar su modo de operación. En nuestro caso, este proceso se ha realizado empleando la herramienta *Vhdldbx* sobre el simulador *VSS* de *Synopsys*. En el hipotético caso de detectarse algún tipo de anomalía, será preciso regresar al paso anterior y quizás rehacer las descripciones.

  Una vez descritos todos los componentes del router, el siguiente paso es integrarlos en una sola unidad. A continuación se construye una red de interconexión, generalmente de reducidas dimensiones, dados los elevados requerimientos de cálculo de los simuladores *VHDL*. En este punto se pasa a simular la red completa con diversos tipos de tráfico sintético hasta probar el correcto funcionamiento del router.

  Una vez verificado el comportamiento correcto del router descrito, el siguiente paso es continuar con el proceso de síntesis *hardware*. Dado que todas las descripciones son a nivel de

transferencia de registros, emplearemos como herramienta de síntesis de alto nivel el *Design Compiler* de *Synopsys*. Generalmente, las herramientas de síntesis soportan la mayor parte del estándar *VHDL* pero siempre existen limitaciones que quizás fuercen la recodificación de algunos módulos del encaminador. Cuando las descripciones VHDL de todos los módulos sean sintetizables, el siguiente paso es elegir la librería tecnológica adecuada y proceder a la compilación y optimización del módulo hasta obtener una implementación a nivel de puertas lógicas. A partir de esta implementación y por medio de reglas aproximadas, la herramienta de síntesis es capaz de extraer las características de tiempo y área de cada módulo. La extracción del *critical path* del módulo se realiza teniendo en cuenta las características precisas de las celdas lógicas empleadas en la implementación lógica y, de forma aproximada, el retraso de las interconexiones. El cálculo se realiza empleando el analizador estático de tiempo *PrimeTime* de *Synopsys*, para cada una de las condiciones de trabajo que establece la librería. Las características de área se extraen también de forma aproximada teniendo en cuenta el área de las celdas empleadas y las interconexiones. Dado el carácter heurístico de las reglas empleadas por estas herramientas a la hora de compilar las descripciones, quizás sea necesario volver a optimizar varias veces cada módulo antes de alcanzar un *pipeline* balanceado y unos tiempos y áreas adecuados para la tecnología empleada. Cabe comentar que el estilo de codificación que se emplee en la descripción *hardware* de los módulos puede ser de vital importancia para lograr unos tiempos y áreas ajustados. Es por ello que en muchas ocasiones es preciso volver a codificar las descripciones VHDL de algún módulo.

- Obtenida una implementación en puertas lógicas del encaminador, el siguiente paso es verificar que la funcionalidad del circuito se mantiene. Como es sabido, los retrasos asociados a las puertas y al conexionado resultantes de la síntesis, puede hacer que una descripción que es correcta desde el punto de vista funcional deje de serlo en la implementación resultante. Este tipo de simulación se denomina *post-síntesis*. De nuevo, en el caso de encontrar alguna dificultad, será preciso rehacer el camino desde el punto adecuado del ciclo de diseño.
- Después de verificar el correcto funcionamiento de la implementación obtenida por la herramienta de síntesis, será preciso descender un nivel de abstracción más. En este punto se pasa a la realización del *Place and Route* de cada módulo del encaminador por separado. En este caso se emplearán herramientas de *auto-layout* como *LAS* de *Cadence.* Aunque no son completamente automáticas, simplifican considerablemente la tarea. A partir de este *P&R* podemos extraer de forma mucho más precisa el retraso de las interconexiones que a partir de la implementación facilitada por la herramienta de síntesis.

- Los retrasos obtenidos del *layout* son tenidos en cuenta a la hora de realizar la simulación *post-layout*. En esta simulación se parte de las descripciones a nivel de puerta lógica obtenidas de la síntesis y se incorporan los retrasos precisos de las interconexiones empleando formatos de intercambio estándares como *SDF*. De nuevo, será necesario retroceder en el proceso de diseño en el caso de encontrar algún problema en el modo de operación del encaminador.
- Cuando el proceso previo ha resultado satisfactorio, el siguiente paso es la extracción eléctrica de los parámetros de *layout*. A partir de estas características y de las de las celdas utilizadas se pasa a la simulación eléctrica y verificación estricta de las reglas de diseño de la *foundry.*
- El ultimo paso es el envio del diseño a fabricación. Una vez fabricado el circuito será preciso verificar su correcto funcionamiento y en el caso de encontrar algún problema quizás sea necesario rehacer todo el camino y volverlo a enviar a fabricación.

Queda claro que el proceso completo puede llegar a ser, pese al empleo de herramientas evolucionadas, extraordinariamente costoso de recorrer totalmente. Fundamentalmente, nuestro interés se centra en establecer los costes *hardware* asociados a las posibles propuestas arquitectónicas a evaluar. Por ello, resulta evidente que si pretendemos descender hasta el último nivel de abstracción en el diseño es más que probable que la mayor parte de nuestro esfuerzo se invierta en el *backtracking* entre los distintos niveles del proceso de diseño. Parece coherente cortar el ciclo de diseño en un punto en el que se considere que el error de los costes asociados al encaminador, extraídos en esta fase, sean pequeños. Desde nuestro punto de vista, el lugar más adecuado en el que se puede parar el diseño es la síntesis de alto nivel. Si descendiésemos hasta el nivel de *layout*, las posibles modificaciones que sería necesario incorporar en los encaminadores serán pequeñas y prácticamente no van a alterar en exceso el coste obtenido previamente.

En lo que respecta a la estimación de costes por parte de la herramienta a partir de la descripción lógica obtenida de la síntesis y las características tecnológicas de la librería empleada, cabe decir que emplea algoritmos cada vez más complejos. Aunque no llegan a la precisión tan elevada como la que se puede alcanzar mediante el análisis eléctrico del *layout*, los resultados son bastante próximos y pueden ser considerados como aceptables.

En resumen: el objetivo es comparar diversas alternativas para los encaminadores de mensajes teniendo en cuenta el coste *hardware* de forma aproximada sin cubrir todo el ciclo de diseño, evitando de esta forma limitar el espacio de diseño a explorar. En al figura siguiente, la zona

más sombreada no ha sido analizada en ningún caso. El caso de la simulación post-síntesis ha sido realizado en algunos casos, observándose que las modificaciones a introducir en las descripciones no eran demasiadas y los requerimientos de cálculo para simular una red eran claramente desorbitados. Esto nos ha conducido a desestimarla en la mayoría de los casos.

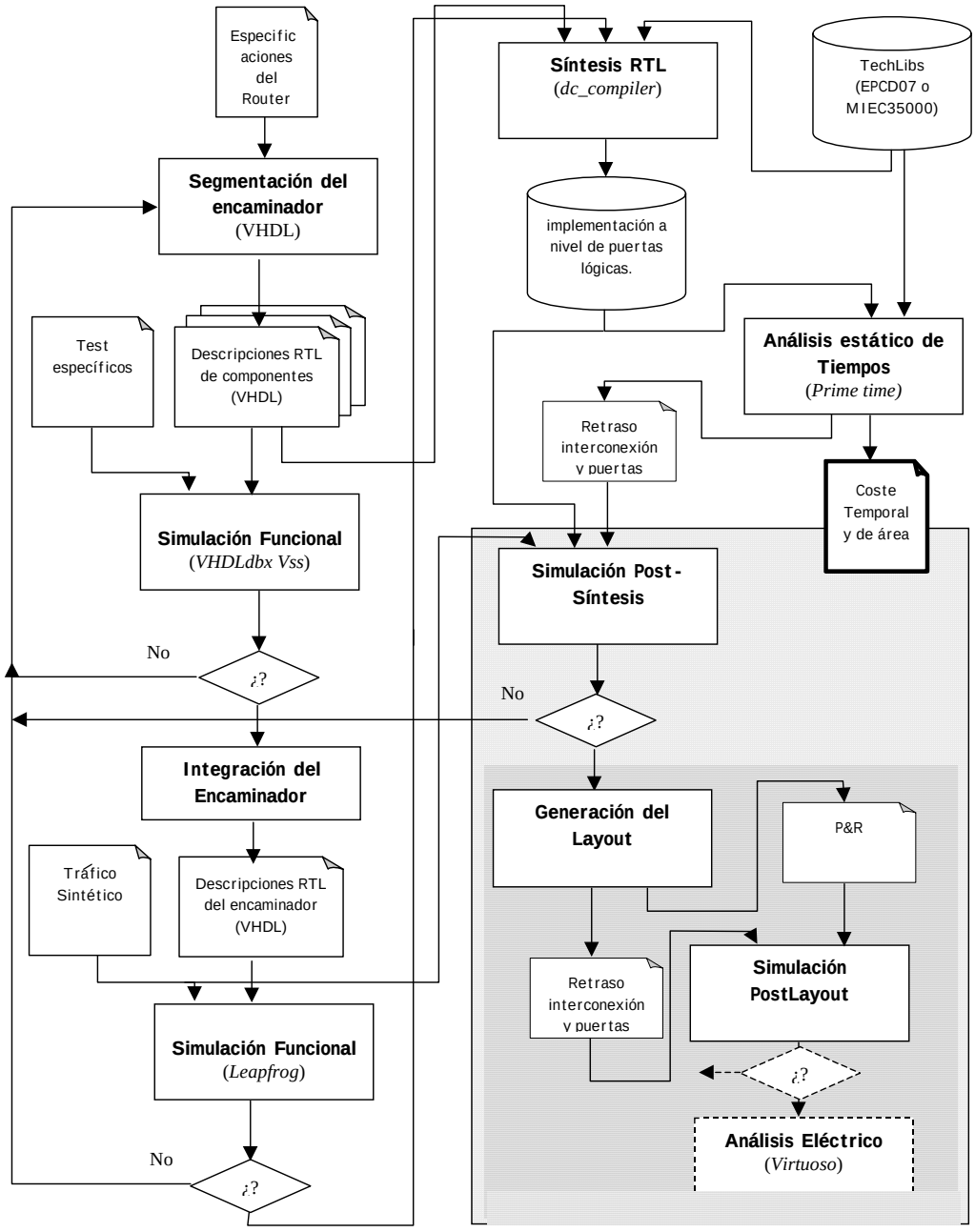

Figura 2-4. Ciclo de diseño completo.

Por último, como es lógico en la mayoría de los casos los encaminadores tienen unidades funcionales comunes entre sí. En estos casos, obviamente, se saca provecho de las características de reutilización que aporta *VHDL* y las herramientas de diseño, de forma que solo ha sido pre-

ciso repetir el ciclo de diseño para los nuevos módulos de cada encaminador, manteniendo los empleados previamente en otros.

2.2.5 Estimación de los Costes *Hardware*.

En el punto anterior se ha expuesto la metodología de diseño seguida a la hora de determinar los costes o implicaciones *hardware* de los encaminadores bajo estudio. Se ha justificado como no parece oportuno recorrer completamente el ciclo de diseño hasta el final, cuando lo que pretendemos es establecer comparativas en los costes *hardware* de distintas alternativas arquitectónicas. Como se ha comentado previamente, no llevar el ciclo de diseño por debajo del nivel de la síntesis lógica puede introducir errores a la hora de establecer los costes. En este punto se pretende determinar, con un circuito de prueba, cuanto puede llegar a ser ese error. Para alcanzar este objetivo, obviamente, ha sido preciso descender hasta el nivel layout para el circuito analizado.

El circuito empleado es la unidad de control de un subcrossbar perteneciente a un encaminador adaptativo. El circuito de control implementa un árbitro tipo *Round Robin* que gestiona las peticiones de paso desde las etapas previas del router. Además, incorpora todo el mecanismo de control del camino de datos del crossbar. En realidad, la unidad de control solo gestiona una de las salidas del crossbar. En la Sección 4.2.1, se puede encontrar una descripción detallada del mismo.

El análisis de la estimación de los costes hardware en cada nivel del circuito ejemplo comienza con una implementación en puertas lógicas que es correcta desde el punto de vista funcional.

La librería tecnológica elegida para realizar esta prueba ha sido *ECD07*. El criterio seguido a la hora de elegir esta librería y este circuito es que por un lado la librería solo soporta dos niveles de metalización y por otro la complejidad del circuito hace que sea uno de los que más interconexionado tiene. Estos dos factores provocarán que el circuito final sea uno en los que más porcentaje del área total del circuito se dedique a interconexión.

A partir de la implementación lógica se ha efectuado el *P&R* del módulo. Como puede apreciarse en la Figura 2-6, no se ha incorporado ningún tipo de *pad* al circuito, ya que se trata tan solo del *core* de una parte del encaminador. Una vez implementado el layout del circuito el siguiente paso es extraer los retrasos característicos de cada línea de interconexión. Empleando un analizador estático de tiempo se ha determinado el *critical path* del circuito. En cuanto al área, como aparece en la figura, se puede calcular fácilmente de forma gráfica.

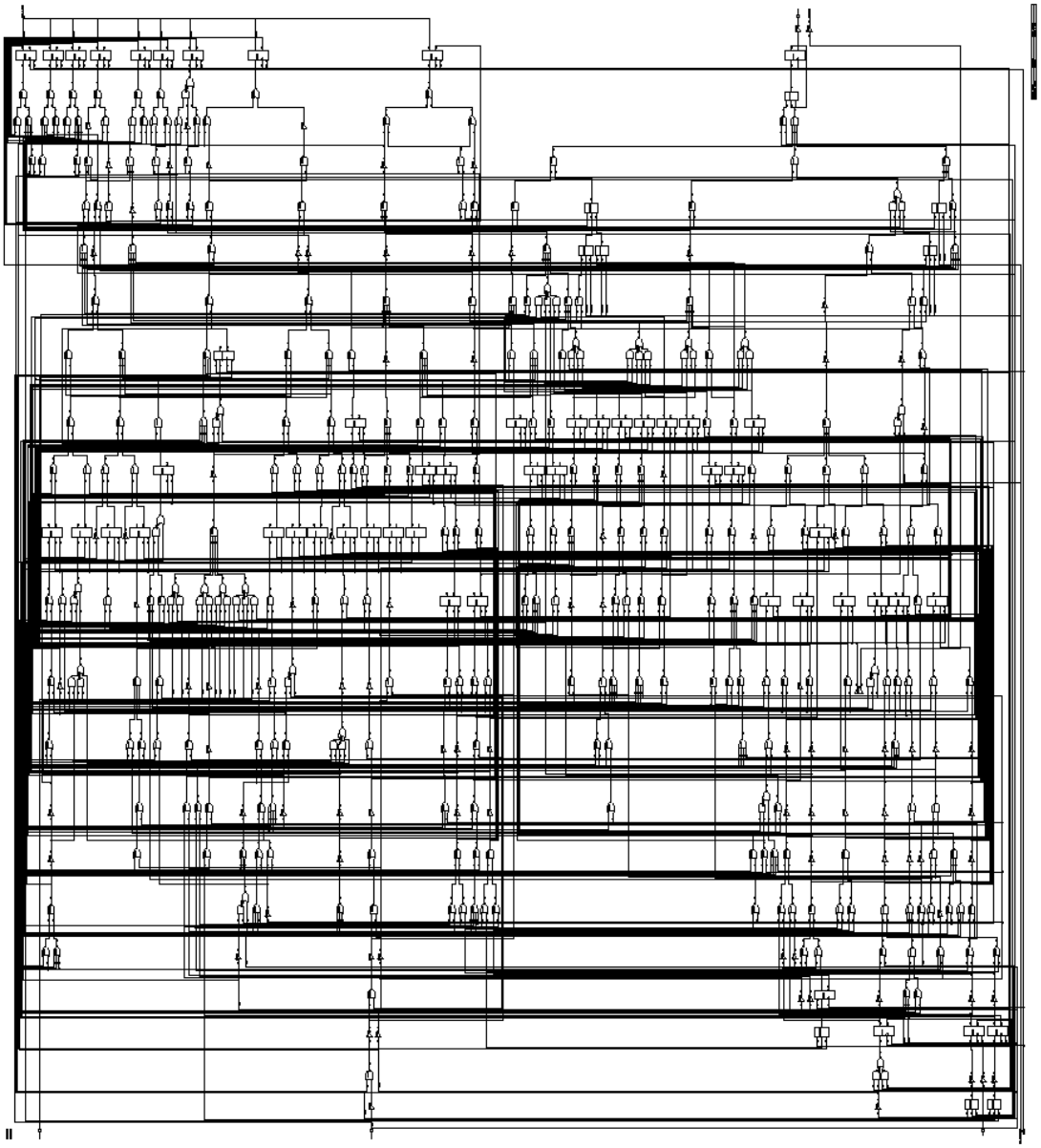

Figura 2-5. Esquemático del circuito bajo estudio.

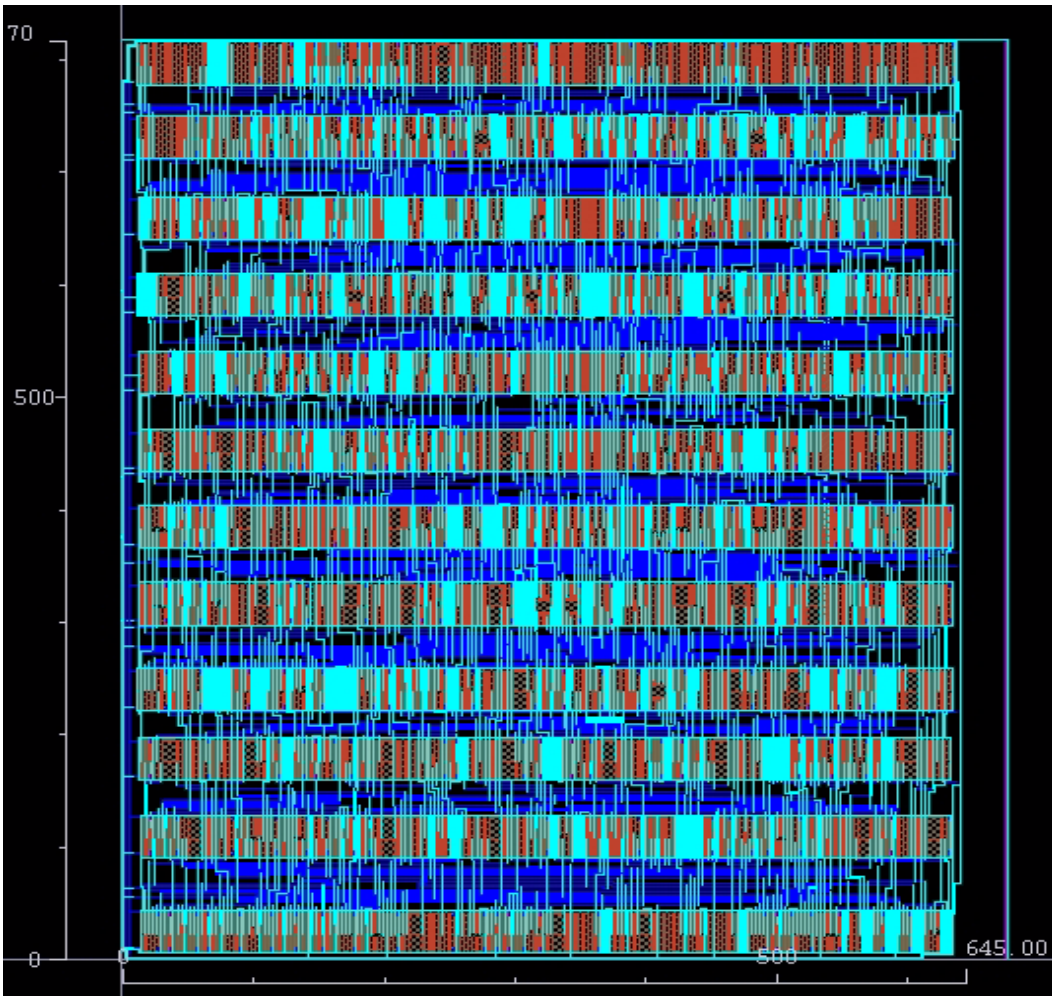

Figura 2-6. Layout del circuito bajo análisis.

Los resultados alcanzados, por cada herramienta, en cada uno de los niveles de abstracción son los siguientes:

	Area (μm^2)	**Critical Path (ns)**		
		MIN	**TYP**	**MAX-IND**
Synopsys (PrimeTime) Nivel Lógico	632003	2.64	5.65	12.33
Cadence (CDC) Nivel Físico	527739	2.51	5.05	10.52
Error (%)	+18%	+5.2%	+11.8%	+17.2%

Tabla 2-1. Estimación de Costes Hardware: desde el nivel lógico al nivel físico.

El camino crítico del circuito bajo análisis es calculado para las tres zonas de operación que establece la tecnología empleada. Estas zonas vienen descritas por:

	Temperatura(ºC)	**Voltaje (V)**
MIN	-55	3.0
TYP	25	3.3
MAX-IND	100	3.6

Tabla 2-2. Condiciones de operación de la librería tecnológica ECPD07.

Es claro que el error cometido por la herramienta de síntesis a la hora de estimar el coste *hardware* es considerable en las condiciones de operación límite del circuito. Es necesario tener en cuenta este hecho a la hora de valorar los datos mostrados en diversas partes del trabajo. Bajo estas circunstancias, en el resto del trabajo analizaremos los encaminadores exclusivamente en la zona típica de operación en lo que respecta al camino crítico. Para este circuito, el error cometido puede considerarse de tolerable, en este caso se encuentra situado aproximadamente en un 11%.

Por otro lado, cabe decir que este circuito, como se ha comentado previamente, se podría considerar como uno de los más desfavorables. La razón está en el hecho de que el porcentaje de área que ocupa las interconexiones es más que considerable (aproximadamente el 41% del área total del circuito). Esto hace que los errores cometidos por la herramienta de síntesis sean mayores, dado que básicamente la fuente del error en el *critical path* se debe a que la herramienta de síntesis ha de estimar el retraso de las interconexiones de forma aproximada, ya que no conoce su disposición final en el *layout*.

Por la misma razón comentada previamente, el error cometido por la herramienta de síntesis a la hora de estimar el área de silicio que ocupa el circuito es incluso mayor que el producido en el camino crítico en las condiciones de trabajo más desfavorables.

Es importante ser conscientes del relativo pequeño error que comete la herramienta de síntesis en condiciones típicas de operación a la hora de sopesar la necesidad de llevar el ciclo de diseño hasta el final. A la vista de los resultados obtenidos para el circuito analizado es correctoconsiderar válidas las estimaciones de coste *hardware* que reporta la herramienta de síntesis. Descender hasta el nivel físico no va a implicar que tengamos que alterar significativamente las posibles conclusiones que saquemos con los resultados previos.

2.3 Evaluación del Rendimiento.

Para determinar de forma cuantitativa el rendimiento de cada red de interconexión, es necesario definir de forma clara y unívoca el escenario empleado. En este trabajo se ha optado por utilizar dos tipos de carga de trabajo. Partiendo de cargas de trabajo sintéticas en las primeras fases del desarrollo de los encaminadores, hasta cargas correspondientes a la ejecución de aplicaciones reales. A continuación pasaremos a establecer cuales han sido las métricas empleadas en cada caso. De la misma forma se definirá, para cada uno de estos dos casos, el banco de pruebas utilizado poniendo especial énfasis, en este análisis previo, en las aplicaciones paralelas.

2.3.1 Cargas de Trabajo Sintéticas.

La evaluación del rendimiento de las redes de interconexión, en primera aproximación, requiere del empleo de cargas de trabajo representativas. En cierta medida, este tipo de estímulos emulan el comportamiento que muestran determinadas aplicaciones reales desde el punto de vista de la red de interconexión. Generalmente, la tendencia es emplear modelos sencillos y fácilmente reproducibles que modelan parcialmente el comportamiento de algunas aplicaciones paralelas de cálculo numérico, en la mayoría de los casos. En su mayor parte, este tipo de aplicaciones hacen uso de operaciones matriciales que pueden ser descompuestas en permutaciones sencillas. Es necesario indicar que, este tipo de modelado, solo cubre lo que se denomina distribución espacial de la aplicación y que depende fundamentalmente de cómo se han distribuido los datos que ha de manejar la aplicación a lo largo de los nodos que componen el sistema. Obviamente, es necesario realizar esta distribución de tal forma que la aplicación no se vea frenada por un posible innecesario acceso a la red de interconexión. Por ello, fundamentalmente, el criterio que prima a la hora de llevar a cabo la distribución de datos es el propio algoritmo en que están basadas las aplicaciones.

Además de la distribución espacial, característica de cada aplicación, tenemos que considerar la distribución temporal. Lograr un modelo sintético válido en este sentido es extremadamente más complejo que en el caso del modelo espacial. Aquí influyen factores más sutiles que el algoritmo en sí. Factores como el tipo de arquitectura sobre la que está siendo ejecutada la aplicación determinan en gran medida su comportamiento temporal.

Teniendo en cuenta estos hechos, a continuación se citarán los aspectos más relevantes del tráfico sintético empleado.

2.3.1.1 Patrón de Destinos.

Como se ha comentado previamente, la distribución espacial de los datos empleados por cada nodo de proceso es fácil de modelar a través de permutaciones origen-destino. Estas permutaciones establecen cómo cada nodo *i* de la red se comunica con el resto. En todos los patrones se excluye la posibilidad de que un nodo se comunique consigo mismo. Es posible que determinadas aplicaciones en un nivel superior empleen comunicaciones de este tipo, sin embargo este tipo de eventos jamás van a llegar a la red, considerándose por lo tanto irrelevantes desde su punto de vista. Atendiendo a estas características podemos clasificar los patrones de tráfico en las siguientes categorías.

- Uniforme

  Es el patrón de destino más empleado en la evaluación de prestaciones de las redes de interconexión. En este caso, no se pretende modelar ningún tipo de aplicación en concreto, sino que intenta determinar cual es el comportamiento de la red en el peor caso promedio, dado que, en mayor o menor medida, las aplicaciones siempre van a tener un porcentaje de tráfico local. De forma simplificada, cada nodo *i* tiene la misma probabilidad de generar un mensaje para cualquier nodo *j* siempre y cuando $i \neq j$ [108].

- No uniforme

  Los patrones de destino no uniformes sí intentan modelar el comportamiento de algunas aplicaciones. En función de este comportamiento se pueden establecer de nuevo dos tipos de tráfico:

  - Tráfico Local

    Intenta modelar el comportamiento de aquellas aplicaciones en que la distribución espacial de las comunicaciones es muy local. Es decir, cuando el algoritmo de la aplicación implica que cada nodo se comunique con mayor probabilidad con sus vecinos más próximos que con los más alejados. Existen diversos métodos para modelar este tráfico desde un punto de vista matemático. El más sencillo es fijar una distribución uniforme por tramos, de tal forma que se fija la probabilidad de que el nodo *i* genere un mensaje dirigido a un nodo *j*, incluido dentro de la esfera de localidad de *i*, con una probabilidad ϕ y que el mensaje sea generado con destino a un nodo, fuera de esa esfera de localidad, con probabilidad $(1-\phi)$ [108]. Otro modo de definir este tráfico es empleando una derivación de una distribución de tipo *gaussiano*.

  - Permutaciones

    Además de los tráficos comentados previamente, es posible caracterizar en cierta medida el comportamiento de determinadas aplicaciones paralelas

numéricas [84]. En estos patrones, cada nodo genera mensajes con el mismo destino. Por lo tanto el grado de ocupación de la red no es uniforme. Entre los tráficos mas importantes podemos citar.

-Bit Reversal

Cada nodo en coordenadas binarias $a_{n-1}, a_{n-2}, \ldots, a_0$ se comunica exclusivamente con el nodo $a_0, a_1, \ldots, a_{n-1}$

-Perfect-Shuffle

Cada nodo en coordenadas binarias $a_{n-1}, a_{n-2}, \ldots, a_0$ se comunica exclusivamente con el nodo $a_{n-2}, \ldots, a_0, a_{n-1}$.

-Matriz Transpuesta

Cada nodo en coordenadas binarias $a_{n-1}, a_{n-2}, \ldots, a_0$ se comunica con el nodo $a_{\frac{n}{2}-1}, \ldots, a_0, a_{n-1}, \ldots, a_{\frac{n}{2}}$.

-Complemento

Cada nodo en coordenadas binarias $a_{n-1}, a_{n-2}, \ldots, a_0$ se comunica exclusivamente con el nodo $\overline{a_{n-1}, a_{n-2}, \ldots, a_0}$.

2.3.1.2 Distribución Temporal.

Este parámetro del modelo pretende describir el comportamiento de distribución temporal del tráfico generado por las aplicaciones reales. Es, sin duda, el más complejo de modelar ya que, como se ha comentado previamente, está fuertemente relacionado con la arquitectura del sistema del que forma parte la red de interconexión. En la mayoría de los estudios se plantean soluciones muy simplificadas.

La primera consiste en suponer que el ritmo de inyección es el mismo para todos los nodos y con una distribución de probabilidad exponencial a lo largo del tiempo. Esto equivale a decir que la probabilidad de que cualquiera de los nodos que forman parte de la red inyecte un mensaje es la misma en cualquier instante de tiempo. Esta probabilidad se suele denominar *carga aplicada.* En la mayoría de los estudios se evalúa el efecto de la carga realizando un barrido desde probabilidades muy bajas hasta probabilidades de inyección que llevan a la red por encima del punto de saturación. El modo de establecer el rendimiento de la red es esperar, en cada uno de estos casos, que la red alcance el estado estacionario y extraer las medidas oportunas en ese estado.

Otra aproximación es la propuesta en [13]. En este caso se intenta modelar la existencia de puntos calientes en la red. Generalmente, el comportamiento de las aplicaciones no se asemeja a una distribución uniforme en la carga aplicada, sino que habrá nodos en los que el ritmo de

inyección será substancialmente superior al resto. Este tipo de fenómenos es usual que aparezcan en los mecanismos de sincronización entre varios procesos o distribución de datos. Por lo tanto, el tráfico generado presenta esencialmente el mismo comportamiento que el anterior pero en éste, existe uno o más nodos que exhiben un ritmo de inyección más alto. Estos nodos juegan el papel de *Hot Spots*. Los parámetros del tráfico vienen determinados por el número de *Hot Spots* que hay en la red, el incremento en la probabilidad de inyección que presentan con respecto al resto de nodos y la distancia a la que se encuentran (en el caso de haber más de uno).

2.3.1.3 Distribución de Longitudes.

El tamaño de los mensajes empleados bajo condiciones de tráfico sintético suele ser fijo. En general es habitual evaluar la influencia de la longitud de los mensajes en el rendimiento de la red de interconexión, realizando para ello varias simulaciones con distintas longitudes de tamaño fijo. Este tipo de tráfico es denominado *Modal*.

El comportamiento de las aplicaciones reales, especialmente en el caso de determinadas arquitecturas, hace que esta aproximación no sea completamente válida. En particular en las arquitecturas DSM [5][85]. Típicamente el tráfico asociado a este tipo de arquitecturas esta constituido por mensajes de petición y mensajes de respuesta. Los mensajes de petición son mensajes de corta longitud que solo incorporan un comando específico. Los mensajes de respuesta contienen los datos a comunicar. En realidad, en la mayoría de los casos, están constituidos por líneas de *cache* más información de la dirección de memoria a la que se refieren, comando, etc... De acuerdo con esto, poseen una longitud más elevada que los de petición. Este hecho se puede tener en cuenta a la hora de modelar la distribución de longitudes de los mensajes. En [76] se modela el comportamiento de este tipo de arquitecturas a través de un tráfico con distribución de longitudes *Bimodal*. Este tráfico está caracterizado por dos longitudes de paquete distintas, denominadas corta y larga. La relación entre las dos longitudes y la probabilidad de generación de un mensaje, para cualquiera de las dos categorías, determina completamente el comportamiento del tráfico.

2.3.2 Métricas y Normalización de los Resultados.

En este punto se introducen las métricas empleadas a la hora de plasmar los resultados alcanzados por la red con las cargas de trabajo de tipo sintético. Bajo este tipo de condiciones, las figuras de mérito fundamentales de la red de interconexión son la latencia y el *throughput*. Ambos parámetros nos darán una idea adecuada del comportamiento de la red para diversos regímenes de carga aplicada y nos permitirán analizar comparativamente cual es el rendimiento alcanzado entre varias redes de interconexión.

Habitualmente, la toma de estas medidas se efectúa cuando la red de interconexión alcanza un régimen estacionario. La red de interconexión se comporta, en cierta medida, como un sistema en el que el parámetro de entrada lo marca el patrón de tráfico con sus características: patrón de destino, distribución de longitudes y distribución temporal. Estos tres parámetros tratan de modelar el comportamiento de las aplicaciones reales y, en general, se describen en términos de distribuciones de probabilidad sencillas. Fundamentalmente, emplear un ritmo fijo de inyección en los nodos a lo largo del tiempo de simulación, permite asegurar que la red alcanzará, en un tiempo finito, un estado cuasi-estacionario donde los parámetros a medir se mantienen prácticamente estables. No obstante, no resulta evidente determinar con precisión cuando se alcanza ese régimen estacionario. Este es un problema de naturaleza fundamentalmente estadística y que esta fuera del campo en el que se desarrolla este trabajo. En nuestro caso se realizarán las medidas pertinentes una vez simulado un número suficientemente alto de ciclos. Esto tiene el inconveniente de que es posible que estemos simulando ciclos de más, pero aseguramos que las medidas obtenidas son correctas.

2.3.2.1 Latencia.

La latencia se define como el tiempo empleado por un mensaje desde el comienzo de su transmisión hasta su recepción. Esta definición es ambigua y en general suele interpretarse de formas muy diversas. Desde nuestro punto de vista nos referiremos a la parte de la latencia causada por la red de interconexión. En general este factor es solo una parte del total ya que éste puede incorporar la parte correspondiente al *overhead software* u otros factores [36]. Esta métrica también es posible definirla para operaciones colectivas, entendiéndola como el tiempo transcurrido desde que la operación comienza en cualquier nodo hasta que el último nodo implicado recibe el último mensaje [45].

A la hora de establecer el retraso que sufren los mensajes al viajar por la red, se pueden considerar varias medidas. De hecho la latencia tiene tres componentes básicas: latencia de red, latencia de inyección y latencia de consumo.

1. Latencia de Red.
   También denominada latencia de transporte y retraso de propagación, representa el retraso del mensaje o paquete desde que el primer *flit* o unidad básica de información entra en la red hasta que alcanza el destino. Es decir, esta componente excluye el tiempo requerido en el proceso de inyección y consumo del mensaje. Esta latencia a su vez se puede descomponer en dos términos:

- •Latencia Base - Es el tiempo requerido por el mensaje, dado un nodo origen, para alcanzar el nodo destino al que se dirige, en ausencia de contención. Depende exclusivamente de las características topológicas de la red, de las características tecnológicas de los encaminadores y del control de flujo empleado. Es sencilla de calcular analíticamente.
- •Latencia de Contención - Es el retraso sufrido por el mensaje debido a la espera para acceder a recursos de la red que están siendo ocupados por otros mensajes. Esta es una componente dinámica que depende fuertemente de las características estructurales de los encaminadores, de los algoritmos de encaminamiento, el tipo de tráfico, etc. Realmente es dificil de calcular de forma teórica y en general es necesario recurrir a simulación para determinarla de modo preciso.

2. Latencia de Inyección.-
   Indica el tiempo de espera del mensaje en las colas de inyección antes de alcanzar la red. Es un parámetro variable con el nivel de carga aplicada. A niveles bajos de carga es prácticamente despreciable frente a la latencia de red. En cambio, a niveles altos de carga, típicamente próximos al punto de saturación, puede llegar a ser más que considerable. De hecho, una vez superado el punto de saturación, esta componente no converge. En gran parte de los estudios relacionados con las redes de interconexión esta componente de la latencia es despreciada, fundamentalmente por la no convergencia. Es necesario señalar que la no convergencia del tiempo de inyección no está relacionado con que la red no alcance el régimen estacionario sino que, simplemente, indica que la red no puede gestionar todo el tráfico ofrecido e implica que en un sistema con colas de inyección limitadas en un tiempo finito comenzaría a perder mensajes. En nuestro caso no se ha despreciado esta parte de la latencia ya que desde un punto de vista superior, esa latencia afecta al rendimiento de las aplicaciones.

3. Latencia de Consumo.-
   También denominado tiempo de *Spooling*, indica cual es el tiempo que pasa desde que el primer *flit* del mensaje alcanza su destino hasta que lo hace el último. Depende de la longitud de los mensajes y de factores externos a la red de interconexión, como puede ser la interface de red o el gestor de comunicaciones del nodo. En el caso de tráfico sintético siempre se supone que el sistema superior es ideal, en el sentido de que si un paquete alcanza su nodo destino, siempre será consumido con éxito en un tiempo finito.

Desde este punto de vista, la latencia individual de los mensajes es irrelevante, salvo quizás la latencia máxima, siendo la latencia promedio de todos los mensajes la verdadera figura de mérito a considerar. Como se ha comentado previamente, es importante que la red haya alcan-

zado un estado estacionario antes de tomar esta medida. Además de la media, en determinados casos resulta interesante observar cual es la desviación estándar dado que muchos sistemas son sensibles a los valores de peor caso de la latencia.

La latencia es medida, obviamente, en unidades de tiempo. Usualmente la unidad temporal que se emplea en sistemas síncronos para cuantificarla son los ciclos de reloj. Esto puede conducir a conclusiones erróneas ya que se están obviando las restricciones tecnológicas de la red, como se ha comentado en la Sección 2.2. Es por ello que, en general, la unidad empleada en este trabajo para su representación serán unidades físicas de tiempo, que para las tecnologías empleadas, su orden de magnitud es de algunos cientos de nanosegundos.

2.3.2.2 Throughput

Representa el caudal de tráfico que puede manejar la red para un nivel de carga determinado. El parámetro realmente importante es el *throughput* máximo. Este representa la mayor cantidad de información que es capaz de manejar la red. El *throughput* puede ser expresado en términos de mensajes entregados por ciclo o por unidad física de tiempo, dependiendo si conocemos o no las dependencias tecnológicas. Sin embargo, el *throughput* depende de parámetros como el tamaño de la red y la longitud de los mensajes, por lo que es habitual normalizar estos resultados en términos de información entregada por unidad de tiempo y nodo. De esta manera es fácil observar cuán distante es el rendimiento máximo alcanzado por la red, ya que sabemos que en una situación ideal a cada nodo llega un *flit* por unidad de tiempo. Existen otras representaciones para expresar tanto la carga ofrecida a la red como la carga aceptada en términos de la bisección de la red. Este tipo de normalización nos parece menos adecuada que la anterior ya que, por ejemplo, la claridad de los resultados al comparar topologías con un número diferente de dimensiones no es correcta.

2.3.2.3 Representaciones Normalizadas

A la hora de representar el rendimiento de una red de interconexión en términos de latencia y *throughput* es habitual recurrir a sistemas de representación estándar y fáciles de interpretar. Como se cita en [45], los formatos más habituales son el BNF (*Burton Normal Form*) y el CNF (*Chaos Normal Form*). La diferencia fundamental entre estos formatos es que el BNF representan en un solo gráfico el tráfico aceptado por la red (*throughput*), la latencia y el tráfico ofrecido. Es un formato compacto, pero parece poco adecuado cuando se pretende representar el tiempo correspondiente a las colas de inyección en la latencia promedio. El formato CNF utiliza gráficos separados para representar el tráfico aceptado frente al tráfico aplicado y la latencia de los

mensajes frente al tráfico aplicado. A continuación se muestran el uso de cada uno de estos dos formatos para una red de 64 nodos con tráfico uniforme.

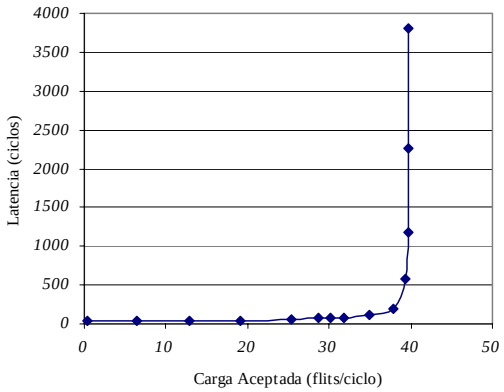

Figura 2-7. Representación BNF de la latencia y el throughput.

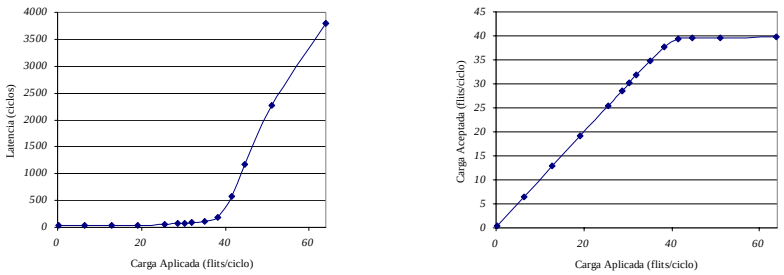

Figura 2-8. Representación del la latencia y el throughput en formato CNF.

En todas las gráficas mostradas se representa la latencia promedio incorporando los tiempos de espera en inyección, por ello su crecimiento para cargas aplicadas por encima del punto de saturación crece considerablemente. En la Figura 2-9 se muestra un ejemplo de lo que representa la incorporación de la componente de la latencia originada por el retraso en las colas de inyección.

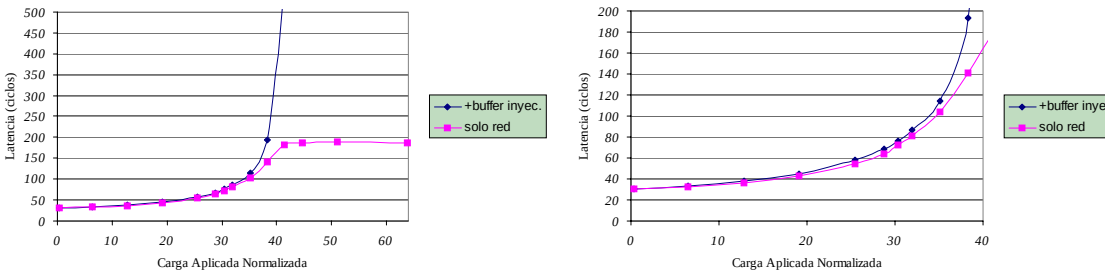

Figura 2-9. (a) Representación CNF de la latencia al incorporar el tiempo de inyección, (b) Detalle.

Dado que en este trabajo se ha logrado determinar, con una precisión adecuada, las implicaciones tecnológicas de los distintos encaminadores analizados, es importante incorporar estos datos a la hora de representar la latencia y el *throughput*. El cambio introducido es sustituir todas las unidades temporales en términos de ciclos de las representaciones previas por unidades físicas de tiempo. Para el tipo de tecnología empleada en los diferentes estudios (Ver Sección 2.2.3) estas unidades siempre van a ser del orden de varios nanosegundos. Tener este hecho en cuenta en la representación que se va a utilizar da lugar a un nuevo tipo de representación que se ha denominado T-CNF. Como ejemplo, en la Figura 2-10 se muestra la representación CNF de dos encaminadores distintos para una red de 64 nodos con tráfico uniforme. En el caso de la red denominada NET1 el tiempo de reloj del encaminador es 3.65 ns. y en el caso de NET2 4.5 ns. Como se puede apreciar, el comportamiento en latencia y *throughput* es muy similar, ofreciendo NET2 unos resultados sensiblemente mejores que NET1. Sin embargo, al pasar a la representación T-CNF se puede observar claramente como el tener en cuenta los tiempos de ciclo de cada una de las redes se invierte la tendencia, pasando a exhibir NET1 un rendimiento notablemente mejor que NET2 tanto en *throughput* como en latencia.

Es importante señalar como las variaciones tecnológicas no sólo afectan a la latencia promedio de los mensajes sino también a la productividad de la red. Por ejemplo, si tenemos un sistema con procesadores a una velocidad fija e incrementamos el periodo de reloj de la red, lo que estamos haciendo es modificar la velocidad relativa de la red con respecto al procesador. Por ello, a un mismo número de ciclos de procesador, el tráfico que puede llegar a manejar la red es mayor.

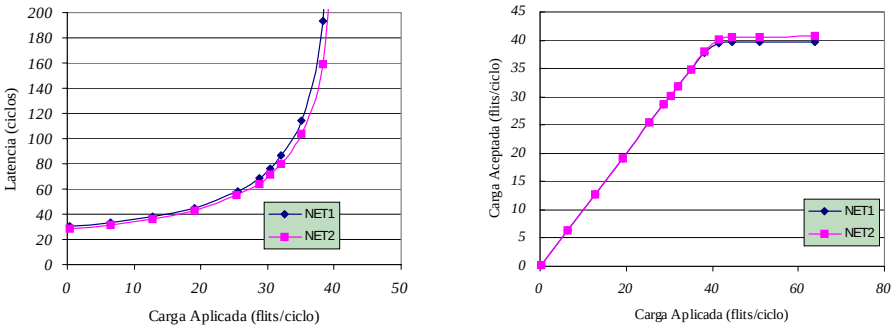

Figura 2-10. Análisis comparativo de dos redes con la representación CNF.

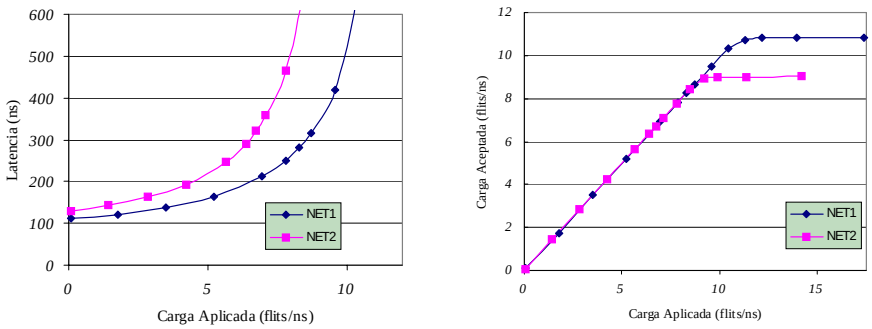

Figura 2-11. Análisis comparativo de dos redes con la representación T-CNF.

2.3.3 Cargas Reales: Impacto del Subsistema de Comunicaciones en una Arquitectura ccNUMA.

El análisis de rendimiento que se emplea habitualmente a la hora de evaluar el comportamiento de las redes de interconexión, se basa en la utilización de tráfico sintético que, como sabemos, solamente permite reproducir de forma aproximada el comportamiento de las aplicaciones reales.

Aquí se empleará una metodología de evaluación que pretende solucionar la problemática asociada a las metodologías basadas en tráfico sintético. Se estudiará cómo es el impacto de las propuestas arquitectónicas a analizar sobre un conjunto significativo de aplicaciones reales. A la hora de plantearnos el sistema de evaluación de la red nos hemos centrado exclusivamente en las arquitecturas tipo ccNUMA con procesadores superescalares o ILP. Un análisis similar puede llevarse a cabo bajo una arquitectura de paso de mensajes, si bien se requiere otro tipo de herramientas.

En este punto se analizará un conjunto de aplicaciones basadas en el paradigma de memoria compartida y se evaluará la sensibilidad que muestran a diversos parámetros de la red de interconexión. Tanto en este análisis previo como a lo largo del trabajo se ha empleado un infraestructura de simulación descrita de forma detallada en el Apéndice A. Una vez conocido el comportamiento de las aplicaciones consideradas, delimitaremos el subconjunto definitivo en base al cual evaluaremos el rendimiento de las diversas propuestas arquitectónicas analizadas en capítulos posteriores.

2.3.3.1 Aplicaciones Consideradas: Banco de pruebas.

Aunque originalmente se consideró la posibilidad de incluir en este estudio algunas aplicaciones desarrolladas por nuestro grupo [133], finalmente se optó por considerar aplicaciones conocidas

y empleadas usualmente en este tipo de estudios. De esta forma, todos los códigos provienen de la *suite* de *benchmarks* SPLASH2 [137].

- FFT

Se trata de un *kernel* de la *suite* SPLASH2. Este *kernel* realiza una FFT compleja 1-D basada en el algoritmo de radio-$\sqrt{n}$ en seis pasos descrito en [10]. Está optimizada para reducir la comunicación entre procesos. El conjunto de datos de entrada, consistente en una lista de *n* complejos dobles, ha de ser transformado en otro conjunto de *n* complejos, denominados raíces únicas o de la unidad. Ambos conjuntos de datos están organizados en matrices de $\sqrt{n} x \sqrt{n}$ elementos, distribuidos de tal manera que a cada proceso se la asigna un conjunto de filas contiguas. De esta manera, cada procesador tendrá en su memoria local una submatriz de tamaño $\sqrt{n}/P x \sqrt{n}$. Las comunicaciones implican realizar tres transposiciones de la matriz previa. Este tipo de operación implica que el patrón de comunicación es *all-to-all*: el procesador *i* envía a cada procesador *j* (*j=1... P* con $i \neq j$), desde la columna $\sqrt{n}\, j/P$ hasta la columna $\sqrt{n}\,(j+1)/P$ de la submatriz almacenada en su memoria local. La transposición del bloque transmitido es realizada por el procesador que mantiene en memoria local los datos, antes de enviarlos al destino. Esta transposición puede ser llevada acabo explotando las características *ILP* del procesador del sistema. En el código original esto no está contemplado de esta forma pero en nuestro caso utilizaremos una versión del código optimizada y que sí lo contempla. Originalmente, la comunicación se realiza de forma escalonada con el fin de evitar posibles *hot-spots* en el acceso al sistema de memoria y por tanto en la red. Esto evita un desbordamiento del subsistema de comunicaciones a costa de alargar el tiempo de ejecución. No hemos intentado suprimir esta limitación con el fin de mantener la estructura original de la aplicación.

Tamaño Problema	Instrucciones (Millones)	Lecturas (Millones)	Escrituras (Millones)	Lecturas Compartidas (Millones)	Escrituras compartidas (Millones)	Número total de Barreras	Número total de Locks
64K puntos	34.79	4.07	2.88	4.05	2.87	6	0

Tabla 2-3. Desglose del número de instrucciones para FFT con el tamaño por defecto (64K complejos) para un caso de 32 procesadores [137].

El número de FLOPS para esta aplicación con esta configuración se sitúa en 6.36 millones. A tenor de los resultados mostrados en [137] este tamaño de problema escala adecuadamente con 64 procesadores para el modelo PRAM, mostrando unas características de balanceo de carga y concurrencia óptimo. Pruebas realizadas con 128 procesos muestran que la aplicación sigue escalando correctamente. De hecho, en determinadas secciones de esta tesis emplearemos esta aplicación sobre 128 procesadores.

Uno de los factores que va a convertir la red de interconexión en un factor determinante es la relación comunicación-cálculo. Es por ello que desde nuestro punto de vista, es interesante conocer si la aplicación va a estresar el subsistema de comunicaciones. En este punto tomaremos en cuenta resultados previos citados en [137]. En ese trabajo se realiza un exhaustivo estudio sobre los efectos del tamaño del problema y el modo en que afectan las características relacionadas con la jerarquía de memoria al comportamiento de la aplicación. En dicho trabajo se dice que la relación comunicación-cálculo crece con el número de procesadores, *P*, y el tamaño del problema, *DS*, de la siguiente manera:

$$CommCalcRatio = \frac{P-1}{P\log DS} \qquad \textbf{[2-2].}$$

En la Figura 2-12 se muestra como evoluciona el tráfico en función del número de procesadores para dos tamaños de la cache de segundo nivel y cómo evoluciona en función del tamaño de las líneas de cache.

Este tipo de información ha de ser tenida en cuenta a la hora de seleccionar la configuración base de la máquina a simular. A priori, la tasa de generación de tráfico hacia y desde nodos remotos es próxima a 0.5 Bytes/FLOP. Asumiendo que se están empleando procesadores del orden de los 500MFLOPS, el tráfico originado total se sitúa entorno a 250Mbytes/seg, lo que nos da una idea de que la red se encontrará situada en zonas de carga media-alta. De cualquier modo, estos datos son promedios y por lo tanto pueden existir fases en la ejecución en las que se ejerza mucha más presión sobre la red. Además, dado el patrón de comunicaciones empleado, la influencia de la latencia de los mensajes será alta, lo que puede hacer que la red de interconexión influya negativamente en el rendimiento de la aplicación.

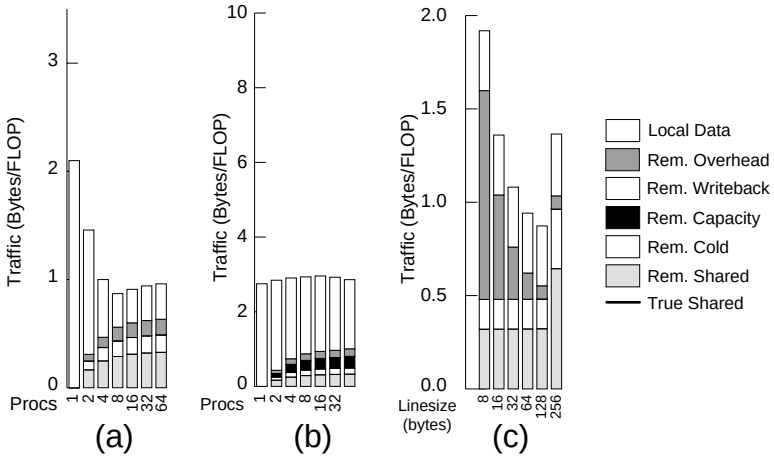

Figura 2-12. Desglose del tráfico originado por FFT en bytes por instrucción para el tamaño por defecto (a.- tamaño L2 =1MB, asociatividad=4 y línea de cache=64 bytes, b.-.- tamaño L2 =8-KB, asociatividad=4 y línea de cache=64 bytes, c.- Variando el tamaño de la línea de cache en L2 para 1 MB y 32 procesadores).

- RADIX

Es un kernel de ordenación de enteros, denominados *keys*, descrito en [15]. El algoritmo es iterativo, realizando una ordenación para cada *r* dígitos, donde *r* es el *radix* (radio) de la iteración y forma parte de los parámetros de entrada. El algoritmo parte de los *b* bits que componen una clave cualquiera y la descompone en un conjunto de $\left\lceil \frac{b}{r} \right\rceil$ claves. El algoritmo procede en $\left\lceil \frac{b}{r} \right\rceil$ iteraciones a ordenar las diferentes claves. Al cabo de estas iteraciones el array con las *n* claves de entrada estará completamente ordenado. Cada fase se compone de 3 partes:

- Cada procesador realiza un histograma con sus *n/p* claves contabilizando el número de claves de cada categoría. Por ejemplo, en la primera iteración es realizado el histograma para los *r* bits más significativos de cada clave. El número de elementos del histograma es 2^r.
- Cuando todos los procesos han pasado la primera parte (este punto se controla mediante barreras de sincronización), se realiza un histograma global con los *p* histogramas locales. A partir del histograma global se realiza el cálculo del prefijo de paralelización, que permitirá determinar cual es el rango dentro del array que será asignado a cada procesador en iteraciones consecutivas.
- A partir del prefijo de paralelización se reordena el array de entrada de tal forma que, a cada procesador, se le pueden asignar todas las claves que en la actual iteración tengan los $\left\lceil \frac{b}{r} \right\rceil$ bits más significativos iguales. Dependiendo de

cómo sean las claves de entrada, el tamaño del problema y el número de procesadores se asignaran las claves al nuevo procesador con los $\left\lceil \frac{b}{r} \right\rceil$ bits más significativos en orden creciente.

Estas fases se repiten hasta recorrer todos los bloques de bits de cada clave, luego es necesario repetir el proceso $\left\lceil \frac{b}{r} \right\rceil$ veces. Bajo estas circunstancias, es claro que el patrón de comunicaciones entre procesos es uniforme, ya que la generación de claves es aleatoria. Esto es debido a que el proceso de comunicación más importante, que esta localizado en la fase tres (fase de permutación) de cada iteración, depende exclusivamente de en qué orden se encontraban previamente las claves en el array de entrada y éstas son siempre generadas en un orden aleatorio.

Un desglose de los parámetros más significativos de la aplicación para el tamaño de problema por defecto aparecen e la Tabla 2-4.

Tamaño Problema	Instrucciones (Millones)	Lecturas (Millones)	Escrituras (Millones)	Lecturas Compartidas (Millones)	Escrituras Compartidas (Millones)	Número total de Barreras	Número total de Pausas
1M Enteros radio 1024	50.99	12.06	7.03	12.06	7.03	10	124*

Tabla 2-4. Desglose del número de instrucciones para Radix con el tamaño por defecto para el caso de 32 procesadores. (*Las pausas son puntos de sincronización basados en *flags*)

En cuanto a la relación comunicación-cálculo de la aplicación, en términos de bytes de tráfico generados por instrucción (no se puede expresar en términos de FLOPS porque todas las operaciones implicadas son sobre enteros) de acuerdo con [137]es de la forma:

$$CommCalcRatio = \frac{(P-1)}{P} \qquad \textbf{[2-3].}$$

En la Figura 2-12 se muestran los resultados de tráfico para diversos casos con el modelo PRAM.

:

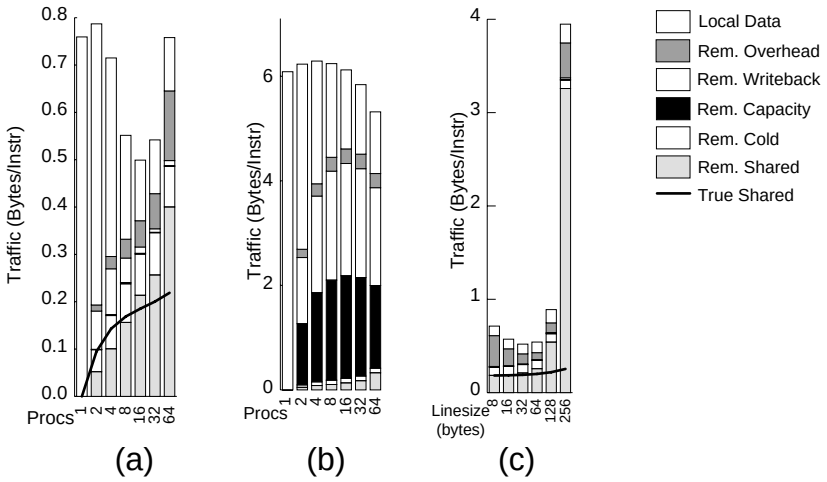

Figura 2-13. Desglose del tráfico originado por Radix en bytes por instrucción para el tamaño por defecto(a.- tamaño L2 =1MB, asociatividad=4 y línea de cache=64 bytes, b.- .- tamaño L2 =8-KB, asociatividad=4 y línea de cache=64 bytes, c.- Variando el tamaño de la línea de cache en L2 para 1 MB y 32 procesadores).

Estos datos indican que la relación comunicación-cálculo es alta, por lo que previsiblemente la demanda de esta aplicación en ancho de banda será tambien elevada. Además, dado que el patrón de tráfico es uniforme, la distancia media recorrida por los mensajes en la red será considerable, lo que implicará que la aplicación sea también sensible a la latencia de los mensajes en las zonas bajas de carga. De cualquier forma, una parte significativa del tráfico de sincronización esta muy localizada (la mayor parte de las pausas son locales).

- LU

Esta aplicación implementa un algoritmo de factorización que permite convertir una matriz densa *A* en el producto de dos matrices *L* y *U*, que son triangulares, inferior y superior respectivamente. El algoritmo de factorización es similar a una eliminación *gaussiana*. La factorización se realiza empleando un algoritmo basado en bloques. La matriz *nxn* a factorizar se divide en bloques *BxB* donde *B* es tomado de los parámetros de entrada de la aplicación. Las operaciones sobre la matriz (como inversión y producto) son aplicadas independientemente sobre cada uno de los bloques. La principal ventaja del plantear el algoritmo desde este punto de vista, es que permite una alta reusabilidad: antes de avanzar al siguiente bloque se realizan todos los cálculos necesarios con el bloque actual.

El algoritmo paralelo se basa en descomponer la matriz a factorizar en forma de teselas. Cada procesador es el encargado de factorizar los bloques incluidos dentro de la tesela asignada y coopera en el cálculo de los bloques restantes actualizando los bloques de *A* incluidos en su tesela a medida que se calculan los bloques de los procesadores previos. Como se puede apreciar

en la Figura 2-14, para el caso de 16 procesadores, el patrón del algoritmo comienza siendo uno a todos (cuando se comienza, el procesador encargado de factorizar la primera tesela de la diagonal comienza), y la tasa de tráfico va decreciendo a medida que avanza el proceso de factorización, hasta llegar al procesador 16. En la fase de factorización de cada bloque se realiza una operación de inversión, multiplicación y transposición. Como se puede intuir, la relación comunicación-cálculo depende del tamaño del bloque y de hecho, es un parámetro crítico de cara a la obtención de un rendimiento adecuado fijado el tamaño del problema y el número de procesadores. Por otro lado, como se puede apreciar en la figura, esta aplicación será propensa a la formación de puntos calientes de carga en los procesadores que mantengan en la memoria local los bloques de la diagonal de la matriz.

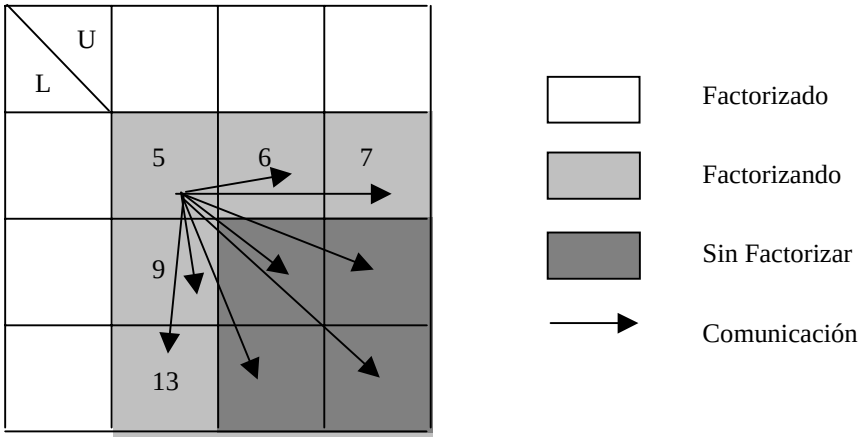

Figura 2-14. Patrón de comunicación del algoritmo LU.

Los parámetros más significativos para esta aplicación, con el tamaño de problema por defecto, aparecen en la Tabla 2-5. Para este caso, el número total de operaciones en punto flotante es de, aproximadamente, 92 millones.

Tamaño Problema	Instrucciones (Millones)	Lecturas (Millones)	Escrituras (Millones)	Lecturas Compartidas (Millones)	Escrituras compartidas (Millones)	Número total de Barreras	Número total de Locks
Matriz 512^2 Bloques 16^2	494.05	104.00	48.00	93.20	44.74	66	0

Tabla 2-5. Desglose del número de instrucciones para LU con el tamaño por defecto para el caso de 32 procesadores.

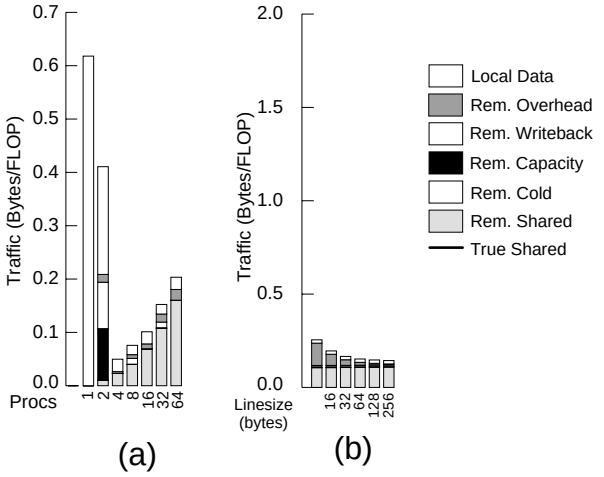

Figura 2-15. Desglose del tráfico originado por LU en bytes por instrucción para el problema por defecto(a.- tamaño L2 =1MB, asociatividad=4 y línea de cache=64 bytes, b.- Variando el tamaño de la línea de cache en L2 para 1 MB y 32 procesadores).

La aplicación escala mejor que *Radix* pero peor que *FFT*. Esto es debido al elevado número de barreras que son necesarias. Estas barreras introducen hasta un 25% de overhead.

En cuanto a la relación comunicación-cálculo, evoluciona como:

$$CommCalcRatio = \frac{\sqrt{P}}{\sqrt{DS}} \qquad \textbf{[2-4].}$$

En la Figura 2-15 aparecen algunos datos que nos pueden dar una idea aproximada de los requerimientos de ancho de banda que va a exigir la aplicación al subsistema de comunicación.

A la vista de estos resultados parece adecuado decir que LU va a ejercer una presión mucho más baja sobre la red que *Radix* o *FFT*. Sin embargo, dado el patrón de comunicaciones empleado, la distancia media va a ser mas elevada que en el resto de aplicaciones, lo que hará que sea substancialmente más crítica la latencia de los mensajes. Ademas será interesante saber cómo afecta la formación de puntos calientes en la red de interconexión.

- WATER-NSQUARED

Es una versión mejorada de la aplicación Water de la *suite* SPLASH-1 [118]. La aplicación calcula las fuerzas y potenciales que aparecen en un sistema de moléculas de agua en estado líquido. La simulación es realizada durante un número de iteraciones temporales que el usuario puede configurar o realizar la simulación hasta alcanzar el estado estacionario. Las principales

diferencias con la aplicación previa es la manera de implementar las regiones de exclusión a la hora de actualizar los parámetros de las moléculas.

La principal estructura de datos de la aplicación es un *array* de registros que mantiene el estado de los parámetros más significativos de cada molécula (la posición y sus 6 primeras derivadas temporales). En cada iteración temporal se calculan las interacciones de los átomos de cada molécula, así como las interacciones con las demás moléculas. Para cada molécula, el procesador asignado calcula las interacciones con las moléculas posteriores y dado que las fuerzas son simétricas, no es necesario establecer las fuerzas con las moléculas anteriores que, en realidad, serán calculadas por procesadores previos.

Tamaño Problema	Instrucciones (Millones)	Lecturas (Millones)	Escrituras (Millones)	Lecturas Compartidas (Millones)	Escrituras compartidas (Millones)	Número total de Barreras	Número total de Locks
512 molec. 3 iteracciones	460.52	81.27	35.25	69.07	26.60	10	17728

Tabla 2-6. Desglose del número de instrucciones para Water con el tamaño por defecto para el caso de 32 procesadores.

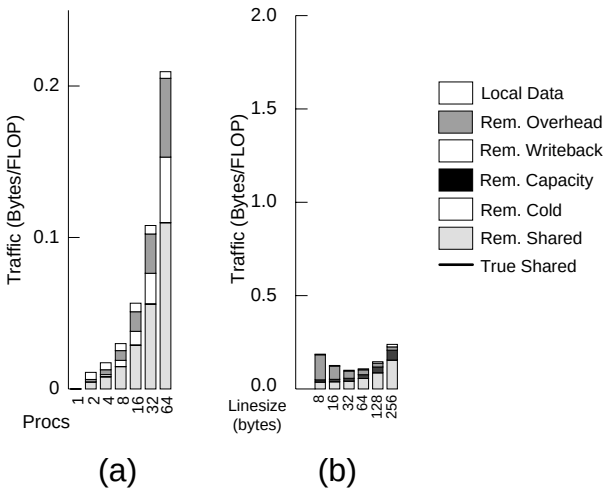

Figura 2-16. Desglose del tráfico originado por Water-Nsq en bytes por instrucción en punto flotante para el problema por defecto(a.- tamaño L2 =1MB, asociatividad=4 y línea de cache=64 bytes, b.- Variando el tamaño de la línea de cache en L2 para 1 MB y 32 procesadores).

Para este tamaño de problema el número de operaciones en punto flotante es de aproximadamente 98.15 millones. Los resultados mostrados en [137] indican que el grado de concurrencia es bastante alto, obteniendo *speedup* considerables.

En cuanto a la relación comunicación-cálculo estimada, se aproxima a:

$$CommCalcRatio = \frac{P}{DS} \quad \textbf{[2-5].}$$

Los resultados PRAM para el problema por defecto se muestran en la Figura 2-16. Como se puede apreciar, la tasa de tráfico generada para este modelo es relativamente baja cuando el número de procesadores del sistema es reducido. Se incrementa sensiblemente a medida que aumenta el número de procesadores, pero no llega nunca a los valores alcanzados por el resto de las aplicaciones, luego parece claro que no van a exigir elevada capacidad en ancho de banda.

2.3.3.2 Configuración del Sistema Evaluado y Metodología Empleada.

RSIM permite configurar un amplio abanico de características del procesador y de la jerarquía de memoria. Frente a estas posibilidades ofrecidas por el simulador nos hemos centrado en establecer unas características fijas para el procesador y la jerarquía de memoria en todos los experimentos a realizar (salvo la red de interconexión). En cuanto al procesador, en la Tabla 2-7 se muestran los parámetros mas significativos. De acuerdo con [97], estos parámetros se aproximan bastante a los del *MIPS R10000*, salvo en la velocidad de reloj que está más de acuerdo con los procesadores actuales.

Velocidad de Reloj	650 MHz
Predictor de Saltos	Basado en históricos de 2-bits, BHT 512 entradas, 4 niveles de profundidad
Unidades Funcionales	2 ALU, 2 FPU, 2 ADU todas con un ciclo de latencia.
Máximo de instrucciones por ciclo	4
Tamaño ventana de instrucciones	64 entradas
Tamaño cola acceso a memoria	32

Tabla 2-7. Principales Parámetros del Procesador.

En cuanto a la jerarquía de memoria se han modificado los tiempos de acceso a memoria y a la cache de segundo nivel con respecto a los parámetros por defecto, intentando que sus características sean más próximas a las empleadas actualmente.

L1	16 KB, write-back, mapeo directo, tamaño de línea 64 bytes, 2 puertos, 8 MSHRs, 1 ciclo de tiempo de acceso en Hit.
L2	64 KB, write-back, asociativa de 4 vías, tamaño de linea 64 bytes, 1 puerto, 8 MSHRs, 8 ciclos de tiempo de acceso en Hit.
Bus	200 MHz, 32 bytes, Multiplexado
Memoria	4-Entrelazada, 18 ciclos de tiempo de acceso.
Directorio	Acceso mínimo en 3 ciclos, Creación de peticiones 8/4 (inicial/consecutivos) ciclos, Tamaño de los comandos 16 bytes, Estructura con mapeo completo
Coherencia	Protocolo de coherencia basado en invalidación con 4 estados (MESI)
Consistencia	Relaxed Consistency (RC)

Tabla 2-8. Principales Parámetros de la Jerarquía de Memoria.

En cuanto a el sistema de evitación del *fetch deadlock* [86] causado por el desbordamiento de los puertos de consumo de las interfaces de red, se ha empleado redes aisladas de peticiones y datos de forma similar a como se planteo en [85].

2.3.3.3 Línea Base de Rendimiento[1].

Una vez descritas tanto las aplicaciones que componen el banco de pruebas como la arquitectura de los nodos de proceso y su jerarquía de memoria, estableceremos cual es la máxima ganancia a obtener variando el subsistema de comunicaciones sobre las aplicaciones propuestas. Para conocer este dato se ha utilizado una red ideal. Esta red se ha modelado empleando un crossbar en el que los enlaces tienen un ancho en bytes igual al tamaño de la línea de cache empleada, más el tamaño de los comandos, es decir el mayor paquete que puede circular por las dos redes es de tan solo un *phit* de longitud. Además se ha planteado de tal forma que la longitud del pipeline del crossbar posea una sola etapa, fijando su frecuencia de reloj igual que la del procesador. Por lo tanto, la contribución de la red a la latencia de las referencias remotas será tan solo un solo ciclo de procesador. Este dato, comparado con las características del resto de elementos de la jerarquía de memoria, hace que el coste temporal introducido por la red de interconexión sea prácticamente despreciable. Los datos de ejecución, en número de ciclos de CPU, resultantes para cada una de las aplicaciones serán empleados posteriormente para normalizar los resultados obtenidos a la hora de emplear arquitecturas de red más realistas. De esta forma, podremos

1. Todos los resultados que se muestran en este punto y posteriores, únicamente incluyen tiempo de cálculo, excluyendo las fases de inicialización y finalización de las aplicaciones en este análisis.

hacer una análisis comparativo entre las diferentes redes evaluadas, así como realizar una interpretación de los resultados en base al mejor caso posible. Además, esta estrategia nos permitirá determinar cuales son las aplicaciones poco o nada sensibles a las características de la red y por tanto podremos restringir el banco de pruebas original a solo aquellas aplicaciones en las que el papel de la red sea relevante.

Una vez ejecutadas cada una de las aplicaciones sobre esta configuración de red (obviamente en los resultados correspondientes a las ejecuciones con un solo procesador, la red no aparece en la jerarquía de memoria, es decir no existen referencias remotas), los resultados mas significativos obtenidos, en lo que respecta a las referencias lanzadas en cada una de las partes de la jerarquía de memoria, aparecem en las Tablas 2-9, 2-10 y 2-11. Notar que estos resultados no se tienen porque asemejar a los mostrados en tablas precedentes, dado que son referencias reales y no las correspondientes al modelo idealizado PRAM. Por ejemplo, la lectura de un dato remoto no tiene porque implicar necesariamente una referencia remota, dado que el dato podría estar cacheado previamente y en estado *clean*.

	Radix	**FFT**	**LU**	**Water-nsq**
L1 Hits	15.25 (94.84%)	3.99 (97.18%)	64.6 (88.5%)	10.8 (86.4%)
L2 Hits	0.53 (2.79%)	0.017 (0.5%)	9.9 (10.7%)	0.57 (4.5%)
Local Ref.	0.45 (2.37%)	0.098 (2.32%)	0.76 (0.7%)	1.13 (9.0%)
Remote Ref	0 (0%)	0 (0%)	0 (0%)	0 (0%)
Total Ref.	18.94	4.10	75.35	12.5

Tabla 2-9. Número de millones de referencias con un único procesador.

	Radix	**FFT**	**LU**	**Water-nsq**
L1 Hits	18.4 (95.4%)	3.72 (95.7%)	86.92 (88.71%)	14.6 (96.2%)
L2 Hits	0.55 (2.8%)	0.068 (1.76%)	9.8 (10.40%)	0.57 (3.74%)
Local Ref.	0.124 (0.64%)	0.49 (1.27%)	0.049 (0.05%)	0 (0%)
Remote Ref	0.123 (0.64%)	0.47 (1.22%)	0.77 (0.82%)	0.202 (0.13%)
Total Ref.	19.27	3.8	94.60	15.23

Tabla 2-10. Número de millones de referencias con 16 procesadores en un Crossbar.

	Radix	FFT	LU	Water-nsq
L1 Hits	32.97 (97.5%)	4.37 (96.7%)	121.4 (91.9%)	22.08 (96.93%)
L2 Hits	0.60 (1.7%)	0.045 (1.01%)	10.3 (7.83%)	0.62 (2.72%)
Local Ref.	0.061 (0.17%)	0.049 (1.08%)	0.053 (0.003%)	0 (0%)
Remote Ref	0.193 (0.5%)	0.054 (1.19%)	0.36 (0.27%)	0.76 (0.33%)
Total Ref.	33.82	4.52	132.21	22.8

Tabla 2-11. Número de millones de referencias con 64 procesadores en un Crossbar.

Por otro lado, en la Tabla 2-12 aparecen los tiempos de ejecución en número de ciclos del procesador para las diferentes aplicaciones y número de procesadores. En principio, el estudio se restringe al caso de 16 y 64 procesadores. A tenor de estos datos se obtienen las curvas de *speedup* de la Figura 2-17.

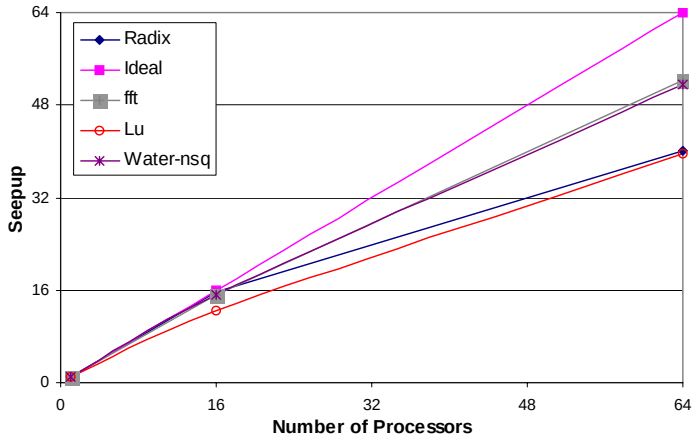

Figura 2-17. Curvas de speedup para cada una de las aplicaciones.

	Radix	FFT	Lu	Water-nsq
1 Procesador	55.6	12.7	173	213
16 Procesadores	3.56	0.832	14	14.7
64 Procesadores	1.29	0.243	4.37	4.14

Tabla 2-12. Tiempo de ejecución sobre un Crossbar (en millones de ciclos de CPU).

Por último, en las Figuras 2-18 y 2-19 aparecen cada uno de estos tiempos desglosados en el porcentaje de acceso a memoria, sincronización y ocupación para 16 y 64 procesadores. Los resultados se muestran normalizando cada uno de los tiempos de ejecución a los que implicaría

un speedup ideal, por lo tanto, el nivel 1 de la escala vertical indica que la aplicación logra alcanzar un *speedup* lineal. Niveles por debajo de 1 indican *speedup* supralineales y por encima infralineales. Se observa como el porcentaje de tiempo de ejecución que consumen las sincronizaciones se mantiene incluso para el caso de 64 procesadores, lo que es coherente con el hecho de poseer una red de interconexión con unas características ideales. Como se podrá observar en apartados posteriores, esta fracción del tiempo de ejecución, como el del tiempo consumido en la lectura y escritura (debido fundamentalmente al incremento en el coste de los accesos remotos) se incrementa notablemente al emplear redes de interconexión más realistas.

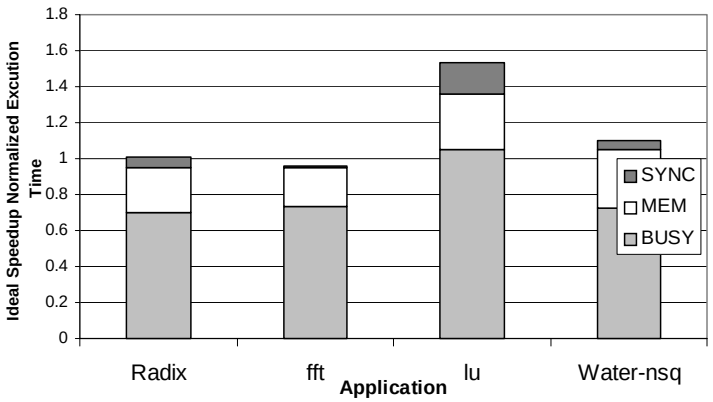

Figura 2-18. Descomposición del tiempo de ejecución para cada una de las aplicaciones con 16 procesadores.

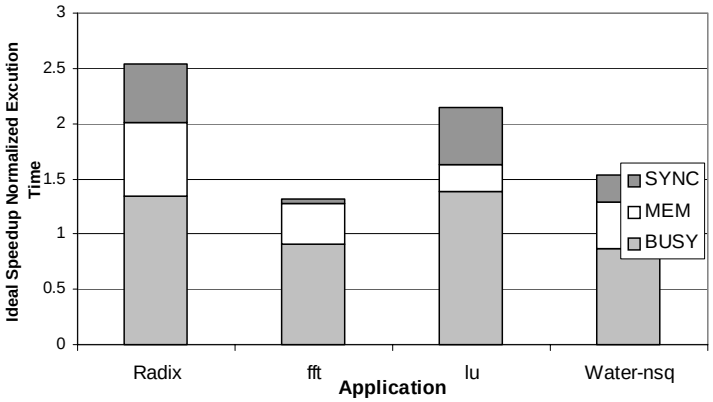

Figura 2-19. Descomposición del tiempo de ejecución para cada una de las aplicaciones con 64 procesadores.

2.3.3.4 Parámetros más significativos de las aplicaciones. Distancia recorrida por los mensajes y carga aplicada sobre la red.

En este punto pasaremos a analizar de forma cualitativa cuales son los parámetros más significativos de cada aplicación desde el punto de vista de la red de interconexión. A tal efecto partiremos del sistema comentado previamente y emplearemos una malla y un toro 2-D con 16 y 64 procesadores. En principio, tanto uno como otro contaran con enlaces de 64 bits y un tiempo de paso de 4 ciclos siendo su ciclo de reloj el doble que el del procesador. En las topologías manejadas se emplearán redes físicamente aisladas de peticiones y datos de cara a la evitación de un *fetch deadlock* como consecuencia de la capacidad limitada de los bufferes de consumo.

Es necesario ser conscientes de que las características temporales y de productividad de la red pueden hacer oscilar sensiblemente los datos que se citan, dadas las características del protocolo de coherencia y el tipo de consistencia empleada en el sistema emulado, si bien los resultados nos van a dar una idea bastante aproximada del comportamiento de cada aplicación desde el punto de vista de la red.

En primer lugar cabe citar las características de tráfico generado. Para obtener éstas, se toma en consideración cual es el número de mensajes generados por cada una de las interfaces de red para cada caso. Estos datos son desglosados en los correspondientes a la red de peticiones y la red de respuestas y se representan los correspondientes a cada una de las redes (toro 2-D) en la tabla siguiente. Los resultados correspondientes a la malla no se muestran por ser muy similares.

		Tráfico Peticiones	**Tráfico Datos**		**Total**	
			Datos	**cohe.**	**Nº Msgs**	**Phits/cyc.**
Radix	4x4	367188	413870	88340	869398	0.94
	8x8	977608	731554	382846	2092008	3.51
Fft	4x4	77320	46435	30987	154742	0.73
	8x8	82790	49836	33342	165968	1.31
Lu	4x4	792168	1062641	20481	1875290	0.66
	8x8	373433	411362	15634	800429	0.88
Water-nsq	4x4	67240	31653	37903	136796	0.0126
	8x8	274553	45545	130440	550539	0.0899

Tabla 2-13. Número de paquetes manejados por cada una de las redes.

A la hora de interpretar los datos de la tabla previa es necesario ser conscientes que representan el comportamiento global de las aplicaciones. Por lo tanto, es más que probable que existan

fases de la ejecución en las que el número mensajes generados por ciclo de procesador sea substancialmente más elevado que los que aparecen en la tabla. Sin embargo, este dato si que nos da una idea comparativa de cual es el comportamiento de cada aplicación en lo que respecta a tráfico generado.

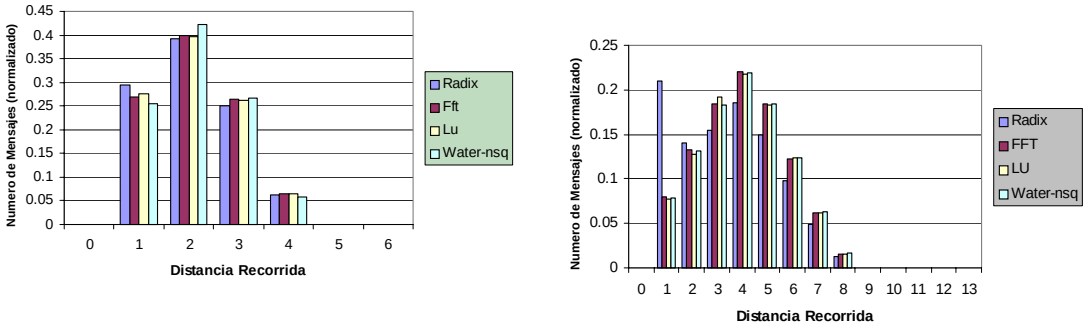

Figura 2-20. Porcentaje de mensajes vs. distancia recorrida en un toro 4x4 y 8x8.

Por otro lado en cuanto a la distribución de distancias recorridas por los mensajes, en la Figura 2-20 aparecen los datos para cada una de las aplicaciones en el toro y en la Figura 2-21 los datos correspondientes a una malla.

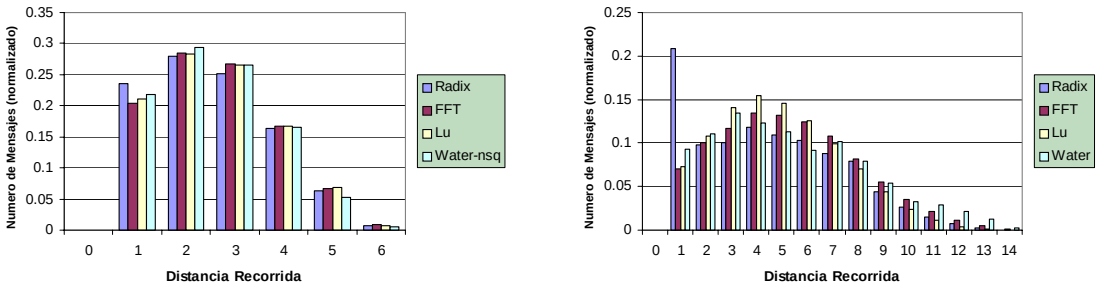

Figura 2-21. Porcentaje de mensajes vs. distancia recorrida en una malla 4x4 y 8x8.

Estos resultados muestran como en todas las aplicaciones en las redes en malla, aproximadamente un 15% mensajes sobrepasan el diámetro topológico de las redes toroidales. Este número de mensajes es mayor para el caso de FFT y LU, lo que puede hacer que las diferencias porcentuales en tiempo de ejecución sean mayores para estas dos aplicaciones entre correr sobre redes toroidales y redes en malla. De la misma manera, este comportamiento se puede cuantificar teniendo en cuenta la Tabla 2-14. En ella aparece la distancia promedio recorrida por cada uno

de los mensajes, cada aplicación y red. Como se puede apreciar, la distancia promedio es mayor para FFT y LU, aproximándose en cada caso a la distancia media topológica de cada red.

	Toro 4x4 ~2	**Toro 8x8 ~4**	**Malla 4x4 ~2.67**	**Malla 8x8 ~5.34**
Radix	2.08	3.47	2.56	4.55
FFT	2.13	3.98	2.63	5.29
LU	2.12	4.01	2.62	4.88
Water-nsq	2.12	3.99	2.56	5.27

Tabla 2-14. Distancia media recorrida por los mensajes en cada aplicación.

A la hora de interpretar estos resultados es necesario ser conscientes que no muestran nada acerca de cómo es el patrón de destino, pero nos permitirán tener una idea de que todas ellas van a ser relativamente sensibles, en principio, a la características topológicas de la red de interconexión.

Por otro lado, estos datos indican el comportamiento de cada aplicación en lo que respecta al número de mensajes que recorren cada distancia, pero es importante señalar, especialmente para la red de respuestas, la distribución de los mensajes correspondientes a datos e invalidaciones. Como se puede apreciar en la Tabla 2-13, tanto para el caso de 16 procesadores como de 64, *Water-nquared* exhibe un comportamiento cuanto menos extraño: en ambos casos el número de mensajes de datos que circula por la red de respuestas es muy pequeño en comparación con el número de mensajes cortos (típicamente invalidaciones). En la Tabla 2-6 se puede observar como el elevado número de sincronizaciones de esta aplicación es la causa de este comportamiento.

2.3.4 Sensibilidad al *Throughput* y a la Latencia. Análisis Inicial.

Existe una cuestión ampliamente debatida en el estudio de las redes de interconexión que podría plantearse como: ¿cuál es la figura de mérito de la red de interconexión que mayor impacto tiene en las aplicaciones reales?: ¿El *throughput* o la latencia?. En este apartado analizaremos esta pregunta para el caso de las aplicaciones descritas en puntos anteriores y para la arquitectura que nos ocupa. Para intentar responder a esta cuestión en el contexto de las máquinas ccNUMA, seleccionaremos un número de redes que nos permitirán observar el comportamiento de cada una de las aplicaciones al emplear cada una de las alternativas.

2.3.4.1 Sensibilidad a la Latencia.

La latencia promedio de la red es dependiente de varios parámetros que en modo alguno nos podemos permitir el analizar completamente. Nos centraremos en analizar como influyen, por un lado, las características topológicas que posee la red de interconexión empleada y por otro lado, las características de tiempo de paso de los encaminadores. Para intentar aislar los efectos causados por la latencia y no incorporar efectos colaterales causados por un mayor ancho de banda total, en el último punto nos centraremos en variar únicamente el número de ciclos de paso de los encaminadores, sin afectar la relación entre el periodo de reloj de la red y el procesador.

Efecto de la topología.

En esta primera comparativa restringiremos el estudio a topologías tipo malla y toro 2-D empleando siempre un mecanismo de encaminamiento determinista. Las características de los enlaces y los encaminadores empleados para estas dos redes se muestran en la Tabla 2-15. Notar que las características se mantienen, dado que en las redes toroidales se emplea un mecanismo de evitación de deadlock que solo requiere una línea física por canal virtual, lo que hace que estructuralmente los encaminadores de las mallas sean idénticos a los encaminadores empleados en los toros (Ver Capítulo 3). Al tener en cuenta este hecho es coherente suponer que ambos encaminadores van a ser similares en sus tiempos de paso y frecuencias de reloj.

Velocidad de los Routers	325 Mhz
Ancho de los enlaces	32 bits
Número de ciclos de paso	4
Tamaño colas de transito	80 flits
Control de Flujo	Virtual Cut-throught

Tabla 2-15. Encaminadores base empleados en el análisis de tiempos de ejecución Malla vs Toro.

En la Tabla 2-16 aparecen resumidas las diferencias porcentuales en tiempo de ejecución para cada red y aplicación con respecto a la red ideal así como el incremento en la latencia promedio de los mensajes. Como se puede observar, la aplicación más sensible a topología es la *LU*: en este caso, las diferencias porcentuales en tiempo de ejecución entre la malla y el toro son los más altos. Sin duda, el patrón de comunicación permite explotar las características de conectividad del toro. En *Radix* y *Fft* se aprecia como la distancia media para las redes de 16 nodos no difieren demasiado, sin embargo al pasar a redes de 64 nodos las diferencias en tiempos de ejecución aumentan considerablemente. Como era de esperar, en estas dos aplicaciones la distancia

promedio de los mensajes es similar a la distancia media de la red lo que hace que, para redes con bajo número de nodos las diferencias topológicas sean pequeñas. Sin embargo, al aumentar el número de nodos, aparecen diferencias acusadas entre los tiempos de ejecución característicos de la malla y del toro. Por otro lado, *Water-nsquared* exhibe un comportamiento muy insensible a las variaciones topológicas. Fundamentalmente este hecho es consecuencia de la baja utilización que hace esta aplicación de la red de interconexión. Por lo tanto, al ser el impacto de la red prácticamente despreciable, este *benchmark* no parece adecuado para realizar comparativas entre diferentes arquitecturas.

			Radix	**FFT**	**Lu**	**Water-nsq**
Incremento en el Tiempo de ejecución con Xbar.	Malla 2-D	4x4	1.31	1.19	2.01	1.02
		8x8	1.50	1.89	1.85	1.01
	Toro 2-D	4x4	1.23	1.13	1.33	1.01
		8x8	1.41	1.61	1.30	1.01

Tabla 2-16. Variación del Tiempo de Ejecución (Tanto por 1).

Los tiempos de ejecución obtenidos, para cada una de las aplicaciones ejecutadas, sobre cada una de las topologías y número de procesadores, pueden observarse en las figuras siguientes:

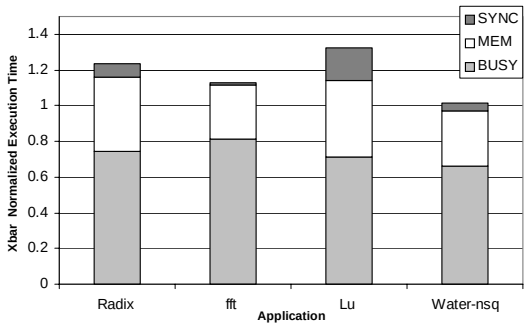

Figura 2-22. Tiempo de ejecución para un toro 4x4, normalizado al tiempo de ejecución de un Xbar 16x16.

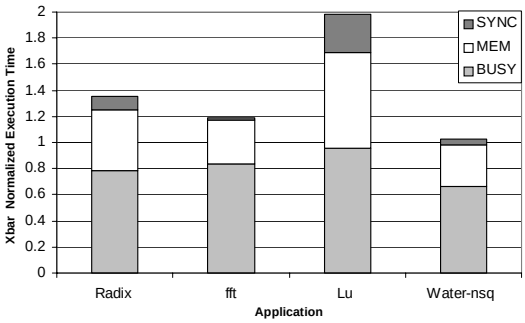

Figura 2-23. Tiempo de ejecución para una malla 4x4, normalizado al tiempo de ejecución de un Xbar 16x16.

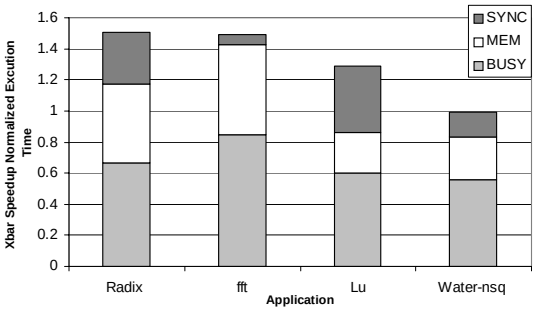

Figura 2-24. Tiempo de ejecución para un toro 8x8, normalizado al tiempo de ejecución de un Xbar 64x64.

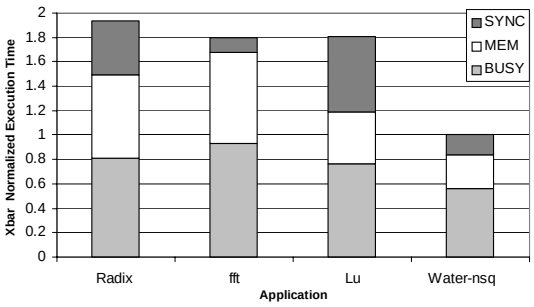

Figura 2-25. Tiempo de ejecución para una malla 8x8, normalizado al tiempo de ejecución de un Xbar 64x64.

Por último, en la Figura 2-26 se muestran las curvas de *speedup*, tanto para la malla como para el toro, de forma comparativa con respecto al crossbar. Estas curvas nos da una idea de cual es impacto de la red en la escalabilidad de cada aplicación. Se observan caídas en todas las aplicaciones en mayor o menor grado, excepto para *Water-nsquared*, donde la influencia de la red es prácticamente despreciable. Estas disminuciones son más acusadas para la malla.

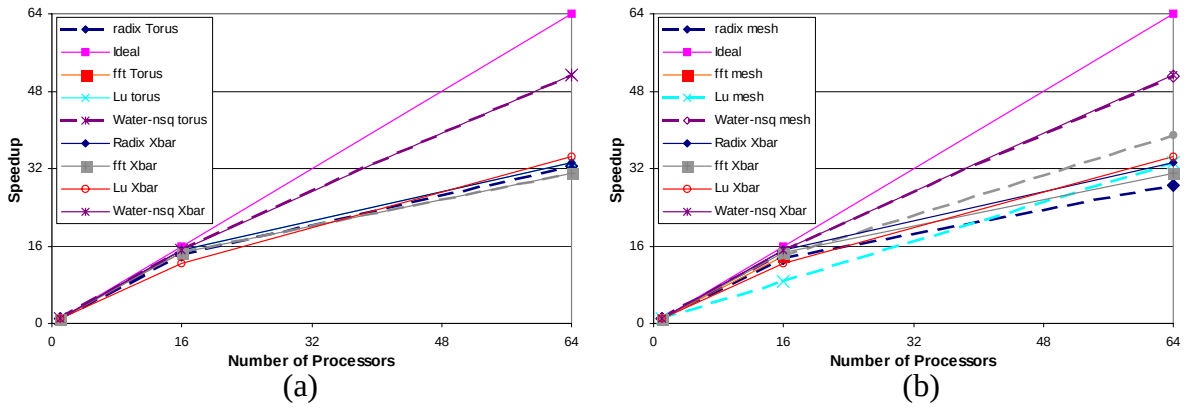

Figura 2-26. *Speedup* para redes toro (a) malla (b). Análisis comparativo frente al Xbar.

Efecto de la latencia de los encaminadores.

En este punto restringiremos el estudio a una sola de las topologías empleadas previamente y analizaremos la variación en el tiempo de ejecución de las aplicaciones al alterar el retraso de los encaminadores. En concreto, las medidas corresponderán a un toro 2-D con 16 y 64 nodos de proceso, donde la anchura de los enlaces, capacidad de las colas de transito y velocidad relativa del reloj de la red con respecto al del procesador se mantendrá como antes, pero variaremos el número de ciclos de paso de cada encaminador. En principio, se efectuarán pruebas con tiempos de paso de 4, 6 y 8 ciclos de red.

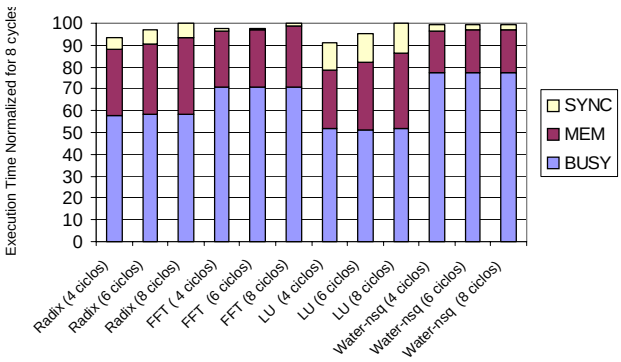

Figura 2-27. Variación en el tiempo de ejecución debido a el retraso de los encaminadores para 16 procesadores (Toro 4x4).

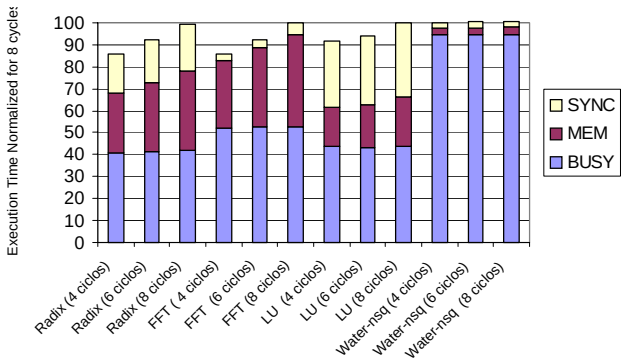

Figura 2-28. Variación en el tiempo de ejecución debido a el retraso de los encaminadores para 64 procesadores (Toro 8x8).

Se observa como en todos los casos existe una disminución en el tiempo de ejecución de las aplicaciones al disminuir el tiempo de paso de los encaminadores. Obviamente, cuanto mayor es el número de procesadores mayor es la variación en el tiempo de ejecución ya que la distancia promedio de los mensajes es mayor. También se observa como, de nuevo, en el caso de *Water-nsquared* las consecuencias de aumentar el tiempo de paso de los routers son prácticamente despreciables, lo que es coherente con los resultados obtenidos previamente.

2.3.4.2 Sensibilidad al ancho de banda de la red. Efectos del tamaño de los *phits*.

En cuanto a determinar la sensibilidad de las aplicaciones al ancho de banda ofrecido por la red, es más complejo encontrar un modo que permita probar distintas configuraciones, con distintos anchos de banda, fáciles de acotar, y sin alterar en gran medida su latencia. Una solución puede ser variar el tamaño de los bufferes de transito, aunque de este modo las variaciones en el ancho de banda de la red dependerán del patrón de tráfico que exhiba la aplicación. Soluciones como emplear algoritmos de encaminamiento distintos (como adaptativos frente a deterministas) puede ser aún mas complicado ya que las diferencias en *throughput* tampoco están acotadas y se pueden producir variaciones en la latencia a cargas bajas dependientes de la implementación.

La solución más sencilla, y que es la finalmente empleada, consiste en alterar el ancho de los enlaces entre los routers. De esta forma, por ejemplo, si pasamos de enlaces de tamaño *b* a *2b* el ancho de banda ofrecido por la red se duplicará. Sin embargo, es preciso puntualizar que la contribución del *spooling* a la latencia de los mensajes se verá afectada dado que los mensajes serán más cortos. En los resultados mostrados a continuación este efecto se ha eliminado artificialmente en la simulación, de manera que el tiempo de entrega en todas las configuraciones, independientemente de la longitud del paquete, sea el mismo. Por otro lado, es importante señalar que debido a la más pequeña longitud de los paquetes, en las redes con superior ancho de

enlace se introduce una mejora adicional en la productividad. En general, este incremento añadido es bastante inferior al causado por la ampliación de los enlaces, lo que hace que se pueda considerar nulo su efecto en esta comparativa de rendimiento.

En el análisis efectuado, nos centraremos en toros 2-D con 16 ó 64 procesadores y en los diferentes experimentos alteraremos el ancho de los enlaces. De esta forma, manteniendo las características temporales de las distintas redes y encaminadores, podremos evaluar, de forma aislada, cual es el impacto real del ancho de banda ofrecido por la red.

Los resultados muestran una relativa baja sensibilidad al ancho de los enlaces para las ejecuciones sobre el toro 4x4, estando situada la perdida máxima de rendimiento relativa en torno a un 15% para el caso de *Radix* o *LU*. Las diferentes variaciones son coherentes con las características de tráfico generado por cada aplicación (ver Tabla 2-13 en la página 67). Es notable, por ejemplo, como en el caso de *Water* las variaciones son prácticamente despreciables, sin duda como consecuencia de que la tasa de generación de tráfico para esta aplicación es extremadamente baja. Un dato curioso que se observa es como *LU*, pese a poseer cualitativamente una tasa muy inferior a Radix, las diferencias porcentuales en el tiempo de ejecución son mayores. La causa de este hecho parece encontrarse en que el patrón de tráfico de *LU* es muy localizado y sin embargo *Radix* es prácticamente uniforme. Como es conocido, el ancho de banda ofrecido por la red depende del tipo de tráfico que maneje, esto puede hacer que *LU*, pese a ejercer una carga menor en términos absolutos, introduzca una mayor contención a nivel de la red de interconexión a causa de la existencia de puntos calientes de carga en la red. Al emplear encaminadores deterministas en estas pruebas la influencia de los puntos calientes puede ser superior.

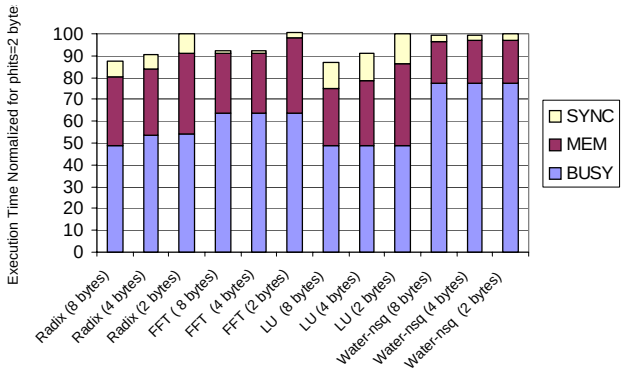

Figura 2-29. Variación en el tiempo de ejecución debido al ancho de los enlaces para 16 procesadores (Toro 4x4).

Por otro lado, como se puede observar en la Figura 2-30, las diferencias entre las diferentes configuraciones tienden a incrementarse al aumentar el número de nodos, llegando a aparecer diferencias de hasta un 40% en la *FFT* y un 30% en *Radix*. Sin duda, cuando la distancia promedio

que han de recorrer los mensajes es mayor, además de un lógico aumento en la latencia de los mensajes, la demanda de ancho de banda por parte de la aplicación es también mayor. En parte, esto es la causa directa de la permanencia por más tiempo de los mensajes en la red. Además es importante notar que, en general, el número de referencias a memoria remota, contabilizadas en *phits* por ciclo de procesador, se incrementa notablemente al incrementar el número de elementos de proceso como consecuencia de un aumento en la relación comunicación-cálculo (Ver Tabla 2-13 en la página 67). Por ejemplo, se observa como Radix y FFT incrementan su tasa de generación de mensajes de forma más pronunciada. Estos datos son coherentes con la tendencia mostrada por las aplicaciones al pasar de 16 a 64 procesadores.

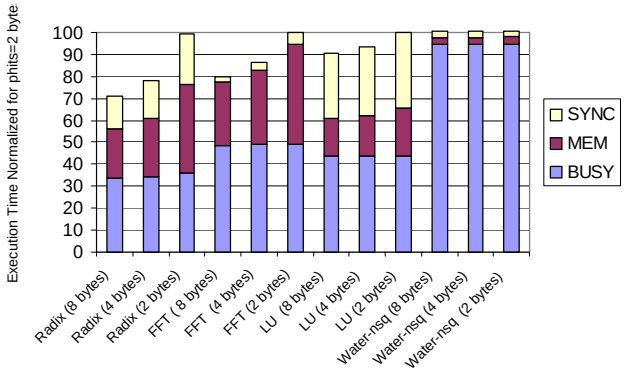

Figura 2-30. Variación en el tiempo de ejecución debido al ancho de los enlaces para 64 procesadores (Toro 8x8).

2.3.5 Efectos Combinados: Correlación Latencia y *Throughput* y su Impacto en las Aplicaciones.

Generalmente, a la hora de proponer nuevas arquitecturas de red, tanto desde el punto de vista de las implementaciones *hardware* de los encaminadores como de la topología empleada, se suele intentar mejorar, de forma aislada, una de las figuras de mérito de la red. Como ejemplo, se puede citar el empleo de mecanismos de encaminamiento que soporten adaptatividad: en este caso se logra importantes ganancias en *throughput* con respecto a su contrapartida determinista. Sin embargo, esta ganancia suele implicar un aumento de la latencia en cargas bajas, ya sea por la necesidad de emplear tiempos de ciclo superiores o un número de etapas más elevado. Es por tanto importante notar que la latencia y el *throughput* están fuertemente correlacionados y a la hora de plantear nuevas arquitecturas es necesario conocer cómo esta correlación puede afectar al rendimiento de las aplicaciones reales. Generalmente, las aplicaciones poseen zonas de gran demanda hacia la red cuya duración puede ser dependiente de la carga máxima soportada y zonas de carga baja, en las que la latencia base puede llegar a ser primordial. Es por tanto fácil de entender que proponer arquitecturas optimizadas únicamente respecto a una sola de las figu-

ras de mérito, puede conducir a que las aplicaciones pierdan las posibles ganancias introducidas en las fases de la ejecución en las que la figura de mérito complementaria sea importante.

Por lo tanto, es esencial tener un especial cuidado en la influencia que se produce sobre la latencia al introducir mejoras en el *throughput* y viceversa.

Frente a la enorme extensión de este problema, en este estudio inicial lo reduciremos a analizar el impacto producido sobre las aplicaciones manteniendo siempre el cociente *throughput* / latencia constante. Es decir, analizaremos el efecto de aumentar la latencia y el *throughput* de la red de manera correlacionada. En este análisis, a tenor de los resultados mostrados en puntos previos, nos centraremos exclusivamente en 3 aplicaciones: Radix, FFT y LU, descartando por lo tanto *Water-nsquared* dado que no parece ser en absoluto determinante, en cuanto a su rendimiento, ni la latencia ni el *throughput* de la red de interconexión.

Los experimentos a realizar con cada aplicación se han separado en tres clases de acuerdo con la Tabla 2-17.

CLASE A	Tiempo de paso=2 ciclos. Ancho de los enlaces=2 bytes
CLASE B	Tiempo de paso=4 ciclos. Ancho de los enlaces=4 bytes
CLASE C	Tiempo de paso=6 ciclos. Ancho de los enlaces=6 bytes
CLASE D	Tiempo de paso=8 ciclos. Ancho de los enlaces=8 bytes

Tabla 2-17. Notación de los experimentos probados.

Los resultados para el número limitado de experimentos seleccionado, muestran la existencia de un mínimo en la ejecución de todas las aplicaciones evaluadas. Este mínimo parece estar situado en los encaminadores de 4 ciclos y 4 bytes por enlace. Encaminadores más rápidos con ancho de banda inferior no permiten alcanzar mejores prestaciones para ninguna de las aplicaciones, como queda claro en las Figuras 2-31 y 2-32 para el caso de 2 *bytes* de ancho de banda y 2 ciclos de paso (de hasta un 20% para *FFT*). Los efectos que introducen las mejoras en ancho de banda o latencia son dependientes de la relación cálculo-comunicación de cada aplicacación. Se observa que, tanto *Radix* como *FFT*, exhiben peores tiempos de ejecución al incorporar routers más rápidos pero con un menor ancho de banda y sin embargo para *LU* la bajada de rendimiento es visible pero inferior a emplear routers con un mayor ancho de banda y mayor latencia.

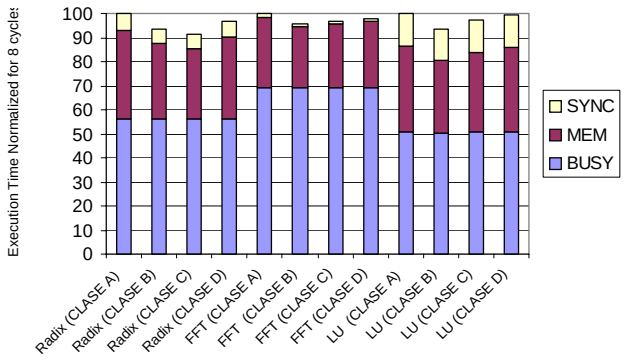

Figura 2-31. Variación del tiempo de ejecución al variar de forma correlacionada la latencia de los encaminadores y su ancho de banda con un Toro 4x4.

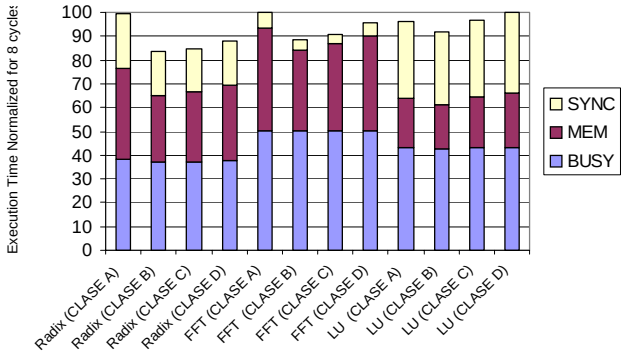

Figura 2-32. Variación del tiempo de ejecución al variar de forma correlacionada la latencia de los encaminadores y su ancho de banda con un Toro 4x4.

2.4 Conclusiones

A lo largo de este capítulo hemos planteado cómo es la metodología empleada en los estudios realizados en los capítulos posteriores. En primer lugar, hemos resaltado la importancia de las restricciones *hardware* y su posible influencia en el rendimiento de la red de interconexión. Se ha descrito la metodología que permitirá considerar en cada caso cuales son los costes de cada posible propuesta arquitectónica. Con esta aproximación es posible ampliar considerablemente el espacio de diseño a explorar, con precisión adecuada. Si llegáramos a descender a niveles de abstracción más bajos limitaríamos considerablemente la amplitud de los estudios presentados en capítulos sucesivos.

A continuación hemos introducido los fundamentos de la metodología de evaluación, constituida tanto por tráficos sintéticos como reales. Hemos descrito de forma somera las características del banco de pruebas basado en aplicaciones reales. Las aplicaciones consideradas nos permitirán posteriormente valorar cuál es impacto real de las posibles mejoras a nivel de la red de interconexión. En esta selección hemos optado por reducir, dado lo demandante de todas ellas, el conjunto a solo tres. De cualquier manera, la red influye de forma determinante en todas ellas. Como se ha podido constatar, las variaciones en el tiempo de ejecución son apreciables al alterar cualquiera de las características de la red de interconexión.