UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DIVISION DE ESTUDIOS DE POSGRADO

UNA ARQUITECTURA PARA EL PROCESAMIENTO PARALELO BASADA EN UNA RED CROSSBAR MODIFICADA

T E S I S

QUE PARA OBTENER EL GRADO DE

DOCTOR EN INGENIERIA

P R E S E N T A

MICHAEL LINDIG BOS 1

DIRECTOR DE TESIS:
DR. FRANCISCO GARCIA UGALDE

MEXICO, D. F. 1996

1. BECARIO CONACYT, reg. 68201

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

AGRADECIMIENTOS

La construcción del prototipo de la máquina, objeto del presente trabajo, no hubiera sido posible sin el financiamiento otorgado por la Dirección de Estudios de Posgrado e Investigación del I.P.N., a través de su programa de apoyo a proyectos de Investigación. Mi más sincero reconocimiento a su titular, Dr. Enrique Villa Rivera, así como a su equipo de trabajo, por el apoyo recibido.

Agradezco, también, el apoyo económico recibido del CONACyT, como becario del programa de Posgrados de Excelencia.

UNA ARQUITECTURA PARA EL PROCESAMIENTO PARALELO BASADA EN UNA RED CROSSBAR MODIFICADA

Michael Lindig Bös

Director de Tesis: Dr. Francisco García Ugalde.
División de Estudios de Posgrado,
Facultad de Ingeniería, UNAM

RESUMEN

Las realizaciones físicas de la máquina paralela de acceso aleatorio (PRAM) más frecuentemente encontradas son el multiprocesador de bus común, y máquinas de memoria compartida basadas en diversos tipos de redes conmutadas. El rendimiento computacional del multiprocesador de bus común disminuye conforme el número total de intercambios de datos entre procesadores, para un cómputo dado, crece. Esta disminución es menor para redes conmutadas, debido al mayor número de trayectorias de intercambio de datos disponible. En este sentido, las ventajas de la red *crossbar* son bien conocidas. Sin embargo, el alto número de elementos de conmutación requerido por esta red ha sido una limitante para su uso en máquinas comerciales. De las configuraciones posibles de redes conmutadas, la red tipo mariposa y otras, cercanamente relacionadas, han recibido mucha atención debido a su alto rendimiento y, comparado con la red *crossbar*, por el número reducido de elementos de conmutación requerido.

En el presente trabajo, se discute brevemente el modelo formal de la máquina paralela de acceso aleatorio. A continuación, dos modelos para el tiempo de acceso a memoria compartida son analizados en detalle. Se comparan los tiempos de acceso de las arquitecturas de bus común, buses múltiples y máquinas basadas en la red *crossbar*, así como de redes de conmutación escalonadas. Uno de estos modelos supone solicitudes de acceso a memoria generados en forma aleatoria, así como tiempos de servicio aleatorios de estos accesos, en ambos casos exponencialmente distribuidos. El segundo modelo considera la ocurrencia de accesos simultáneos, con tiempo de servicio constante. Para este modelo, se consideran tanto accesos a módulos distintos, esto es, libres de contienda, como accesos a direcciones de memoria uniformemente distribuidas. El modelo basado en accesos simultáneos proporciona mejores estimaciones para el tiempo esperado de conclusión de un cómputo paralelo, que el modelo basado en teoría de colas.

Los modelos para el tiempo de acceso a memoria compartida existentes no consideran los tiempos de propagación de los elementos de conmutación y de las líneas físicas que los interconectan, ni los retardos asociados al arbitraje de acceso. Por su importancia práctica, estos factores son analizados extensamente. En particular, se propone un esquema de arbitraje distribuido para la red

crossbar, que permite la realización de esta función por los elementos de conmutación y que simplifica considerablemente la distribución física de las líneas de interconexión requeridas por la máquina.

Para una cierta cantidad de elementos de conmutación requerida por una arquitectura dada, la cantidad de dispositivos en los que pueden ser integrados estos elementos de conmutación es función del número de líneas físicas conmutadas por cada elemento. Por otra parte, la complejidad relativa de los elementos de conmutación constituye un factor relevante para la factibilidad de una arquitectura dada. Un análisis de estos factores para las arquitecturas citadas arriba es efectuado, y se llega a la conclusión de que la red *crossbar* es una elección competitiva para máquinas de paralelismo limitado, tanto en lo que al número total de componentes se refiere, como en velocidad de conmutación.

En el presente trabajo se propone una variante de la red *crossbar* basada en $p(p-1)$ elementos de conmutación, donde p es el número de procesadores, y que, para el intercambio de datos, elimina una de las dos travesías por la red requeridas. Se discute el diseño de un conjunto de dispositivos propietario para una máquina de 8 procesadores, basado en arreglos de compuertas programables en el campo (FPGA's). Una breve descripción de las funciones incorporadas a cada dispositivo es proporcionada, seguida de una discusión detallada de las trayectorias para el intercambio de información entre los procesadores. La descripción continúa con un análisis de la función de arbitraje de acceso incorporada a los elementos de conmutación. Finalmente, se describe un dispositivo de interface entre el multiprocesador, y un procesador anfitrión, dedicado a funciones de entrada/salida. Estimaciones de la velocidad de ejecución de dos algoritmos de uso frecuente son comparadas con los rendimientos correspondientes a la máquina de bus común y diseños basados en redes escalonadas, que demuestran la superioridad de la arquitectura propuesta.

Una extensión lógica de la arquitectura propuesta es el uso de memorias caché para datos compartidos por los procesadores. Se demuestra que, para la red *crossbar*, es factible incorporar estas memorias caché a los elementos de conmutación, conservando un nivel de complejidad para estos dispositivos similar al requerido para elementos de conmutación con memoria propia, utilizados en redes conmutadas. La máquina resultante se aproxima a las características del modelo de máquina paralela de acceso aleatorio, en el sentido de permitir el acceso simultáneo de todos los procesadores a una misma localidad de memoria. Finalmente, se muestra que una extensión trivial de la arquitectura permite su uso como nodo computacional en una máquina de paralelismo masivo, de red de interconexión fija. En este sentido, se concluye que la propiedad intrínseca al diseño de proporcionar consistencia de memoria en todo momento, simplifica considerablemente los protocolos de manejo de memoria.

AN ARCHITECTURE FOR PARALLEL PROCESSING BASED ON A MODIFIED CROSSBAR NETWORK

Michael Lindig Bös

Thesis Director: Francisco García Ugalde, Ph.D.
División de Estudios de Posgrado,
Facultad de Ingeniería, UNAM

ABSTRACT

The most frequently found implementations of the parallel random access machine (PRAM) are the common-bus multiprocessor and shared-memory machines based on several types of switching networks. The performance of the common-bus multiprocessor decreases as the total number of required data exchanges among processors, for a given computation, grows. The same applies to switching networks but, because of the higher number of available data paths, to a lesser degree. In this sense, the advantages of the crossbar network as compared to other switching networks are well known. However, the high number of switching elements required by this network has been a limitation on its use in practical machines. Of the possible switch configurations, the butterfly network and other closely related networks have received much attention because of their high performance and, as compared to the crossbar switch, considerably lower component count.

In this work, a brief characterization of the ideal PRAM is given. Two models for the access time to shared memory are analysed in detail. Access times for common-bus and multiple-bus machines, as well as for architectures based on crossbar and multi-level networks, are compared. One model considers randomly generated, exponentially-distributed, access requests and service times, while a second model considers simultaneous access requests and constant service time. For the latter, both contention-free and uniformly distributed memory references are discussed. The model based on simultaneous occurrence of memory requests provides better estimates for the expected execution time of algorithms computed in parallel, than performance models based on queuing theory.

Existing shared-memory access time models do not consider the propagation delays of switching elements and physical interconnections, nor the delays introduced by access priorization. Because of their practical importance, these factors are extensively studied. In particular, a distributed arbitration scheme for the crossbar network is proposed, that allows the inclusion of this function within the switching elements themselves and simplifies considerably the physical layout of the machine.

For a certain amount of switching elements required by a given architecture, the amount of devices into which these switching elements may be integrated is a function of the number of physical lines switched by each element. Also, the relative switch complexity is a relevant factor in the feasibility of a

given architecture. An analysis of these factors for the architectures mentioned above is performed, and the conclusion is reached that the crossbar network is a competitive choice for machines of limited parallelism, both in terms of total component count, and in terms of switching performance.

In this work a variation of the crossbar network is presented, based on $p(p-1)$ switching elements, where p is the number of processors. The proposed architecture requires only one pass through the switching network for data exchange among processors. The design of a chip set for an 8-processor machine, based on field-programmable gate arrays (FPGA's), is discussed. A brief description of the functions implemented in each device is given, followed by a detailed discussion of the data paths provided by the architecture. The description then concentrates on the access priority arbitration function implemented in the switching elements, and concludes with a discussion of an interface device between the multiprocessor and a host processor, dedicated to input/output functions. Performance estimations for two frequently used algorithms are given and compared with the common-bus multiprocessor and multi-level switching network designs, which show the proposed architecture to have superior performance.

A logical extension of the architecture is the use of cache memories for shared data. It is shown that for the crossbar network it is feasible to incorporate cache memories within the switching elements, while conserving a switch complexity similar to buffered switches found in multi-level networks. The resulting machine approaches the characteristics of the ideal PRAM, in the sense that simultaneous memory requests to the same physical address are supported. Finally, it is shown that a trivial extension of the architecture allows its use as a node in massively-parallel, fixed interconnection network machines, providing a building block that is inherently memory consistent, thus simplifying memory management protocols.

INDICE

NOTACION

H	precedida por una cadena de números y/o los caracteres A - F, denota una cantidad hexadecimal, esto es, expresada en base 16.
log x	es el logaritmo base 2 de un número positivo x;
N	es el conjunto de los número naturales;
Z	es el conjunto de los números enteros;
R	es el conjunto de los números reales;
$\lfloor x \rfloor$	es el mayor entero menor o igual a x;
$\lceil x \rceil$	es el menor entero mayor o igual a x;
$\perp$	es un valor indeterminado;
$<x_1, x_2, ..., x_n>$	denota la secuencia ordenada de elementos $x_1, x_2, ..., x_n$

Por otra parte, entre los símbolos definidos a lo largo del presente trabajo, y más frecuentemente usados, se encuentran los siguientes:

b	número de buses de interconexión de una arquitectura dada;
BW	ancho de banda de memoria, definido como el número promedio de módulos de memoria activos en un ciclo de transferencia de una red de interconexión síncrona.
m	número de módulos de memoria de una arquitectura dada;
p	número de procesadores de una arquitectura dada;
P	número de procesadores activos en un momento dado;
PIN	número de identificación de un procesador dado en un multiprocesador;
η	relación entre rendimientos de dos arquitecturas de multiprocesadores;
μ	cantidad de solicitudes de acceso a memoria atendidas por unidad de tiempo;
λ	cantidad de solicitudes de acceso a memoria generadas por unidad de tiempo;
ρ	tráfico en el bus de datos, definido como λ/μ;
τ_A	tiempo medio de arbitraje;
τ_c	tiempo medio de acceso a memoria para el caso de contienda por módulos, incluyendo el tiempo de servicio a la última solicitud en cola;
τ_E	tiempo medio de ejecución previo a una solicitud de acceso a memoria;
τ_m	tiempo medio de acceso a memoria para el caso de contienda por módulos, observado por un procesador dado;
τ_μ	tiempo medio de respuesta de un módulo de memoria a una solicitud de acceso;
τ_T	suma del tiempo medio de ejecución previo a una solicitud de acceso a memoria, más el tiempo de servicio de la solicitud de acceso.

ORDENES DE MAGNITUD

Sean f y g funciones reales definidas sobre el conjunto de los números naturales, f,g:N $\rightarrow$ R. Entonces:

1.- $f(n) = O(g(n))$ denota que f es de orden *no mayor* que g, esto es, $|f(n)| = \alpha |g(n)|$ para toda n, donde α es una constante real que no depende de n.

2.- $f(n) = \Omega(g(n))$ denota que f es de orden *no menor* que g, esto es, $g(n) = O(f(n))$.

3.- $f(n) = \theta(g(n))$ denota que f es del mismo orden que g, esto es:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \in R$$

I. INTRODUCCION

I.1 Antecedentes y Motivación

El paralelismo en computadoras, entendido éste como la ejecución simultánea de más de una función por una misma máquina, forma parte de prácticamente todas las computadoras modernas. Ya sea en la forma de unidades de procesamiento dedicadas a tareas específicas, o, más recientemente, en la forma de unidades de procesamiento independientes operando de manera concurrente en el marco de un mismo dispositivo, el paralelismo se encuentra inclusive en máquinas personales. Estaciones de trabajo, máquinas tipo *main frame* y algunas supercomputadoras son ejemplos de máquinas de paralelismo limitado, en las que el procesador central es sustituido por un número pequeño de procesadores, que pueden, ya sea, ejecutar tareas independientes entre sí, o ejecutar un mismo programa sobre uno, o varios, conjuntos de datos. Por otra parte, se han construido máquinas de paralelismo masivo de miles de procesadores, tanto para propósitos específicos, como para aplicaciones de tipo más general , especialmente en el campo de la ciencia. En todos los casos, el objetivo es, desde luego, reducir el tiempo de cómputo necesario para obtener un resultado. Sin embargo, la mayor velocidad, aunque deseable, no siempre es necesaria para determinadas aplicaciones. Un motivo adicional para el estudio del paralelismo en cómputo, reside en la consideración de que algunos problemas computacionales, por su complejidad y magnitud, solamente podrán ser resueltos con base en máquinas altamente paralelas. Como ejemplos, se pueden citar los campos de la inteligencia artificial, la visión robótica, la biotecnología, y otros.

Si bien es cierto de que se dispone de la tecnología para construir, en principio, máquinas con un número arbitrario de procesadores (asumiendo que el costo no es un factor limitante), es igualmente correcto afirmar que, con esta misma tecnología, la programación eficiente de tales máquinas es tarea altamente compleja, y que frecuentemente el rendimiento computacional real obtenido constituye una pequeña fracción del rendimiento teóricamente disponible. Parece que lo anterior es, por lo menos en parte, consecuencia de que el modelo de programación disponible no se ajusta a la realidad impuesta por una arquitectura específica. Más específicamente, el algoritmo paralelo debe ser optimado para la estructura de intercomunicaciones entre procesadores proporcionado por la arquitectura de la máquina.

. Las computadoras paralelas pueden ser clasificadas en dos grandes categorías: computadoras en que una misma instrucción opera sobre un conjunto de datos (*single-instruction, multiple data*, SIMD), y máquinas en las que un conjunto de instrucciones no necesariamente iguales opera sobre un conjunto de datos, en un momento dado (*multiple-instruction, multiple data*, MIMD)[1]. La primera categoría presupone operación síncrona entre procesadores, una condición cada vez más difícil de satisfacer a las velocidades de operación de unidades de procesamiento modernas. En consecuencia, esta categoría está limitada a un número relativamente pequeño de procesadores, y asume formas específicas tales como procesadores vectoriales. La segunda categoría, a su vez, puede ser dividida

entre arquitecturas de memoria compartida, y arquitecturas basadas en el paso de mensajes entre procesadores que cuentan con memoria propia. Las máquinas de memoria compartida cuentan con una memoria global, a la que tienen acceso todos los procesadores por medio de una red de interconexiones. La memoria global puede consistir en, ya sea, un sólo banco de memoria física, o un sólo espacio de direccionamiento físicamente dividido en un conjunto de módulos de memoria. La comunicación entre procesadores se efectúa mediante accesos de escritura y lectura a la memoria global.

En máquinas basadas en el paso de mensajes, cada procesador posee memoria local propia. Los procesadores se comunican por medio de una red de interconexiones que consiste en enlaces directos entre ciertas parejas de procesadores. Cada nodo de la red es un sistema de cómputo completo, que incluye una o varias unidades de procesamiento y memoria local, un sistema de entrada/ salida y un sistema operativo. Las máquinas basadas en este modelo reciben frecuentemente el nombre de multicomputadores. La figura 1 es un esquema de la clasificación discutida.

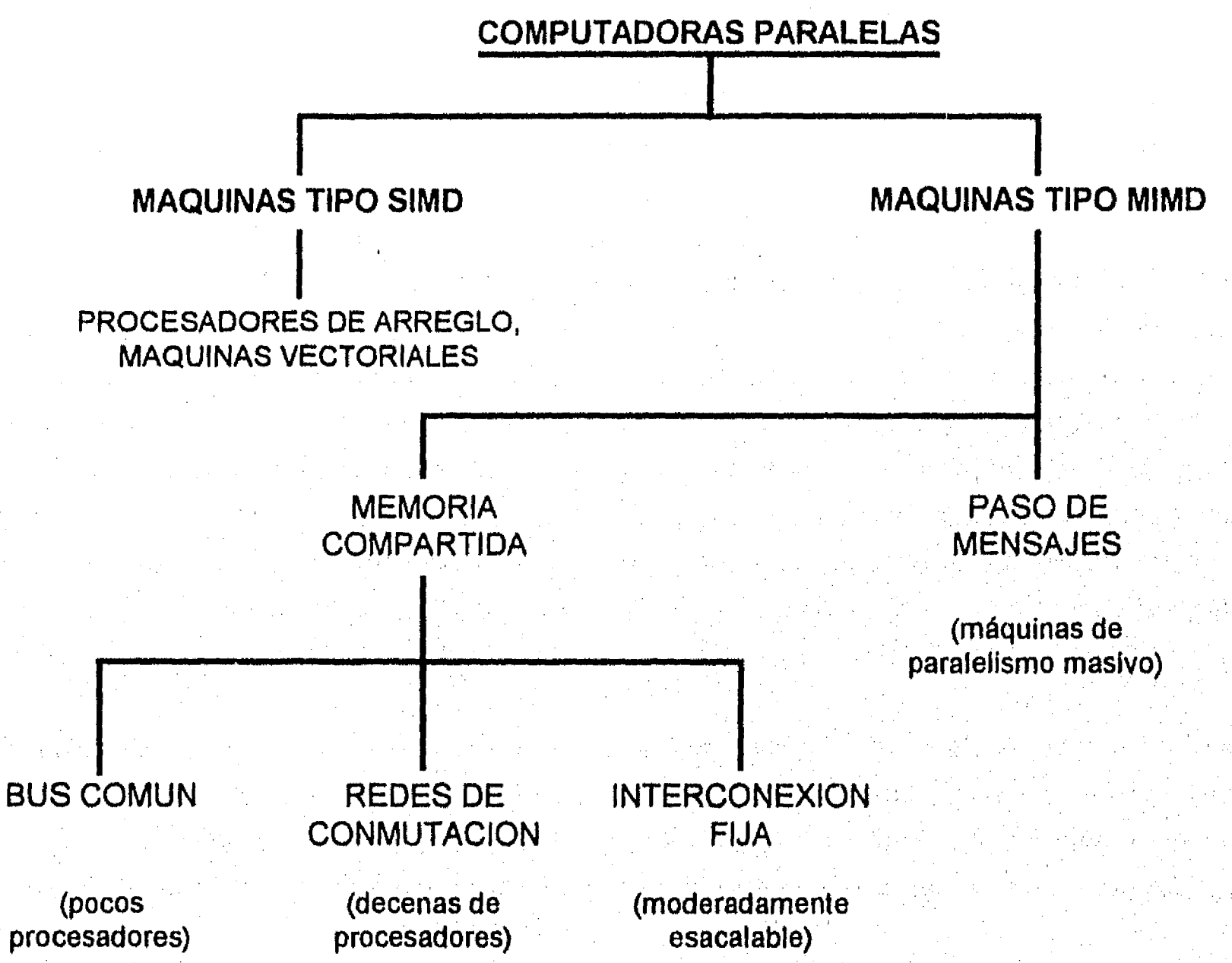


figura 1:
Clasificación de máquinas paralelas

En general; solamente las estructuras de interconexión fija, y, en particular, solamente aquellas que proporcionan un número de enlaces constante entre un nodo dado con respecto a otros nodos, son genuinamente escalables sin límite. Sin embargo, estas estructuras privilegian la comunicación entre nodos vecinos inmediatos, en detrimento de la comunicación entre nodos más alejados, y generan problemas de enrutamiento, cuya solución y ejecución puede constituir una fracción significativa del tiempo de cómputo total requerido por la solución de un problema. Está claro que estas estructuras son tanto más eficientes en la medida en que el principio de localidad espacial de memoria pueda ser aprovechado en los nodos individuales, reduciendo, así, la necesidad de comunicación entre nodos distantes. Lo anterior sugiere esquemas híbridos de multicomputadores, en los que cada nodo, a su vez, es un multiprocesador de memoria compartida.

En la actualidad, los multiprocesadores de memoria compartida frecuentemente asumen un esquema de interconexión en el cual los procesadores interactúan con la memoria global mediante un sólo conjunto de líneas físicas de interconexión (*bus* de interconexión). Esta realización, aunque la más sencilla, no permite la interconexión de muchos procesadores, en virtud de que el bus común constituye un cuello de botella para la intercomunicación entre procesadores. Para reducir este problema, se han desarrollado sistemas basados en memorias caché con protocolos de control complejos, que permiten la comunicación directa entre procesadores por medio de un segundo bus de muy alta velocidad (bus de indagación). Este esquema posee dos aspectos negativos: por una parte, incrementa la complejidad (y costo) de la máquina y, por otra, genera el llamado "problema de consistencia de memoria". En efecto, la intercomunicación directa entre procesadores, que reduce la necesidad de acceder a la memoria global para fines de intercambio de información, tiene por consecuencia que ésta no se encuentre actualizada en el transcurso de un cómputo. Para sistemas aislados, lo anterior requiere que al término de un cómputo, la memoria global deba ser actualizada previamente a operaciones de entrada/salida. Pero en sistemas de paralelismo masivo, que utilizan grupos (*clusters*) de procesadores como elementos de procesamiento en cada nodo de una red de interconexión fija, la falta de consistencia entre memorias caché y la memoria global genera la necesidad de mecanismos de control que aseguren que los datos transferidos entre nodos correspondan, efectivamente, a los valores más recientemente generados. Estos mecanismos adicionales, nuevamente, incrementan la complejidad del sistema resultante y reducen la escalabilidad del mismo.

La motivación del presente trabajo es contribuir, mediante un examen crítico de algunas arquitecturas de multiprocesadores conocidas, y con base en la tecnología disponible, a una valoración más objetiva de ideas consideradas previamente como imprácticas para la realización de máquinas físicas. Concretamente, el objetivo consiste en proponer una arquitectura de multiprocesador que:

1.- Para un nivel de complejidad dado, posea rendimientos computacionales superiores a los alcanzables con soluciones convencionales.
2.- De manera intrínseca no posea el problema de consistencia de memoria aludido arriba, facilitando, con ello, la generación de máquinas de paralelismo masivo.
3.- Alcance un mayor nivel de coincidencia con el modelo conceptual de "máquina paralela", simplificando la programación eficiente.

I.2 Formulación del problema.

Una alternativa para reducir el problema de congestionamiento generado por un bus de interconexión común y, en consecuencia, poder aumentar el número de procesadores capaces de interactuar eficientemente en una máquina de memoria compartida, consiste en utilizar una red de interconexión más compleja, que une un espacio de direccionamiento global dividido entre módulos de memoria individuales, con los procesadores. En un multiprocesador, la red asume la forma de un esquema de conmutación de uno o más niveles, que establece la conexión de un procesador con un módulo de memoria determinado, de acuerdo con las necesidades planteadas durante la ejecución del programa. Con relación al multiprocesador de bus común, la pregunta que se genera de inmediato es si la complejidad adicional representada por la red de conmutación es justificable en términos de una mayor velocidad de ejecución o, alternativamente, si la complejidad de la red es comparable a la de un esquema de intercomunicación directa basado en memorias caché y bus (adicional) de indagación. Para cualquier esquema de conmutación, una segunda pregunta es la relativa a la cantidad máxima de procesadores que pueden ser interconectados dentro de límites preestablecidos de complejidad, confiabilidad y costo del sistema.

La interconexión de *p* procesadores y *m* módulos de memoria, tal que cada procesador tenga acceso a todos y cada uno de los módulos de memoria, puede visualizarse como la gráfica bipartita completa G[p,m]. Cada procesador requiere m puertos de comunicación. Similarmente, cada módulo de memoria es interconectado por medio de p puertos a los procesadores. En virtud de que procesadores y memorias físicas poseen un número limitado de puertos, éstos son sustituidos por elementos de conmutación que permiten la conexión entre un punto de entrada y dos, o más, puntos de salida.

Es bien sabido que la red de conmutación *crossbar* es la única red capaz de generar la gráfica bipartita completa, permitiendo el establecimiento de min[p,m] conexiones simultáneas. Esta red requiere *p* elementos de conmutación de 1 entrada y *m* salidas o, alternativamente, *pm* conmutadores del tipo encendido/apagado. Por su complejidad, esta red se considera como impráctica para máquinas físicas, excepto de un número reducido de procesadores [5]. Como alternativa, se han desarrollado y caracterizado un gran número de redes de conmutación que, conservando la comunicación punto a punto entre cualquier procesador y todos los módulos de memoria, requieren una menor cantidad de elementos de conmutación. Esta reducción en componentes se logra reduciendo el conjunto de todas las relaciones entrada/salida simultáneas posibles a un determinado subconjunto, cuya naturaleza satisface los requerimientos de cierta clase de algoritmos. El problema de diseño de un multiprocesador consiste, entonces, en la elección correcta de la red de conmutación que mejor se adapte a los requerimientos computacionales, dentro de especificaciones de velocidad y costo. En este sentido, parece pertinente la siguiente observación:

> *It seems that enough research has already been done in evaluating interconnection networks in isolation. We strongly feel that more work is needed at the system level that includes the interconnection network as a major component. For example, evaluation of multiprocessor systems with prefetching, bulk data read or write, and solving cache coherence with interconnection networks shows promise for future work.* [25]

Si bien es cierto que redes de interconexión han sido extensamente documentadas, en un contexto de alta velocidad en el cambio tecnológico, sin embargo, conviene examinar, también, alternativas previamente consideradas como poco viables. El presente trabajo parte de la hipótesis que la red *crossbar* constituye una alternativa válida en términos de complejidad y costo para máquinas de hasta decenas de procesadores. Esta hipótesis se fundamenta no solamente en avances tecnológicos recientes, sino también en la apreciación de que algunos resultados formales están basados en modelos que no consideran la totalidad de los elementos pertinentes a una realización física concreta y que, en consecuencia, conviene re-examinar. Parámetros tales como retardos de propagación en líneas de interconexión, retardos asociados a dispositivos de conmutación, el número y complejidad de estos dispositivos para distintos tipos de redes de interconexión, y, finalmente, los esquemas de arbitraje requeridos para estas redes, son aspectos importantes para la viabilidad de una arquitectura y que no son tratados de manera integral por los modelos de rendimiento computacional existentes.

Con base en una red tipo *crossbar* modificada, un esquema de arbitraje de acceso distribuido, incorporado a los elementos de conmutación de la red, así como un esquema de memorias caché para datos compartidos por los procesadores igualmente incorporado a los elementos de conmutación, el presente trabajo pretende constituir un acercamiento a la máquina paralela ideal, entendida ésta como el modelo formal de máquina paralela aleatoria (PRAM)[2].

I.3 Revisión bibliográfica y contribuciones.

Existe una gran riqueza bibliográfica relacionada tanto con los fundamentos teóricos de la computación paralela, y, como ya fue mencionado, con la caracterización y evaluación de redes de interconexión. Por otra parte, el estudio de algoritmos paralelos, y de su desempeño en diferentes clases de arquitecturas, ha sido, y es, realizado por un gran número de investigadores. A continuación se citan algunos de los trabajos consultados, de acuerdo con la división temática del presente trabajo.

En lo referente a los fundamentos teóricos, en particular, la definición del modelo de máquina paralela de acceso aleatorio, se consultó "Introduction to Parallel Processing" de B. Codenotti y M. Leoncini [2]. Arquitecturas paralelas son tratadas en el libro "Computer Architecture", de M. De Blasi [3]. Una visión global de la computación paralela se encuentra en la obra "Highly Parallel Computing", de G.S. Almasi y A. Gottlieb [5]. Esta obra incluye una amplia bibliografía, que resultó de gran utilidad para este trabajo.

Para la caracterización y evaluación de las distintas arquitecturas paralelas existentes, se consultaron las siguientes antologías: "Multiprocessor Performance Measurement and Evaluation", de L. Bhuyan y X. Zhang [9], "Interconnection Networks for High-Performance Parallel Computers", de I. Sherson y A. Youssef [10], así como "Interconnection Networks for Multiprocessors and Multicomputers", de A. Varma y C.S. Raghavendra [11]. Por otra parte, se consultaron las obras "Performance Measurement and Visualization of Parallel Systems" de G. Haring y G. Kotsis [12], y "Computer Benchmarks", de J. Dongarra y W. Gentzsch [13].

Para el modelado estocástico de arquitecturas de bus común, buses múltiples y la red crossbar, resultó de particular valor la obra "Performance Models of Multiprocessor Systems", de M. Marson y G. Balbo [14]. Los fundamentos matemáticos de la teoría de colas se consultaron en el libro "Stochastic Models in Queueing Theory", de J. Medhi [15]. Los trabajos "Analysis of the Instruction Rate in Certain Computer Structures" de W.D. Strecker [23], "Performance of Multiprocessor Interconnection Networks" de L.N. Bhuyan, Q. Yang y D.P. Agrawal [25], así como "Performance of Processor-Memory Interconnections for Multiprocessors" de J.H.Patel [26] son importantes para el análisis de redes escalonadas. En este sentido, un trabajo determinante en tanto que considera elementos de conmutación con memoria propia, es "The Distribution of Waiting Times in Clocked Multistage Interconnection Networks", de C.P. Kruskal, M. Snir y A. Weiss [28].

La casi totalidad de las referencias citadas consideran intervalos de acceso, así como períodos de servicio de la o las memorias que constituyen a la arquitectura, como variables aleatorias exponencialmente distribuidas con, ya sea, igual o distintos valores esperados. En algunos casos, el análisis se amplía a distribuciones generales. No se encontraron referencias para el caso de accesos simultáneos, o de alta concentración temporal. El modelo para este caso se encuentra en el apéndice I, y deseo expresar aquí mi agradecimiento al Dr. David Romero por su ayuda para la formalización del modelo.

Uno de los primeros trabajos relacionados con el problema del arbitraje de acceso parece ser el artículo "Asynchronous Arbiter Module", de R.C. Pearce, J.A. Field y W.D. Little [37]. Un trabajo relevante para el caso de las arquitecturas basadas en buses múltiples es "M - users B - servers Arbiter for Multiple-Buses Multiprocessors", de T. Lang y M. Valero [38]. Un análisis riguroso del costo en tiempo asociado a esquemas de arbitraje se encuentra en "Multiple Bus Architectures", de T.N. Mudge, J.P. Hayes y D.C. Winsor [39].

Los elementos de conmutación, y los elementos de conmutación con memoria propia, han sido descritos por una serie de autores. Se pueden citar, entre otros, los siguientes trabajos: "The Butterfly Parallel Processor", de W. Crowther et al. [40], "The Torus Routing Chip", de W.J. Dally y C.L. Seitz [41], "VLSI Communication Components for Multicomputer Networks", de R.M. Fujimoto [42], "Communication Element for the Versatile MultiComputer", de Y. Rimoni et al. [43]. Y. Tamir y G.L. Frazier proponen memorias para múltiples colas en el trabajo "High-Performance Multi-Queue Buffers for VLSI Communication Switches" [44].

El uso de memorias caché en multiprocesadores, y, en particular, el problema de consistencia de memoria, es un tema de gran importancia y objeto de estudio en la actualidad. Uno de los primeros trabajos, en este sentido, parece ser "A new solution to coherence problems in multicache systems", de L. Censier y P. Feautrier [45]. Conviene citar, también, los trabajos "A low overhead coherence solution for multiprocessors with private cache memories", de M.S. Papamarcos y J.H. Patel [46], y "Memory consistency and event ordering in scalable shared-memory multiprocessors", de G. Gharachorloo et al. [47]. La reducción de tráfico entre memorias caché ha sido objeto de numerosos trabajos, como por ejemplo "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes", de A. Gupta, W.D. Weber y T. Mowry [48], "Policies for efficient memory utilization in a remote caching architecture", de A. Leff, P.S. Yu y J.L. Wolf [51] y "An approach to solve the cache thrashing problem", de Z. Fang, M. Lu y H. Lin [53]. Un trabajo que parece ser representativo de la investigación actual en máquinas de paralelismo masivo, basadas en multiprocesadores individuales con memorias caché escalonadas, es "The Stanford Dash Multiprocessor", de D. Lenoski et al. [55]. Por otra parte, se han propuesto arquitecturas basadas en el uso exclusivo de memorias cache: "DDM - A Cache-Only Memory Architecture", de E. Hagersten, A.Landin y S. Harid [56].

En materia de análisis de algoritmos para las diferentes estructuras paralelas, particularmente las estructuras de interconexión fija, un libro sin duda de la mayor importancia es la obra de F.T. Leighton: "Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes" [31]. En particular, su tratamiento de redes hypercúbicas y su relación con redes conmutadas escalonadas fue de la mayor utilidad para el desarrollo de este trabajo.

El diseño de la máquina se apoyó en una serie de paquetes de software de edición y programación de circuitos para arreglos lógicos programables (FPGA's - *field-programmable gate arrays*)[62], así como de circuitos impresos [63]. La documentación de estos paquetes puede, en general, considerarse como adecuada, aunque existen algunas lagunas en dicha documentación que generaron atrasos en el desarrollo del trabajo. Por otra parte, la documentación de INTEL relativa al microprocesador Pentium™ [32], [67] puede juzgarse como excelente. Lo mismo es aplicable a los diferentes manuales consultados en materia de componentes [64], [65] y [69].

I.4 Organización de la tesis.

En el capítulo dos se presenta un breve resumen de la fundamentación teórica de las máquinas paralelas y, en particular, la caracterización de la máquina paralela de accesos aleatorio (PRAM - *parallel random access machine*) en sus diferentes variantes.

En el capítulo tres se exponen evaluaciones de rendimiento de cuatro clases de arquitecturas, basadas tanto en la ocurrencia de accesos simultáneos a memoria, como en modelos estocásticos de la razón de arribo de solicitudes de acceso a memoria, y de períodos de servicio de las mismas. Las evaluaciones asumen el principio de costo unitario tanto para la ejecución de instrucciones, como para

accesos a memoria, y no consideran aspectos tecnológicos específicos. Dos algoritmos son evaluados, como ejemplos concretos de rendimiento computacional. Se concluye que la red crossbar es superior a las demás alternativas consideradas.

El capítulo cuatro analiza la complejidad de diseño de las cuatro clases de arquitecturas evaluadas en el capítulo anterior. Aspectos tales como la cantidad y complejidad de los dispositivos requeridos por los elementos de conmutación, así como esquemas de arbitraje de acceso, son analizados y vinculados a una tecnología concreta. Tomando en cuenta estos factores, la velocidad de ejecución de los algoritmos, descritos en el capítulo tres, es nuevamente evaluada. Se concluye que la red crossbar constituye una alternativa atractiva para máquinas de hasta decenas de procesadores.

El capítulo cinco describe el diseño de la arquitectura que se propone, una red *crossbar* modificada. Los elementos de conmutación, así como los esquemas de arbitraje, son descritos en detalle. La arquitectura incorpora mecanismos de sincronización entre procesadores, soportados a nivel físico. Las estimaciones de tiempos de acceso a memoria compartida, basadas en diagramas de tiempo obtenidos por la simulación de las componentes que conforman la arquitectura, son presentadas.

En el capítulo seis se propone un esquema de memorias caché distribuidas en los elementos de conmutación que conforman la red. Este esquema es comparado con memorias caché convencionales, asociadas a los módulos de memoria que conforman la arquitectura. Se concluye que este esquema, por una parte, acerca el rendimiento de la máquina al modelo teórico de la PRAM. Por otra, su factibilidad de implementación parece estar ligada a la naturaleza de la red de interconexión, siendo la red *crossbar* óptima en este sentido. El esquema no conduce a problemas de consistencia de memoria, por lo que la máquina ofrece importantes ventajas como elemento de procesamiento en una estructura de paralelismo masivo, de interconexión fija.

Un capítulo de conclusiones, de problemas aún no resueltos y de posibles líneas de investigación futuras, complementa el trabajo.

II.- EL MODELO COMPUTACIONAL DE LA MAQUINA PARALELA

El modelo formal más frecuentemente utilizado para caracterizar a las computadoras paralelas es el de la máquina paralela de acceso aleatorio, o PRAM, por sus siglas en inglés. El modelo constituye una generalización del concepto de máquina de acceso aleatorio (RAM), un modelo computacional de referencia para computadoras seriales, y que se describe a continuación.

II.1 La máquina de acceso aleatorio.

La RAM (figura 2) consta de una unidad de procesamiento y un conjunto ilimitado de registros de trabajo, r_0, r_1, ... [2]. Los registros pueden contener enteros de longitud arbitraria. La unidad de procesamiento posee dos registros de significado especial: el acumulador A, usado para instrucciones aritméticas y durante transferencias de información, y el contador de programa, PC, que en cada paso del cómputo contiene la dirección de la siguiente instrucción a ejecutar. El programa es considerado parte integral de la máquina, esto es, existen tantas RAM's como programas. Los datos de entrada son obtenidos de un dispositivo de lectura, direccionado implícitamente por un apuntador, PI, que es incrementado después de cada lectura. Similarmente, datos pueden ser escritos en un dispositivo de salida, direccionado por un apuntador PO que incrementa su contenido previamente a una escritura. El lenguaje de máquina acepta expresiones consistentes en etiquetas y operandos. Una etiqueta es cualquier cadena arbitraria de caracteres y números que comienza con un caracter. Los operandos válidos son el inmediato, directo e indirecto. Cualquier registro puede ser utilizado como operando indirecto. El conjunto de instrucciones de la RAM incluye las de transferencia de datos LOAD y STORE, las aritméticas ADD y SUB, la de control de ejecución incondicional JMP, y las condicionales JZ (acumulador = 0) y JGTZ (acumulador > 0). Las instrucciones READ y WRITE transfieren información entre, respectivamente, el dispositivo de entrada o salida, y el acumulador. Finalmente, la instrucción HALT detiene la ejecución.

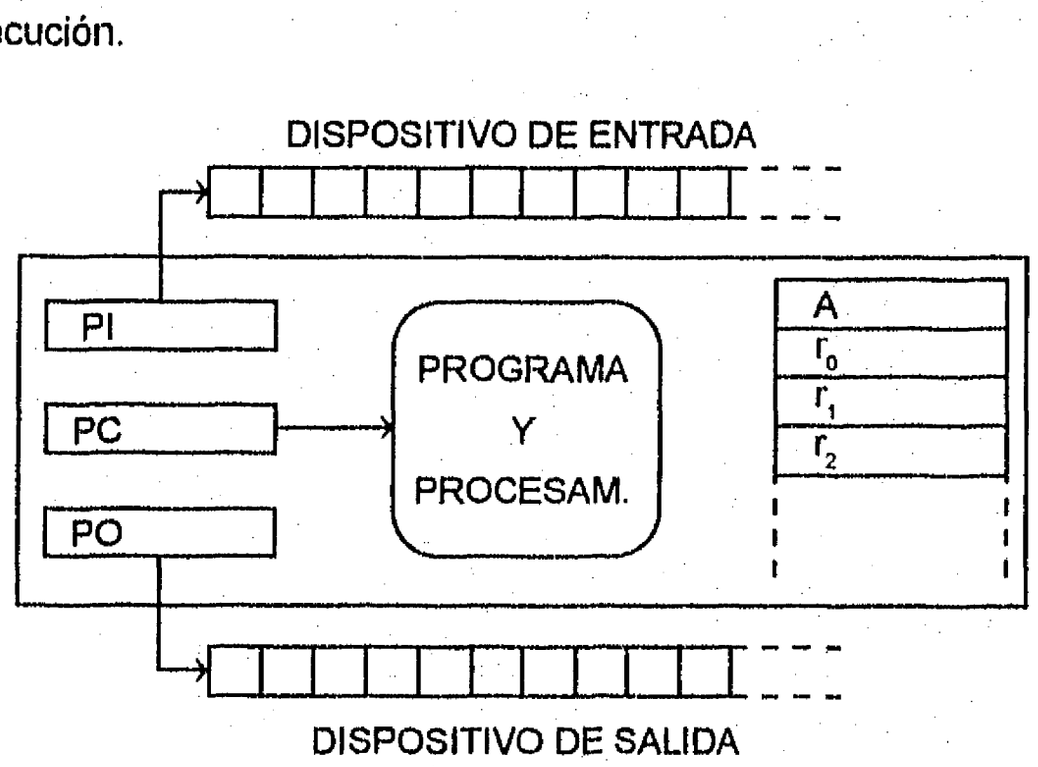


figura 2:
La máquina de acceso aleatorio

Una RAM efectúa un cómputo como sigue: Inicialmente, n datos están almacenados en las primeras n localidades del dispositivo de entrada. El apuntador PI apunta al primer dato de entrada. El valor de n se encuentra en el acumulador, y todos los demás registros poseen el valor cero. Finalmente, el contador de programa PC apunta a la primera instrucción. Entonces:

Definición 1.

Una RAM efectúa el cómputo de la función f ($f:Z^n \rightarrow Z$) , dado $< i_1, i_2, ..., i_n >$ como entrada, si se cumplen las siguientes condiciones:

a.- Si $f(i_1, i_2, ..., i_n)$ está definida, el cómputo de R concluye al generar el primer valor de salida *u* tal que $u = f(i_1, i_2, ..., i_n)$.
b.- Si $f(i_1, i_2, ..., i_n)$ no está definida, el cómputo de la RAM no concluye, esto es, la instrucción HALT no es ejecutada. □

Este modelo puede relacionarse con la máquina de Türing [4] con base en los conceptos de costo temporal y espacial asociados a un cómputo. Sea C{M} el conjunto de todos los cómputos posibles en una máquina M. En general, el costo de un cómputo en tiempo, T, y en espacio de almacenamiento, S, son funciones de C{M} a $R \cup \{\perp\}$. Un cómputo es una secuencia de estados en una máquina. Una posible medida del costo unitario en tiempo consiste en contar el número de transiciones de estado, bajo la suposición de que el tiempo requerido para ejecutar una instrucción es el mismo para todas las instrucciones posibles, e independiente del estado en que se encuentra la máquina. Lo anterior constituye la hipótesis de costo unitario: La longitud de un cómputo convergente $\ell(C)$, $C = S_0, S_1, ..., S_k$ es el número de transiciones de estado, esto es, $\ell(C) = k$. El costo en tiempo de este cómputo está dado por $T(C) = \ell(C)$.

Por otra parte, el costo en espacio de almacenamiento de un dato es la longitud de su representación como secuencia de elementos de un alfabeto dado. La hipótesis de costo unitario considera el caso de representaciones de datos de longitud comparable, de tal manera que un dato ocupa una "unidad de memoria": El costo en espacio S(C) de un cómputo convergente es el número total de localidades de memoria distintas usadas durante el cómputo.

La definición anterior no considera la longitud de los datos. En virtud de que esta longitud es función del logaritmo base r de la representación del dato, donde r es el número de elementos del alfabeto usado, se define el costo logarítmico $S_l(C)$. Sea c(i) el valor contenido en el registro i, y L(c(i)) la longitud de la representación de c(i). Entonces:

$$S_l(C) = \sum_{i \in I} \max\{L(c(i))\}, \text{ donde I es el conjunto de registros usados durante el cómputo.}$$

Similarmente, para el caso en que el tiempo de ejecución es distinto para instrucciones diferentes, se define el costo logarítmico en tiempo, $S_l(T)$, como la suma de los costos individuales de las instrucciones ejecutadas por el programa.

Para una máquina de Türing [TM] se define la complejidad temporal T(n) como el costo máximo en tiempo de todos los cómputos posibles sobre una entrada de longitud n. Si, sobre una entrada de longitud n, al menos un cómputo no concluye, el valor de T(n) es indefinido. Una definición similar se puede establecer para la complejidad espacial, S(n). La relación entre una TM y una RAM queda, entonces, establecida por la siguiente proposición:

Proposición 1.

Un cómputo en una TM de complejidad temporal $T(n) \geq n$ puede ser simulado por una RAM en $O(T(n))$. □

Por otra parte, la relación entre una RAM y una TM toma la siguiente forma:

Proposición 2.

Un cómputo en una RAM de complejidad temporal $T(n) \geq n$ puede ser simulado en una TM en un tiempo $O(T(n)^3)$. □

Una máquina de Türing [4] posee k+2 cintas, de las cuales una corresponde a datos de entrada, y otra es usada para datos de salida. Las restantes k cintas constituyen las cintas de trabajo de la máquina. La prueba de las proposiciones 1 y 2 simula el espacio de direccionamiento de una RAM en localidades de las cintas de trabajo de una TM, y viceversa. Por otra parte, cada paso de una TM puede ser simulado en tiempo constante (independientemente del valor de n) en una RAM. También, las instrucciones aritméticas de una RAM pueden ser simuladas en $O(T(n)^2)$ en una TM. Como el número de operaciones a simular es T(n), el tiempo total de simulación es, respectivamente, $O(T(n))$ y $O(T(n)^3)$. La prueba completa se encuentra en las referencias [2] y [7].

Se dice que dos funciones f y g definidas en **N** están polinomialmente relacionadas [2] si existe una constante positiva k tal que, para cada n, $f(n) = O(g(n)^k)$. De las proposiciones 1 y 2 se desprende que la RAM y PM son modelos computacionales polinomialmente relacionados.

II.2 La máquina paralela de acceso aleatorio.

En su forma original [7], el modelo consta de una colección infinita de RAM's, cada una de las cuales tiene acceso a una memoria global, utilizada para comunicación entre procesadores (figura 3). Las localidades de la memoria global no pueden ser usadas como registros índice. Una instrucción adicional, FORK, se discute más adelante. Tanto la memoria local, como la global, tienen capacidad para almacenar enteros de longitud arbitraria. Además del acumulador y contador de programa, asociado a cada procesador, existe un registro de significado especial, el indicador de estado F, que contiene 1, si el procesador está activo, y 0 en caso contrario.

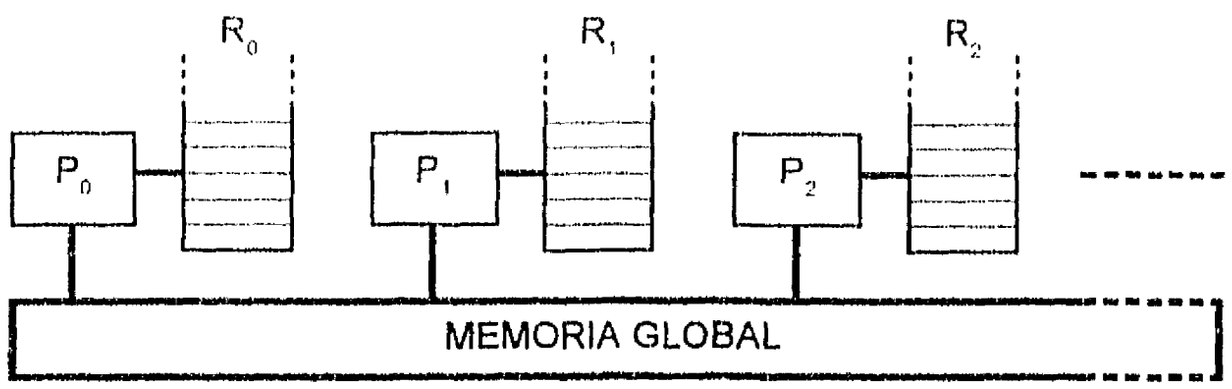


figura 3:
La máquina paralela de acceso aleatorio

Al igual que la RAM, el modelo descrito en la referencia [7] incluye dispositivos de entrada/salida, así como las instrucciones correspondientes. Todos los procesadores ejecutan un mismo programa, en general, sobre datos diferentes, que forma parte de los procesadores. Esto es, existen tantos modelos PRAM, como programas. La condición de que el programa ejecutado por los procesadores sea el mismo, no implica que la secuencia de ejecución también lo sea. La actividad de los procesadores es controlada por las instrucciones FORK y HALT. Esta última detiene la operación de un procesador, y escribe 0 en el registro de estado F. La instrucción FORK, ejecutada por el procesador P_i, genera los siguientes efectos:

1.- Un procesador inactivo, P_j, es encontrado, y activado.
2.- El acumulador de P_j, A_j, es inicializado con el contenido del acumulador A_i.
3.- El contador de programa de P_j es inicializado con el valor de una etiqueta asociada a la instrucción FORK. La primera instrucción ejecutada por P_j es la indicada por esta etiqueta.

Supóngase que el procesador encontrado es el de menor índice de entre los inactivos. Los n valores de entrada están almacenados en los registros de memoria global $m_1, m_2, ..., m_n$. Los registros A_0, PC_0 y F_0 del procesador P_0 contienen, respectivamente, n, 1 y 1. Todos los demás registros, tanto locales como globales, contienen cero. Esto es, el único procesador activo es P_0. La computación se detiene cuando una de las siguientes condiciones se cumple:

1.- Dos o más procesadores intentan escribir a la misma localidad de memoria al mismo tiempo. En ese caso, el resultado del cómputo es indefinido.
2.- El procesador P_0 ejecuta HALT. En este caso, el resultado del cómputo se encuentra en m registros consecutivos de la memoria global, a partir de una dirección preestablecida.

El modelo respeta el principio de costo unitario, incluyendo accesos a memoria global y operaciones de entrada/salida.

En [7], la PRAM es definida como una pareja (ℓ,P), donde ℓ es un programa de longitud finita, y $P: N \rightarrow N$ es un límite al número de procesadores disponibles para un cómputo.

Definición 2:

Sean g, h:N $\rightarrow$ N $\cup$ {$\perp$}. Una PRAM efectúa el cómputo de la función $f = \langle f_n \rangle_{n \in N}$, donde $f_n: Z^{g(n)} \rightarrow Z^{h(n)} \cup \{\perp\}$, si, para cualquier entrada $x \in Z^{g(n)}$ se cumple:

a.- Si $f_n(x)$ está definida, el cómputo se detiene cuando P_0 ejecuta la instrucción HALT, y se aplica la condición (2) dada arriba.
b.- Si $f_n(x)$ es indefinida, el cómputo no se detiene, o, alternativamente, se aplica la condición (1) dada arriba. □

De manera similar al modelo RAM, la PRAM puede ser relacionada con una máquina de Türing [TM]. Sean S y T dos funciones positivas, no-decrecientes, definidas sobre los números naturales. Sean:

1.- $ESPACIO_{TM}(S(n))$, $ESPACIO_{PRAM}(S(n))$ la clase de problemas que puede ser resuelta por, respectivamente, una TM y una PRAM de complejidad espacial O(S(n)) y
2.- $TIEMPO_{TM}(T(n))$, $TIEMPO_{PRAM}(T(n))$ la clase de problemas que puede ser resuelta por, respectivamente, una TM y una PRAM de complejidad temporal O(T(n)).

Entonces, una formulación de la llamada "tesis de computación paralela" toma la siguiente forma:

Proposición 3.

Si $S(n) \geq \log n$, entonces: $\bigcup_{k=1}^{\infty} TIEMPO_{PRAM}(S^k(n)) = \bigcup_{k=1}^{\infty} ESPACIO_{TM}(S^k(n))$ □

Esto es, cualquier cómputo ejecutado por una máquina de Türing con un costo espacial S(n) puede ser simulado por una PRAM en un tiempo $T(n) = O(S(n)^c)$, donde c es una constante que depende del modelo de PRAM usado.

Como ejemplo, sea $TIEMPO\text{-}ESPACIO_{TM}(n^k, \log^h n)$ la clase de problemas que puede ser resuelta por una TM de complejidad temporal $T(n) = O(n^k)$, y complejidad espacial $S(n) = O(\log^h n)$, donde h, $k \geq 1$. De la proposición 3 se desprende que existe un algoritmo paralelo que resuelve esta clase de problemas en un tiempo dado por $O(\log^r n)$. Si se compara el tiempo de ejecución secuencial $O(n^k)$ con el de ejecución paralela $O(\log^r n)$, se concluye que esta clase de problemas es eficientemente paralelizable en lo que a tiempo de ejecución se refiere.

La prueba de la proposición 3 [7] considera a las máquinas de Türing y las PRAM's como reconocedoras de lenguajes. La prueba consiste en simular una TM con una PRAM, y viceversa. La técnica fundamental es la de construir la gráfica del cómputo de la máquina M y resolver el problema de accesibilidad de la gráfica (o, equivalentemente, el de cerradura transitiva de la gráfica del cómputo de la máquina M).

II.2.1 Variaciones sobre el modelo original.

La fase de inicialización requiere O(log n) unidades de tiempo, donde n es el tamaño de entrada (número de procesadores). Si el cómputo requiere Ω(log n) unidades, la omisión de la fase de inicialización no altera el orden del costo de tiempo. En consecuencia, se puede considerar que todos los procesadores están inicialmente activos y, por tanto, omitir la instrucción FORK. Cada procesador es identificado por un número, **PIN** (*processor identification number*), que puede ser leído por cada procesador en un registro de lectura dedicado. Por otra parte, todos los procesadores ejecutan la misma instrucción. Si esta instrucción es función de un PIN en particular, aquellos procesadores que no poseen este PIN ejecutan lazos de sincronización.

Desde el punto de vista del juego de instrucciones, el modelo original aplica las siguientes restricciones [7]:

1.- Cada instrucción de complejidad temporal S(n) debe poder ser emulada por una máquina de Turing determinista en, a lo más, $S(n)^{o(1)}$ pasos.
2.- Cada instrucción de complejidad temporal S(n) debe poderse simular en una máquina de Turing determinista con, a lo más, $S(n)^{o(1)}$ cintas de trabajo. □

Se han propuesto las siguientes extensiones al modelo [2]:

1.- Juego de instrucciones mínimo. Incluye la asignación adicional $r \leftarrow \lfloor r/2 \rfloor$. Esta asignación evidentemente satisface las condiciones (1) y (2) dadas arriba.
2.- Juego de instrucciones restringido. Incluye la asignación adicional $A \leftarrow \lfloor A*2^r \rfloor$. Las condiciones (1) y (2) se satisfacen (bajo el criterio de costo unitario) si el número de bits, W(n), de la representación de los operandos está acotado a $W(n) = n^{O(1)}$.
3.- Juego de instrucciones completo. Corresponde al juego restringido, con las adiciones de multiplicación y división:
   $A \leftarrow A*r$
   $A \leftarrow \lfloor A/r \rfloor$
   Se aplica la misma condición dada para el caso del juego restringido. □

En lo referente a accesos a memoria global, el modelo original permite la lectura simultánea de la misma localidad de memoria por más de un procesador. Esto es, accesos simultáneos de lectura no requieren arbitraje de acceso. Escrituras simultáneas, en cambio, conducen a la condición HALT. Actualmente, se manejan las siguientes alternativas:

1.- EREW (lectura y escritura exclusivas). Accesos simultáneos de lectura y escritura son detenidos.
2.- CREW (lectura concurrente, escritura exclusiva). Corresponde al modelo original.

CRCW (lectura y escritura concurrentes). Accesos simultáneos son permitidos. De acuerdo con el valor efectivamente escrito, esta categoría se divide como sigue:

3.- CRCW DEBIL. Escrituras simultáneas son permitidas, si y sólo si el valor escrito es cero.
4.- CRCW MODO COMUN. El valor escrito por los procesadores debe ser el mismo.
5.- CRCW GANADOR ARBITRARIO. El valor escrito es arbitrario, y no se sabe, *a priori*, cuál de los procesadores efectúa la escritura.
6.- CRCW MODO PRIORITARIO. El valor escrito corresponde al procesador cuyo PIN es el mayor (o, equivalentemente, el menor).
7.- CRCW FUERTE. El valor escrito es el mayor (o, equivalentemente, el menor).

Sea la clase de problemas que puede ser resuelta por una PRAM de cualesquiera de las categorías definidas arriba en tiempo O(T(n)), con un número de procesadores O(P(n)). Entonces, la secuencia de categorías establecida define una jerarquía tal que la clase de problemas que puede ser resuelta por una categoría dada incluye a la clase de problemas que puede ser resuelta por la categoría anterior. La prueba es obvia para las categorías 1, 2 y hasta la 6, en virtud de que la definición de la categoría inmediata anterior constituye un caso particular de la siguiente. Para la categoría 7, la prueba consiste en simular una escritura en modo prioritario con una PRAM en modo fuerte, y viceversa. La siguiente proposición relaciona la PRAM EREW con la CRCW en modo fuerte:

Proposición 4:

Un cómputo paralelo que puede ser efectuado en una PRAM CRCW fuerte en tiempo T(n) utilizando P(n) procesadores, también puede ser efectuada en una PRAM EREW en tiempo O(T(n)logP(n)), utilizando O(P(n)) procesadores [7].

II.3 Limitación de los modelos PRAM.

Los modelos PRAM discutidos arriba se fundamentan en la premisa de que todos los procesadores tienen acceso simultáneo a una memoria global, compartida. La estructura de interconexión con la memoria, así como la organización de la misma, no forman parte del modelo. En general, una arquitectura paralela de memoria compartida puede ser vista como un conjunto de procesadores, N, interconectados a un conjunto de módulos de memoria, M, donde el espacio de direccionamiento de la memoria global está constituido por la suma de los espacios de direccionamiento de los módulos individuales. La red de interconexión puede ser representada por una gráfica, en la cual los nodos representan procesadores y módulos de memoria, y los arcos, conexiones físicas entre nodos. La complejidad de la red se define como el número de nodos, multiplicada por el número de arcos que entran, o salen, de cada nodo. Las redes de mayor interés son las correpondientes a complejidad O(N), O(NlogN) y $O(N^2)$, donde N es el número de nodos.

El hecho de que más de un arco conecta con un nodo dado, implica que el procesador o módulo de memoria posee más de un puerto de comunicación. Alternativamente, procesadores y módulos de memoria pueden ser limitados a un sólo puerto, y elementos de conmutación externos pueden substituir la función de los puertos de comunicación adicionales.

En un multiprocesador típico, la memoria global es dividida en M módulos de la misma capacidad de almacenamiento. Cada uno de N procesadores tiene acceso a los M módulos por medio de una red conmutada. Para cada entrada a la red (procesador) existen O(M) trayectorias de uno o más arcos que unen la entrada con M salidas (módulos de memoria). Los nodos interiores de la gráfica representan puntos de conmutación.

Está claro que la estructura de interconexión genera un costo en tiempo de ejecución, cuyo valor depende, entre otros factores, del retardo asociado a cada nodo, y del número de nodos existente en una trayectoria. Supóngase que cada nodo cumple la hipótesis de costo unitario, y que el número de nodos por trayectoria es el mismo para todas las trayectorias. Entonces:

1.- En la estructura de complejidad N, exactamente un punto de conmutación une a cada uno de N procesadores con una sola memoria global (M=1). Por lo tanto, solamente un procesador tiene acceso a la memoria global en un tiempo dado. La lectura de N datos tiene un costo O(N).

2.- Estructuras de complejidad O(NlogN) son típicamente realizadas con base en un esquema de conmutación escalonado, en los que cada trayectoria entrada/salida incluye logN puntos de conmutación. Sea M≥N. Entonces, hasta N valores de entrada pueden ser leídos por N procesadores al mismo tiempo, con un costo T(N) = O(logN).

3.- Finalmente, en estructuras de complejidad O(NM), cada procesador se comunica con cada módulo de memoria por medio de un punto de conmutación. En consecuencia, hasta N datos pueden ser leídos en O(1) unidades de tiempo.

□

Sea una PRAM uniforme, esto es, que el programa de longitud finita que controla el cómputo sea el mismo para cualquier posible número de entradas. Entonces, el valor mínimo del tiempo de ejecución T(n) de una PRAM uniforme es función de la complejidad de la estructura de interconexión entre procesadores y memoria global, no pudiendo ser inferior a, respectivamente, O(N), O(logN) y O(1), para las estructuras discutidas. Lo anterior es consecuencia de que cada procesador tiene que leer, al menos, un dato de entrada de memoria global (véase la definición 2).

La conclusión anterior afecta la validez de la tesis de computación paralela para ciertas clases de problemas. Por ejemplo, considérese la clase de problemas que puede ser resuelta por una TM con espacio logarítmico. De la proposición 3 se obtiene:

$$ESPACIO_{TM}(\log n) = TIEMPO_{PRAM}(O(\log n))$$

Por lo dicho arriba, lo anterior no se cumple para una PRAM basada en una estructura de interconexión de complejidad N. En una máquina física, la estructura de intercomunicación determina en gran medida tanto el rendimiento computacional, como el costo. Estos aspectos se analizan en el siguiente capítulo.

III.- ESTRUCTURAS DE INTERCONEXION EN MULTIPROCESADORES

La estructura de interconexión de un multiprocesador introduce retardos adicionales a los generados por accesos a memoria propiamente dichos, tanto por el tiempo de tránsito asociado a una trayectoria dada, como por posibles conflictos derivados del uso de un mismo recurso por más de un procesador. La importancia relativa de estos retardos con relación al tiempo de cómputo total depende, evidentemente, de la frecuencia relativa con que cada procesador requiere servicios de la memoria global y, en consecuencia, de la red. Por otra parte, depende de la distribución temporal de estas solicitudes en un período dado, esto es, de la probabilidad de que más de una solicitud se presente de manera coincidente con otra. Ambos factores dependen del modelo de programación utilizado. De manera informal, la frecuencia relativa de acceso conduce al concepto de *granularidad*. Se dice que un algoritmo es de granularidad más fina que otro si requiere de un mayor número de accesos a memoria en un período dado y si estos accesos involucran transferencias de una cantidad menor de información. En contraposición, un algoritmo es de granularidad gruesa si genera transferencias de bloques de datos en intervalos de tiempo relativamente grandes.

Por otra parte, la distribución temporal de los accesos depende de la función desarrollada por cada procesador, en relación con los demás. Si esta función es independiente, como por ejemplo el cómputo de una tarea no relacionada con otros procesadores, los accesos generados por esta tarea no estarán temporalmente correlacionados con accesos generados por otros procesadores. En cambio, si dos o más procesadores ejecutan el mismo algoritmo en el mismo tiempo, solicitudes simultáneas de acceso son más frecuentes.

Existe una vasta literatura relacionada con el modelado, simulación y evaluación del rendimiento de multiprocesadores. Entre los libros consultados, se cuentan las referencias [9-13]. En la gran mayoría de los trabajos ahí expuestos se asume un modelo de programación orientado a la ejecución de tareas independientes por cada procesador, y/o algoritmos de granularidad gruesa. El presente trabajo se orienta hacia la ejecución paralela de un mismo algoritmo, de granularidad fina. Las demandas de comunicación impuestas a un red dada son mayores, especialmente en lo que a solicitudes de acceso simultáneas se refiere. En lo subsecuente, se analizan algunas estructuras de interconexión representativas de la literatura, para los casos de solicitudes de acceso simultáneas desde más de un procesador, y solicitudes de procesadores no correlacionadas entre sí. Por sencillez, se asume que los procesadores operan en forma síncrona, y que el principio de costo unitario es satisfecho por todas las componentes del multiprocesador.

III.1 EL MULTIPROCESADOR DE BUS COMUN.

De entre las estructuras de complejidad O(N), la tal vez más frecuentemente utilizada es la de bus común (figura 4). Se entiende por *bus* un conjunto de líneas de intercomunicación, donde cada línea transmite un valor binario y donde el conjunto de valores transmitidos constituye determinada unidad

de información. Todos los procesadores se comunican con la memoria global por medio de este bus, con base en un mecanismo de arbitraje. Solamente un procesador puede escribir en un instante dado, aunque lectura simultánea, en principio, es posible. El costo de esta estructura, donde por "costo" debe entenderse el número de componentes físicas, crece aproximadamente en forma lineal con **p**, donde **p** es el número de procesadores.

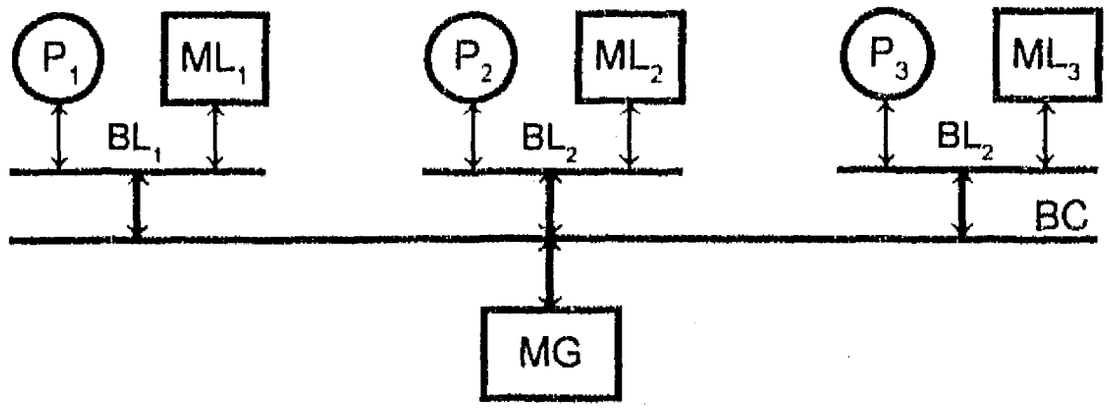


figura 4:
Multiprocesador de bus común de tres unidades de procesamiento compuestas de procesador (P) y memoria local (ML) interconectados por un bus local (BL). La memoria global (MG) es conectada por medio de un bus común (BC) a las unidades de procesamiento.

III.1.1 Accesos simultáneos.

Considérese un modelo de programación SIMD (*single instruction, multiple data* [1]), en el cual todos los procesadores ejecutan la misma secuencia de instrucciones. Posterior a un tiempo de ejecución τ_E, todos los procesadores escriben a la memoria global. El proceso de escritura se realiza en un tiempo $\tau_A + p\tau_\mu$, donde τ_A es el tiempo requerido por el arbitraje de acceso, p, el número de procesadores y τ_μ el tiempo de acceso a memoria. Nótese que el tiempo de arbitraje es requerido solamente para el primer acceso. El arbitraje de los subsecuentes accesos es efectuado de manera concurrente a los accesos a memoria, por lo que el tiempo de arbitraje puede considerarse como despreciable [14]. El tiempo total requerido por el cómputo es, entonces, $\tau_T = \tau_E + p\tau_\mu$. Sea P el número de procesadores activos, esto es, sea:

$$P = p\tau_E/\tau_T = p\tau_E/(\tau_E + p\tau_\mu) \qquad \text{..... 1}$$

Bajo la hipótesis de costo unitario, $P = p\tau_E/(\tau_E + p)$. Es fácil ver que P crece más lentamente conforme p crece. En particular, P tiende a τ_E conforme p tiende a infinito. Supóngase que $\tau_E = 10$. Entonces, P = 2.857 para p = 4, pero solamente 6.154 para p = 16.

La ecuación (1) corresponde al caso en que el procesamiento no se considera concluido sino hasta que el último acceso se haya efectuado, esto es, ningún procesador puede iniciar un nuevo procesamiento hasta que todos los procesadores hayan concluido el anterior. Si esta condición de

dependencia entre procesos no se aplica, conviene expresar P en términos del tiempo de acceso promedio τ_m observado por cada procesador, donde $\tau_m = \tau_\mu p/2$:

$$P = p\tau_E/[\tau_E + \tau_\mu p/2] \quad \text{..... 2}$$

Para los valores del ejemplo anterior, se obtiene, respectivamente, P = 3.333 y P = 8.889.

III.1.2 Accesos estadísticamente independientes.

Un modelo ampliamente utilizado para el estudio del rendimiento de multiprocesadores está basado en una subclase de procesos estocásticos, las cadenas de Markov [14, 15]. El modelado considera que los procesadores se encuentran inicialmente activos. Después de un tiempo aleatorio, los procesadores emiten una solicitud de acceso a la memoria global. Si el bus no está disponible, los procesadores tienen que esperar hasta recibir el acceso al bus. La memoria es utilizada durante un tiempo aleatorio para completar un acceso, a cuya conclusión el procesador atendido retoma al estado activo.

Nótese que la descripción anterior no considera el tiempo de arbitraje requerido, esto es, el modelo supone que el bus es otorgado (en caso de estar disponible) en cero tiempo, y liberado, una vez concluido el acceso, en cero tiempo. En virtud de que la contienda por el bus es la única fuente de degradación considerada, el modelo toma la forma de la figura 5. La descripción completa del modelo requiere la especificación del número de procesadores p, las distribuciones de las variables aleatorias $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_p$ y $\sigma_1, \sigma_2, \ldots, \sigma_p$ que representan, respectivamente, el tiempo activo y el tiempo de acceso a memoria de cada procesador, así como el protocolo de priorización de accesos. Supóngase que los vectores ε y σ están conformados por variables aleatorias exponencialmente distribuidas, con parámetros λ_i y μ_i, respectivamente. El modelo más sencillo considera que estos parámetros son iguales

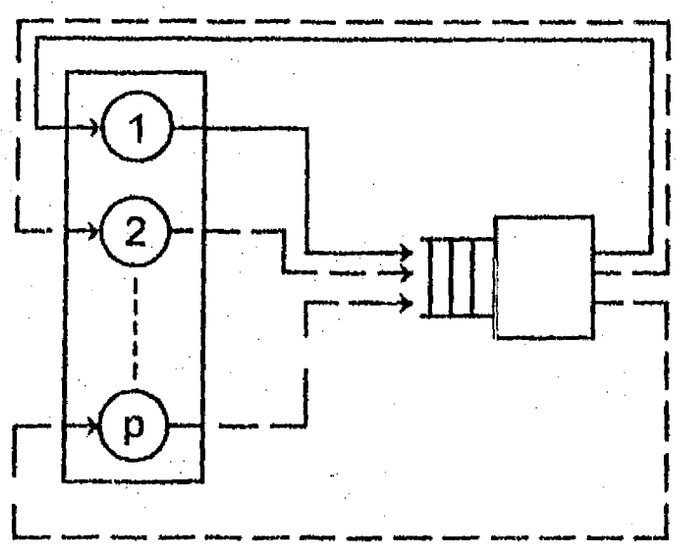


figura 5:
Modelo de cola del multiprocesador de bus común.

para i = 1, 2, ..., p. El esquema de priorización es irrelevante para este caso, y el modelo reduce a un sisterna de cola de M arribos y otros tantos procesos de servicio, con base en una sola estación de servicio (cola del tipo M/M/1 en la notación de Kendall)[16]. El modelo puede ser analizado como un proceso nacimiento-muerte con:

$$\lambda'_k = \begin{cases} \lambda(p - k), & 0 \le k < p, \\ 0, & k \ge p \end{cases}$$

$$\mu'_k = \begin{cases} \mu, & 1 \le k \le p, \\ 0, & k = 0, k \ge p + 1 \end{cases} \qquad \text{..... 3}$$

donde λ'_k y μ'_k son las razones de arribo y de servicio por unidad de tiempo, respectivamente, cuando k solicitudes se encuentran esperando o recibiendo servicio.

La distribución de estado estacionario en la estación de servicio, π, está dada por:

$$\pi_k = \pi_0(\lambda/\mu)^k p!/(p-k)!, \qquad 0 \le k \le p, \text{ y}$$

$$\pi_0 = \left[\sum_{k=0}^{p} (\lambda/\mu)^k p!/(p-k)!\right]^{-1} \qquad \text{..... 4}$$

donde $\pi_k = \lim_{t \to \infty}$ PR{N(t) = k} y N(t) es el número de clientes presentes en el sistema en el tiempo t.

El número de procesadores activos está dado por el número total de procesadores, menos el número de procesadores cuyas solicitudes de acceso a memoria global se encuentran en proceso. Por lo tanto,

$$P = \sum_{k=0}^{p} (p - k)\pi_k \qquad \text{..... 5}$$

Como ejemplo, considérense los valores usados para el caso de accesos simultáneos, esto es, sea $\lambda = 0.1$ y $\mu = 1$. De (3) y para cuatro procesadores, se obtiene $\pi_0 = 0.6467$. Similarmente, para 16 procesadores resulta $\pi_0 = 0.0223$. Substituyendo en (4), se obtiene para P, respectivamente, 3.5334 y 9.7613.

En [14] se analizan varios modelos de la arquitectura de bus común, considerando tanto funciones de distribución no-exponenciales, así como parámetros desiguales de estas funciones. Con relación al modelo sencillo presentado arriba, diferencias significativas en rendimiento son obtenidas solamente cuando existen variaciones importantes entre los parámetros asociados a cada procesador. En estos casos, el criterio de asignación de prioridad también puede influir en el rendimiento global, siendo preferible un criterio de rotación al de otros, como el de prioridad fija, o uno basado en el orden de arribo.

III.2 EL MULTIPROCESADOR DE BUSES MULTIPLES.

Estos sistemas constituyen una extensión natural de la máquina de bus común analizada en el punto anterior. Un conjunto de *p* procesadores pueden acceder a un conjunto de *m* memorias comunes por medio de un conjunto de *b* buses (figura 6). Se considerará que el espacio de direccionamiento de la memoria global es dividido en m partes iguales. Por otra parte, se supondrá que b < min[p,m]. Cuando el número de buses es igual o mayor al menor de entre el número de procesadores o de memorias comunes, la red de interconexión es del tipo *crossbar*, que será analizada en el siguiente punto.

En el multiprocesador de buses múltiples se pueden presentar dos tipos de conflictos de acceso. Por una parte, la cantidad de solicitudes que pueden ser atendidas por unidad de tiempo es limitada por la cantidad de buses disponibles. Por otra, la memoria común direccionada puede estar atendiendo una solicitud previa o, equivalentemente, más de una solicitud puede estar dirigida a la misma memoria. El protocolo de arbitraje desempeña un papel importante en el rendimiento de la red. Algunos autores [17,18] asignan un bus a una solicitud solamente si la memoria direccionada está, también, disponible. En virtud de que este protocolo parece difícil de ser realizado en una máquina física, lo subsecuente asume una asignación de buses aleatoria para el caso de solicitudes simultáneas, y basada en el criterio de prioridad de arribo (*first come, first served*).

III.2.1 Accesos simultáneos.

Supóngase que p solicitudes son generadas de manera simultánea. Sea τ_c el tiempo medio de servicio de estas solicitudes, bajo la restricción de que todos los accesos a memoria deben estar concluidos antes de que se inicie un procesamiento posterior. Por otra parte, sea τ_m el tiempo medio de servicio observado por cada procesador, sin la restricción anterior. Supóngase que una solicitud a la memoria j es igualmente probable para toda j, $1 \leq j \leq m$, y que el evento de que una solicitud seleccione la memoria j es independiente de que otra seleccione la memoria k, donde $k \neq j$.

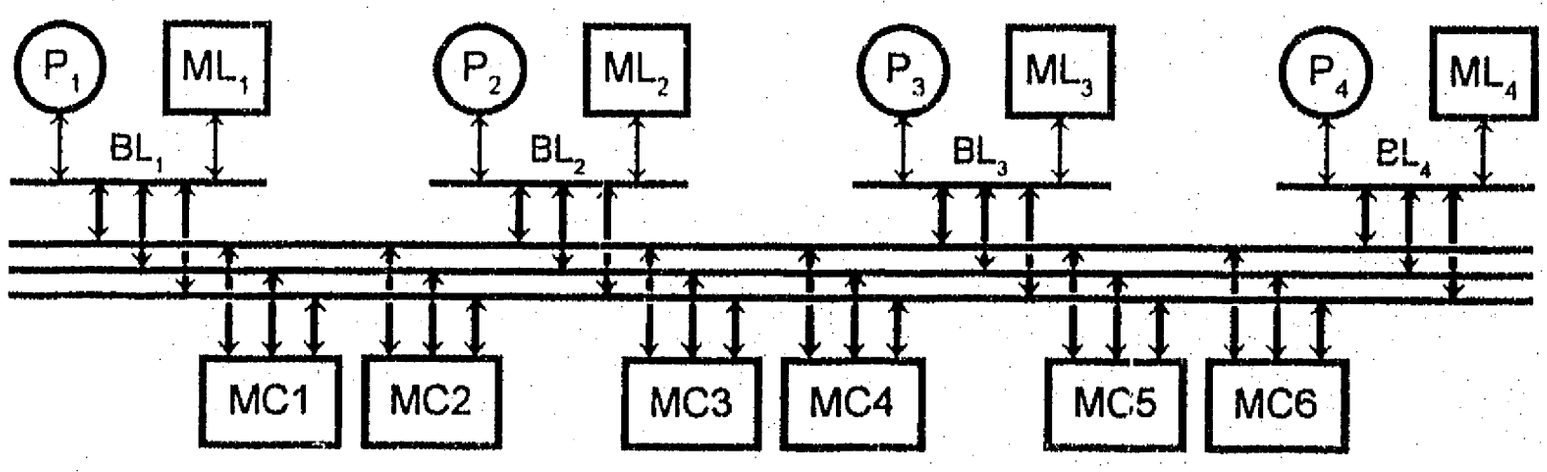


figura 6:
Multiprocesador de tres buses, cuatro unidades de procesamiento y seis memorias comunes (MC1 - MC6)

Dado que el número de buses es inferior al de procesadores, a lo más b solicitudes pueden ser atendidas al mismo tiempo. Esto es, la fracción de solicitudes atendidas en un tiempo dado es b/p, por lo que el tiempo total de acceso es, respectivamente, $(p/b)\tau_c$ y $(p/b)\tau_m$. En consecuencia, el valor medio de procesadores activos, P, es:

$$P = p\tau_E/[\tau_E + (p/b)\tau_c] \text{ y}$$
$$P = p\tau_E/[\tau_E + (p/b)\tau_m] \qquad \ldots\ldots 6$$

donde τ_c y τ_m pueden ser calculados como se muestra en el apéndice I, y τ_E es el tiempo de ejecución previo a la generación de las solicitudes de acceso. Como ejemplo, considérese el sistema de la figura 6. Para 3 buses y 6 memorias, se obtiene $\tau_c(3,6) = 1.4722$ y $\tau_m(3,6) = 1.2639$. De (6), y para $\tau_E = 10$, resulta, respectivamente, para 4 procesadores P = 3.3437 y P = 3.4231. Comparado con el ejemplo del procesador de bus común de 4 procesadores (P = 2.857 y P = 3.200), el resultado anterior es significativamente mayor (obtenido, desde luego, con una mayor complejidad de la red de interconexión).

III.2.2 Accesos estadísticamente independientes.

Las suposiciones en que se fundamentan los modelos descritos en [18,19] son similares a las expuestas en el punto III.1.2. A la conclusión de un período activo, los procesadores generan solicitudes a memorias específicas elegidas al azar, con la misma probabilidad 1/m, donde m es el número de memorias. El comportamiento del sistema puede entonces ser representado por un sistema de cola como se muestra en la figura 7, donde λ y μ son los valores esperados de las razones de arribo y de servicio, respectivamente. El sistema supone que las razones de arribo de solicitudes y de atención a las mismas corresponden a distribuciones exponenciales con parámetros iguales para ambas variables

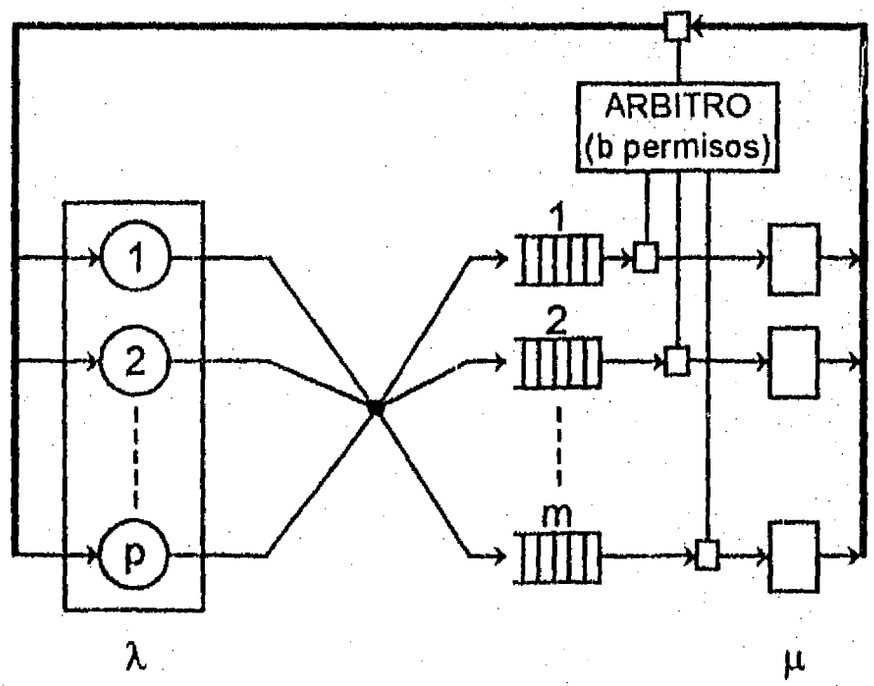


figura 7: Modelo de cola del multiprocesador de buses múltiples.

aleatorias. El comportamiento de la arquitectura puede ser modelado, entonces, con base en una cadena de Markov de tiempo continuo. Sin embargo, estos modelos solamente pueden ser aplicados a multiprocesadores de tamaño moderado, debido al crecimiento exponencial de la cantidad de estados, conforme el número de componentes del sistema aumenta. Se han obtenido soluciones aproximadas de estos modelos [19], dos de las cuales se presentan a continuación.

Cota superior.

Si el número de memorias es considerablemente mayor al número de buses, se puede suponer que la memoria seleccionada se encuentra libre [14] para la gran mayoría de las solicitudes de acceso. En este caso, la contienda por recursos compartidos se limita a la contienda por buses, esto es, la interferencia de memoria es despreciada. El resultante sistema de cola puede, entonces, ser analizado como un proceso nacimiento-muerte caracterizado por:

$$\lambda_k = \lambda(p - k), \quad 0 \le k \le p,$$

$$\mu_k = \begin{cases} k\mu, & 1 \le k < b, \\ b\mu, & b \le k \le p \end{cases} \qquad \text{..... 7}$$

donde λ_k y μ_k son las razones de arribo y de servicio, respectivamente, cuando p - k procesadores se encuentran activos. La probabilidad de que k solicitudes se encuentren en la cola (incluyendo la solicitud que está siendo atendida), π_k, satisface la relación:

$$\pi_k = \pi_{k-1}(\lambda_{k-1}/\mu_k), \quad 1 \le k \le p, \quad \sum_{k=0}^{p} \pi_k = 1 \qquad \text{..... 8}$$

La relación anterior puede ser resuelta para obtener:

$$\pi_k = \pi_0 \binom{p}{k} (\lambda/\mu)^k, \quad 0 \le k \le b,$$

$$\pi_k = \pi_0 \binom{p}{k} (\lambda/\mu)^k [k!/\, b! b^{k-b}], \quad b \le k \le p, \qquad \text{..... 9}$$

donde:

$$\pi_0 = \left[(1 + \lambda/\mu)^p + \sum_{j=b}^{p} \binom{p}{j} \left(\frac{j! b^{b-j}}{b!} - 1 \right) (\lambda/\mu)^j \right]^{-1} \qquad \text{..... 10}$$

La cota superior para el número de procesadores activos, P_U, es evaluada como:

$$P_U = \sum_{k=0}^{p} (p - k)\pi_k \qquad \text{..... 11}$$

Como ejemplo, considérese el sistema de la figura (6), con los valores indicados en el punto III.2.1. Substituyendo en (10), se obtiene $\pi_0 = 0.6830$ y, de (9) y (11) resulta $P_U = 3.6363$. Este valor es, aproximadamente, 9% mayor al obtenido en el punto anterior (accesos simultáneos).

Cota inferior.

Si en el modelo anterior se reduce el número de memorias, la probabilidad de que una memoria seleccionada se encuentre ocupada, crece. Para p > b, el valor más desfavorable para el número de memorias es igual al número de buses, en virtud de que, para un número menor de memorias, uno o más buses no tendrían conexión a módulos de memoria. El sistema puede ser modelado de manera similar al caso anterior, caracterizado ahora como:

$$\lambda_k = \lambda(p - k), \qquad 0 \le k \le p$$

$$\mu_k = b\mu \frac{k}{k + b - 1}, \quad 0 \le k \le p \qquad \text{..... 12}$$

Las probabilidades de estado estacionario satisfacen la relación (8) y pueden ser expresadas como:

$$\pi_k = \pi_0 (\lambda/b\mu)^k \binom{p}{k} \frac{(b + k - 1)!}{(b - 1)!}, \quad 0 \le k \le p \qquad \text{..... 13}$$

donde:

$$\pi_0 = \left[\frac{1}{(b - 1)!} \sum_{k=0}^{p} (\lambda/b\mu)^k \binom{p}{k} (b + k - 1)! \right]^{-1} \qquad \text{..... 14}$$

La cota inferior del número de procesadores activos, P_L, puede ser obtenida mediante la expresión:

$$P_L = \sum_{k=0}^{p} (p - k)\pi_k \qquad \text{..... 15}$$

Para los valores dados en el ejemplo anterior y utilizando (13), (14) y (15), se obtiene $P_L = 3.6049$. Nótese que para este ejemplo, P_U y P_L son comparables al rendimiento obtenido para la arquitectura de bus común (3.5389). Esto difiere de los resultados obtenidos para accesos simultáneos, para los cuales la arquitectura de la figura 6 es claramente superior.

III.3 EL MULTIPROCESADOR CON RED CROSSBAR.

En un multiprocesador con red de interconexión *crossbar*, un conjunto de p procesadores puede acceder a m memorias, $m \geq p$, por medio de p buses (figura 8). En consecuencia, la única fuente de conflicto consiste en la posibilidad de que una memoria seleccionada está ocupada atendiendo una solicitud previa, o, equivalentemente, que dos o más solicitudes simultáneas especifican la misma memoria. La red es de complejidad O(pm), lo que puede considerarse como un caso límite de las arquitecturas de buses múltiples discutidas arriba. A pesar de que esta red se considera como impráctica para máquinas físicas por su alto costo, ha recibido sin embargo considerable atención en la literatura [17] - [26]. Lo anterior es consecuencia de que constituye la realización de la gráfica bipartita completa y, en este sentido, constituye la referencia para otras arquitecturas basadas en redes de conmutación.

III.3.1 Accesos simultáneos.

Bajo los supuestos especificados en el punto III.2.1 y reconociendo que, a lo más, p solicitudes pueden presentarse al mismo tiempo, resulta de inmediato:

$$P = p\tau_E/[\tau_E + \tau_c]$$
$$P = p\tau_E/[\tau_E + \tau_m] \qquad \text{..... 16}$$

donde τ_c y τ_m son tiempos medios de acceso definidos arriba. Como ejemplo, considérese un sistema de cuatro procesadores y seis memorias comunes. Del apéndice I se obtiene $\tau_c = 1.8241$ y, para $\tau_E = 10$, resulta P = 3.3829. Comparado con el ejemplo dado en el punto III.2.1 (P = 3.3436), el incremento es sólo marginal. Sin embargo, el sistema correspondiente a la red crossbar (figura 8) es menos complejo que el multiprocesador de la figura 6, como se verá mas adelante.

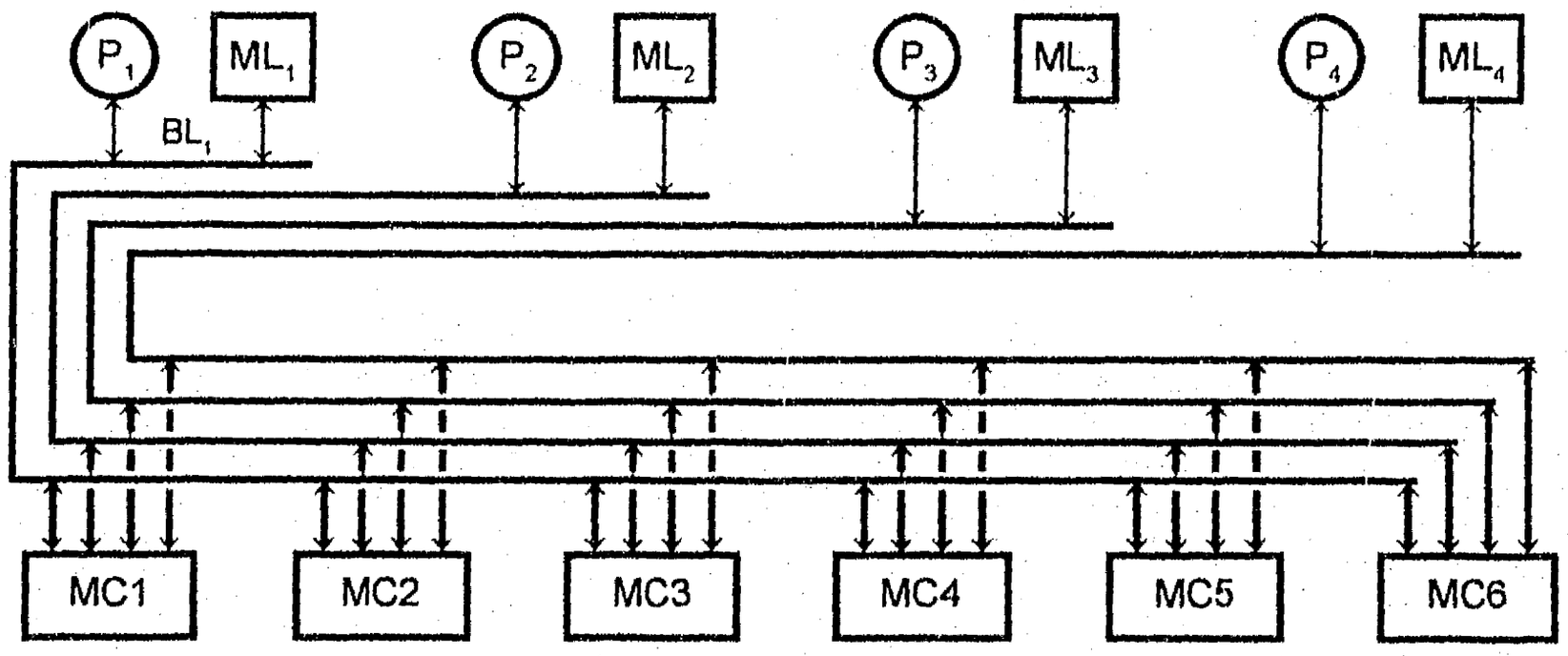


figura 8: Multiprocesador con red *crossbar*, de cuatro unidades de procesamiento y seis memorias comunes.

III.3.2 Accesos estadísticamente independientes.

En la referencia [14] se presenta una discusión amplia de los diferentes modelos que han sido utilizados para caracterizar la red *crossbar*. En general, se trata de modelos discretos en los cuales el tiempo es dividido en unidades iguales. Los procesadores emiten solicitudes de acceso al inicio de cada unidad de tiempo, con una probabilidad α. La medida de calidad es el número de solicitudes exitosas, esto es, atendidas, por unidad de tiempo. Bajo estas condiciones, el problema de interferencia de memoria es analizado por medio de un modelo de cola de tiempo discreto, en el cual p procesadores son representados como p estaciones de retardo, y m módulos de memoria como m estaciones de servicio. Colas se pueden formar solamente en frente de las memorias (figura 9). La operación de los módulos de memoria es síncrona, y cada módulo atiende a lo más una solicitud por unidad de tiempo. El direccionamiento de los módulos es aleatorio, y más de una solicitud puede ser generada para un módulo en un tiempo dado. Los modelos pueden ser clasificados con base en tres características fundamentales:

1.- Tiempo activo. Algunos modelos suponen que $\alpha = 1$. Al inicio de cada unidad de tiempo, un procesador cuya solicitud previa ha sido atendida, genera una nueva solicitud. Los procesadores se comportan como estaciones de cero retardo. Otros modelos suponen $\alpha < 1$, permitiendo una distribución geométrica del tiempo activo de los procesadores.

2.- Patrón de referencia de memoria. La mayoría de los modelos asumen una distribución uniforme para la selección de los módulos de memoria. En algunos casos se consideran patrones no-uniformes (principio de localidad de accesos de memoria).

3.- Solicitudes perdidas. Para simplificar la construcción de los modelos, muchos autores asumen que, en caso de presentarse varias solicitudes a un mismo módulo, solamente una es atendida y las demás se pierden, esto es, los procesadores que no fueron atendidos emiten una nueva solicitud al inicio de la siguiente unidad de tiempo, que pueden ser dirigidas a módulos diferentes. Con ello se elimina, de hecho, la existencia de colas y se simplifica considerablemente el modelo, pero los resultados subestiman la interferencia de memoria.

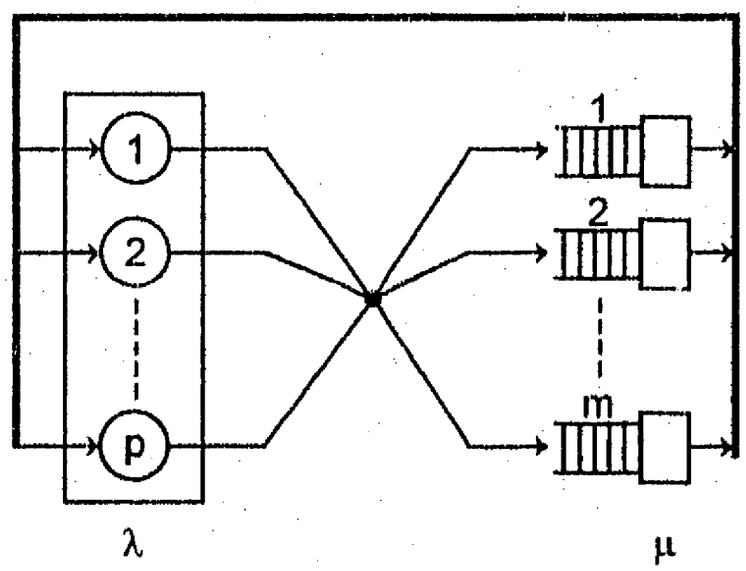


figura 9:Modelo de cola de la red *crossbar*.

Un primer análisis riguroso para el modelo de cero retardo, permitiendo la existencia de colas, se encuentra en [20]. Este modelo fue considerablemente simplificado por Bhandarkar [21], y posteriormente ampliado para un tiempo activo geométricamente distribuido. Este autor obtuvo una solución cerrada para el número promedio de módulos activos β, para el caso de 2 procesadores y 2 módulos de memoria:

$$\beta = (1 - \alpha)\frac{\alpha^2 + 1}{\alpha^2 - \alpha + 2} \qquad \ldots\ldots 17$$

El modelo de cero retardo y solicitudes perdidas fue inicialmente propuesto por Strecker [23]. Bajo estas suposiciones, el estado actual del sistema es independiente del estado anterior, y β puede ser obtenida como:

$$\beta = m\{1 - [1 - (1/m)]^p\} \qquad \ldots\ldots 18$$

Bhandarkar observó una condición de simetría entre m y p, en el sentido de que un sistema de p procesadores y m memorias se comporta de manera similar a otro, de m procesadores y p memorias. En consecuencia, propuso la siguiente modificación al resultado anterior:

$$\beta = r\{1 - [1 - (1/r)]^s\} \qquad \ldots\ldots 19$$

donde r = max(m,p) y s = min(m,p).

Finalmente, el problema de accesos no-uniformemente distribuidos fue tratado por Mudge y Makrucki [24]. Ellos observaron que β aumenta, si un procesador dado direcciona con mayor frecuencia un módulo determinado, con relación a los demás.

Actualmente, la estimación de rendimiento más aceptada está basada en el ancho de banda de memoria BW [25], definido como el número promedio de módulos de memoria activos en un ciclo de transferencia de la red de interconexión síncrona. El término "activo" significa que un procesador está realizando exitosamente una transferencia de memoria (lectura o escritura). Bajo la condición de solicitudes perdidas, el ancho de banda está dado por:

$$BW = m\{1 - [1 - (\lambda/m)]^p\} \qquad \ldots\ldots 20$$

donde p es el número de procesadores, m el de módulos de memoria y λ la probabilidad de que un procesador genere una solicitud de acceso durante la unidad de tiempo. La ecuación (20) supone probabilidades de acceso uniformemente distribuidas para todos los módulos. Bhandakar [21] demostró que esta aproximación está limitada a un error de 8% para $m/p > 0.75$.

El número esperado de procesadores activos, P, puede obtenerse de (20) como [25] :

$$P = (\mu/\lambda)BW = (m\mu/\lambda)\{1 - [1 - (\lambda/m)]^p\} \quad \text{..... 21}$$

En la ecuación anterior, μ/λ representa la relación entre el número de solicitudes de acceso que pueden ser atendidas por un módulo de memoria en la unidad de tiempo, y el número de solicitudes generadas en el mismo lapso. Nótese que $\mu/\lambda \geq 1$. Como ejemplo, considérese el sistema de 4 procesadores y 6 módulos de memoria, para los mismos valores de λ y μ usados en los ejemplos anteriores. De (21) se obtiene P = 3.9011, significativamente el mayor valor para las arquitecturas previamente consideradas.

Es interesante comparar (21) con la cota inferior para una arquitectura de buses múltiples, discutida en el punto anterior. Para 4 procesadores y 4 módulos de memoria, de (21) resulta P = 3.8525. Por otra parte, aplicando las ecuaciones (13), (14) y (15), resulta P = 3.6130. Estos resultados difieren en un 6.22% (dentro del límite dado por [21]) y reflejan el criterio optimista implícito en la condición de solicitudes perdidas. Pero, por otra parte, la validez de aplicar las ecuaciones correspondientes a la cota inferior resulta cuestionable, en virtud de que, por hipótesis, el número de buses considerado debe ser inferior al número de procesadores.

III.4 REDES CONMUTADAS ESCALONADAS.

Una red conmutada escalonada comunica *p* procesadores con *m* módulos de memoria a través de una secuencia de elementos de conmutación (figura 10). La unidad básica de la red es el elemento de conmutación, un dispositivo que interconecta *k* entradas con *s* salidas (figura 11). Este dispositivo

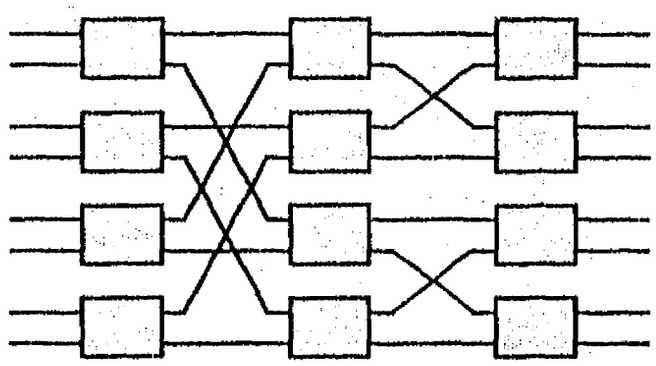


figura 10: La red conmutada tipo mariposa.

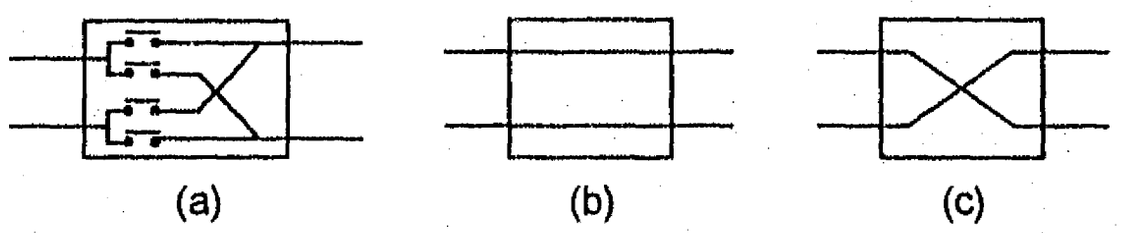


figura 11: Un elemento de conmutación de dos entradas y salidas (a).
Las comunicaciones simultáneas permitidas son horizontal (b) y cruzada (c).

constituye, a su vez, una red tipo *crossbar*. El número total de elementos de conmutación requerido para una red dada depende del número de entradas y salidas de que dispone cada elemento. Supóngase que $p = m = N = 2^r$, r un entero. Entonces, el número de elementos de conmutación varía entre 1 (esto es, una sola red *crossbar*) para $k = s = N$, hasta $(N/2)\log N$, para $k = s = 2$. De la figura 11 se desprende que un elemento de conmutación de 2 entradas y salidas requiere cuatro interruptores del tipo encendido/apagado. Una red de orden r, de 2^r entradas y salidas requiere, por lo tanto, $r2^{r+1}$ interruptores, mientras que una red crossbar utilizaría 2^{2r}. La economía en el número de interruptores es la razón principal por la cual las redes escalonadas han sido estudiadas extensamente. En virtud de que esta economía es máxima para elementos de conmutación de 2 entradas y salidas, en lo subsecuente se considerará únicamente este caso.

Las redes escalonadas exhiben conflictos de acceso a nodos, aun para destinos finales distintos, como se muestra en la figura 12. En consecuencia, cada elemento de conmutación requiere arbitraje para resolver accesos conflictivos. En virtud de que a lo más un dato puede ser transmitido al siguiente nodo en la unidad de tiempo, los datos no transmitidos deben ser re-emitidos por la fuente (procesadores) en un tiempo posterior, o el dato no transmitido debe ser almacenado localmente hasta que el nodo se encuentre disponible. En el segundo caso, la capacidad de almacenamiento generalmente está dada por una memoria tipo FIFO (primer arribo, primera salida)(figura 13).

Supóngase que en cada uno de N nodos de entrada a la red se encuentra un dato para ser transmitido a una de N salidas. Para fines de análisis, conviene distinguir entre dos tipos de relaciones entrada/salida, como sigue:

a.- Permutaciones. Cada dato presente en un nodo de entrada $<u,e>$ está destinado a un nodo de salida $<\pi u,s>$, donde $\pi:[1,N] \rightarrow [1,N]$ es una permutación. Estas relaciones no generan contiendas en los nodos destino.

b.- Aleatorio. Cada dato presente en un nodo de entrada está destinado a un nodo de salida s, tal que $s \in \{1, 2, ..., N\}$ seleccionado al azar. Esto es, se pueden producir contiendas en los nodos destino.

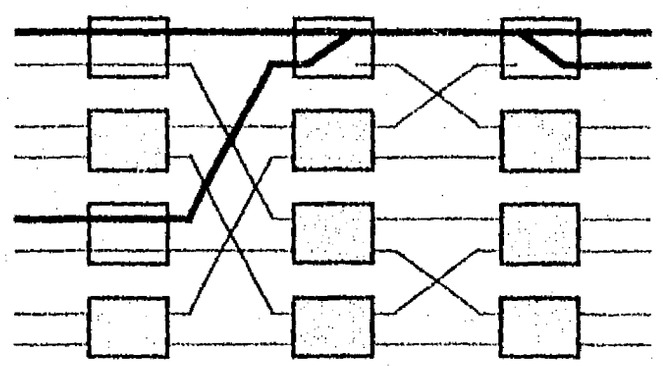


figura 12: Ejemplo de trayectorias conflictivas en la red conmutada tipo mariposa.

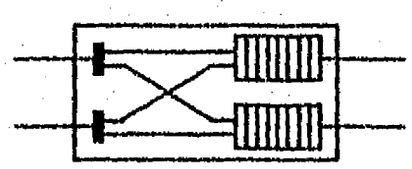


figura 13: Un elemento de conmutación con memoria propia

La cantidad total de relaciones entrada/salida posibles es, desde luego, N^N. De esta cantidad, N! corresponden a permutaciones. Para una red de orden r = logN construida con base en elementos de conmutación de 2 entradas y 2 salidas, se cumple, además, que solamente $N^{N/2}$ no generan contiendas de acceso en nodos internos a la red. Lo anterior se desprende del siguiente argumento. Para cada elemento de conmutación existen dos vías de comunicación que no involucran contiendas de acceso a la salida, esto es, ambas vías horizontales, o ambas cruzadas (figura 11). Denótese por H un elemento de conmutación que establece comunicación horizontal, y por C al que establece comunicación cruzada. Sea S el número de elementos de conmutación en la red. Entonces, la cantidad de permutaciones que no causan contienda está dado por:

$$\{0C,SH\} + \{1C, (S - 1)H\} + \{2C, (S - 2)H\} + \ldots + \{SC,0H\} = \sum_{k=0}^{S} \binom{S}{k} = 2^S,$$

donde la última igualdad se desprende del teorema binomial.

Pero, el número de elementos de conmutación S es (N/2)logN. Substituyendo, se obtiene el resultado. □

Como ejemplo, sea N = 8. Entonces, existen 16,777,216 relaciones de entrada/salida diferentes, de las cuales 40,320 no generan conflictos en los nodos de salida (permutaciones). De éstas, solamente 4096 no generan conflictos en nodos internos de la red.

Está claro que únicamente las permutaciones que no generan contiendas internas pueden ser transmitidas en tiempo mínimo. Supóngase que cada elemento de conmutación introduce un retardo unitario, τ. La cota inferior del tiempo de transmisión es, entonces, (logN)τ. Se han definido muchos tipos de redes, todas óptimas para cierta clase de algoritmos. Como ejemplo, considérese la red tipo mariposa (figura 14), que puede caracterizarse como sigue [31].

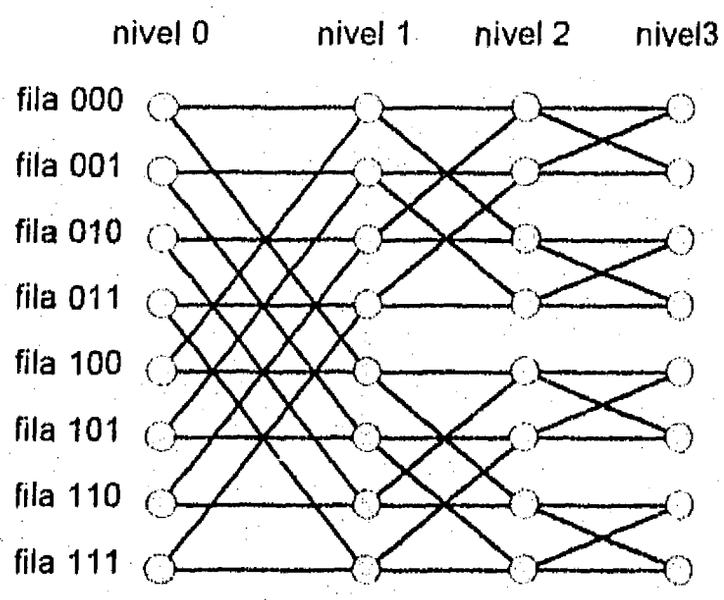


figura 14: La red mariposa de orden 3.

La red mariposa de orden r posee $(r + 1)2^r$ nodos y $r2^{r+1}$ arcos. Los nodos corresponden a parejas <w,i>, donde i es el nivel del nodo, $0 \leq i \leq r$, y w es un número de r bits que designa la fila en que se encuentra el nodo. Dos nodos, <w,i>, y <w',i'>, están unidos por un arco si y solo si i' = i + 1 y:

ya sea w = w', o, alternativamente,
w y w' difieren en el i-ésimo bit.

Nótese que la red está definida en términos de una gráfica. La relación entre esta gráfica y su realización como una red conmutada se obtiene agrupando cuartetas de nodos de la gráfica, interconectados por dos arcos horizontales y dos cruzados, en un elemento de conmutación. Así, los nodos <0,000>, <1,000>, <0,100> y <1,100> conforman el primer elemento de conmutación de la figura 10, etc. Algunas de las propiedades más importantes de esta red son las siguientes:

1.- La red permite la transmisión entre cualquier nodo de entrada <0,i> y de salida <(r + 1),j>. En efecto, la trayectoria entre un nodo de entrada y salida comprende r nodos. En cada uno de ellos se dispone de dos elecciones para la trayectoria, esto es, horizontal y cruzada. Por lo tanto, desde cualquier nodo de entrada se dispone de $2^r = N$ trayectorias distintas. Como éste es, también, el número de salidas, la proposición sigue. Por otra parte, y por la construcción de las trayectorias, se desprende que éstas son únicas (red tipo banyan).

2.- Para una red en la que a lo más un dato inicia en un nodo de entrada dado, y a lo más un dato está destinado a cualquier nodo de salida, el retardo que experimenta un dato al transitar entre la entrada y la correspondiente salida, debido a contiendas por un mismo nodo por otros datos, está acotado a $O(N^{1/2})$ unidades.

La prueba de la propiedad (2) se encuentra en [31]. En particular, para r = log N, r impar, el retardo máximo obtenido es $(3N^{1/2}/2^{1/2})$ - logN -2. Para logN par, se puede demostrar que el retardo máximo es $2N^{1/2}$ - logN - 2. Para permutaciones π:[1,N] $\rightarrow$ [1,N]se tiene, entonces, que el tiempo de tránsito está acotado a:

$$\log N \leq \tau_T \leq (3N^{1/2}/2^{1/2}) - 2, \quad \log N \text{ impar y}$$
$$\log N \leq \tau_T \leq (2N^{1/2}) - 2, \quad \log N \text{ par} \qquad \ldots\ldots 22$$

3.- Una consecuencia importante de la propiedad (2) es que la profundidad máxima de la cola asociada a cada nodo está acotada a $\theta(N^{1/2})$. Este valor puede crecer, desde luego, para asignaciones entrada/salida arbitrarias.

El impacto que posee una red escalonada en el tiempo de acceso a memoria depende de las magnitudes relativas de los retardos de propagación de la red, y de los tiempos de acceso a memoria. En las arquitecturas previamente consideradas, un número constante de interruptores está involucrado en cada acceso. En consecuencia, el retardo asociado a este interruptor puede ser agregado al tiempo de acceso de memoria y no requiere ser considerado explícitamente. Para redes escalonadas, el número de elementos de conmutación que intervienen en cada trayectoria depende del orden de la red. Por otra parte, se presentan dos fuentes de conflicto: contiendas por nodos en la red, y contiendas por un módulo de memoria determinado. Estas fuentes de conflicto claramente no son estadísticamente independientes. Se pueden distinguir dos casos:

a.- El tiempo de acceso a memoria es mayor al retardo asociado a cada elemento de conmutación. En este caso, colas pueden acumularse en frente de cada módulo de memoria, cuya longitud varía como función de las longitudes de cola existentes en los nodos de la red.

b.- El tiempo de acceso a memoria es igual (o menor) al retardo asociado a cada elemento de conmutación. En este caso, y como a lo más un dato puede ser transmitido por cada nodo de salida por unidad de tiempo, la formación de colas se limita a los nodos de la red, y el tiempo de acceso para cada dato es igual al tiempo de tránsito por la red.

El caso (a) no parece ser tratado en la literatura. En general, se asume que cada módulo de memoria incluye una memoria tipo FIFO, capaz de recibir los datos con la velocidad con que son transmitidos por la red. Por otra parte, para tecnologías modernas, el tiempo de acceso a memoria es comparable a los retardos de propagación de un elemento de conmutación. En consecuencia, se considerará únicamente el caso (b).

III.4.1 Accesos simultáneos.

No ha sido posible obtener una expresión matemática para el tiempo medio de tránsito por la red. En el apéndice II se describe un algoritmo de simulación, que ha sido utilizado para generar los resultados que se presentan más adelante. Se propone aquí una expresión empírica para el tiempo de tránsito, τ_c (véase, también, el punto III.2.1):

$$\tau_c = [1 + 2/3]\log N \qquad \ldots\ldots 23$$

La expresión anterior genera un error inferior al 3% comparado con los resultados de simulación reportados en el apéndice II. Para una red de p entradas y salidas, donde p es el número de procesadores y módulos de memoria, y considerando un tiempo de acceso a memoria unitario, la cantidad esperada de procesadores activos, P, toma la forma:

$$P = p\tau_E/[\tau_E + \tau_\mu(1 + 2/3)\log p] \qquad \ldots\ldots 24$$

Para 4 procesadores y $\tau_E = 10$, $\tau_\mu = 1$, de (24) se obtiene P = 3.0. Similarmente, para p = 16, resulta P = 9.6. Comparado con los resultados obtenidos para el multiprocesador de bus común (respectivamente 2.857 y 6.154), esta red es significativamente superior. Por otra parte, existe una clase de algoritmos importantes para la cual el tiempo de tránsito es logp. Para estos algoritmos, y para 4 y 16 procesadores, se obtiene respectivamente P = 3.3333 y P = 11.4286.

III.4.2 Accesos estadísticamente independientes.

Patel [26] sugirió un análisis probabilístico para redes escalonadas de tamaño $a^n \times b^n$, integrado por elementos de conmutación *crossbar* de a entradas y b salidas. Bajo la suposición de accesos a módulos de memoria uniformemente distribuidos, las solicitudes presentes en cualquier módulo están uniformemente distribuidas sobre b destinos. El número esperado de solicitudes que son transferidas a las b salidas puede ser obtenido, entonces, de la ecuación (20), substituyendo p y m por a y b, respectivamente. Dividiendo esta cantidad entre b, se obtiene la probabilidad de que existe una solicitud en una salida del elemento de conmutación, en función de la probabilidad de entrada:

$$P_i = 1 - [1 - (P_{i-1}/b)]^a, \quad 1 \leq i \leq n \qquad \ldots\ldots 25$$

La probabilidad de salida en la última etapa de la red determina el ancho de banda de la misma, esto es,

$$BW = P_n b^n \qquad \ldots\ldots 26$$

Kruskal y Snir [27] obtuvieron una expresión límite para la ecuación (25). Sea P_m la probabilidad de que existe un dato en una entrada dada de un elemento de conmutación de k entradas y salidas. Entonces:

$$P_m = 2k/[(k - 1)m + 2k/P_p \qquad \ldots\ldots 27$$

donde P_p es la probabilidad de que una solicitud es generada por un procesador. De las ecuaciones (26) y (27) se puede obtener el número esperado de procesadores activos y, para k = 2, resulta:

$$P = (\mu/\lambda)BW = 4p(\mu/\lambda)/[p + 4/\lambda] \qquad \ldots\ldots 28$$

donde p es el número de procesadores. Como ejemplo, para $\lambda = 0.1$, $\mu = 1$ y cuatro procesadores, de (28) se obtiene P = 3.6364. Similarmente, para p = 16, P = 11.4286.

La ecuación (28) constituye una cota superior estricta para redes que no contienen memorias propias, esto es, en caso de conflicto el dato a ser transmitido es seleccionado al azar, y los datos no seleccionados se pierden. Kruskal, Snir y Weiss [28] estudiaron el caso de elementos de conmutación con memoria propia (figura 13). Estos autores concluyen que los elementos de conmutación ubicados en las entradas de la red pueden ser modelados como un sistema de cola del tipo M/M/1. Para etapas posteriores, y debido a la existencia de memoria local en los elementos de conmutación, el principio de independencia estadística en el tráfico de la red no se cumple. Basados en observaciones del comportamiento de las estadísticas del tiempo de espera en cola en etapas posteriores, y postulando la existencia de una distribución límite para este tiempo, los autores obtienen una serie de expresiones aproximadas que concuerdan bien con resultados obtenidos por simulaciones. En particular, para

elementos de conmutación de 2 entradas y 2 salidas, el tiempo medio de espera en cola para la primera etapa, w_1, está dado por:

$$w_1 = \frac{\rho[2 - (3/2)\mu)]}{2\mu(1 - \rho)} \qquad \ldots\ldots 29$$

donde $\rho = \lambda/\mu$. Para etapas posteriores, los autores sugieren la expresión:

$$w_\infty \approx (1 + 2\lambda/5)w_1 \qquad \ldots\ldots 30$$

El tiempo total de servicio, τ_S, está dado, entonces, por $\tau_\mu \log p + w_1 + (\log p - 1)w_\infty$, donde τ_μ es el retardo asociado a cada elemento de conmutación:

$$\tau_S = \tau_\mu \log p + \left\{\frac{\rho[2 - (3/2)\mu)]}{2\mu(1 - \rho)}\right\}\left[1 + (\log p - 1)(1 + 2\lambda/5)\right] \qquad \ldots\ldots 31$$

Para $\mu = 1$, la expresión (31) reduce a:

$$\tau_S = \log p + \left\{\frac{\lambda}{4(1 - \lambda)}\right\}\left[1 + (\log p - 1)(1 + 2\lambda/5)\right] \qquad \ldots\ldots 32$$

y la cantidad esperada de procesadores activos puede ser calculada como:

$$P = p\tau_E/(\tau_E + \tau_S) \qquad \ldots\ldots 33$$

Para los valores utilizados en los ejemplos anteriores, y para p = 4, de la expresión (32) resulta $\tau_S = 2.0567$, y, de (33), P = 3.3177. Similarmente, para p = 16, se obtiene $\tau_S = 4.1144$, y P = 11.3359.

III.5 COMPARACION ENTRE MODELOS.

A continuación se presentan algunas gráficas correspondientes al valor esperado de procesadores activos para los diferentes modelos analizados. Se han discutido tres situaciones:

1.- Tiempos de ejecución iguales, y accesos simultáneos de todos los procesadores. El tiempo de acceso considerado es el valor medio del tiempo requerido para el último dato en ser escrito (o leído). Esto es, se considera que existe dependencia de datos entre procesos.
2.- Tiempos de ejecución iguales, y tiempos de acceso dados por el valor promedio experimentado por cada procesador. Cada procesador puede iniciar un segundo proceso al concluir su respectivo acceso a memoria, por lo que el instante de inicio de un segundo proceso será, en general, distinto para cada procesador.

elementos de conmutación de 2 entradas y 2 salidas, el tiempo medio de espera en cola para la primera etapa, w_1, está dado por:

$$w_1 = \frac{\rho[2 - (3/2)\mu)]}{2\mu(1 - \rho)} \qquad \ldots\ldots 29$$

donde $\rho = \lambda/\mu$. Para etapas posteriores, los autores sugieren la expresión:

$$w_m \approx (1 + 2\lambda/5)w_1 \qquad \ldots\ldots 30$$

El tiempo total de servicio, τ_S, está dado, entonces, por $\tau_\mu \log p + w_1 + (\log p - 1)w_m$, donde τ_μ es el retardo asociado a cada elemento de conmutación:

$$\tau_S = \tau_\mu \log p + \left\{\frac{\rho[2 - (3/2)\mu)]}{2\mu(1 - \rho)}\right\}\left[1 + (\log p - 1)(1 + 2\lambda/5)\right] \qquad \ldots\ldots 31$$

Para $\mu = 1$, la expresión (31) reduce a:

$$\tau_S = \log p + \left\{\frac{\lambda}{4(1 - \lambda)}\right\}\left[1 + (\log p - 1)(1 + 2\lambda/5)\right] \qquad \ldots\ldots 32$$

y la cantidad esperada de procesadores activos puede ser calculada como:

$$P = p\tau_E/(\tau_E + \tau_S) \qquad \ldots\ldots 33$$

Para los valores utilizados en los ejemplos anteriores, y para p = 4, de la expresión (32) resulta $\tau_S = 2.0567$, y, de (33), P = 3.3177. Similarmente, para p = 16, se obtiene $\tau_S = 4.1144$, y P = 11.3359.

III.5 COMPARACION ENTRE MODELOS.

A continuación se presentan algunas gráficas correspondientes al valor esperado de procesadores activos para los diferentes modelos analizados. Se han discutido tres situaciones:

1.- Tiempos de ejecución iguales, y accesos simultáneos de todos los procesadores. El tiempo de acceso considerado es el valor medio del tiempo requerido para el último dato en ser escrito (o leído). Esto es, se considera que existe dependencia de datos entre procesos.
2.- Tiempos de ejecución iguales, y tiempos de acceso dados por el valor promedio experimentado por cada procesador. Cada procesador puede iniciar un segundo proceso al concluir su respectivo acceso a memoria, por lo que el instante de inicio de un segundo proceso será, en general, distinto para cada procesador.

3.- Arribos de solicitudes de acceso acorde a un proceso de Poisson, con tiempos de ejecución y de acceso exponencialmente distribuidos. El momento de inicio de un proceso no está correlacionado con el inicio de otros procesos.

Las situaciones (1) y (3) pueden considerarse como extremas con relación a situaciones reales, por lo que se usarán como base para las comparaciones. La figura 15 muestra la cantidad esperada de procesadores activos P con base en la ecuación (1) para $\tau_m = 1$, y valores para τ_E de 5, 10 y 20. El caso de accesos estadísticamente independientes (ecuaciones 4 y 5) se muestra en la figura 16, donde ρ es el tráfico en el bus, definido como $\rho = \lambda/\mu$. Nótese que P tiende a $1/\rho$ conforme el número de procesadores, p, crece.

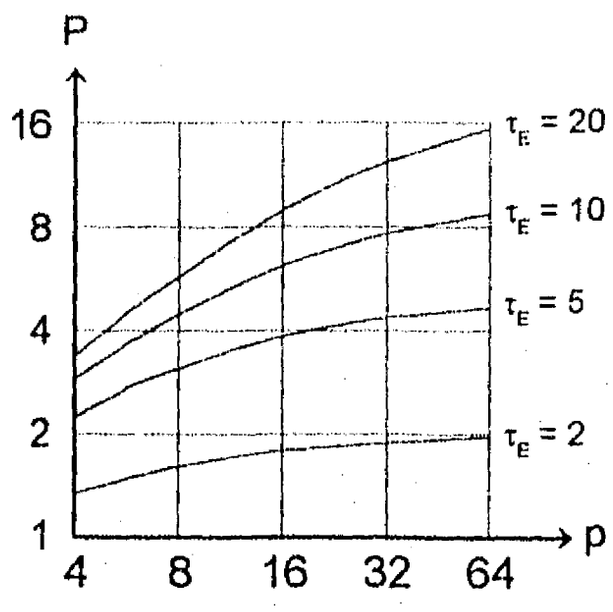


figura 15: Número de procesadores activos, multiprocesador de bus común, accesos simultáneos

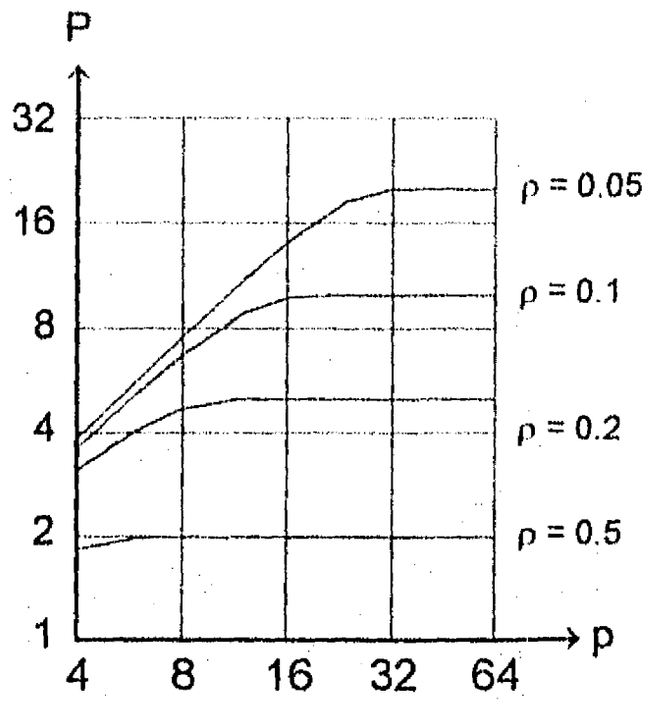


figura 16: Número de procesadores activos, multiprocesador de bus común, accesos exp. distribuidos

El multiprocesador de buses múltiples requiere un total de b(p + m) interruptores, como se desprende de la figura 6, en la que cada bus es conectado, por una parte, a cada procesador y, por otra, a cada módulo de memoria. En particular, el esquema de la figura 6 requiere 30 interruptores.

Por otra parte, la red crossbar requiere mp interruptores. En general, el esquema de buses múltiples es más económico que una red crossbar solamente si se cumple la relación $b(p + m) < mp$. Para el caso de que m = p, se obtiene $(2b/p) < 1$. Las figuras 17 y 18 muestran el valor de P para una máquina de 2 buses, para el caso de accesos simultáneos y exponencialmente distribuidos, respectivamente. Para estas arquitecturas, P tiende a b/ρ para valores grandes de p. En la figura 18 se graficaron las cotas inferior y superior para P, de acuerdo con las ecuaciones dadas en el punto III.2.2.

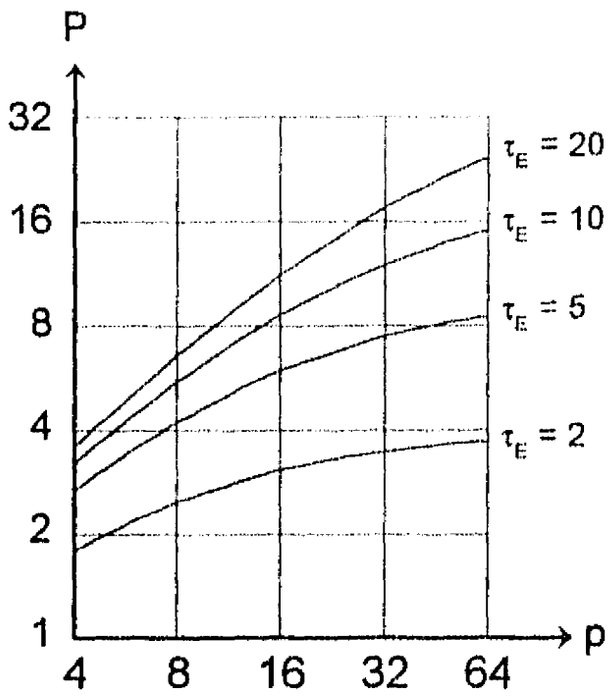


figura 17: Número de procesadores activos del multiprocesador de 2 buses, accesos simultáneos

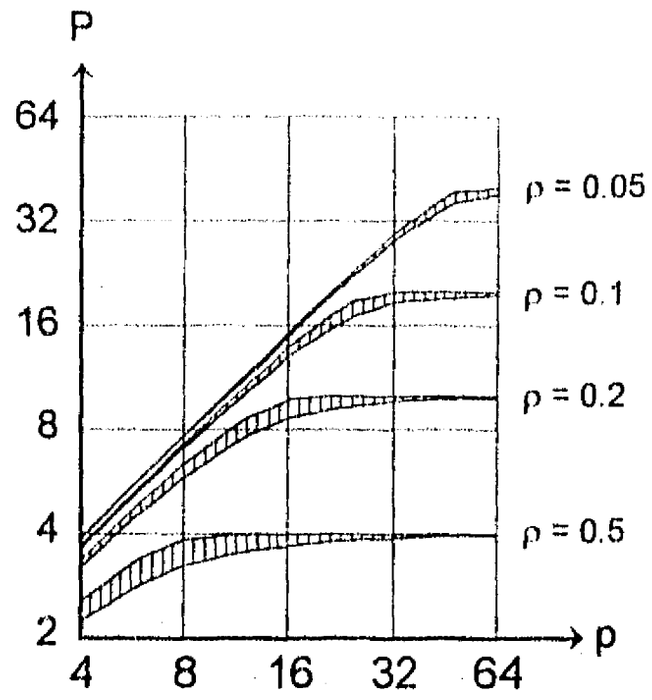


figura 18: Número de procesadores activos del multiprocesador de 2 buses, accesos exp. distribuidos

Las figuras 19 y 20 corresponden al caso en que la relación entre procesadores y buses es constante (aquí igual a 4). Para la relación indicada, y comparado con una red *crossbar*, una máquina de buses múltiples requiere la mitad de interruptores.

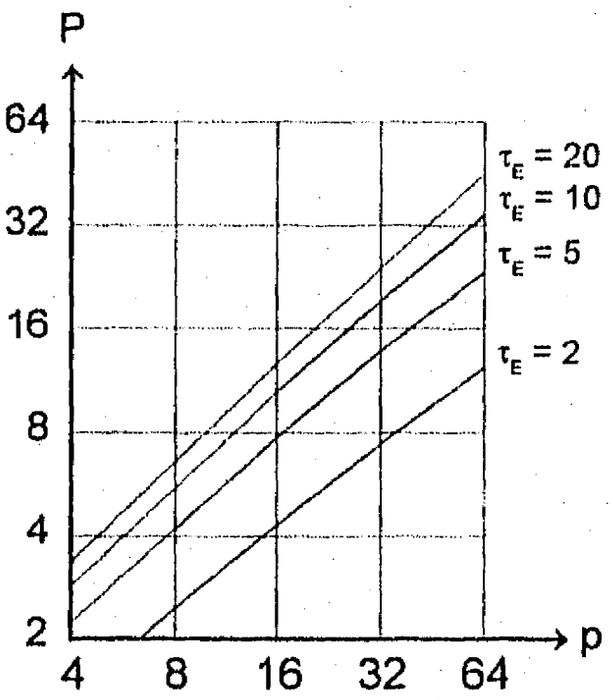


figura 19: Número de procesadores activos, multiprocesador de p/4 buses, accesos simultáneos

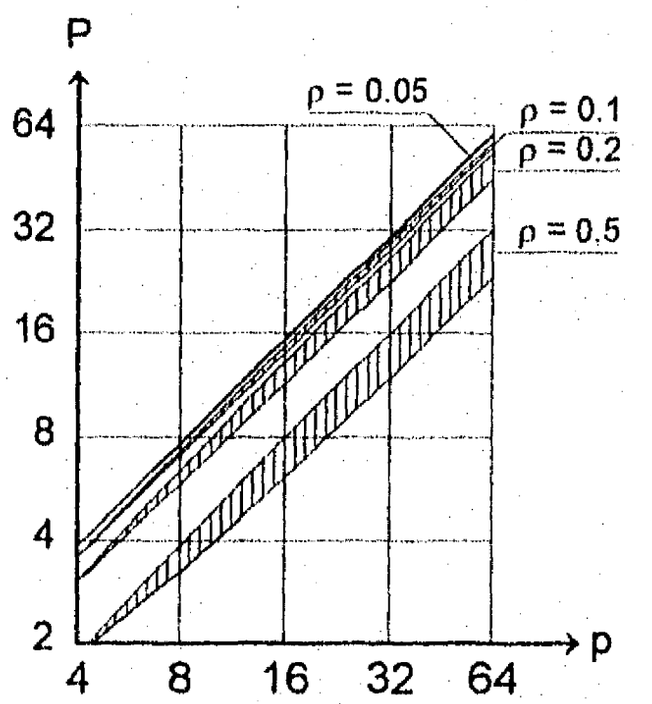


figura 20: Número de procesadores activos, multiprocesador de p/4 buses, accesos exp. distribuidos

El valor esperado de P para una red *crossbar* se muestra en las figuras 21 y 22. Finalmente, las figuras 23 y 24 reproducen este valor para una red escalonada, en particular, para la red mariposa.

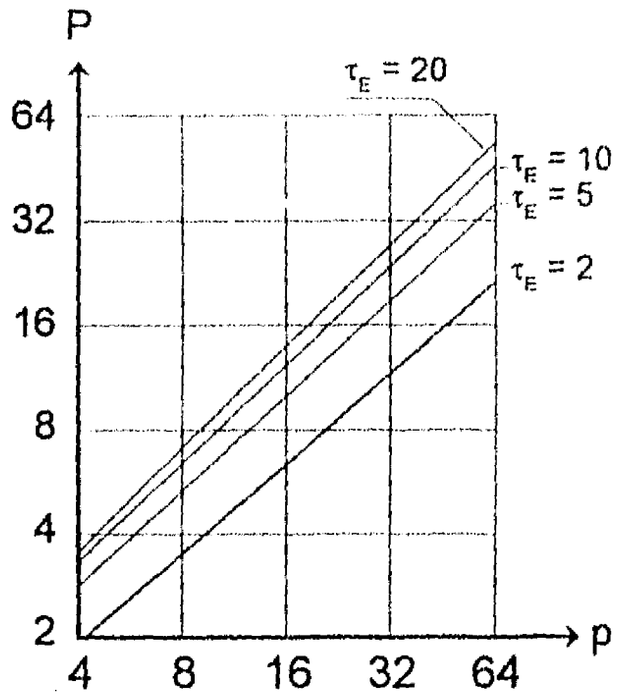


figura 21: Número de procesadores activos
red *crossbar*,
accesos simultáneos

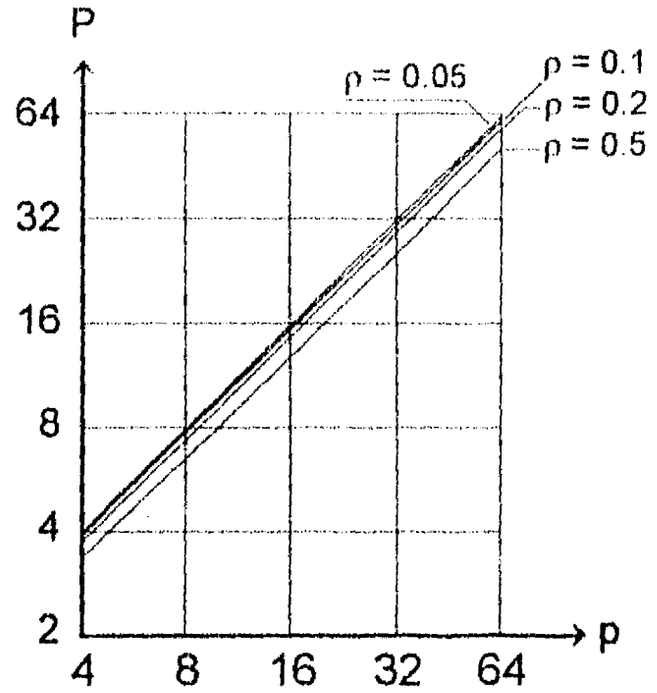


figura 22: Número de procesadores activos
red *crossbar*,
accesos exp. distribuidos

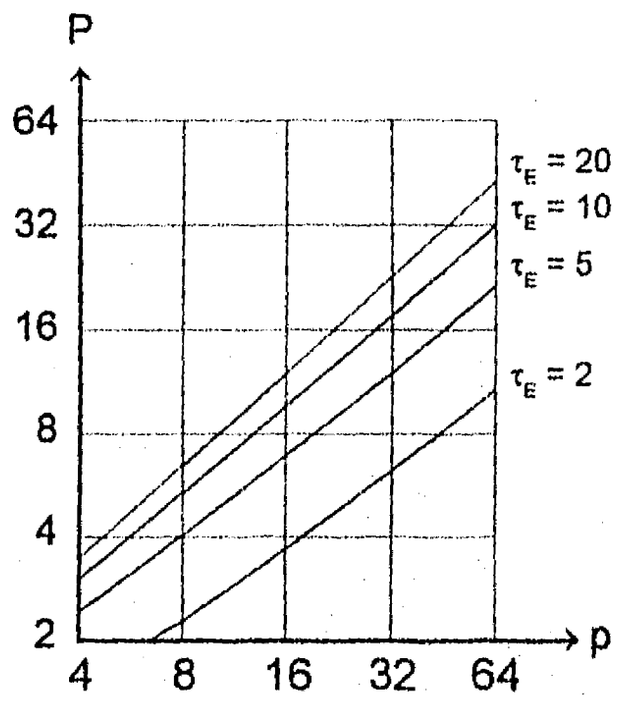


figura 23: Número de procesadores activos
red escalonada,
accesos simultáneos

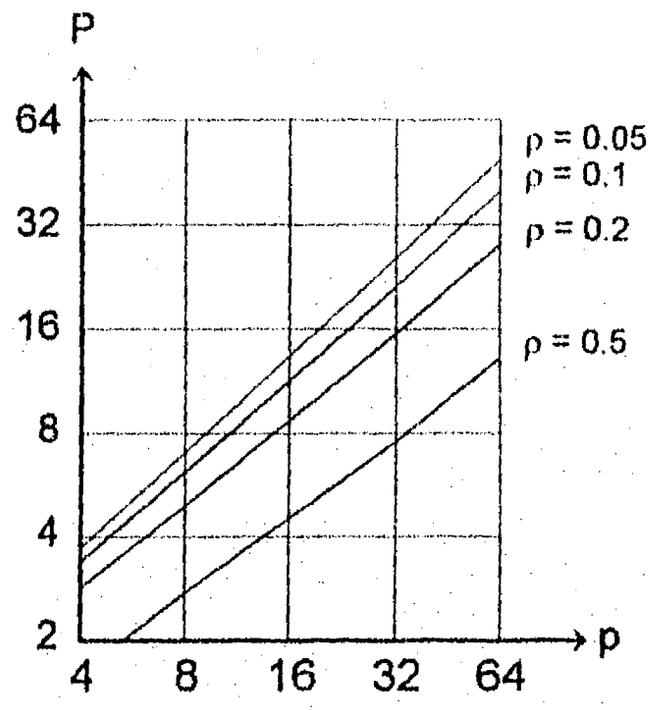


figura 24: Número de procesadores activos
red escalonada,
accesos exp. distribuidos

Las figuras 20 y 22 muestran la superioridad de las arquitecturas de buses múltiples y crossbar con relación a la de bus común y red escalonada, tanto en lo que al valor de P se refiere para un número de procesadores dado, como en la menor dependencia de este valor con respecto a la intensidad de tráfico. Con objeto de facilitar la comparación, las figuras 25 - 30 muestran la relación η entre el valor de P para las diferentes arquitecturas consideradas, con respecto al correspondiente valor obtenido para la red crossbar. Para las arquitecturas de buses múltiples, se consideró la media aritmética entre las cotas superior e inferior. Para estas arquitecturas y para accesos exponencialmente distribuidos, η es aproximadamente constante con respecto al número de procesadores. La tabla 1 reproduce los valores de η para las relaciones p/b = 8 y p/b = 4.

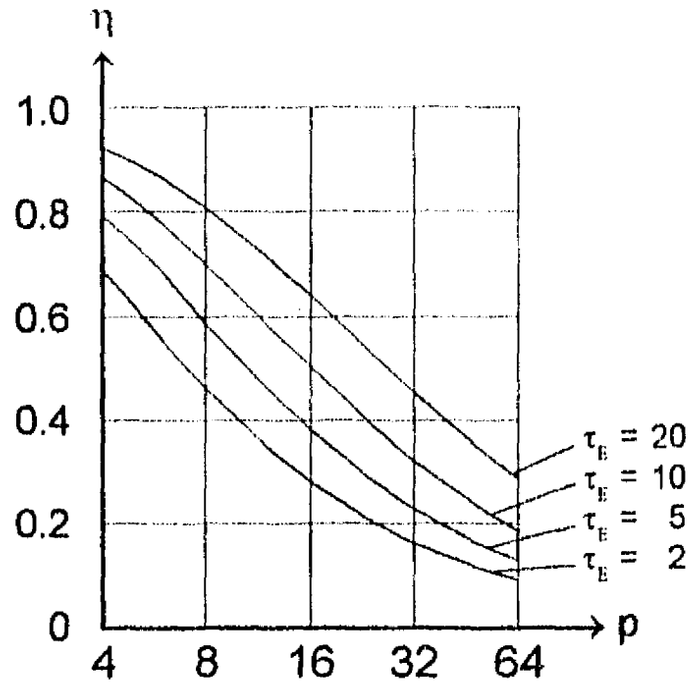


figura 25: Relación entre rendimientos, multiprocesador de bus común, accesos simultáneos

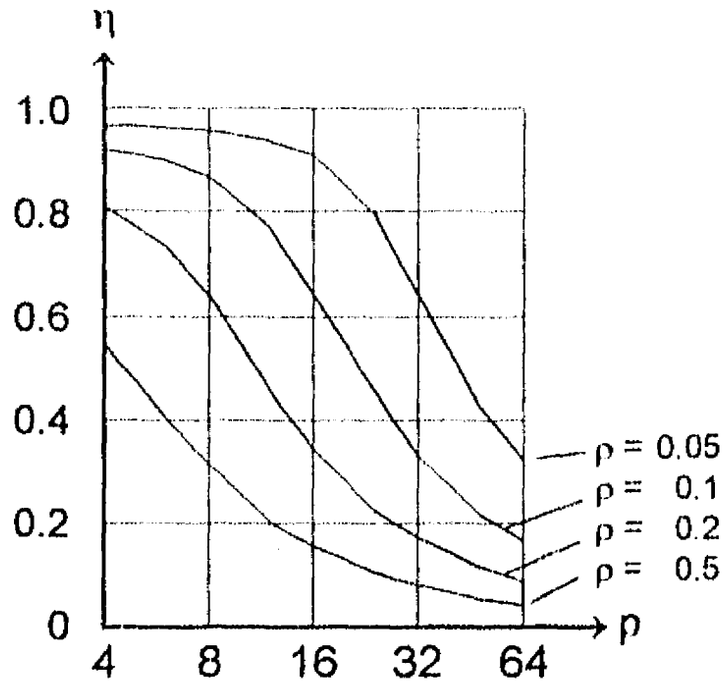


figura 26: Relación entre rendimientos, multiprocesador de bus común, accesos exp. distribuidos

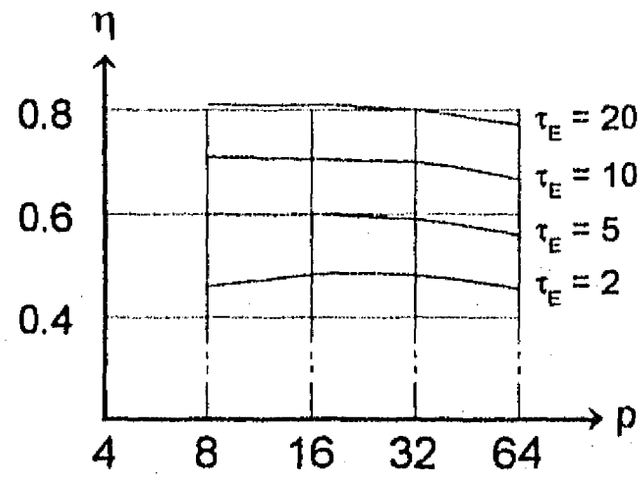


figura 27: Relación entre rendimientos, multiprocesador de p/8 buses, accesos simultáneos

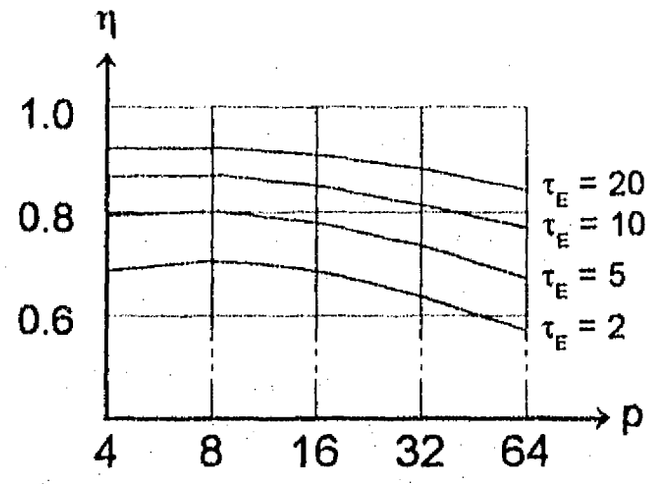


figura 28: Relación entre rendimientos, multiprocesador de p/4 buses, accesos simultáneos

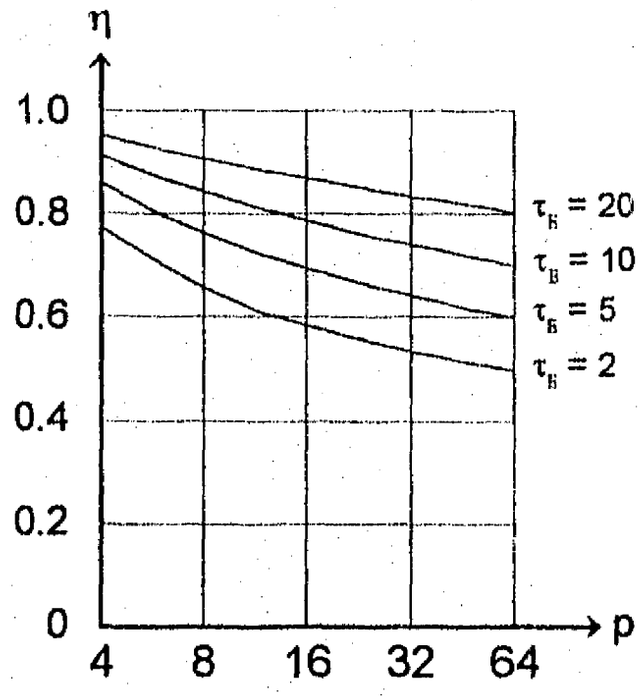


figura 29: Relación entre rendimientos, red escalonada, accesos simultáneos

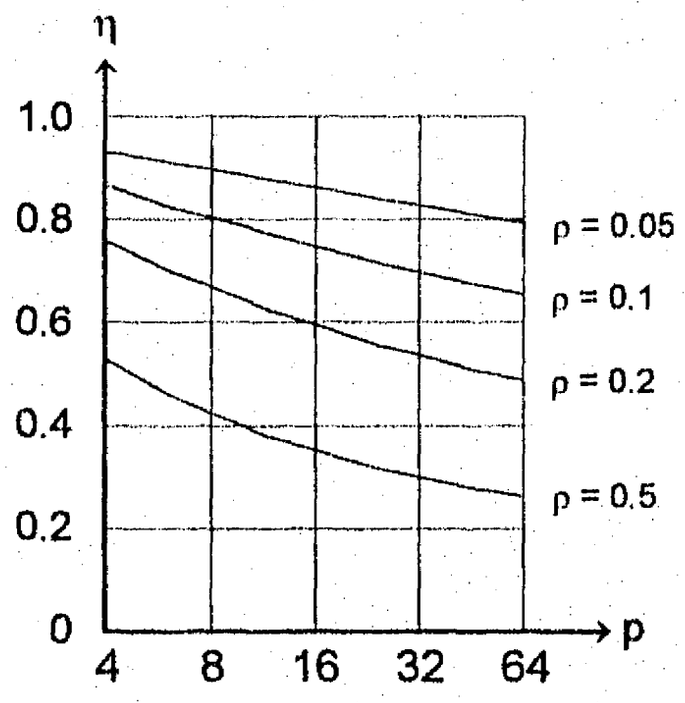


figura 30: Relación entre rendimientos, red escalonada, accesos exp. distribuidos

Tabla 1: Valores de η para el multiprocesador de buses múltiples

ρ	p/b = 8	p/b = 4
0.5	0.49	0.55
0.2	0.63	0.84
0.1	0.89	0.93
0.05	0.96	0.97

De las figuras 25 - 30, así como de la tabla 1, se desprende que la red *crossbar* es superior a las demás arquitecturas consideradas, especialmente para altas intensidades de tráfico. Por otra parte, y con excepción de las redes conmutadas escalonadas, solamente se han considerado accesos uniformemente distribuidos sobre todos los módulos. Para arquitecturas de buses múltiples y para la red *crossbar*, las permutaciones posibles sobre la secuencia <1, 2, ..., p> constituyen direccionamientos óptimos, en el sentido de que los accesos pueden ser ejecutados en tiempo constante para cualquier relación p/b. En particular, para la red *crossbar* este tiempo es igual al tiempo de acceso unitario. Para un número de procesadores dado, y para una intensidad de tráfico esperada en función al tipo de aplicaciones de la máquina, la cuestión relativa a la "mejor" arquitectura se puede, entonces, responder con base en una comparación entre el nivel de complejidad de las mismas, y de su rendimiento relativo.

En este sentido, es necesario resaltar que el modelo probabilístico proporciona información sobre el nivel de utilización esperado de los procesadores, pero es poco adecuado para responder a otras preguntas que pueden ser relevantes para una aplicación dada. Esto es, el modelo es útil para situaciones en las que una serie de programas son ejcutados concurrentemente, con relativamente poca comunicación y poca dependencia de control entre los mismos. Pero supóngase que se desea estimar el tiempo de ejecución de un programa ejecutado en paralelo en más de un procesador. En este caso, p procesadores inician la ejecución del programa en forma simultánea, y el resultado del cómputo no está completo sino hasta que el último procesador ha concluido la ejecución del programa. Supóngase que las mismas condiciones impuestas al modelo probabilístico son satisfechas, con la excepción del instante de inicio de ejecución. Esto es, supóngase que los tiempos de ejecución son variables aleatorias exponencialmente distribuidas, con igual media λ. Entonces, $pr(\tau_E) = \lambda e^{-\lambda t}$, $t \geq 0$, es la función de densidad de probabilidad de τ_E. Bajo estas condiciones, la probabilidad de que un procesador concluya en un tiempo τ_E, $r \leq \tau_E \leq s$, es $Pr(\tau_E) = e^{-\lambda r} - e^{-\lambda s}$.

Bajo la interpretación de $pr(\tau_E)$ como la frecuencia relativa de ocurrencia de eventos, y para k repeticiones del experimento en p procesadores, el tiempo promedio de ejecución de cada procesador puede graficarse como un histograma de p columnas, delimitadas por valores de t tales que la probabilidad de ocurrencia de un evento es 1/p. Entonces, la probabilidad de que un evento corresponda a una columna dada es la misma para todas las columnas, y, para k repeticiones cuando $k \to \infty$, el número esperado de eventos en cada columna es k. Los eventos de mayor duración están,

entonces, ubicados en el intervalo limitado por $r = (1/\lambda)\ln p$ y $s = \infty$. Sea t_M una variable aleatoria exponencialmente distribuida, $(1/\lambda)\ln p \leq t_M \leq \infty$. Bajo la substitución $\tau = t - (1/\lambda)\ln p$, la función de densidad de probabilidad de t_M toma la forma:

$$p_r(\tau_M) = \frac{1}{p}\left[\frac{1}{\lambda} e^{-\lambda\tau}\right] \qquad \ldots\ldots 34$$

El valor esperado de $p_r(\tau_M)$ es $1/p\lambda$., de donde se desprende que el valor esperado para el mayor tiempo de ejecución, $E[(t_M)$, es:

$$E[(t_M) = \frac{1}{\lambda}(\ln p + \frac{1}{p}) \qquad \ldots\ldots 35$$

Como ejemplo, considérese una máquina de 8 procesadores y $\lambda = 0.1$. Substituyendo en (35), se obtiene $E[(t_M) = 22.04$, esto es, más del doble del tiempo de ejecución promedio ($\tau_E = 1/\lambda$). Está claro que este resultado es poco útil para estimar el tiempo de conclusión de un algoritmo paralelo, para el cual el tiempo de ejecución es similar en todos los procesadores. Por otra parte, el modelo de accesos simultáneos, que presupone tiempos de ejecución idénticos para todos los procesadores, conduce a tiempos de ejecución totales dados por $\tau_E + \tau_c$, donde τ_c es el tiempo medio de acceso a memoria compartida (véase, también, el punto III.2.1). Típicamente, la necesidad de sincronizar a los procesadores entre sí (dependencia de datos y/o de control) se presenta únicamente después de una serie de accesos a memoria compartida por cada procesador, por lo que este modelo es claramente pesimista. Finalmente, y aun para tiempos de ejecución iguales, la contienda por recursos comunes tiende a dispersar el inicio de un segundo período de procesamiento entre procesadores, posterior a un acceso a memoria compartida. Esta dispersión se traduce en una menor probabilidad de contienda para accesos subsecuentes. Un modelo más realista, entonces, tiende a asumir formas complejas, que no poseen solución conocida. Por otra parte, es de dudarse que una posible solución sea de utilidad general, en virtud de que los parámetros involucrados dependen de algoritmos específicos. A continuación, se analizan dos casos concretos.

III.5.1 LA TRANSFORMADA RAPIDA DE FOURIER.

La familia de algoritmos conocida colectivamente como transformada rápida de Fourier ha sido estudiada extensamente [29,30,31]. Uno de ellos, conocido por "decimación en tiempo", se ilustra en la figura 31. Para una secuencia de N valores de entrada, $N = 2^L$, L un entero positivo, el algoritmo consiste en la ejecución de logN iteraciones, donde cada iteración comprende N/2 operaciones conocidas como "mariposa". Esta operación se ilustra en la figura 32, en la cual A, B y W son, en general, números complejos. Los resultados parciales generados en cada iteración son los valores de entrada para la subsecuente, de acuerdo con la gráfica de la figura 31. Si i es la iteración, i = 0, 1, ..., logN - 1, entonces en cada iteración existen $N/2^{i+1}$ grupos, donde cada grupo consta de 2^i mariposas. Sea g el número del grupo de la i-ésima iteración, g = 0, 1, ..., $(N/2^{i+1}) - 1$, y c el número de la mariposa dentro

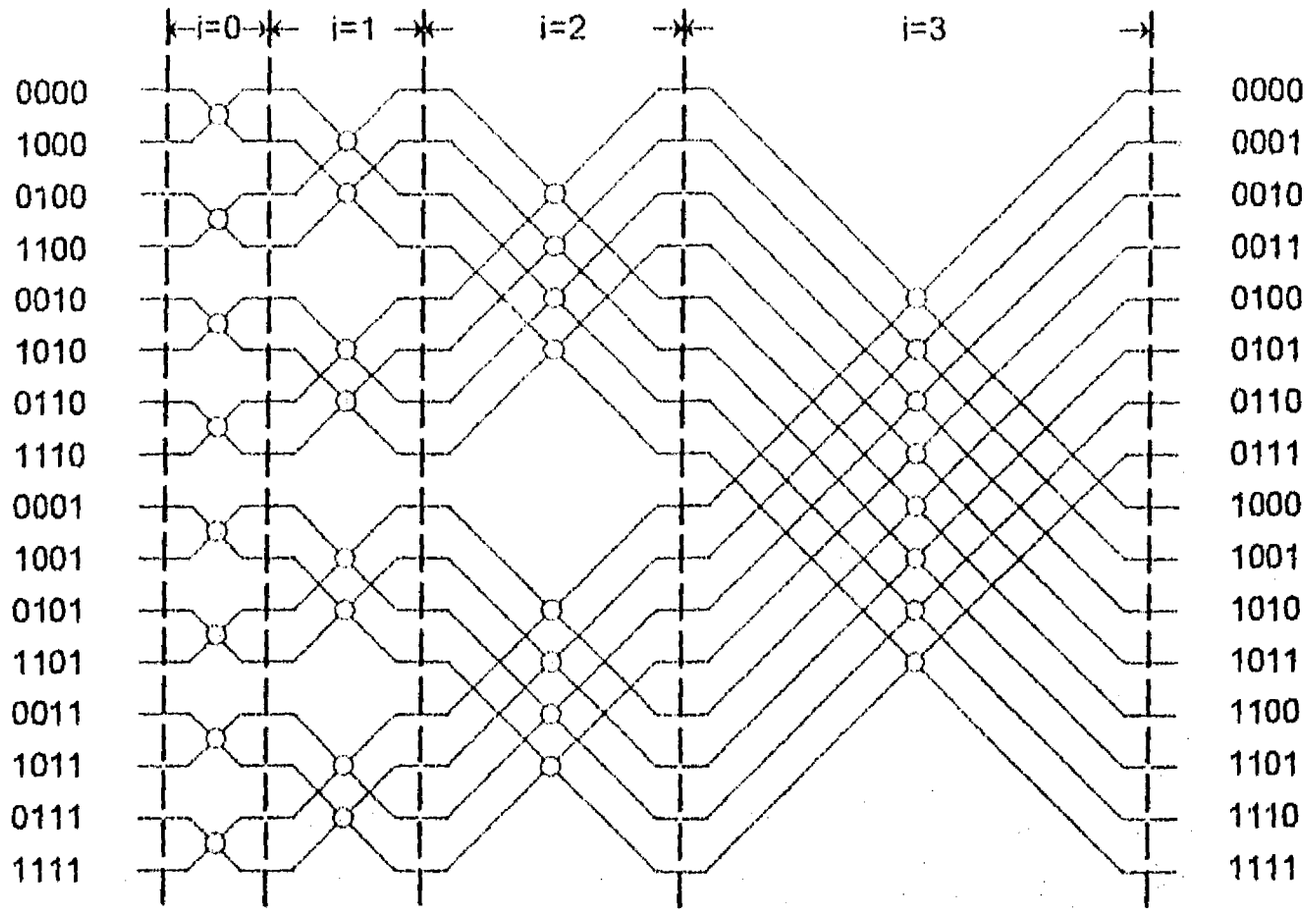


figura 31: El algoritmo de decimación en tiempo para N = 16

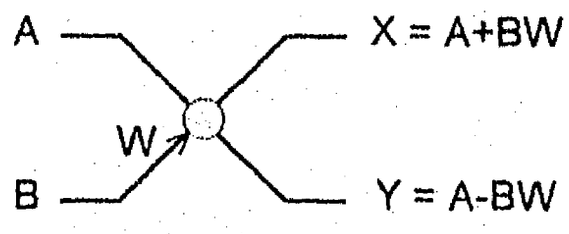


figura 32: La operación mariposa.

de ese grupo, c = 0, 1, ..., 2^i- 1. Entonces, la ubicación de A en la secuencia de entrada, $D_A(i,g,c)$, está dada por:

$$D_A(i,g,c) = c + g2^{i+1} \quad \text{..... 36}$$

Similarmente, la ubicación de B, $D_B(i,g,c)$, puede obtenerse como:

$$D_B(i,g,c) = c + (2g + 1)2^i = D_A(i,g,c) + 2^i \quad \text{..... 37}$$

Sea <W> una secuencia de N elementos. Entonces, a cada mariposa corresponde el elemento de la secuencia ubicado en la posición $D_W(i,c)$ dada por:

$$D_W(i,c) = [cN/2^{i+1}]\big|_{\text{mod } N/2} \quad \text{..... 38}$$

Finalmente, sea <S> = <s_0, s_1, ..., s_{N-1}> la secuencia a la que se desea aplicar el algoritmo. La secuencia de entrada <D> es obtenida por medio de una permutación de <S>, tal que la ubicación de cada elemento en la secuencia <D> corresponde a la representación binaria de su ubicación en la secuencia <S>, invertida bit a bit (véase, también, la figura 31).

De la figura 32 se desprende que, dados A, B y W complejos, cada mariposa consta de 10 operaciones aritméticas: 4 multiplicaciones, una suma y una resta para generar el producto BW, y dos sumas y restas adicionales para generar X y Y, respectivamente. Para la subsecuente evaluación de tiempos de ejecución, se supondrá lo siguiente:

1.- El programa, la tabla de coeficientes W, así como una tabla de conversión de direcciones para los datos de entrada, se encuentran en la memoria local de cada procesador. Los datos de entrada se encuentran en memoria compartida, y los resultados son escritos a esa memoria.
2.- Las operaciones aritméticas son efectuadas en notación de punto decimal flotante. El tiempo de ejecución es el mismo para los tres tipos de operación involucrados, y comparable al tiempo de acceso a memoria compartida.
3.- El cálculo de direcciones, así como la ejecución de instrucciones de control, no requiere tiempo adicional a la ejecución de las instrucciones aritméticas. Lo anterior es razonable para procesadores modernos que cuentan con una unidad aritmética de punto decimal flotante, que opera de manera concurrente, e independientemente de la, o las, unidades aritméticas y lógicas [32]. El tiempo de acceso a memoria local se considera despreciable.
4.- El formato de los datos es el de palabras de 32 bits. Un dato complejo requiere dos palabras para su representación. Las transferencias entre procesador y memoria compartida son efectuadas con base en palabras individuales.
5.- La cantidad de procesadores p que se considerará es 1, 2, 4, ..., N/2. La memoria compartida es dividida en p módulos. El espacio de direccionamiento de cada módulo es np + PIN, donde PIN es el número de identificación del procesador, $0 \leq PIN \leq (p - 1)$, y $n = 0, 1, 2, ...$

Tiempo de ejecución del uniprocesador.

Una máquina de un sólo procesador ejecuta, en general, N lecturas de datos complejos, esto es, 2N lecturas. El algoritmo consta de (N/2)logN mariposas. Los resultados intermedios pueden ser escritos a memoria local (memoria caché) en cero tiempo. Los resultados finales son escritos en 2N tiempos a memoria principal. Si el tiempo de ejecución de una mariposa es τ_E (aquí, $\tau_E = 10$), el tiempo total requerido por el algoritmo es:

$$T_U = 4N + \tau_E(N/2)\log N \qquad \text{..... 39}$$

Tiempo de ejecución del multiprocesador de bus común.

De la figura 31 se desprende que, para cada iteración $i \geq 1$, de los dos valores de entrada requeridos para el cálculo de una mariposa, el procesador afectado dispone de uno de ellos en su propia memoria local. Esto es, de los dos valores X y Y generados por un procesador en una iteración dada, solamente uno de ellos es compartido con otro procesador. El valor a escribir es Y, si el i-ésimo bit del PIN es 0, y X en caso contrario. Supóngase que se dispone de N/2 procesadores. Entonces, la dirección destino de este valor es:

$D_D = PIN + 2^i$, si el i-ésimo bit de PIN = 0 y
$D_D = PIN - 2^i$, en caso contrario. 40

De (40) se desprende de inmediato que las N/2 direcciones generadas durante una iteración dada son todas diferentes. Para el caso de que p < N/2, supóngase que la secuencia de N/2 mariposas de cada iteración es dividida en p subsecuencias, y que las p mariposas de cada subsecuencia son ejecutadas en paralelo. En general, la ejecución de una mariposa requiere dos accesos de lectura y escritura a memoria compartida. Para i = 0, cada mariposa requiere 4 accesos de lectura. También, para i = logN - 1, cada mariposa genera 4 accesos de escritura. Para estimar el efecto de contienda de bus, supóngase que la máquina opera con base en un esquema de prioridad rotatorio, en el cual el primer procesador atendido recibe la última prioridad para un acceso posterior. Si la secuencia de accesos, para la primera lectura de datos (i = 0) es 0, 1, ..., p-1 (de hecho, el orden inicial con que se atienden las solicitudes en la primera subsecuencia es irrelevante), la ejecución del algoritmo puede representarse como se indica en la figura 33 (E = escritura, L = lectura de memoria compartida). El tiempo de ejecución de una subsecuencia es $\tau_M = (3p + 1) + \tau_E$. Para i = 0, así como para i = logN - 1, el tiempo de ejecución se eleva en 2p unidades por las lecturas y escrituras adicionales requeridas. Nótese que no se requiere de sincronización entre subsecuencias, ni entre iteraciones, dado que el esquema de priorización de accesos garantiza que la ejecución de cada subsecuencia está efectivamente concluida antes del inicio de la siguiente. En virtud de que existen N/2p subsecuencias por cada iteración, el tiempo de ejecución total está dado por:

$$T_{BC} = [(N/2p)\log N][(3p + 1) + \tau_E] + 2N \quad \text{..... 41}$$

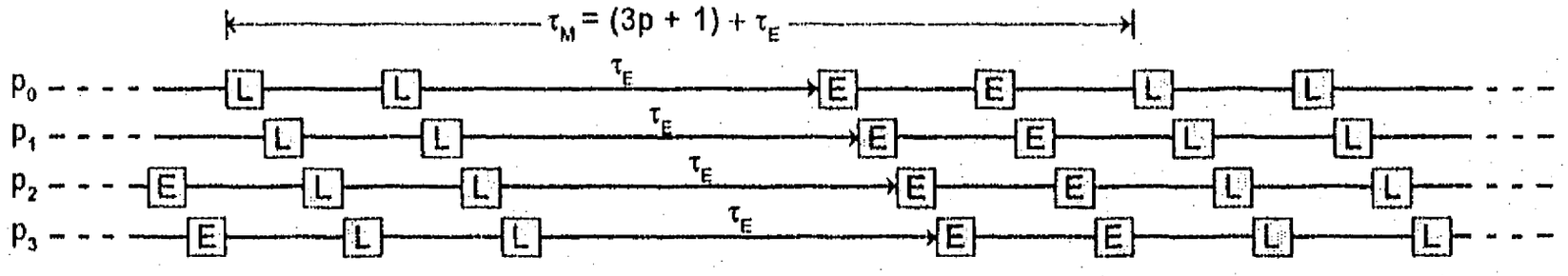


figura 33: Secuencia de ejecución para el multiprocesador de bus común, p = 4

Tiempo de ejecución de multiprocesadores con memoria compartida segmentada.

Para las arquitecturas en las cuales la memoria compartida está segmentada, esto es, dividida en módulos, y para la asignación de espacios de direccionamiento dada en la suposición 5 (pág. 43), la lectura inicial de datos genera una cantidad de solicitudes de acceso a un mismo módulo dada por min[p, N/2p]. Por ejemplo, de la figura 31 se desprende que para una máquina de cuatro procesadores, las primeras cuatro parejas de datos se encuentran en los módulos 0 y 2. Los accesos a memoria compartida en iteraciones subsecuentes, en cambio, no generan contienda por módulos. Para p < N/2 y bajo la substitución de PIN por (kp + PIN), k = 0, 1, ..., (N/2p) - 1, de (40) resulta:

$D_D = kp + PIN + 2^i$, si el i-ésimo bit de $kp + PIN = 0$ y
$D_D = kp + PIN - 2^i$, en caso contrario. 42

Si $(PIN \pm 2^i) \leq p$, entonces $D_D \mid_{mod\ p} = (PIN \pm 2^i)$ y se aplica la consideración hecha para (40).
Para $2^i \geq p$, $D_D \mid_{mod\ p} = PIN$ y la conclusión es obvia.

La generación de la secuencia de datos de entrada en orden invertido bit a bit, requerida por el algoritmo, puede evitarse si el orden de ejecución de las mariposas de la primera iteración es alterado. Supóngase que se ejecutan en paralelo p mariposas correspondientes a las parejas de datos (k+n, k+n+1) para una n dada, donde:

$k = 0, N/p, 2N/p, ..., (p - 1)(N/p)$, y $n = 0, 1, ..., (N/2p) - 1$.

Por ejemplo, para $p = 4$ y $N = 16$, se ejecutan en paralelo las mariposas correspondientes a las parejas de datos (0,1), (4,5), (8,9) y (12,13). Después, se ejecutan en paralelo las mariposas para (2,3), (6,7), (10,11) y (14,15). Esta secuencia de ejecución no genera contienda por módulos, por el siguiente argumento. Sea $j = \log p$, $k = \log N$, $k > j$. Con respecto a la secuencia de datos invertida, j asume la posición k - j. Pero, para $n = k - j$, existen 2^n datos en la secuencia invertida cuya posición en la secuencia está dada por un número cuya representación binaria posee los mismos valores para los bits 0, 1, ..., n - 1. Consecuentemente, si mariposas son ejecutadas sobre datos cuya ubicación se encuentra a una distancia $r2^n$, $r = 0, 1, ..., j - 1$, la representación binaria de su ubicación difiere en los bits j, j + 1, ..., k para toda r, y la conclusión sigue.

Para un valor de N, y una cantidad de procesadores p dada, el tiempo de ejecución del algoritmo depende, entonces, del número de buses con que cuenta la arquitectura. Para el caso de una red tipo mariposa, se puede probar [31] que el esquema de direccionamiento inherente al algoritmo no genera contienda por nodos. Por otra parte, bajo el mismo esquema de priorización de accesos usado para el caso del multiprocesador de bus común, en general no se requiere de sincronización entre ejecuciones de subsecuencias o iteraciones. Para la arquitectura de buses múltiples, la sincronización es requerida solamente entre la primera y segunda iteraciones. El tiempo de ejecución se obtiene como:

$T_{BM} = [(N/2p)\log N][2(1 + p/b) + \tau_E] + N/b$ 43

Para la red crossbar, la expresión anterior se reduce a:

$T_{CR} = [(N/2p)\log N][4 + \tau_E] + N/p$ 44

Finalmente, para la red mariposa se obtiene:

$T_{RM} = [(N/2p)\log N][4\log p + \tau_E] + (N/p)\log p$ 45

La figura 34 reproduce el incremento de velocidad logrado por las arquitecturas consideradas con respecto al uniprocesador para N = 1024. Para la arquitectura de buses múltiples se consideró la relación p/b = 4, por lo que el mínimo valor de p graficado, para este caso, es 8. En la figura 35 se graficó el rendimiento de las arquitecturas consideradas, con relación al rendimiento de la red tipo *crossbar*.

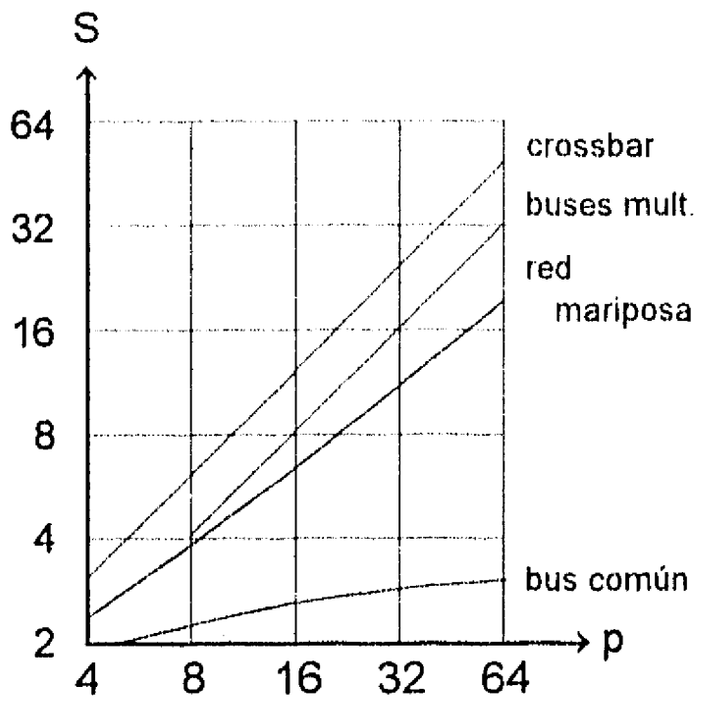


figura 34: Incremento de velocidad, algoritmo de decimación en tiempo.

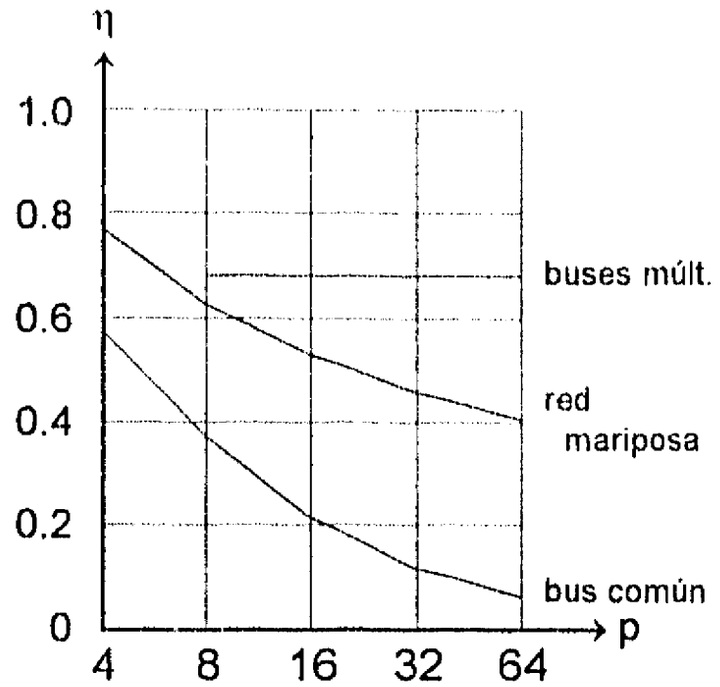


figura 35: Relación entre rendimientos, algoritmo de decimación en tiempo.

III.5.2 EL ALGORITMO DE ORDENAMIENTO POR FUSION PAR - IMPAR.

De entre los algoritmos de ordenamiento deterministas conocidos, el de fusión par - impar es, tal vez, el más utilizado en la actualidad. Originalmente descrito por Batcher [33] hace aproximadamente 30 años, el algoritmo ordena una lista de N entradas en $O(\log^2 N)$ unidades de tiempo. Aunque existen algoritmos de mejor comportamiento asintótico, como el descrito por Cypher y Plaxton [34], ninguno puede competir en términos de sencillez y elegancia con el de fusión par - impar, y no resultan particularmente eficientes para $N < 2^{20}$ [31]. En la descripción subsecuente, se asumirá que N es de la forma 2^L, L un entero positivo.

El algoritmo divide la lista de N entradas en N sublistas de longitud 1. Posteriormente, a partir de parejas de sublistas de longitud 1, integra N/2 sublistas ordenadas de longitud 2, N/4 sublistas ordenadas de longitud 4, etc. La clave del algoritmo reside en la integración (fusión) de dos listas de longitud M. Sean dos listas ordenadas de longitud M, las listas A y B. El procedimiento es el siguiente:

1.- Dadas $A = a_0, a_1, \ldots, a_{M-1}$ y $B = b_0, b_1, \ldots, b_{M-1}$, se integran sublistas a partir de las entradas pares e impares de A y B, esto es, las sublistas $a_0, a_2, \ldots, a_{M-2}$ y $a_1, a_3, \ldots, a_{M-1}$ por una parte, y $b_0, b_2, \ldots, b_{M-2}$ y $b_1, b_3, \ldots, b_{M-1}$, por otra.
2.- Las sublistas A_{par} y B_{impar} son recursivamente fusionadas para generar la lista ordenada C, de M entradas. Similarmente, las sublistas A_{impar} y B_{par} son recursivamente fusionadas para integrar la lista ordenada D, de M entradas.
3.- Las listas C y D son fusionadas para generar una lista, no necesariamente ordenada, L', de 2M entradas, insertando las entradas de C y D en el orden $c_0, d_0, c_1, d_1, \ldots, c_{M-1}, d_{M-1}$.
4.- Finalmente, la lista ordenada L de 2M entradas es obtenida comparando, para toda i, i = 0, 1, ..., 2M-1, las entradas l'_i de la lista L' e invirtiendo entradas si $l'_{i+1} < l'_i$.

El algoritmo está basado en el lema de ordenamiento 0-1 de Knuth [35]. Como ejemplo, sean A = 2, 6, 7, 8, 9, 10, 12 y 14 y B = 0, 1, 3, 4, 5, 11, 13, 15 dos listas ordenadas de 8 entradas cada una. El procedimiento descrito arriba consiste, entonces, en generar las sublistas:

A_{par} = 2, 7, 9, 12 $\qquad$ A_{impar} = 6, 8, 10, 14 y
B_{par} = 0, 3, 5, 13 $\qquad$ B_{impar} = 1, 4, 11, 15

Estas sublistas son recursivamente fusionadas, para obtener las listas ordenadas C y D:

C = 1, 2, 4, 7, 9, 11, 12, 15 y $\qquad$ D = 0, 3, 5, 6, 8, 10, 13, 14

C y D se agregan, para obtener L' = 1, 0, 2, 3, 4, 5, 7, 6, 9, 8, 11, 10, 12, 13, 15, 14. Finalmente, por comparación de entradas sucesivas de L', e inversión, en caso necesario, se obtiene la lista ordenada de 16 entradas L = 0, 1, ..., 15. La naturaleza recursiva del algoritmo se observa en la siguiente figura, para una lista de 16 entradas.

En la figura 36, cada pareja de nodos unida por una línea vertical representa una comparación entre dos números, tal que el menor valor es escrito al nodo superior, y el mayor valor al nodo inferior. De la rnisma figura se desprende que el algoritmo consta de logN iteraciones, donde cada iteración está integrada por i + 1 pasos, i = 0, 1, ..., logN - 1. Cada paso, a su vez, involucra N/2 comparaciones. Por lo tanto, el número total de comparaciones C es (N/2)[1 + 2 + 3 +... + logN], esto es:

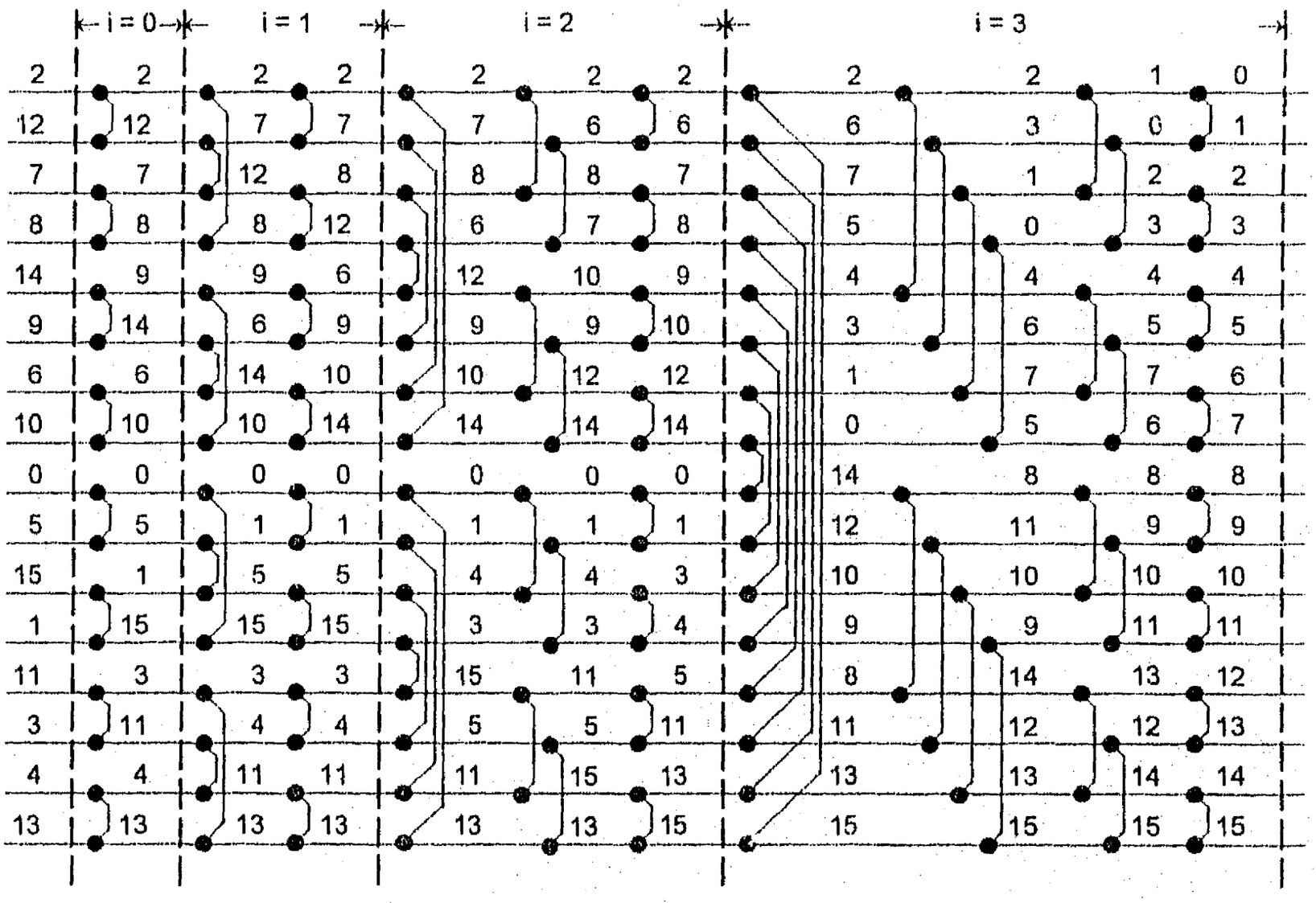


figura 36: Algoritmo de ordenamiento por fusión par - impar, para N = 16

$$C = (N/2)\log N(\log N + 1)/2 = (N/4)[\log^2 N + \log N] \quad \ldots\ldots 46$$

Con objeto de facilitar la comparación entre rendimientos, supóngase que los valores a ordenar son números expresados en notación de punto decimal flotante y, por otra parte, que N es mayor a la capacidad de almacenamiento de la, o las, memorias caché de la máquina. En lo subsecuente se considerará que las consideraciones (1 - 5) de la página 43 resultan aplicables.

Tiempo de ejecución del uniprocesador.

Cada comparación es precedida por 2 lecturas de datos. Parece razonable suponer que, en promedio, la mitad de las comparaciones conducen a la necesidad de invertir la posición relativa de los datos, esto es, que cada comparación conduce, en promedio, a una escritura. Para tiempos de lectura, comparación y escritura iguales, cada comparación representa un tiempo de ejecución de 4 unidades. De (46) resulta, entonces, que una máquina de un sólo procesador requiere un tiempo de ejecución, T_{U1}, dado por:

$$T_{U1} = N(\log^2 N + \log N) \quad \ldots\ldots 47$$

Supóngase que el procesador posee una memoria caché de capacidad M, tal que N/M = k, donde $k \geq 1$. La velocidad de ejecución del algoritmo puede, entonces, incrementarse significativamente si la lista de N entradas es dividida en k sublistas, y cada sublista es ordenada por separado. Al concluir el ordenamiento de las sublistas, éstas son a su vez fusionadas, para generar la lista ordenada final.

Cada paso del ordenamiento de una sublista requiere M lecturas, M/2 comparaciones y M escrituras. Una sublista puede ser ordenada, entonces, en $2M + (M/4)(\log^2 M + \log M)$ tiempos. En términos de N, las k sublistas pueden ordenarse en $2N + (N/4)[\log^2(N/k) + \log(N/k)]$ tiempos.

Es fácil ver que la fusión de las k sublistas puede ser efectuada en $N\log k[2\log N - \log k + 1]$ tiempos, por lo que el tiempo de ejecución total es:

$$T_{U2} = 2N + (N/4)[\log^2(N/k) + \log(N/k)] + N\log k[2\log N - \log k + 1] \quad \ldots\ldots 48$$

Como ejemplo, supóngase que $N = 2^{16}$, y $M = 2^{14}$, esto es, k = 4. Entonces, de (47), $T_{U1} = 17.826 \times 10^6$. Similarmente, de (48) resulta $T_{U2} = 7.635 \times 10^6$, o sea, $T_{U2} = 0.428 T_{U1}$.

Tiempo de ejecución del multiprocesador de bus común.

Por medio de un análisis similar al indicado en la figura 33, y para la ejecución paralela del algoritmo por p procesadores, se obtiene el tiempo de ejecución, T_{BC1}, del multiprocesador de bus común:

$$T_{BC1} = (N/2p)(1 + p)[\log^2 N + \log N] \quad \ldots\ldots 49$$

Supóngase ahora que los p procesadores cuentan con memorias caché de capacidad igual o mayor a N/p datos, de tal manera que p sublistas pueden ser ordenadas secuencialmente cada una, en forma concurrente. Para $N \geq 2p$, las p sublistas se pueden ordenar en un tiempo dado por:

$$2N + (N/4p)[\log^2(N/p) + \log(N/p)]$$

La fusión de estas sublistas requiere un tiempo adicional de:

$$(N/4p)\log p(2 + 5p)(2\log N - \log p + 1)/2$$

suponiendo, como arriba, que en promedio N/2 datos son escritos como resultado de las comparaciones. El tiempo de ejecución total es, entonces:

$$T_{BC2} = 2N + (N/4p)\{\log^2(N/p) + \log(N/p) + \log p(2 + 5p)(2\log N - \log p + 1)/2\} \quad \ldots\ldots\ldots\ldots 50$$

Tiempo de ejecución de multiprocesadores con memoria compartida segmentada.

Para las arquitecturas en las cuales la memoria compartida está dividida en módulos, y para la asignación de espacios de direccionamiento dada en la suposición 5 (pág. 43), la lectura de datos en el orden indicado en la figura 36 genera dos solicitudes de acceso a un mismo módulo en un tiempo dado. Esta contienda puede evitarse, sin incrementar el tiempo de ejecución, si los procesadores 0, 1, ..., (p/2) - 1 leen primero direcciones pares, y los procesadores (p/2), (p/2) + 1, ..., P - 1 direcciones impares. En la segunda lectura se procede en modo inverso. Desafortunadamente, en el caso de la red mariposa, este procedimiento conduce en general a contiendas en los nodos de la red. Una alternativa para evitar toda contienda consiste en generar, a partir de la lista inicial, dos listas concatenadas tales, que la primera contiene datos ubicados en direcciones pares, y la segunda datos ubicados en direcciones impares. Entonces, la secuencia de direccionamiento para un procesador cuyo número de identificación es par, es par - impar, y en orden inverso en caso contrario. Esta lista puede ser generada en paralelo en tiempo 2N/p, mismo que será considerado despreciable, dado la magnitud del tiempo de ejecución del algoritmo.

Para el procesador de buses múltiples, y suponiendo la ejecución paralela del algoritmo, se obtiene:

$$T_{BM1} = (N/4p)[1 + (5p/2b)][\log^2 N + \log N] \quad \ldots\ldots\ldots\ldots 51$$

Para la red *crossbar*, resulta:

$$T_{CR1} = (N/p)[\log^2 N + \log N] \quad \ldots\ldots\ldots\ldots 52$$

Para la red mariposa se obtiene:

$$T_{RM1} = (N/4p)[3\log p + 1][\log^2 N + \log N] \quad \ldots\ldots\ldots\ldots 53$$

Bajo el esquema del ordenamiento secuencial, concurrente, de p sublistas, y fusión paralela de estas sublistas en la lista ordenada final, el tiempo de ejecución para el procesador de buses múltiples puede ser obtenido como:

$$T_{BM2} = (N/4b)[6 + \log^2(N/p) + \log(N/p)] + (N/4p)[1 + (5p/2b)]\log p[2\log N - \log p + 1] \quad \ldots\ldots 54$$

Similamente, para la red *crossbar* resulta:

$$T_{CR2} = (N/4p)[6 + \log^2(N/p) + \log(N/p)] + (N/p)\log p[2\log N - \log p + 1] \quad \ldots\ldots 55$$

Y, finalmente, para la red mariposa se obtiene:

$$T_{RM1} = (N/4p)\{6\log p + \log^2(N/p) + \log(N/p) + \log p(3\log p + 1)(2\log N - \log p + 1)\} \quad \ldots\ldots 56$$

De acuerdo con la ecuación (47), una máquina de un sólo procesador ordena una lista de 2^{16} entradas en 17.826×10^6 unidades de tiempo. Para un tamaño dado de la memoria caché, este tiempo puede ser reducido en términos de la ecuación (48). Obviamente, el tiempo de ejecución es mínimo, si la capacidad de la memoria caché es suficiente para contener la lista completa. En este caso, el tiempo de ejecución se reduce a 4.588×10^6. La tabla 2 enlista los tiempos de ejecución de esta máquina, así como del multiprocesador de bus común en términos de p, donde p asume el significado de N/M para el caso del uniprocesador. Nótese que para el caso del multiprocesador de bus común, el mejor rendimiento se obtiene con solamente dos procesadores ordenando secuencialmente dos sublistas de 2^{15} entradas, previo a la fusión final.

Tabla 2: Tiempos de ejecución ($\times 10^6$) para el uniprocesador y multiprocesador de bus común

p	1	2	4	8	16	32	64
Uniprocesador	4.588	6.160	7.635	9.011	10.289	11.469	12.555
Bus común, ec. (49)	--------	13.369	11.141	10.027	9.470	9.191	9.052
Bus común, ec. (50)	--------	3.670	3.785	4.375	5.161	6.005	6.836

La figura 37 muestra el incremento de velocidad logrado por las diferentes arquitecturas consideradas, en comparación con el mínimo tiempo de ejecución del uniprocesador, para $N = 2^{16}$ y ejecución paralela. Para la arquitectura de buses múltiples se consideró una relación p/b = 4. El caso del ordenamiento secuencial, concurrente, de sublistas se muestra en la figura 38. Es interesante observar que, comparado con el mejor desempeño del uniprocesador, en una máquina paralela la ejecución paralela del algoritmo es más rápida solamente si $p \geq 4$ para la red *crossbar*, y $p \geq 16$ para la mariposa. Este ejemplo ilustra la dependencia del rendimiento de máquinas paralelas, para la ejecución de un mismo algoritmo, con relación al modelo de programación utilizado.

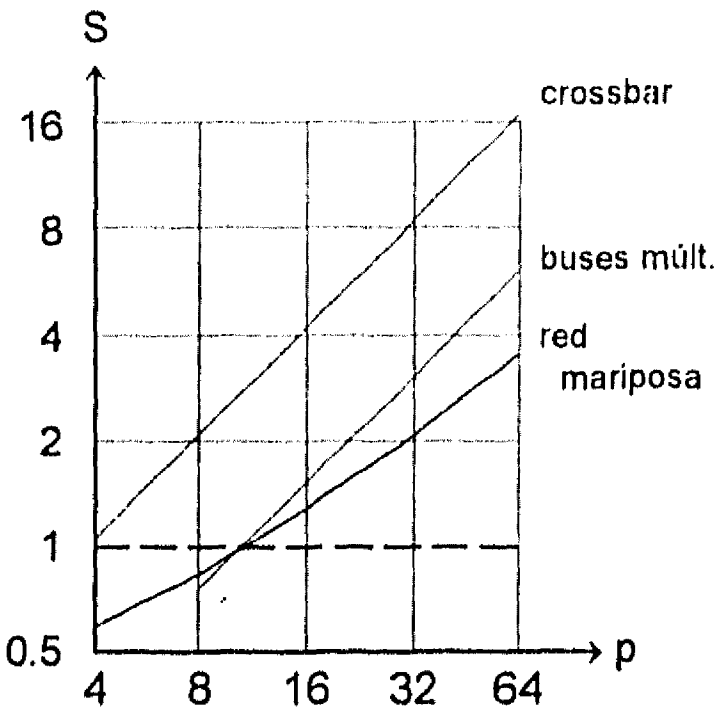


figura 37: Incremento de velocidad, algoritmo de ordenamiento con ejecución paralela.

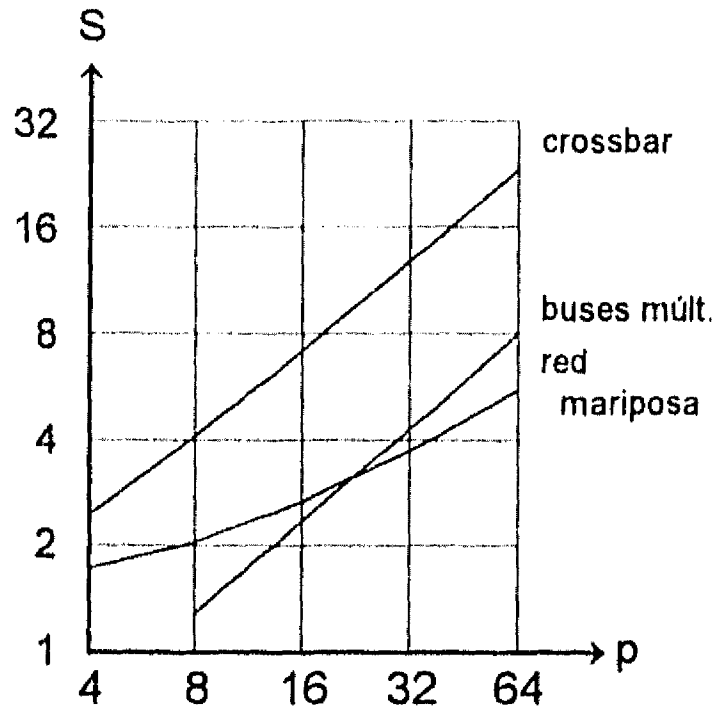


figura 38: Incremento de velocidad, algoritmo de ordenamiento con ejecución secuencial

En la discusión previa se trató de respetar el principio de costo unitario. Esta restricción conduce a estimaciones de rendimiento pesimistas para redes escalonadas, para el caso en que el tiempo de tránsito por un nodo de la red es inferior al tiempo de acceso a memoria. Por otra parte, los modelos discutidos no incluyen una serie de factores que pueden modificar significativamente algunos de los resultados obtenidos. En particular, el problema de arbitraje afecta, en especial, a las arquitecturas de buses múltiples. No parece viable analizar estos aspectos al margen de una tecnología concreta. En el siguiente capítulo se presenta un análisis de complejidad de las arquitecturas consideradas, tomando en cuenta estos factores.

IV.- ASPECTOS DE IMPORTANCIA PRACTICA EN ESTRUCTURAS DE INTERCONEXION

En un multiprocesador, la red de interconexión constituye, tal vez, la componente que en mayor medida define la complejidad de la máquina, no solamente por la cantidad y tipo de elementos de conmutación que la conforman y por la naturaleza de interconexiones físicas que requiere, sino también por el tipo de interface que impone tanto a los elementos de procesamiento, como a los módulos de memoria. Por "interface" se debe entender, en este contexto, tanto la estructura física, como el protocolo de comunicación exigido. Una discusión general de todos los aspectos involucrados parece poco factible en el marco de este trabajo. Se proponen, a continuación, las siguientes condicionantes para delimitar la discusión posterior:

1.- Elementos de procesamiento. Se considerará que este elemento es un microprocesador representativo del estado del arte en la materia. En particular, y cuando la discusión de posibles alternativas lo requiera, se considerarán las características y especificaciones del dispositivo Pentium de INTEL [58], utilizado en el desarrollo del prototipo del presente trabajo. Cabe señalar que este dispositivo posiblemente no sea el más idóneo en términos de la arquitectura que se propondrá en el siguiente capítulo[1], pero constituye la elección de mas fácil acceso y, tal vez, de mayor soporte de *software* disponible.

2.- Módulos de memoria. Parece razonable asociar físicamente estas componentes a las unidades de procesamiento, por las siguientes razones. Cada procesador requiere acceso a una memoria de programa, así como de datos exclusivos de este procesador. El programa no necesariamente es el mismo para todos los procesadores, por lo que conviene proporcionar una memoria a cada procesador. Por otra parte, la comunicación entre un procesador y la memoria que contiene el programa a ejecutar, debe ser lo más rápida posible. Este factor, particularmente la operación síncrona de estas componentes, exige cercanía física. Finalmente, esta disposición permite el uso de un sólo módulo de memoria tanto para programa y datos exclusivos, como para datos compartidos por todos los procesadores.

1 El procesador PENTIUM constituye, en la gran mayoría de las aplicaciones, el elemento de procesamiento único, o el procesador central en una arquitectura con pocos procesadores adicionales, dedicados a tareas específicas como aceleradores de video, procesadores de comunicaciones, etc. Para aumentar su rendimiento, cuenta con un mecanismo de predicción de instrucciones de control de flujo de ejecución (saltos), basado en la historia de la secuencia de instrucciones ejecutadas. Este mecanismo involucra la lectura de código ubicado en espacios de direccionamiento cuya referencia sea la mas probable en el futuro inmediato, independientemente de que este código sea, o no, ejecutado. Esta característica, si bien aumenta la velocidad de ejecución, tambien incrementa el número de accesos a memoria y, en consecuencia, el tráfico en el bus del sistema. Es cuestionable que el incremento de velocidad logrado corresponda, en una arquitectura paralela, a la disminución del rendimiento global originado por este incremento de tráfico.

Por otra parte, el procesador ofrece una serie de características que aumentan su versatilidad, pero que resultan poco útiles en el contexto de un multiprocesador. Como ejemplo, se puede citar la gran riqueza de modos de operación de la memoria. Estas características consumen área de silicio, que posiblemente podría ser aprovechada para incrementar la capacidad de las memorias caché residentes en el dispositivo. Claramente, lo anterior requiere un estudio más detallado que, por otra parte, trasciende a los objetivos del presente trabajo.

Las frecuencias de operación de microprocesadores modernos (superior a 100 MHz) excluyen, en la práctica, la operación síncrona entre procesadores, con la posible salvedad del uso de estructuras de intercomunicación sumamente sofisticadas. Este aspecto será considerado en el capítulo V.

Se discutirán, a continuación, algunos aspectos no considerados por los modelos de redes de interconexión presentados en el capítulo III. En particular, se analiza:

a.- La cantidad y complejidad de los elementos de conmutación.
b.- La cantidad y tipo de arbitraje requerido.
c.- El protocolo de memoria caché.
d.- La estructura de interconexiones físicas.

IV.1 Elementos de conmutación.

Asumiendo que el multiprocesador de bus común posee una sola memoria compartida, y que para las restantes arquitecturas consideradas el número de procesadores, *p*, es igual al número de módulos de memoria *m*, la cantidad de elementos de conmutación *S* requerida para, respectivamente, la estructura de bus común, S_{BC}, de buses múltiples, S_{BM}, la red crossbar, S_{CR} y la red mariposa, S_{RM}, está dada por:

$$S_{BC} = p \quad \text{..... 57}$$
$$S_{BM} = 2p^2/\eta \quad \text{..... 58}$$
$$S_{CR} = p^2 \quad \text{..... 59}$$
$$S_{RM} = (p/2)\log p \quad \text{..... 60}$$

En (58), η es la relación entre el número de procesadores y de buses, $\eta = p/b$. La tabla 3 reproduce los valores de S para p = 2, 4, ..., 64 y $\eta = 4$:

Tabla 3: Número de interruptores, S.

p	2	4	8	16	32	64
S_{BC}	2	4	8	16	32	64
S_{BM}	2	8	32	128	512	2048
S_{CR}	4	16	64	256	1024	4096
S_{RM}	1	4	12	32	80	192

El número de buses interconectados por los elementos de conmutación es 2, para todas las estructuras consideradas excepto redes escalonadas, y 4 para estas últimas (véase la figura 11, pág. 29). El número de líneas físicas interconectadas depende del formato de la información transmitida.

Para el caso de multiprocesadores de bus común, el formato considerado es típicamente el de una unidad de almacenamiento de la memoria caché. Para el microprocesador de referencia, la configuración tal vez mas representativa define esta unidad (llamada *línea* en la literatura de INTEL)[58] como de 256 bits, transmitidos secuencialmente por el controlador de memoria caché como 4 palabras de 64 bits. Para redes conmutadas, se considera la mayor unidad de información manejada por el conjunto de instrucciones del microprocesador, aquí una palabra de 32 bits. De lo anterior se desprende que el multiprocesador de bus común requiere elementos de conmutación capaces de manejar dos buses de 64 bits cada uno, el esquema de buses múltiples y la red crossbar, dos buses de 32 bits, y redes escalonadas, cuatro buses de 32 bits.

En lo que al direccionamiento se refiere, el formato es de 32 bits para todas las transferencias posibles (bytes, palabras de 16 y de 32 bits, y líneas de 256 bits). El direccionamiento es generado para cada transferencia, excepto para lecturas o escrituras de líneas de la memoria caché. En este caso, la dirección es emitida para la primera palabra de 64 bits únicamente (transferencias tipo *ráfaga* en la literatura de INTEL). El controlador de memoria debe calcular las direcciones restantes, de acuerdo con la dirección inicial y la secuencia de transmisión dictada por el dispositivo. En lo subsecuente se asumirá que los buses de la red de interconexión operan en forma multiplexada, transmitiendo primero la dirección, y posteriormente el dato.

IV.1.1 Cantidad de dispositivos.

Está claro que el elemento de conmutación más complejo corresponde a las redes escalonadas. La figura 39 muestra una configuración *crossbar* bidireccional de dos entradas y dos salidas, requerida para desempeñar la función de ese elemento. Los bloques marcados con M corresponden a multiplexores, y los marcadas con B a dispositivos de reforzamiento (*buffers*) con salidas deshabilitables (salidas de tres estados).

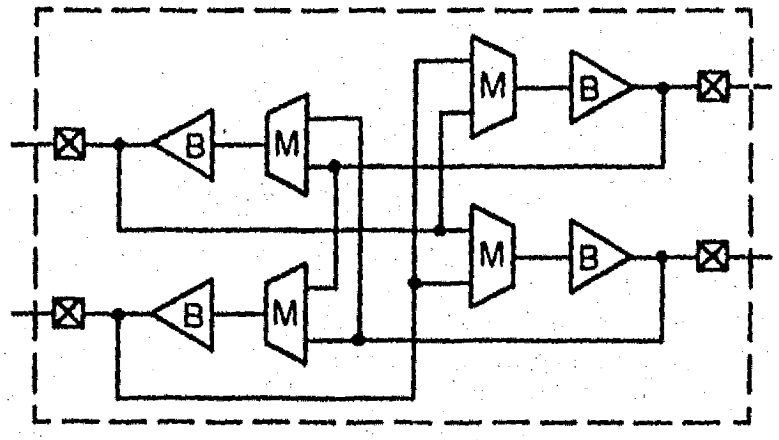


Figura 39: Configuración *crossbar* bidireccional de dos entradas y salidas.

Supóngase que la función de la figura 39 puede ser alojada en un sólo dispositivo, que proporciona el número de terminales suficiente para los cuatro buses de datos, señales de control y de alimentación requeridos. Entonces, la densidad de elementos de conmutación por dispositivo es cuatro veces mayor para arquitecturas de buses múltiples y red *crossbar*, comparado con redes escalonadas. Como ya fue señalado por A. A. Sawchuk *et al.* [36], el número de interruptores que pueden ser integrados en un dispositivo está limitado, no tanto por la complejidad del interruptor en sí, sino por el número de terminales requerido para las señales de entrada/salida. Para un número de terminales dado, la cantidad de interruptores que puede ser integrada en un dispositivo crece inversamente proporcional al cuadrado del número de líneas que conforman a los buses de datos. Conviene,

entonces, analizar la opción de dividir un formato de datos dado en unidades más pequeñas y transmitir estas unidades secuencialmente por la red. Comparado con redes escalonadas, y si el formato de datos es divido por logN, donde N es la dimensión de la red (número de entradas), el tiempo de transmisión total de una red *crossbar* es el mismo que la cota inferior de una red escalonada. La tabla 4 muestra el número de dispositivos requerido por la red mariposa, así como por una red *crossbar* para diferentes cantidades de líneas de bus W, donde $W \geq 32/\log N$. Se supone que un elemento de conmutación para la red escalonada, para buses de 32 bits, puede ser integrado en un sólo dispositivo.

Por otra parte, por lo discutido arriba el número de dispositivos requerido por una estructura de bus común es N/2, asumiendo buses de 64 bits.

Tabla 4: Número de dispositivos, D, de las redes mariposa y *crossbar*, para diferentes valores de W.

N	4	8	16	32	64
D_{RM}, W = 32	4	12	32	80	192
D_{CR}, W = 32	4	16	64	256	1024
D_{CR}, W = 16	1	4	16	64	256
D_{CR}, W = 8	-	-	4	16	64

IV.1.2 Retardo de propagación.

En lo subsecuente se asumirá que todas las funciones lógicas pueden ser ejecutadas en un retardo de compuerta unitario. Por otra parte, la suma de los retardos asociados a una terminal de entrada y una de salida de un dispositivo dado se considerará igual a un retardo de compuerta. Sea n el número de entradas (*fan-in*) de una función lógica determinada. Valores típicos para n varían entre 5, para arreglos de compuertas programables (FPGA's - *field-programmable gate arrays*) [64], [65], hasta 16 para otros tipos de lógica programable [66].

Para la estructura de bus común, cada interruptor está constituido por una compuerta de reforzamiento (*buffer*) con salida de tres estados. Esto es, la conexión puede ser establecida en dos tiempos. Para redes escalonadas, la función lógica asociada al multiplexor puede ser ejecutada en 2 tiempos, por lo que el retardo de propagación total es 4, para cada elemento de conmutación. En el caso de estructuras de buses múltiples y la red *crossbar*, el retardo asociado al multiplexor depende del número de interruptores, S, integrados en el dispositivo. Sea s el número de entradas del multiplexor. Entonces, el retardo asociado a esta función es $\tau_{MUX} = \lceil \log(\log s + 1)/\log n \rceil + \lceil \log s/\log n \rceil$, y, para el retardo total, se obtiene:

$$\tau_{CR} = 2 + \lceil \log(\log s + 1)/\log n \rceil + \lceil \log s/\log n \rceil \quad \text{..... 61}$$

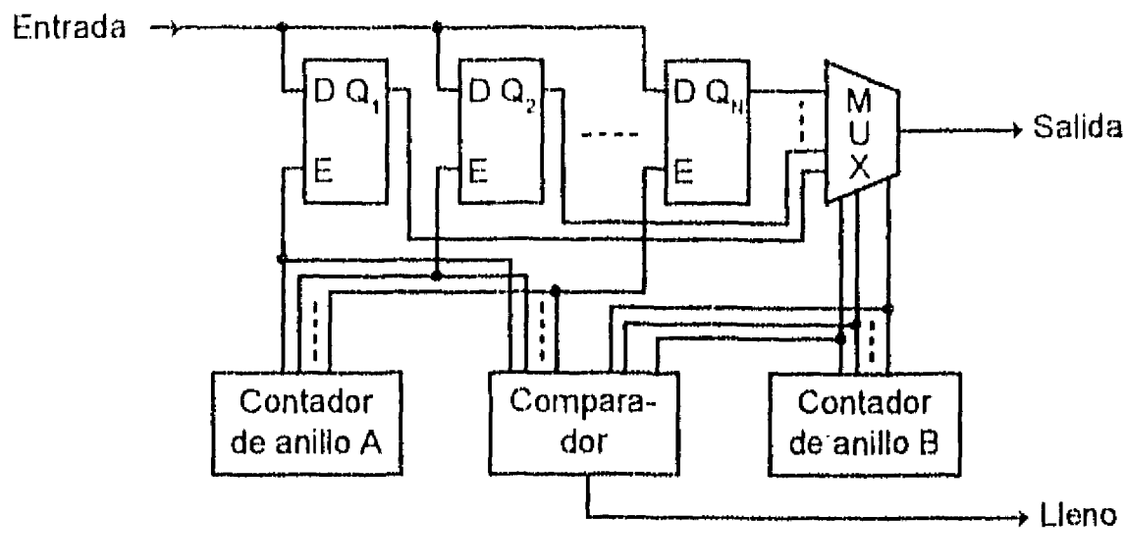


Figura 40: Memoria tipo FIFO

En redes escalonadas se presenta, en general, contienda por nodos (véase, también, el punto III.4). Como ya fue mencionado con anterioridad, este problema se puede reducir proporcionando capacidad de almacenamiento local a los elementos de conmutación (figura 13). Una memoria tipo FIFO construida a partir de N registros y dos contadores de anillo como apuntadores de entrada y salida, respectivamente, se muestra en la figura 40. La cuenta inicial de ambos contadores es cero. La salida del contador A habilita la escritura del registro Q_1 por medio de la señal E. Se asume operación sincrona, con base en una señal de reloj (no mostrada). El arribo de un dato ocasiona que éste es escrito en el registro habilitado, y que el contador A incremente su cuenta. Con ello, se habilita al registro Q_2, y así sucesivamente. La secuencia de cuenta es 0, 1, ..., N -1, 0, 1, ... De manera similar, el contador B, por medio de un multiplexor de N entradas, habilita la salida del registro Q_1. Al ser leído un dato, este contador incrementa su cuenta, habilitando al registro Q_2, etc. Sean C_A y C_B las cuentas de los contadores A y B. Entonces, la memoria está llena cuando se efectúe una escritura y $C_A = C_B$. Un comparador detecta esta condición, e inhibe escrituras adicionales hasta que lecturas liberen espacio de almacenamiento.

El retardo de propagación del esquema es la suma de los retardos asociados a un registro, y del multiplexor. Un registro genera dos retardos unitarios. En virtud de que de las salidas del contador de anillo solamente una está activa en un tiempo dado, el retardo del multiplexor se reduce a 1 + logN/logn. Si esta memoria es incorporada al elemento de conmutación mostrado en la figura 39, el retardo total del dispositivo es:

$$\tau_{RM} = 4 + \lceil \log 2N/\log n \rceil \qquad \ldots\ldots 62$$

La cantidad de registros N requeridos para un número de procesadores p puede estimarse como sigue. La condición de peor caso se produce, evidentemente, cuando p procesadores transfieren líneas de memoria caché a un mismo módulo de memoria. Entonces, hasta (p - 1) paquetes de 256 bits de datos, más 32 bits de direccionamiento, pueden acumularse en un nodo de la red. Asumiendo que esta situación no se repite antes de que la información acumulada haya podido abandonar el nodo, el valor requerido de N es 9(p - 1) palabras de 32 bits. Similarmente, para el caso de transferencias de datos de 32 bits, se obtiene N = 2(p - 1) palabras. La tabla 5 reproduce los retardos de propagación

correspondientes a ambos casos, así como la cantidad de memoria requerida en bytes, para un elemento de conmutación con 4 FIFO's para operación bidireccional. Se asume, además, que n = 5, como un valor representativo de dispositivos de arreglos de compuertas programables (FPGA's).

Tabla 5: Retardo de propagación y (cantidad de bytes) de elementos de conmutación.

p	2	4	8	16	32	64
N = 2(p - 1)	5 (32)	6 (96)	7 (224)	7 (480)	7 (992)	8 (2016)
N = 9(p - 1)	6 (144)	7 (432)	8 (1008)	8 (2160)	8 (4464)	9 (9072)

En contraste con lo anterior, el elemento de conmutación para el procesador de bus común genera un retardo de 2 unidades, y, para la red *crossbar*, de (61) se obtiene un retardo de 4 para hasta 16 interruptores por dispositivo, y de 5, para S = 64.

IV.2 Arbitraje de acceso.

En general, una máquina de bus común proporciona dos mecanismos de intercambio de información entre procesadores: un mecanismo de intercambio directo entre memorias caché por medio de un bus de indagación de alta velocidad (*snoop bus*), y un segundo mecanismo por medio de la memoria compartida [58]. El acceso a cualquiera de estos buses involucra un árbitro para seleccionar una, de entre p solicitudes. El elemento de conmutación afectado es el correspondiente al procesador p, y al bus involucrado en la transferencia.

Los esquemas de buses múltiples requieren dos tipos distintos de árbitros: uno, que selecciona uno de b buses para una de entre p solicitudes, y un segundo árbitro que selecciona uno de p módulos de memoria [39]. Los elementos de conmutación involucrados están definidos por p y b, por una parte, y por la dirección del dato, por otra.

En una red tipo crossbar, cada interruptor es común a un bus de, ya sea, una fila, o una columna (figura 41). Supóngase que las filas corresponden a procesadores, y las columnas a módulos de memoria. La selección de un módulo en particular involucra un árbitro de una de p solicitudes. El interruptor afectado está definido por p, y la dirección del dato. Finalmente, en una red escalonada cada elemento de conmutación debe seleccionar una de dos solicitudes, y decidir si ambas, o solamente una, pueden ser transmitidas a la vez. Por otra parte, la cadena de elementos de

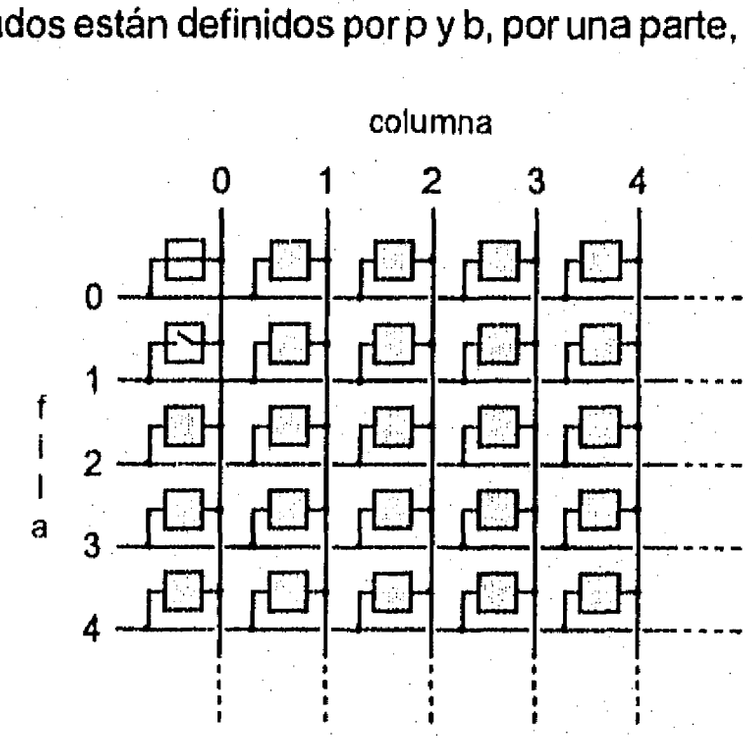


Figura 41: La red *crossbar*

conmutación afectada por una transferencia debe decidirse de acuerdo con algún algoritmo de enrutamiento. Como se observa, el tipo y cantidad de arbitraje requerido difiere para las cuatro arquitecturas consideradas.

IV.2.1 El árbitro uno-de-N.

El protocolo de arbitraje más frecuentemente utilizado es del tipo solicitud-otorgamiento (*request-grant*), con base en un árbitro centralizado. Pearce, Field y Little [37] propusieron una realización física para un módulo de arbitraje de 1-de-2 solicitudes, integrado por 12 compuertas (figura 42). Para N una potencia entera de 2, (N - 1) módulos pueden ser interconectados en forma de árbol para construir árbitros de una-de-N solicitudes. Para cada módulo, el retardo desde las entradas de solicitud hasta la salida de solicitud es 2τ, donde τ es el retardo de compuerta. Similarmente, el retardo de la señal de otorgamiento es τ. El retardo por módulo es, entonces, 3τ, y el arbitraje uno-de N puede efectuarse en:

$$\tau_A = 3\tau \log N \qquad \text{..... 63}$$

El árbitro soporta esquemas de prioridad fija o rotatoria. En el primer caso, un circuito biestable registra cuál de las dos solicitudes es transmitida, y en consecuencia, cuál de las dos salidas de otorgamiento, en su caso, es activada. En el segundo caso, el biestable cambia de estado después de recibir una entrada de otorgamiento activa.

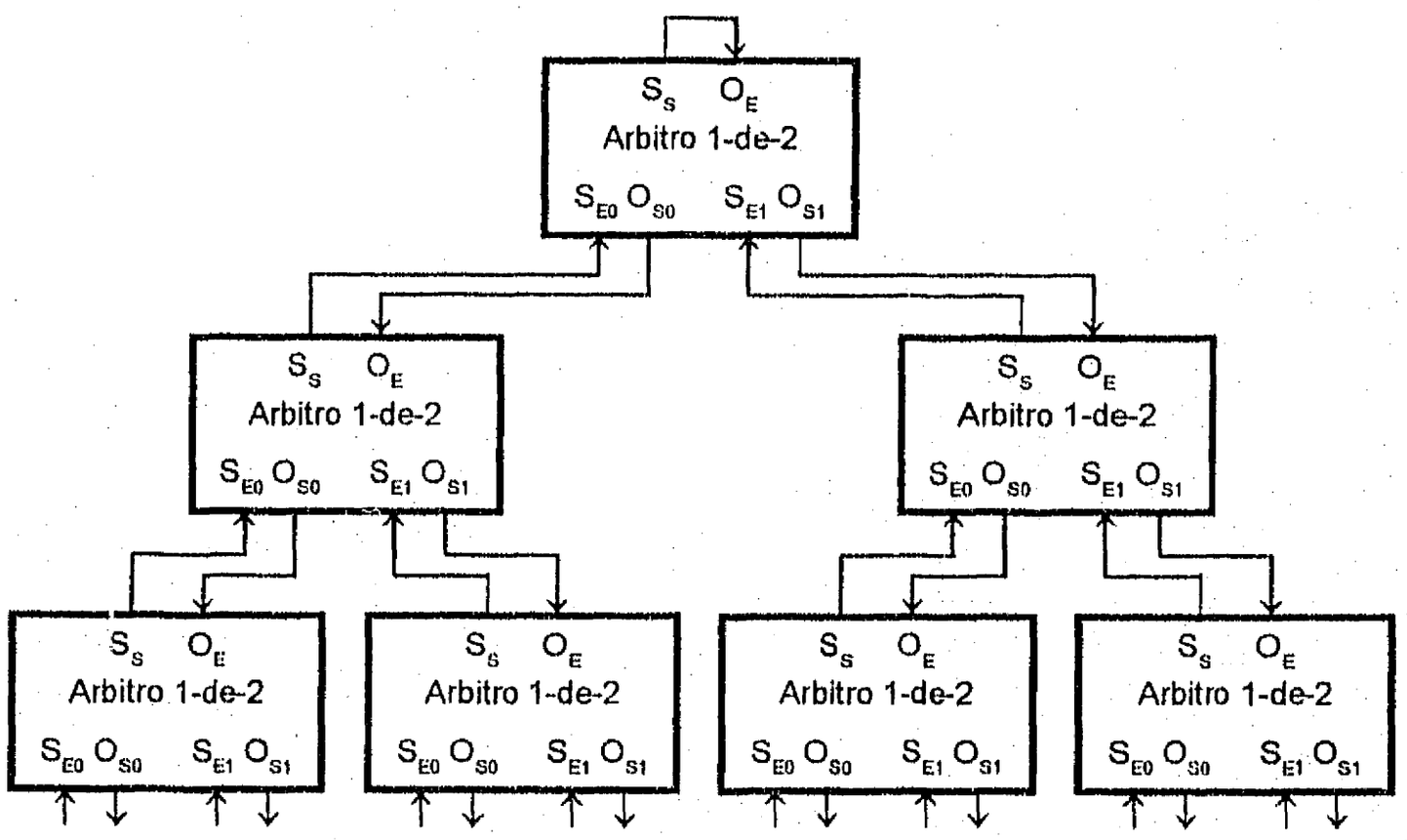


Figura 42: Arbitro 1-de-8. S_E - entrada de solicitud, O_E - entrada de otorgamiento, S_O - salida de solicitud, O_S - salida de otorgamiento.

La desventaja fundamental de un esquema de arbitraje centralizado reside en el hecho de que representa una, o varias, componentes adicionales a las de la red de interconexión propiamente dicha, con los consecuentes problemas de enrutamiento de señales físicas. A pesar de ello, el protocolo solicitud-otorgamiento es, tal vez, el más usado hasta la fecha [67].

Se propone aquí un esquema de arbitraje descentralizado, basado en la réplica de árbitros en cada elemento de conmutación, y físicamente integrados a los mismos. Considérese la figura 43. Una solicitud S_n es encodificada, de acuerdo con un determinado esquema de priorización, en una de N líneas de solicitud s_j, comunes a todos los elementos de conmutación del sistema. Si la prioridad del elemento j es diferente a la del elemento k, donde j, k = 1, 2, ..., N, está claro que cada encodificador activa, a lo más, una de N líneas de solicitud. La encodificación requiere un retardo unitario. Por otra parte, cada árbitro asociado a un elemento de conmutación dado debe decidir si, de hasta N solicitudes, la de mayor prioridad es la correspondiente al mismo elemento de conmutación. Sea pr_j una señal que identifica la prioridad j, j = 1, 2, ..., N. Cada árbitro dispone de N señales pr_j, de las cuales N - 1 están inactivas, y una, la generada por el encodificador de prioridad asociado, está activa. Si la suma y producto denotan, respectivamente, las operaciones lógicas O (*or*) e Y (*and*), y si la negación es representada por x, donde x es una variabale binaria, entonces la función de arbitraje toma la forma:

$$A = (s_1)(pr_1) + (s_2)(pr_2)(\bar{s}_1) + (s_3)(pr_3)(\bar{s}_1)(\bar{s}_2) + ... + (s_N)(pr_N)(\bar{s}_1)(\bar{s}_2)...(\bar{s}_{N-1}) \quad \text{..... 64}$$

De (64) se desprende que la operación Y se efectúa sobre, a lo más, (N + 1) entradas. Similarmente, la operación O se efectúa sobre N entradas. Consecuentemente, la función de arbitraje genera $\lceil \log(N+1)/\log n \rceil + \lceil \log N/\log n \rceil$ retardos que, sumados al retardo generado por el encodificador, representan:

$$\tau_A = 1 + \lceil \log(N+1)/\log n \rceil + \lceil \log N/\log n \rceil \quad \text{..... 65}$$

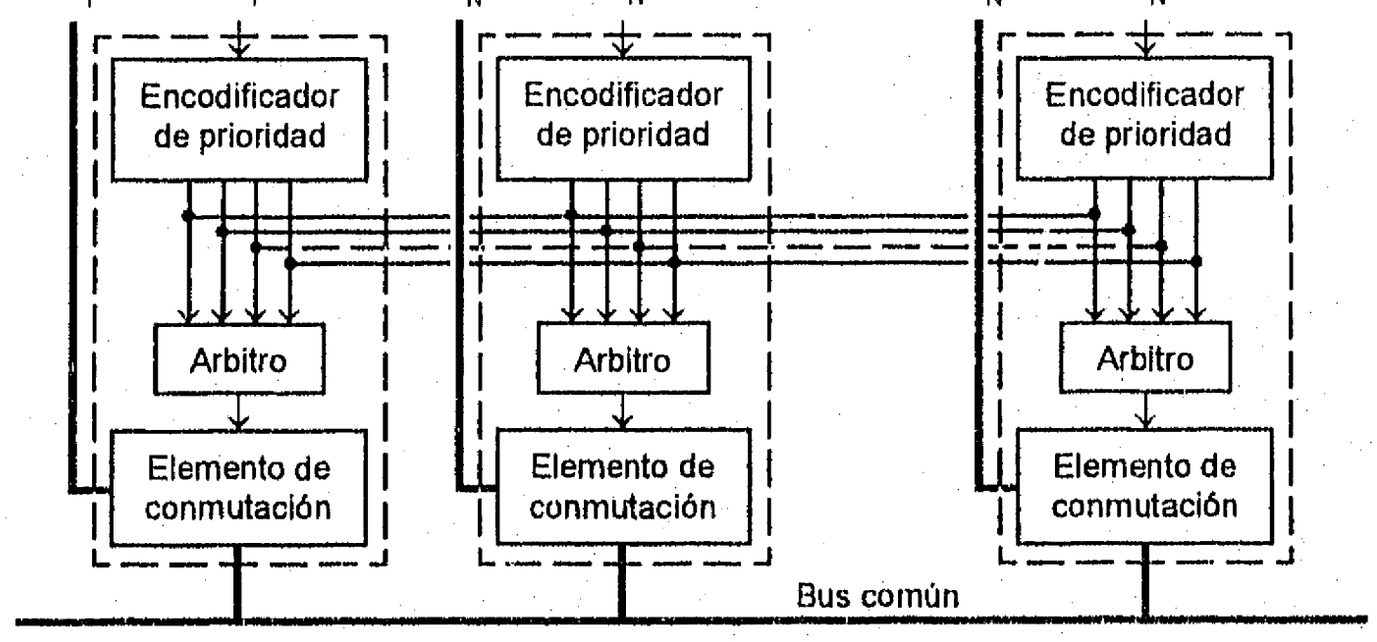


Figura 43: Arbitro 1-de-N descentralizado. S_n - entrada de solicitud, D_n - bus de datos

Como ejemplo, supóngase que N = 16, y n = 5. Entonces, de (65) se obtiene $\tau_A = 5$. El esquema dado por (63) genera $\tau_A = 12$. La reducción de tiempo es consecuencia de que el esquema no depende de circuitos biestables para generar las señales de otorgamiento. Por otra parte, (65) incluye solamente un retardo unitario asociado a las señales de entrada y salida, en virtud de que la función de arbitraje se asume integrada al dispositivo de conmutación. Finalmente, las señales s_j pueden físicamente ser multiplexadas en el bus común, y no representan recursos adicionales.

IV.2.2 El árbitro B-de-N.

Lang y Valero [38] proporcionaron realizaciones detalladas de árbitros B-de-N para multiprocesadores de buses múltiples. El esquema consiste en un anillo de N módulos de arbitraje que efectúan la asignación de hasta B buses a solicitudes pendientes, y un registro que almacena el estado de los árbitros, posterior a cada ciclo de arbitraje. La prioridad más alta es otorgada al módulo inmediatamente posterior al último en ser atendido, esto es, el esquema implementa una priorización rotatoria convencional. Un análisis detallado del esquema fue hecho por Mudge, Hayes y Winsor en [39]. Para el caso de que g módulos de arbitraje pueden ser integrados en un solo dispositivo, estos autores concluyen que el tiempo de arbitraje requerido es $\tau_A \approx (3N/g)\tau$. En general, se cumple que:

$$O(\log N) < \tau_A < O(\log^2 N) \qquad \text{..... 66}$$

Por lo tanto, y para valores de N grandes, este árbitro puede constituir el elemento limitante del rendimiento de la máquina.

IV.2.3 Selección del elemento de conmutación.

En las estructuras de bus común, el elemento de conmutación que interviene en una transferencia dada es activado directamente por el árbitro asociado, ya sea, al bus de indagación, o al bus común.

En las estructuras de buses múltiples de P procesadores y M módulos de memoria intervienen, como se mencionó, dos niveles de arbitraje. Primero, M árbitros seleccionan una solicitud de P posibles para un módulo de memoria dado, y, posteriormente, un árbitro asigna uno de B buses a cada una de las solicitudes de acceso. De los elementos de conmutación asociados a cada procesador (véase la figura 6), el árbitro B-de-P activa, a lo más, uno, en forma directa mediante la señal de otorgamiento $O_{bj,}$ donde b = 1, 2, ..., B y j = 1, 2, ..., P. Sea $O(m)_j$ la señal de otorgamiento generada por el árbitro m, m = 1, 2, ..., M. Entonces, la señal de activación del elemento de conmutación $C_m(b)$ está dada por:

$$C_m(b) = O_{bj}[\, O(1)_j + O(2)_j + \ldots + O(M)_j \,] \qquad \text{..... 67}$$

donde la suma y el producto representan las operaciones lógicas O e Y, respectivamente. Si la función lógica dada por (67) está integrada al elemento de conmutación, el tiempo requerido para la activación de este elemento es, entonces:

$$\tau_s = \lceil 1 + \log M/\log n \rceil \quad \ldots\ldots 68$$

Para redes crossbar, solamente un elemento de conmutación interviene en cada transferencia (figura 41). Un procesador dado emite una solicitud a todos los elementos de conmutación ubicados en la fila correspondiente a ese procesador, y el árbitro asociado a la columna, esto es, a un módulo de memoria en particular, selecciona un elemento de entre P filas. Consecuentemente, la activación de un elemento de conmutación involucra la decodificación de la columna destino a partir del direccionamiento proporcionado por el procesador. Un comparador de identidad integrado al dispositivo de conmutación puede realizar esta función en un tiempo dado por:

$$\tau_s = 1 + \lceil \log(\log P)/\log n \rceil \quad \ldots\ldots 69$$

En una red escalonada, un algoritmo de enrutamiento debe ser resuelto para una asignación de entrada/salida dada. Si se desea que el control de la red sea transparente para el programador, las posiciones de los interruptores individuales que conforman el elemento de conmutación deben decidirse en línea, por la red misma. Para la red mariposa, y para cada interruptor, esta decisión, en su forma más sencilla, se basa en la comparación de un sólo bit de las direcciones de entrada y salida. Sea i el nivel de un interruptor en la red, y las dos posiciones del interruptor definidas como recta, si w = w', esto es, si la fila de entrada es igual a la de salida, y cruzada en caso contrario (véase la definición de la red mariposa en la página 31). Entonces, la posición del interruptor es recta si el i-ésimo bit de la dirección de entrada es igual al de la dirección de salida, y cruzada en caso contrario.

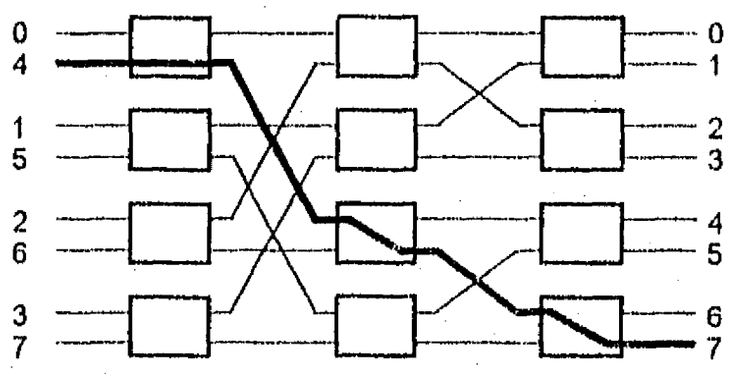


Figura 44:
La trayectoria entre la entrada 4 y la salida 7

En la figura 44 se muestra la trayectoria desde la entrada 4, a la salida 7. La representación binaria de las direcciones de entrada y salida es, respectivamente, 100 y 111. La comparación de los i-ésimos bits arroja una posición de los interruptores recta para i = 0, y cruzada para i = 1 e i = 2. Nótese que la numeración de las entradas y salidas son distintas. La numeración de las entradas se obtiene rotando la numeración de salida 1 bit a la derecha.

En virtud de que cada elemento de conmutación cuenta con dos entradas, debe decidirse, adicionalmente, si las posiciones de ambos interruptores son iguales, o no. En consecuencia, cada elemento de conmutación requiere arbitraje. Para una red de P entradas, el proceso de decisión genera

$$\tau_A = 3 + \lceil \log(\log P)/\log n \rceil \quad \ldots\ldots 70$$

retardos, incluyendo el enmascaramiento del i-ésimo bit.

IV.3 Tiempos de tránsito.

En general, el retardo global experimentado por un dato en su tránsito por la red de conmutación es la suma de los retardos asociados al esquema de arbitraje, de selección del dispositivo de conmutación, y el retardo de propagación de este último. Para el caso de solicitudes de acceso simultáneas, esto es, cuando se generan colas en la red, es necesario distinguir entre el tiempo de tránsito del primer dato, y de los subsecuentes. Lo anterior es consecuencia de que el retardo de arbitraje para subsecuentes transferencias puede ser parcial - o totalmente enmascarado por los restantes retardos asociados a la transferencia previa.

La suma T de los retardos asociados a la transmisión de datos en una máquina de bus común de p procesadores es 2 para el dispositivo de conmutación, más τ_A dado por (65) si el esquema de arbitraje de la figura 43 es adoptado. Esto es, para una primera transferencia:

$$T_{BC} = 3 + \lceil \log(p + 1)/\log n \rceil + \lceil \log p/\log n \rceil \quad \text{..... 71}$$

Para transferencias subsecuentes, el tiempo de tránsito es igual al tiempo de arbitraje. Para un abanico de entrada n, este tiempo es 3, para $p \leq n$.

En las arquitecturas de buses múltiples, el tiempo de tránsito total está dado por dos retardos de propagación especificados en (61), los tiempos de arbitraje de los árbitros 1-de-P (65) y B-de-P (66), y el retardo de asignación dado en (68). Desafortunadamente no se dispone de una expresión exacta para el árbitro B-de-P. Una estimación optimista puede consistir en considerar la cota inferior dada en (66). Sea p el número de procesadores y de módulos de memoria. Suponiendo que cuatro elementos de conmutación son integrados a cada dispositivo, y para un abanico de entrada n = 5 para estos dispositivos, se obtiene:

$$T_{BM} = 11 + 2\lceil \log(p + 1)/\log n \rceil + 3\lceil \log p/\log n \rceil \quad \text{..... 72}$$

En la ecuación anterior, $9 + \lceil \log p/\log n \rceil$ retardos corresponden a los retardos de propagación y asignación de interruptores. Por lo tanto, para los accesos subsecuentes, el tiempo de tránsito es 10, para $p \leq n$, y $5 + 2\lceil \log(p + 1)/\log n \rceil + 3\lceil \log p/\log n \rceil$, en caso contrario.

Para la red *crossbar*, el retardo de propagación está dado por (61), el de arbitraje por (65) y la asignación del interruptor por (69). Asumiendo, al igual que para el caso anterior, que cuatro interruptores son integrados a cada dispositivo, se obtiene:

$$T_{CR4} = 6 + \lceil \log(p + 1)/\log n \rceil + \lceil \log p/\log n \rceil + \lceil \log(\log p)/\log n \rceil \quad \text{..... 73}$$

Para accesos subsecuentes, el tiempo de tránsito es 6, para $p \leq n^2$, y determinado por el tiempo de arbitraje (65) en caso contrario. Supóngase ahora que 16 interruptores son integrados en un dispositivo de conmutación, y el ancho del bus de datos es reducido a la mitad. En virtud de que los retardos de arbitraje y de asignación de interruptor permanecen invariables, se obtiene:

$$T_{CR16} = 10 + \lceil \log(p + 1)/\log n \rceil + \lceil \log p/\log n \rceil + \lceil \log(\log p)/\log n \rceil \quad \ldots\ldots 74$$

Para accesos subsecuentes, el tiempo de tránsito es, entonces, 10, para $p \leq n^4$.

Finalmente, para la red mariposa, el retardo de propagación está dado por (62), y el proceso de decisión de asignación de interruptores por (70). Supóngase que las memorias internas a los elementos de conmutación poseen 2(p - 1) registros. Entonces, (62) reduce a:

$$\tau_{RM} = 4 + \lceil [2 + \log(p - 1)]/\log n \rceil \quad \ldots\ldots 75$$

Considerando que una trayectoria dada involucra logp elementos de conmutación, resulta:

$$T_{RM} = \log p\{7 + \lceil \log(\log p)/\log n \rceil + \lceil [2 + \log(p - 1)]/\log n \rceil\} \quad \ldots\ldots 76$$

Para accesos subsecuentes, el tiempo de tránsito está claramente dominado por (75).

La tabla 6 muestra los tiempos de tránsito para las diferentes arquitecturas consideradas, asumiendo que el abanico de entrada de las compuertas lógicas es 5. Cantidades entre paréntesis corresponden a accesos posteriores al primero, en caso de existir una cola en la red. La entrada T_{CR4} se refiere a una red crossbar con 4 interruptores integrados en un dispositivo de conmutación (buses de 32 bits), mientras que la entrada T_{CR16} corresponde a 16 interruptores por dispositivo, esto es, a buses de 16 bits.

Tabla 6: Tiempos de tránsito.

p	2	4	8	16	32	64
T_{BC}	5 (3)	5 (3)	7 (5)	7 (5)	9 (7)	9 (7)
T_{BM}	--- ---	--- ---	21 (15)	21 (15)	26 (20)	26 (20)
T_{CR4}	9 (6)	9 (6)	11 (6)	11 (6)	13 (7)	14 (7)
T_{CR16}	13 (10)	13 (10)	15 (10)	15 (10)	17 (10)	18 (10)
T_{RM}	9 (5)	20 (6)	33 (7)	44 (7)	60 (7)	78 (8)

De la tabla anterior se desprende que la arquitectura de buses múltiples parece constituir una pobre opción, comparada con redes *crossbar* de 1/2 de ancho de bus. En efecto, para 16 interruptores integrados en un dispositivo de conmutación y buses de 16 bits, la red crossbar es aún significativamente más rápida que la opción de buses múltiples operando con 4 interruptores por dispositivo, y buses de 32 bits. Por otra parte, de la ecuación 58 y tabla 4 se obtiene, para una misma cantidad de dispositivos para ambas opciones, $\eta = 8$. Pero, para esta relación entre procesadores y buses, el rendimiento de la máquina de buses múltiples es significativamente menor al de una máquina con red crossbar, como se desprende de la figura 27 y tabla 3 del capítulo anterior.

Redes escalonadas generan grandes retardos para primeros accesos, pero su desempeño es comparable a la red *crossbar* para accesos subsecuentes que se encuentren en cola. En consecuencia, son tanto más eficientes conforme la cantidad de información, por cada transferencia, crece. En términos absolutos, el menor tiempo de tránsito corresponde, obviamente, a la estructura de bus común.

Para una máquina física, la importancia relativa del tiempo de tránsito por la red depende del tiempo de acceso a memoria, por una parte, y de la frecuencia de operación del procesador, por otra. En términos de la tecnología utilizada en el presente trabajo, un período de reloj del procesador equivale aproximadamente a 4 retardos unitarios de compuerta, y el tiempo de acceso a memoria corresponde a 15 retardos unitarios. Para máquinas de 8 procesadores, los tiempos de tránsito dados en la tabla 6 equivalen a entre 2 y 8 períodos de reloj del procesador, y entre 0.45 a 2.2 veces el tiempo de acceso a memoria compartida, aproximadamente.

Nótese que las ecuaciones (72) a (74) definen tiempos de tránsito como $T = O(\log p)$, y la ecuación (76) como $T = O(\log^2 p)$. Esto difiere de la suposición implícita que, en este sentido, fue hecha para los modelos discutidos en el capítulo III, para los cuales el tiempo de tránsito es cero. Por otra parte, la importancia relativa del tiempo de tránsito es mayor para accesos exponencialmente distribuidos, que para el modelo de accesos simultáneos.

IV.4 Protocolos de memoria caché.

Un protocolo ampliamente utilizado para el manejo de memorias caché en los multiprocesadores de bus común es el llamado MESI, por las siglas en inglés de modificado, exclusivo, compartido (*shared*) e inválido [68]. Para la arquitectura mostrada en la figura 45, este protocolo se traduce en dos alternativas de operación:

1.- Escritura transparante (*writethrough*). Una solicitud de escritura a memoria generada por un procesador altera el contenido de la memoria caché de este procesador (pero no de las memorias caché de los demás procesadores), y de la memoria compartida. En consecuencia, esta última se encuentra siempre actualizada, y el contenido de la memoria caché es una réplica exacta de un subconjunto del espacio de direccionamiento de la memoria compartida. La dirección afectada por la escritura debe ser invalidada en las memorias caché de los demás procesadores.

2.- Escritura de actualización (*writeback*). Una solicitud de escritura a memoria generada por un procesador altera exclusivamente el contenido de la memoria caché de este procesador (asumiendo, por supuesto, que la dirección correspondiente está contenida en la memoria caché). Este modo de operación reduce considerablemente el tráfico en el bus de la memoria compartida, pero genera el problema de inconsistencia de memoria, esto es, que la memoria compartida no se encuentre actualizada con relación a las memorias caché, ni, obviamente, las memorias caché de los demás procesadores.

En el modo de escritura de actualización, una solicitud de escritura de uno de los procesadores marca la localidad afectada en la memoria caché propia como modificada, e invalida esta localidad en las restantes memorias caché del sistema. Una solicitud de lectura que no puede ser satisfecha desde

la memoria caché local, genera una solicitud de indagación en las restantes memorias caché, con objeto de localizar el dato más recientemente modificado. El controlador asociado a la memoria caché, que dispone del dato más reciente, inicia entonces una escritura de actualización a la memoria compartida. Esta transferencia puede ser interceptada por el procesador solicitante en el bus de indagación, permitiendo, así, la comunicación directa entre memorias caché. Si ninguna de las memorias caché del sistema contiene la dirección solicitada, la solicitud es atendida por la memoria compartida.

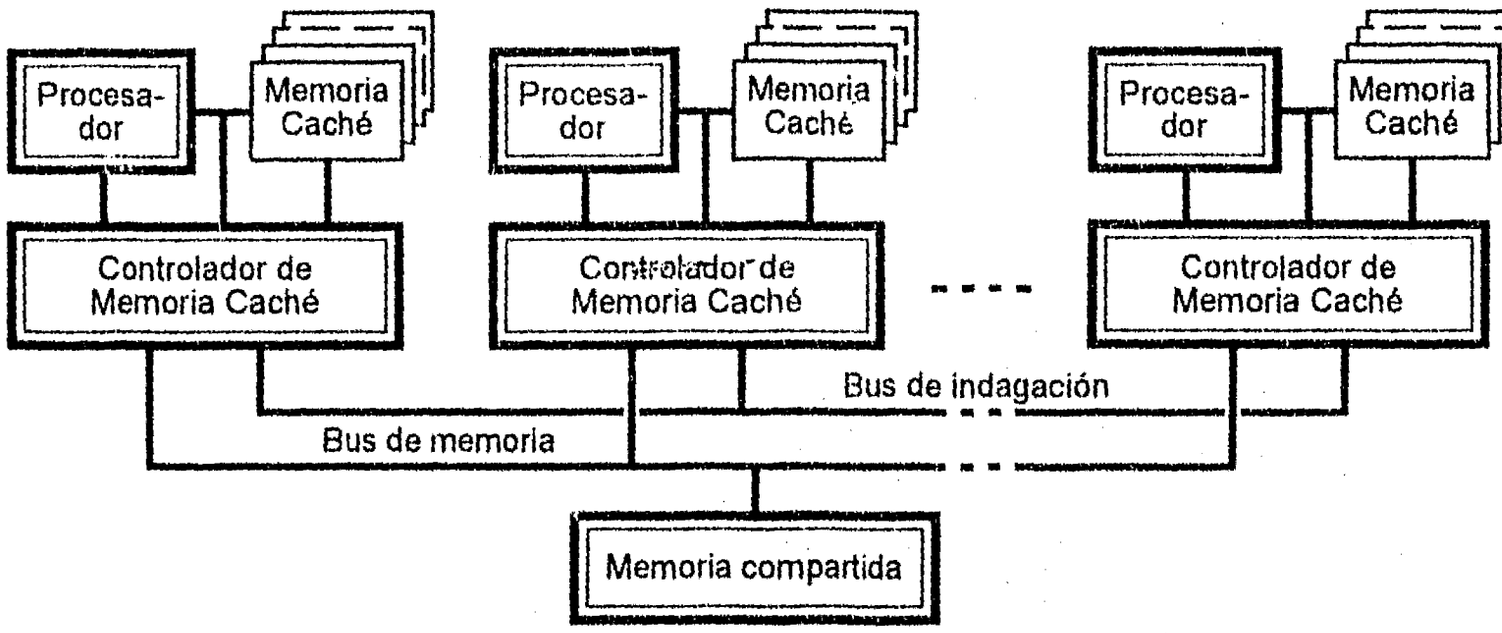


Figura 45: Multiprocesador de bus común y comunicación directa entre memorias locales

Es una característica de los controladores modernos de memoria caché que solicitudes de invalidación no generan retardos en la ejecución del programa del procesador asociado a ese controlador [67]. Por otra parte, la memoria caché está organizada como una memoria de dos puertos, y permite el acceso concurrente desde el procesador local, así como desde el bus de indagación. Para la comunicación de datos entre procesadores, el factor más importante es, desde luego, el ahorro de tiempo posible gracias a la mayor velocidad de la memoria caché, comparada con la memoria compartida. Supóngase que los tiempos de acceso a memoria caché y memoria compartida son, respectivamente, τ_C y τ_M. La comunicación directa entre procesadores requiere, entonces, un tiempo dado por τ_C, más el tiempo de tránsito T_{BC} especificado en (71). En cambio, la comunicación por medio de la memoria compartida involucra un acceso de escritura, seguido de un acceso de lectura, esto es, $2(T_{BC} + \tau_M)$.

Una limitante de la comunicación directa entre procesadores, ya señalada con anterioridad, es el formato fijo de la información, aquí de 256 bits. En consecuencia, para datos de 32 bits o menos, la comunicación directa es más rápida que el intercambio por memoria compartida solamente si se satisface:

$$\tau_C < T_{BC} + 2\tau_M \qquad \text{..... 77}$$

Una condición que, en general, no se cumple para la tecnología aquí considerada. Así, la efectividad del esquema depende, en gran medida, del formato de los datos compartidos.

El protocolo MESI resulta de difícil aplicación en las restantes arquitecturas consideradas. La limitante fundamental consiste en que los accesos de encuesta, esto es, la determinación de que el dato solicitado por un procesador se encuentre, modificado, en la memoria caché de otro procesador, son efectuados secuencialmente cuando una, o más solicitudes se presentan antes de que la actual haya sido atendida. Esto es, la cuestión se reduce a decidir si la atención secuencial de n solicitudes por el bus de indagación es más rápida que la atención paralela de estas mismas solicitudes por medio de la memoria compartida. Para $\tau_M \leq \tau_C$, la respuesta es obvia para datos de 32, o menos, bits. Para líneas de memoria caché de 256 bits, se tiene:

1.- El tiempo requerido por transferencia directa es la suma de $n\tau_C$, más un tiempo de tránsito (ecuación 71) para el primer acceso, mas (n - 1) retardos de propagación de los elementos de conmutación. En lo anterior se consideró que $T_{BC} < \tau_C$ para toda p.

2.- El tiempo de intercambio por memoria compartida de 8 palabras de 32 bits es 2 veces la suma de $8\tau_M$, más el tiempo de tránsito para la arquitectura considerada (ecuaciones 72 a 76), más 7 retardos de propagación de los elementos de conmutación. Como arriba, se consideró que $T < \tau_M$, donde T es el tiempo de tránsito para la arquitectura considerada.

Para n intercambios, el tiempo anterior debe multiplicarse por la cantidad esperada de accesos a un mismo módulo. Considérese el caso de accesos simultáneos en una máquina basada en la red *crossbar*. La figura 46 muestra los tiempos de intercambio de información requeridos, considerando parámetros representativos de la tecnología aquí empleada, para la cual $\tau_C \approx 88\tau$, donde τ es el retardo unitario de compuerta. Para la memoria compartida se reproducen los valores correspondientes a tiempos de acceso, τ_M, iguales a 15τ (60 ns) y 25τ (100 ns). La gráfica reproduce los tiempos para el bus de indagación (trazo a), intercambio por memoria compartida sin contienda por módulos (trazos b y c) e intercambio por memoria compartida, bajo las condiciones dadas en III.2.1 (acceso aleatorio) (trazos d y e).

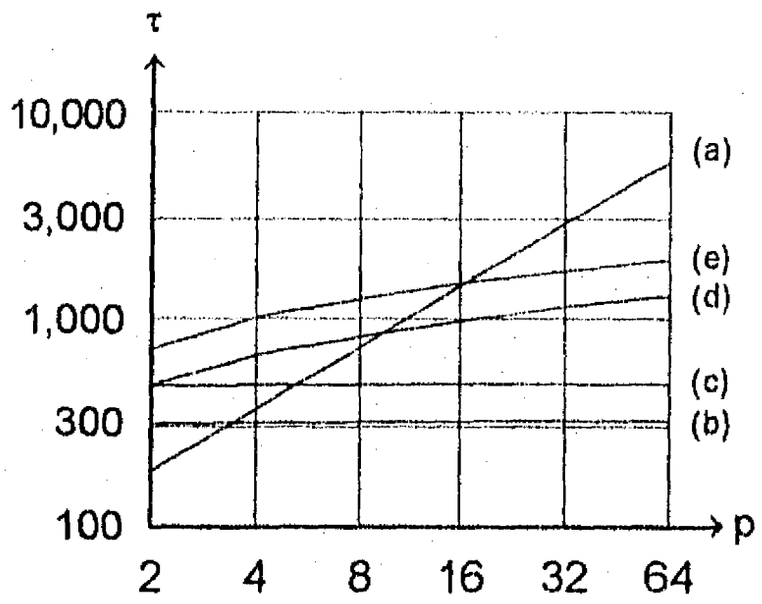


Figura 46: Tiempos de intercambio de datos por bus de indagación (a), intercambios sin contienda (b y c) y aleatorios (d y e).

Para accesos aleatorios, de la gráfica adjunta se desprende que el uso de una memoria caché para datos compartidos ofrece ventajas para $p < 8$, y para $p < 16$, respectivamente. Parece difícil establecer un modelo general para todas las arquitecturas consideradas, dada la alta dependencia de factores tecnológicos específicos. Sin embargo, está claro que el protocolo MESI es poco adecuado para multiprocesadores con arquitecturas de buses múltiples o redes escalonadas. En el capítulo VI se presenta una solución alternativa para redes *crossbar*.

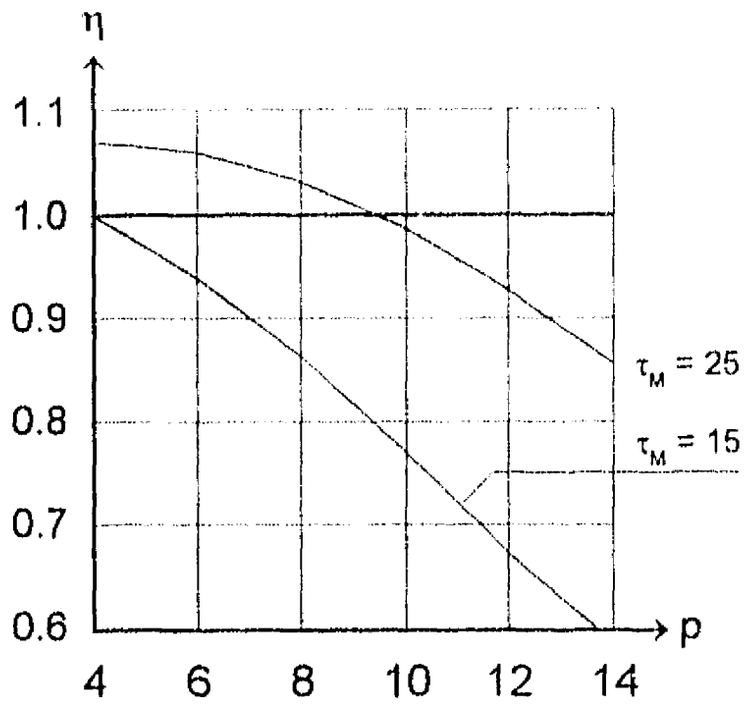


Figura 47: Rendimiento relativo del multiprocesador de bus común, comparado con la red *crossbar*, para intercambios de 256 bits.

Una observación importante que se desprende de la figura 46 consiste en que un multiprocesador de bus común, para hasta un cierto valor máximo de p y formatos de datos grandes, puede ofrecer rendimientos iguales o superiores a los de multiprocesadores con redes de interconexión más complejas, que no cuentan con memorias caché para datos compartidos. La figura 47 muestra el rendimiento relativo, para accesos exponencialmente distribuidos, de una máquina de bus común, comparado con la red *crossbar*. Se utilizaron los parámetros dados arriba y un valor de $\rho = 0.5$ para esta última.

IV.5 Estructura de interconexiones físicas.

En lo subsecuente se asumirá que la red de interconexión está constituida por un circuito impreso multicapa convencional. Es de interés comparar el número de capas requerido, la densidad de líneas de señal y los efectos de carga a que están sujetas estas líneas para las diferentes arquitecturas consideradas. Para los dispositivos de conmutación se considera tecnología C-MOS, esto es, las entradas de señal constituyen cargas esencialmente capacitivas.

IV.5.1 Número de líneas.

Con objeto de unificar la discusión subsecuente, se considerará que un bus está constituido por las líneas de señal propiamente dichas, así como por las líneas de control requeridas, tales como comandos de lectura/escritura, arbitraje, etc. Como una simplificación adicional, se asume que las líneas de control constituyen una proporción constante del número de líneas de señal. Sea W el número de líneas físicas de un bus de 32 bits, incluyendo señales de control.

Bus común. De la figura 45 se desprende que esta arquitectura da lugar a una geometría de interconexión ortogonal, constituida por (2p) buses asociados a p unidades de procesamiento, y dos buses que interconectan a estos últimos. Se ha supuesto, en general, una operación multiplexada de los buses de interconexión. Para máquinas de bus común, y, en particular, para el bus de indagación, este supuesto se traduce en limitaciones de rendimiento cuando la operación del bus requiere una inversión en el sentido del tráfico. Por lo tanto, es preferible la utilización separada de buses de direccionamiento y de datos para este último. Sea W_V el número de líneas de un bus asociado a un procesador o memoria compartida (bus "vertical"), y W_H el correspondiente a un bus que interconecte

a estos últimos (bus "horizontal"). Para buses de datos de 64 bits, y de direccionamiento de 32 bits, el número total de líneas W_T está dado, entonces, por:

$$W_{T,BC} = 5[pW_V + W_H] \quad \text{..... 78}$$

Las líneas W_V constituyen comunicaciones punto a punto, esto es, la carga observada por el dispositivo que alimenta a la línea es la capacidad de entrada de un sólo dispositivo (carga unitaria). En cambio, las líneas W_H constituyen, respectivamente, (p -1) cargas para el bus de indagación, y p cargas para el bus común.

Buses múltiples. De la figura 6 se desprende que la geometría de interconexión es ortogonal. Reconociendo que la interconexión entre un procesador con los b buses comunes constituye un interruptor (b x 1) y que lo mismo se aplica a los módulos de memoria, se obtiene:

$$W_{T,BM} = 2pW_V + bW_H \quad \text{..... 79}$$

Las líneas W_V constituyen b cargas unitarias, y las líneas W_H, 2p cargas unitarias cada una.

Red *crossbar*. De la figura 41 se desprende que la geometría de interconexión es ortogonal. Procediendo como arriba, se obtiene:

$$W_{T,CR} = p[W_V + W_H] = 2pW \quad \text{..... 80}$$

Las líneas W constituyen p cargas unitarias cada una. Si se emplea el esquema de buses de ancho reducido discutido en el punto IV.1.1, (80) debe ser afectado por el factor de reducción correspondiente.

Redes escalonadas. En general, la geometría de interconexión de estas redes es función de la gráfica que les da origen (véase la figura 14). De la figura 10 se desprende que:

$$W_{T,RM} = p[(\log p) + 1]W \quad \text{..... 81}$$

La ecuación (80) es válida para todas las redes escalonadas hipercúbicas [31]. Es importante observar que todas las líneas constituyen comunicaciones punto a punto, esto es, representan una carga unitaria cada una. La siguiente tabla reproduce los valores dados por (78 - 81) para diferentes valores de p, considerando para el caso de buses múltiples una relación $\eta = 4$:

Tabla 7: Número de líneas L, en unidades W, para las arquitecturas consideradas.

p	4			8			16			32			64		
	W_H	W_V	W_T	W_H	W_V	W_T	W_H	W_V	W_T	W_H	W_V	W_T	W_H	W_V	W_T
L_{BC}	20	5	25	40	5	45	80	5	85	160	5	165	320	5	325
L_{BM}	--	--	--	16	2	18	32	4	36	64	8	72	128	16	144
L_{CR}	4	4	8	8	8	16	16	16	32	32	32	64	64	64	128
L_{RM}	--	--	12	--	--	32	--	--	80	--	--	192	--	--	448

Para topologías ortogonales, y considerando una tecnología de montaje de superficie para dispositivos de conmutación, el circuito impreso requiere solamente tres planos de señal (lado componentes, líneas horizontales y líneas verticales de interconexión). Considerando planos de tierra y de voltaje de alimentación, estas topologías pueden montarse en un circuito de 5 capas. Para densidades de interconexión mayores, sin embargo, planos adicionales de señal, separados por planos de tierra, pueden ser necesarios. En este sentido, la red crossbar posee la menor densidad de interconexiones para un número de procesadores dado.

Redes escalonadas, en general, generan trayectorias cruzadas entre sí, que no pueden estar ubicadas en un mismo plano. Por este hecho, aunado al comparativamente alto número de líneas de interconexión requeridas, parece razonable afirmar que para un número de procesadores dado, estas redes requieren un mayor número de planos de señal que estructuras de interconexión ortogonales. En consecuencia, la realización física es más costosa.

Como fue señalado arriba, la carga capacitiva que representa una línea es función del número de interruptores interconectados por esa línea. Cuando más de un interruptor está alojado en un dispositivo de conmutación, la carga capacitiva es la correspondiente a los dispositivos interconectados.

En contraste, en una red escalonada, cada línea interconecta dos dispositivos, y la carga capacitiva es la de uno sólo, independientemente del valor de p. La carga capacitiva crece adicionalmente por la naturaleza del circuito impreso que, a las frecuencias de operación de interés, se comporta como una línea de transmisión. No es objetivo del presente trabajo desarrollar este tópico. De manera resumida, una carga capacitiva ubicada a una distancia *d* de una fuente crece en forma no-lineal (desde el punto de vista de la fuente) conforme *d* aproxima $\lambda/4$, donde λ es la longitud de onda de la tarjeta a una frecuencia dada. El efecto de carga alterna en forma cíclica entre capacitivo e inductivo, como función de *d*. En un circuito actual, el efecto de carga capacitivo puede ser varias veces mayor a la capacidad de entrada especificada para un dispositivo dado.

Resumiendo, el tamaño, esto es, la cantidad de dispositivos interconectados de una red está limitado por la capacidad de manejo de cargas capacitivas de los dispositivos de conmutación utilizados. Las redes menos afectadas por esta limitante son las escalonadas. Para la tecnología utilizada en el presente trabajo, el retardo generado por cargas capacitivas equivale aproximadamente a 0.5τ por dispositivo interconectado, donde τ es el retardo unitario de compuerta.

IV.6 Síntesis.

La elección de una determinada arquitectura depende de un gran número de factores adicionales a los aquí considerados, tales como tipo de tecnología, costo, escalabilidad, sensibilidad a fallas (robustez) y otros. Limitando la elección a lo aquí presentado, parece claro que las arquitecturas de buses múltiples no ofrecen ventajas importantes, al menos para valores de $p < 100$. Por otra parte, y para el mismo límite de p, redes escalonadas, aunque más económicas en términos de elementos de conmutación, no ofrecen ventajas significativas en lo que a cantidad de dispositivos se refiere. Considerando la complejidad de estos dispositivos por la necesidad de almacenamiento local, así como la gran latencia para establecer una comunicación, la ventaja de una menor carga capacitiva no parece suficiente argumento para preferir esta opción a otras, como la red *crossbar*. En este sentido resulta sorprendente que la mejor opción, para $p < 8$, parece constituir la arquitectura de bus común. Conviene señalar, sin embargo, que esta solución no necesariamente es la mas económica. La arquitectura de bus común debe su efectividad a memorias caché para datos compartidos, y un bus de indagación de muy alta velocidad cuya viabilidad en términos de la tecnología aquí utilizada como referencia parece cuestionable. La comparación efectuada a lo largo de la discusión previa no considera memorias caché para datos compartidos en las restantes arquitecturas, por lo que el costo de los dispositivos de conmutación adicionales puede considerarse compensado. En términos absolutos, la red *crossbar* resulta la elección de mayor velocidad posible para $p \geq 8$, y eminentemente viable de ser construida para, al menos, $p \leq 64$. A continuación se presenta un diseño experimental de una máquina de 8 procesadores, basado en una red *crossbar* modificada, que confirma lo antes expuesto.

V.- DISEÑO DE UNA MAQUINA EXPERIMENTAL

En las páginas anteriores se justificó el porqué de la elección de una arquitectura *crossbar*. La arquitectura que se describe a continuación difiere en un aspecto significativo de las arquitecturas analizadas en los capítulos anteriores, en tanto que reconoce que, para el intercambio de información entre procesadores, no es necesario que un mismo dato transite dos veces por la red de interconexión. El principio es, desde luego, aplicable a cualquier red conmutada. Supóngase que cada procesador cuenta con una memoria propia, y que el espacio de direccionamiento de esta memoria es lógicamente dividida en una área exclusiva a este procesador, y una segunda área para datos compartidos entre procesadores. Si esta memoria es diseñada como una memoria de dos puertos, uno de los cuales comunica con el procesador local, y el otro con la red de interconexión, está claro que el tránsito de información entre procesadores involucra una sola travesía por la red. Para árbitros descentralizados (véase el punto IV.2.1), este esquema requiere dos niveles de arbitraje, uno, para decidir entre accesos desde el procesador local y procesadores remotos, y otro para decidir entre (p - 1) procesadores remotos posibles. El nivel adicional de arbitraje requerido, sin embargo, no introduce retardos adicionales significativos, por las siguientes razones:

a.- Se requiere arbitraje adicional aun sin la modificación propuesta, para permitir ciclos de refresco de memoria [69]. Por otra parte, para un abanico de entrada de funciones lógicas mayor a dos, el tiempo de arbitraje para dos solicitudes simultáneas es el mismo que para tres.

b.- El procesador PENTIUM posee la característica de generar accesos en modo *pipeline*, esto es, genera la dirección de un subsecuente acceso, así como la caracterización del mismo, previo a la conclusión del acceso en curso [32]. Por lo tanto, si el tiempo de arbitraje es menor al tiempo requerido para concluir el acceso en curso, el arbitraje no genera retardos adicionales.

En la figura 48 se muestra un diagrama de bloques simplificado de la máquina, integrada por 8 unidades de procesamiento (en la figura se muestran 3), y la red de interconexión. Cada unidad de procesamiento cuenta con un procesador y memoria caché integrado al mismo (PENTIUM), y una memoria dinámica (RAM) de 8 Mbytes. Cada procesador se comunica, por medio del bus BD_j, con la red de interconexión. Similarmente, cada memoria se comunica con la red por medio del bus RD_k, donde j, k = 0, 1, ... , p - 1 y p es el número de unidades de procesamiento. Los buses BD_j y RD_k, $j \neq k$, se interconectan por medio de un elemento de conmutación. Cuando j = k, la interconexión se efectúa en la unidad de procesamiento. Esto es, la red propiamente dicha requiere p(p - 1) elementos de conmutación. Una fila adicional de elementos de conmutación interconecta el esquema de multiprocesamiento, por medio de una interface, con un procesador anfitrión. Una tarjeta maestra IBM-PC/AT compatible de tipo comercial, también basada en el procesador PENTIUM, constituye al anfitrión, encargado del procesamiento de entrada/salida (despliegue, unidades de disco etc.), así como del control y monitoreo del multiprocesador. Desde el punto de vista del anfitrión, el multiprocesador constituye un periférico ubicado en el bus PCI [70], integrado por una serie de registros de control y p

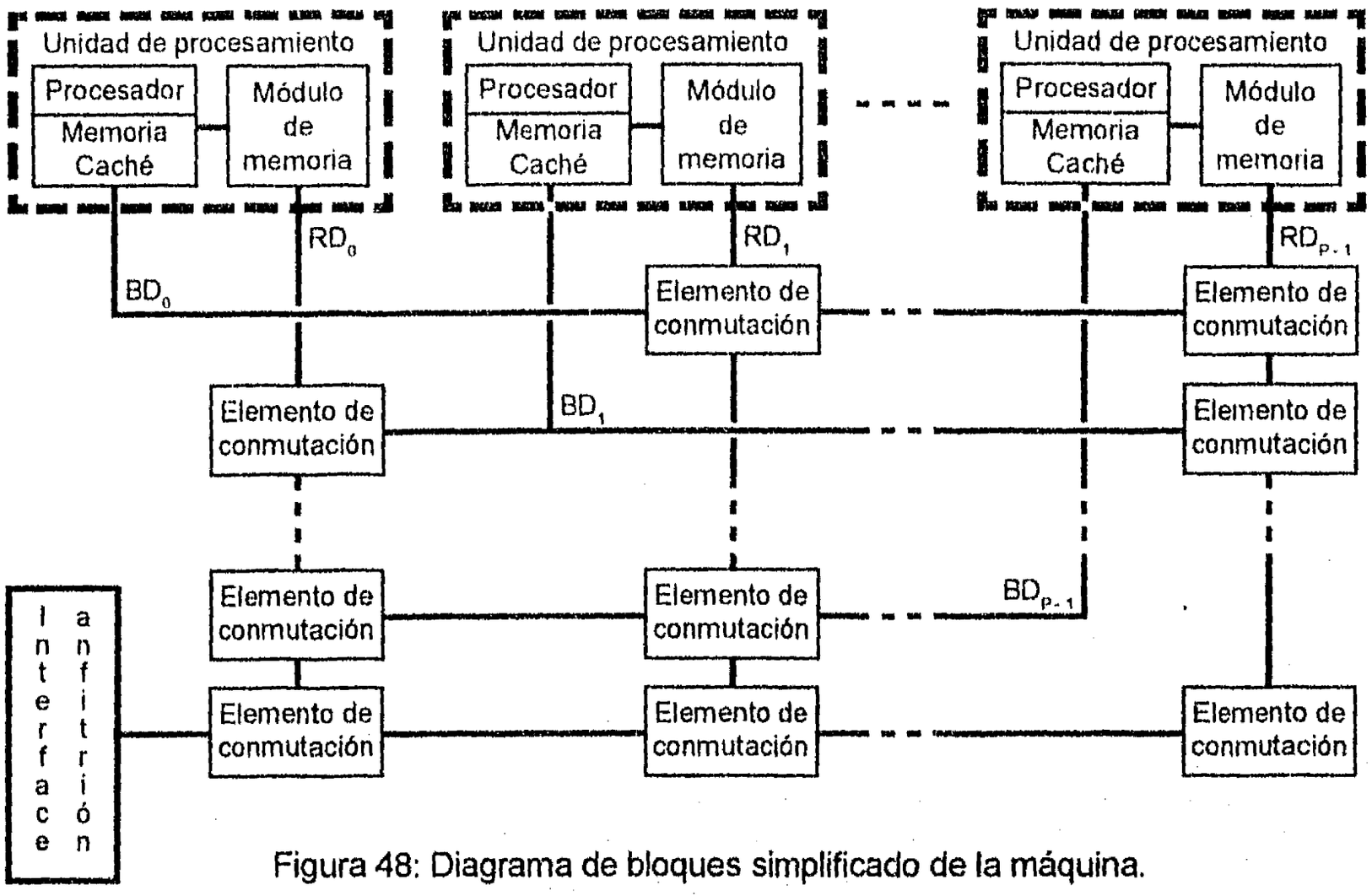


Figura 48: Diagrama de bloques simplificado de la máquina.

bancos de memoria de 8 Mbytes cada uno, ubicados a continuación del espacio de direccionamiento del anfitrión. Este último tiene acceso a estos bancos, ya sea en forma individual, y, para escritura, en forma paralela (*broadcasting*). Esto es, código y/o información común a los procesadores pueden ser escritos en forma simultánea a todas las memorias del multiprocesador.

Una característica importante del procesador PENTIUM es la de incluir, en el mismo dispositivo, dos memorias caché de 8 Kbytes, una para el código, y otra para los datos [32]. En cada solicitud de acceso a memoria, el dispositivo especifica la categoría a la que corresponde la transferencia deseada. Por otra parte, puede ser programado para dividir determinado espacio de direccionamiento en páginas, con atributos individualmente seleccionables para cada una. En particular, una página dada puede ser inhibida para su potencial inclusión en la memoria caché del procesador. Esto es, un espacio de direccionamiento para los datos compartidos puede ser especificado por *software* en forma tal, que las memorias caché no generen réplicas de los mismos. Las señales de control activadas por el dispositivo para cada acceso a memoria, informan a la lógica externa de los atributos de la página direccionada, permitiendo, así, la separación lógica de los espacios de direccionamiento, sin necesidad de dispositivos externos adicionales.

La figura 49 muestra un diagrama de bloques de las unidades de procesamiento. Con la salvedad del procesador PENTIUM y los módulos de memoria, cada bloque corresponde a un dispositivo de arreglo de compuertas programable (FPGA, [64]), diseñado para la presente aplicación. De manera resumida, las funciones de cada dispositivo son las siguientes:

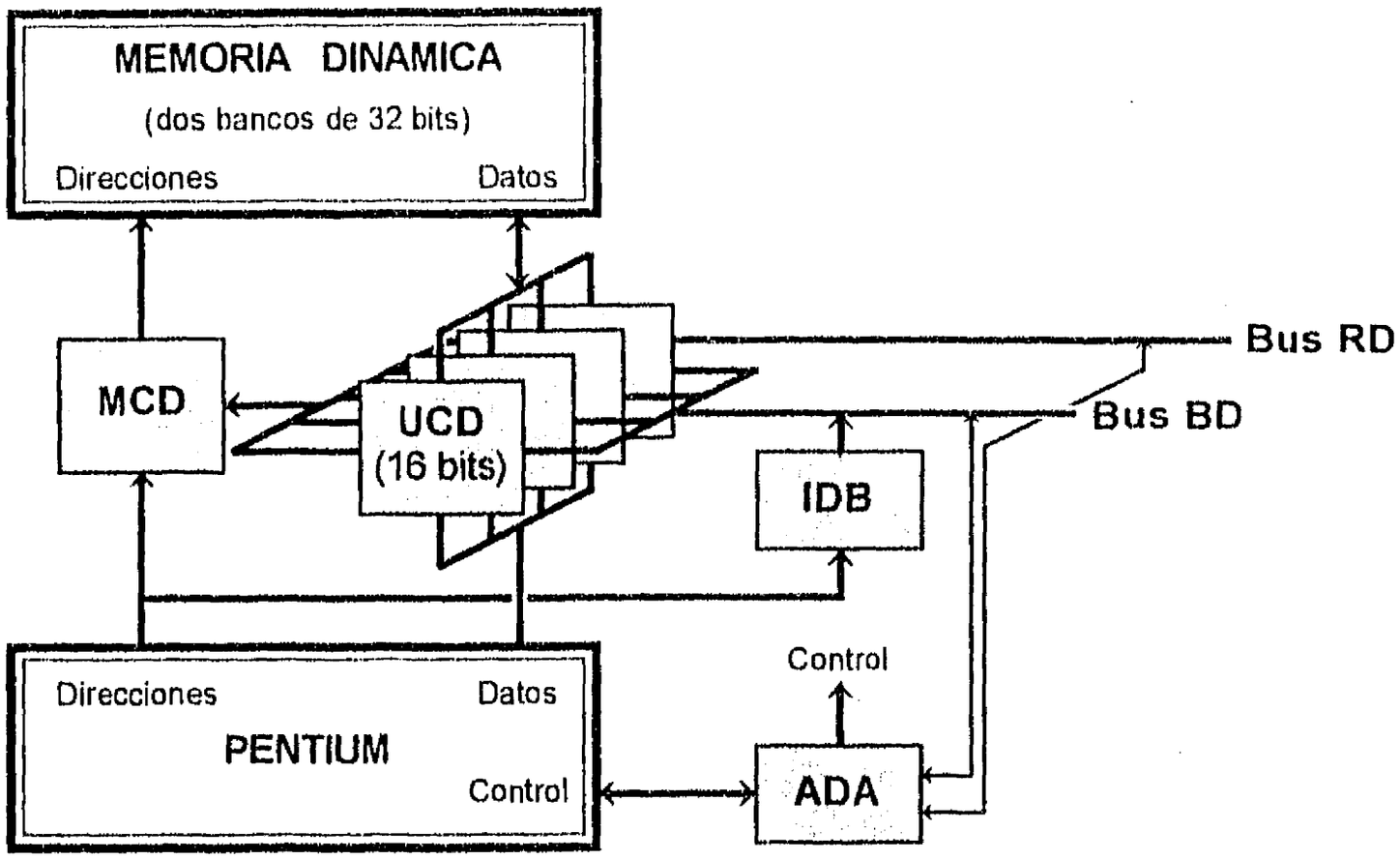


Figura 49: Diagrama de bloques de la unidad de procesamiento.

Arbitraje y control de memoria. El dispositivo ADA resuelve el arbitraje entre solicitudes de acceso a memoria de refresco, del procesador local y de procesadores remotos, y genera las señales de control tanto para la memoria, como para los demás dispositivos que conforman el diseño.

Multiplexaje y control de direccionamiento. El dispositivo MCD transmite el direccionamiento generado, ya sea, por el procesador local, o por alguno de los procesadores remotos. El dispositivo conforma el bus de direcciones para uno de tres tipos de acceso posibles (acceso local a código/datos exclusivos, acceso local a datos compartidos y acceso remoto a datos compartidos) y efectúa la conmutación entre direcciones de fila y columna requerida por la memoria dinámica, bajo control del dispositivo ADA. Por otra parte, genera las direcciones adicionales requeridas para accesos tipo ráfaga, esto es, para transferencias de líneas de 256 bits entre la memoria caché y la memoria dinámica.

Conmutación de datos. Cuatro dispositivos UCD (16 bits por dispositivo) efectúan la conmutación bidireccional de datos entre los diferentes buses de la unidad de procesamiento. Las posibles vías de comunicación son las siguientes:

a.- Procesador local - Memoria. Comunicación directa para escritura, y por medio de registros para lectura, de datos de hasta 64 bits.
b.- Procesador local - Elementos de conmutación de la red de interconexión. Igual como arriba, por medio del bus BD.
c.- Elemento de conmutación de la red de interconexión - Memoria. Comunicación bidireccional, por medio de registros. Las transferencias son controladas por el elemento de conmutación de la red y efectuadas por medio del bus RD.

Interface de Bus. El dispositivo IDB constituye el dispositivo de interface entre el procesador, y los elementos de conmutación de la red de interconexión asociados al mismo. Bajo el control del dispositivo ADA, por conducto del dispositivo IDB, el procesador envía el direccionamiento requerido para los accesos a los módulos de memoria remotos.

El dispositivo IDB cumple una serie de funciones adicionales. Por una parte, contiene un puerto de lectura, dedicado al número de identificación (PIN) del procesador. Este número es establecido por medio de un banco de interruptores (*DIP switch*), y puede ser interrogado por el procesador. Por otra parte, y para facilitar la sincronización entre procesadores, el dispositivo implementa un semáforo en *hardware*, mediante un bit ubicado en un byte de lectura/escritura, en la posición correspondiente al número de identificación del procesador. Este bit puede ser leído por los demás procesadores. Así, la conclusión de un proceso en cada uno de los procesadores puede ser identificada mediante la activación del bit correspondiente a ese procesador. La activación de uno de estos bits, o, alternativamente, la de todos, puede ser indagada por *software* o causar una interrupción al procesador, por medio de un rudimentario controlador de interrupciones ubicado en el mismo dispositivo. Finalmente, incluye una bandera de lectura/escritura para propósitos de control desde el procesador anfitrión.

Elemento de conmutación (dispositivo EC). Asociado a cada procesador, existen 8 elementos de conmutación localizados en la red de interconexión (figura 48), de los cuales 7 permiten la comunicación con otros tantos módulos de memoria remotos, y el octavo con el procesador anfitrión del multiprocesador (figura 48). Dos elementos de conmutación son alojados en cada dispositivo, que contiene la lógica de priorización de acceso requerido por la arquitectura, y maneja el protocolo de comunicación entre procesador, por una parte, y memoria remota, por otra. Para ello, y como respuesta a la solicitud enviada por el dispositivo ADA del procesador solicitante, genera la solicitud de acceso al controlador de memoria asociado a la memoria destino, y las diferentes señales de habilitación requeridas por el controlador de direcciones (dispositivo MCD) y por los conmutadores de datos (dispositivo UCD).

Interface anfitrión (dispositivo PCI). Finalmente, el dispositivo PCI (figura 48) efectúa la conversión de protocolo entre el bus PCI del procesador anfitrión, y el protocolo utilizado en el esquema de multiprocesamiento, para permitir la transferencia de datos entre ambos. Por otra parte, desempeña funciones de inicialización y de control del multiprocesador. El dispositivo incluye registros de estado, que permiten la supervisión, desde el procesador anfitrión, del esquema de multiprocesamiento.

Para poder describir los dispositivos citados arriba en mayor detalle, es necesario analizar previamente algunos de los diferentes tipos de solicitudes de acceso generadas por el procesador PENTIUM. Estos se presentan a continuación.

V.1 Secuencias de acceso a memoria.

Para propósitos de la presente descripción, los tipos de acceso de interés, generados por el procesador PENTIUM, son los siguientes:

1.- Accesos de lectura/escritura a memoria, de uno, dos y cuatro bytes.
2.- Accesos de lectura/escritura a memoria tipo ráfaga (burst cycles), esto es, transferencias de 256 bits como 4 palabras de 64 bits cada una.
3.- Accesos a memoria como respuesta a interrupciones.
4.- Accesos de lectura/escritura a puertos de entrada/salida.

Estos tipos de acceso, así como otros de naturaleza más especializada, se encuentran descritos en forma detallada en el manual del dispositivo [32]. Aquí se describirán brevemente los tipos (1) y (2), por constituir los accesos que, en mayor medida, determinan los requisitos de diseño de los dispositivos de la máquina. En lo subsecuente, a una señal activa en cero lógico se le agregará el símbolo "#". Por otra parte, a la transición cero a uno lógico de una señal se le denotará por "transición positiva", y a la transición uno a cero por "negativa".

El procesador PENTIUM opera con base en un oscilador de reloj externo (aquí, de 60 MHz). Todas las señales de entrada y salida al dispositivo están referidas a la transición positiva del oscilador. En la figura 50 se muestra un diagrama de tiempos de una secuencia de accesos a memoria, consistente en un acceso de lectura, seguido de un acceso de escritura y un acceso de lectura tipo ráfaga de 256 bits. En T_1, el procesador emite la solicitud de lectura activando la señal ADS# (*address strobe*), y establece las señales de identificación de la transferencia solicitada. La señal M/IO# distingue entre accesos a memoria (=1) y espacio de entrada/salida (=0). D/C# distingue entre datos (=1) y código (=0) y, finalmente, W/R# entre accesos de escritura (=1) y de lectura (=0). En conjunto con estas señales, la dirección de la transferencia deseada es ubicada en el bus de direcciones del dispositivo (no mostrada).

T_2 constituye un estado de espera, originado por la incapacidad del sistema externo de proporcionar la información deseada en este tiempo. El sistema externo señala la disponibilidad del dato activando la señal BRDY# (*burst ready*) en T_3. La información es leída por el procesador en la misma transición en que BRDY# es detectada activa, y debe encontrarse en las líneas del bus de datos correspondientes al byte, o los bytes, especificados por la transferencia. El dispositivo activa hasta 8 líneas de habilitación de byte (BE0# - BE7#), y no proporciona las direcciones A_0 - A_2.

Durante T_1, el procesador informa al sistema externo si la lectura deseada es susceptible de ser almacenada en la memoria caché local, activando la señal CACHE# (no mostrada). Por otra parte, la señal KEN# (*cache enable*) informa al procesador si el sistema externo puede proporcionar (=0), o no (=1) una línea completa. Para este ejemplo, el procesador detecta en T_3 (en conjunto con BRDY#) que solamente un dato será proporcionado, y no espera datos adicionales.

T_4 constituye un estado de espera generado por el procesador, en virtud de que el siguiente acceso es de escritura, y requiere una inversión del sentido del bus de datos. Durante este tiempo, el estado de las líneas de identificación de ciclo está indefinido. En T_5 se inicia un ciclo de escritura, de manera similar a T_1 (exceptuando el estado de W/R#). El dato a escribir es ubicado por el procesador en el bus de datos en T_6.

Durante T_6, el sistema externo informa al procesador que puede recibir una siguiente solicitud, activando NA# (*next address*). Dos ciclos después (en T_8), el procesador genera ADS# y las señales de control correspondientes a una solicitud de lectura, susceptible de ser almacenada en memoria caché. Nótese que esta solicitud se produce independientemente de si el acceso de escritura está, o no, concluido. Aquí, el acceso de escritura se concluye en T_7. La activación de KEN# en T_{10} informa al procesador que el sistema externo puede proporcionar una línea de datos completa, por lo que éste espera tres datos adicionales para concluir la transferencia.

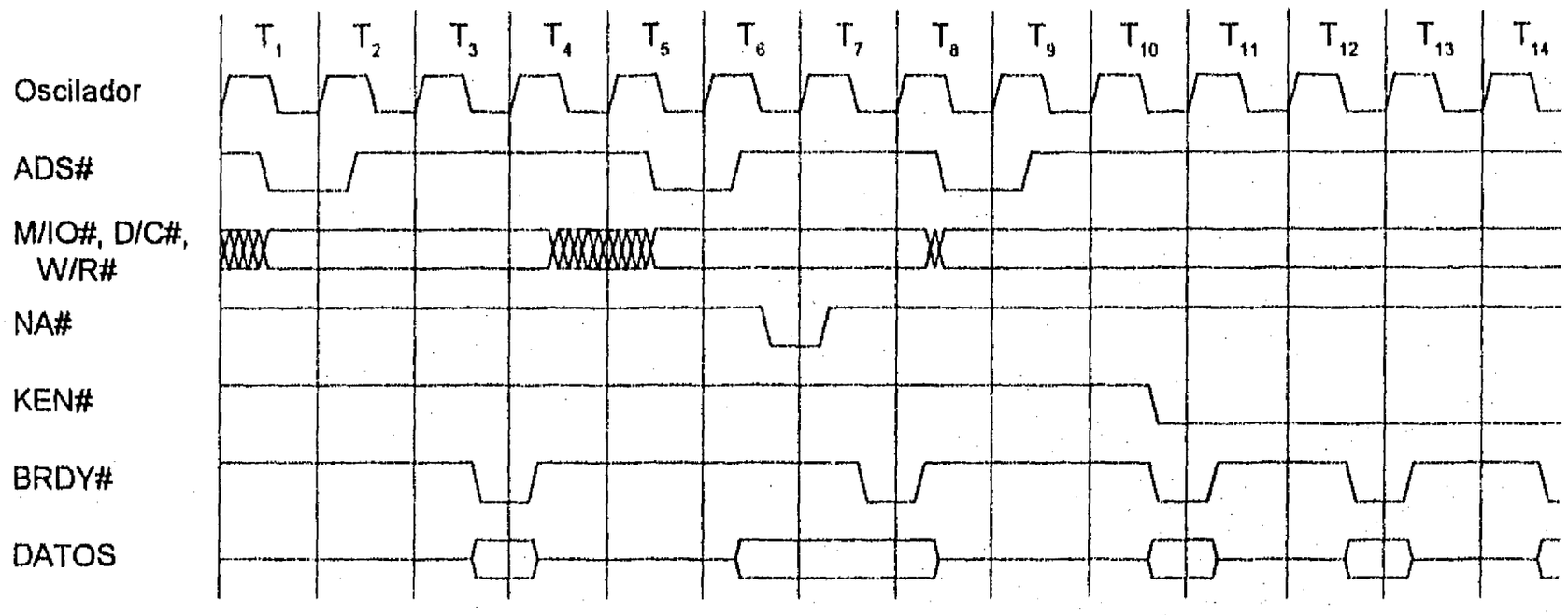


Figura 50: Accesos de lectura y escritura, seguidos de un acceso de lectura de línea.

Las memorias dinámicas modernas ofrecen una serie de modos de operación [69], de los cuales los utilizados en el presente diseño se muestran en la figura 51. En virtud de que la mayoría de los accesos a código y datos exclusivos involucran a las memorias caché del procesador, se buscó optimizar la velocidad de transferencia para estos accesos. Por lo tanto, el diseño proporciona la operación en modo página (*page mode*) para palabras de 64 bits. La figura reproduce las señales de control RAS# (*row address strobe*) y CAS# (*column address strobe*), requeridas para un acceso de escritura, seguido de un acceso de lectura de línea de 256 bits, en modo página.

En T_3 el dispositivo de memoria registra la dirección de fila, proporcionada por lógica externa, mediante la transición negativa de RAS#. Al término de T_3, la lógica externa proporciona la dirección de columna. El dispositivo de memoria registra la dirección de columna mediante la transición negativa de CAS#.

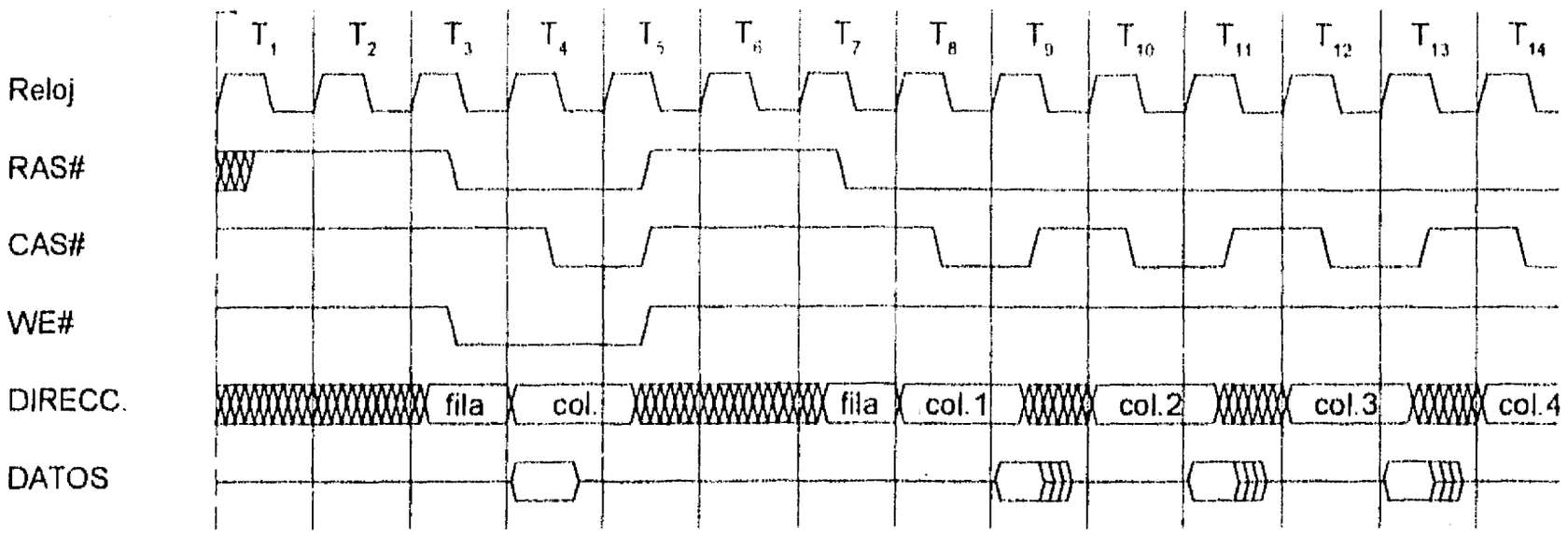


Figura 51: Acceso de escritura, seguido de un acceso de lectura de línea.

Para escritura (WE# = 0), el dato debe ser suministrado previo a la transición negativa de CAS#, y mantenido hasta un cierto tiempo posterior a esa transición. Durante lectura, el dispositivo de memoria proporciona el dato un cierto tiempo posterior a la transición negativa de CAS#, y lo mantiene, al menos, hasta la desactivación de esta señal.

La frecuencia de repetición de accesos al dispositivo está limitada, por entre otros factores, el tiempo de desactivación mínimo de RAS# (pre-carga de RAS#), aquí dado por los períodos T_1, T_2 y T_5, T_6. Cuando entre dos, o más, accesos subsecuentes la dirección de fila permanece constante, es posible controlar al dispositivo mediante la señal CAS# únicamente. En este modo de operación (modo página), la máxima frecuencia de operación es función del tiempo de pre-carga de CAS#, aquí, los períodos T_9, T_{11} y T_{13}. Obsérvese que las direcciones generadas durante las transferencias de líneas de 256 bits están alineadas sobre fronteras de 8 bytes, y no cruzan fronteras de múltiplos enteros de 32 bytes. Consecuentemente, estas transferencias siempre pueden ejecutarse en modo página, si no trascienden fronteras dadas por el dispositivo de memoria (aquí, de 1 Kbyte).

V.2 Condicionantes del diseño.

En el capítulo IV se hizo alusión a la dificultad de operar al multiprocesador con base en una señal de reloj común a todos los procesadores, consistente en los retardos inherentes a los circuitos impresos. Por el mismo argumento, se excluyó la posibilidad de lograr una intercomunicación síncrona.

Los dispositivos utilizados en el diseño son arreglos de compuertas programables (FPGA) de Texas Instruments, Inc., números de parte TPC1225A-1, y TPC1280A-1, en empaque de plástico tipo PQFP de, respectivamente, 100 y 160 terminales [64]. Estos dispositivos ofrecen un retardo típico de compuerta de 4.5 ns, esto es, un período de reloj de 60 MHz equivale a menos de cuatro retardos de compuerta. En consecuencia, se decidió reducir la frecuencia de operación de las componentes a la

mitad de la frecuencia de operación de los procesadores, y especificar para éstos una frecuencia máxima de 66 MHz. La única componente que opera a esta frecuencia (además del procesador) es el dispositivo ADA que, entre otras funciones, constituye el generador de reloj para cada unidad de procesamiento. A partir de un oscilador maestro, el dispositivo genera:

1.- La señal de reloj del procesador, de la misma frecuencia que la señal de entrada, pero con un retardo que optimiza la diferencia de fase con relación a las demás señales de reloj de la unidad de procesamiento, y el procesador.
2.- La señal de reloj de sistema, a la mitad de la frecuencia del oscilador maestro (SNCLK). Esta señal es la referencia para todas las demás componentes de la unidad de procesamiento, así como de los elementos de conmutación del bus de interconexión, asociados a la memoria de esa unidad.
3.- Una señal de reloj de igual frecuencia, pero desfasada 180 grados con relación a la señal de reloj del sistema (SDCLK). Esta señal es utilizada por el dispositivo MCD, para efectuar la conmutación entre direcciones de fila y columna para los dispositivos de memoria.

Resumiendo, la operación de un procesador es asíncrona con relación a los demás. Los elementos de conmutación (EC) de la red de interconexión asociados a una memoria en particular operan en forma síncrona con ésta. En consecuencia, la comunicación entre un procesador y una memoria remota involucra la sincronización entre procesador y el elemento de conmutación únicamente. Los dispositivos pertenecientes a una unidad de procesamiento operan, desde luego, en forma síncrona con el procesador.

A continuación se describe el diseño de la máquina en mayor detalle. La descripción se basa en el análisis de los diferentes tipos de transferencia de datos posibles en la arquitectura, mas que en la descripción de cada uno de los dispositivos que la conforman. En particular, se analizan:

- Transferencias entre el procesador y la memoria local.
- Transferencias entre el procesador y la memoria remota.
- Transferencias entre el procesador anfitrión y las memorias de las unidades de procesamiento.
- Los esquemas de arbitraje entre los accesos.

Se concluye con una breve descripción de las funciones de monitoreo y control proporcionadas por la arquitectura. Cabe señalar que el diseño fue extensamente simulado antes de ser construido. Para ello se utilizaron los paquetes de simulación de Viewlogic Systems [62], y los parámetros de retardo para los diferentes dispositivos que conforman a la arquitectura, generados por el paquete de programación de los dispositivos FPGA de Actel Corp. [65]. Los valores numéricos de retardos proporcionados a continuación, así como las gráficas de tiempo, son resultados de estas simulaciones.

V.3 Transferencias entre procesador y memoria local.

Las vías de comunicación para las direcciones y los datos utilizadas para transferencias entre el procesador y la memoria local se muestran en la figura 52. Para los accesos de escritura, el dispositivo UCD constituye esencialmente un multiplexor con salida deshabilitable, que permite seleccionar entre las transferencias del bus de datos del procesador (PD), y de la red de conmutación. Para los accesos de lectura, el dato proporcionado por la memoria es almacenado en un registro transparente, esto es, la salida Q del registro refleja el estado de la entrada D, mientras la señal de habilitación G está activa, y retiene el último valor de la entrada cuando la señal de habilitación se desactiva. El registro es requerido debido a que la memoria proporciona el dato únicamente mientras CAS# está activo, y no el tiempo requerido por el procesador para la lectura.

El direccionamiento de fila y columna es retenido por el dispositivo MCD por medio de dos registros (a) para cada línea de direccionamiento de memoria (en la figura se muestra la línea MA_3). Una característica importante de los dispositivos FPGA utilizados en el diseño, consiste en que la unidad funcional básica es el multiplexor, y no una compuerta lógica convencional. Los registros (a) incluyen la función de un multiplexor de dos entradas, y el valor almacenado depende del estado de la línea de selección S. Los valores almacenados en los registros (a) son secuencialmente transferidos a las líneas de direccionamiento de la memoria por medio de los registros (b) y (c), que constituyen, en conjunto, un registro maestro-esclavo convencional. Para las líneas de direccionamiento MD_0 - MD_2, los registros (b) forman parte de una máquina de estado, que genera las direcciones adicionales requeridas para accesos a líneas de 256 bits, dada la dirección del primer acceso.

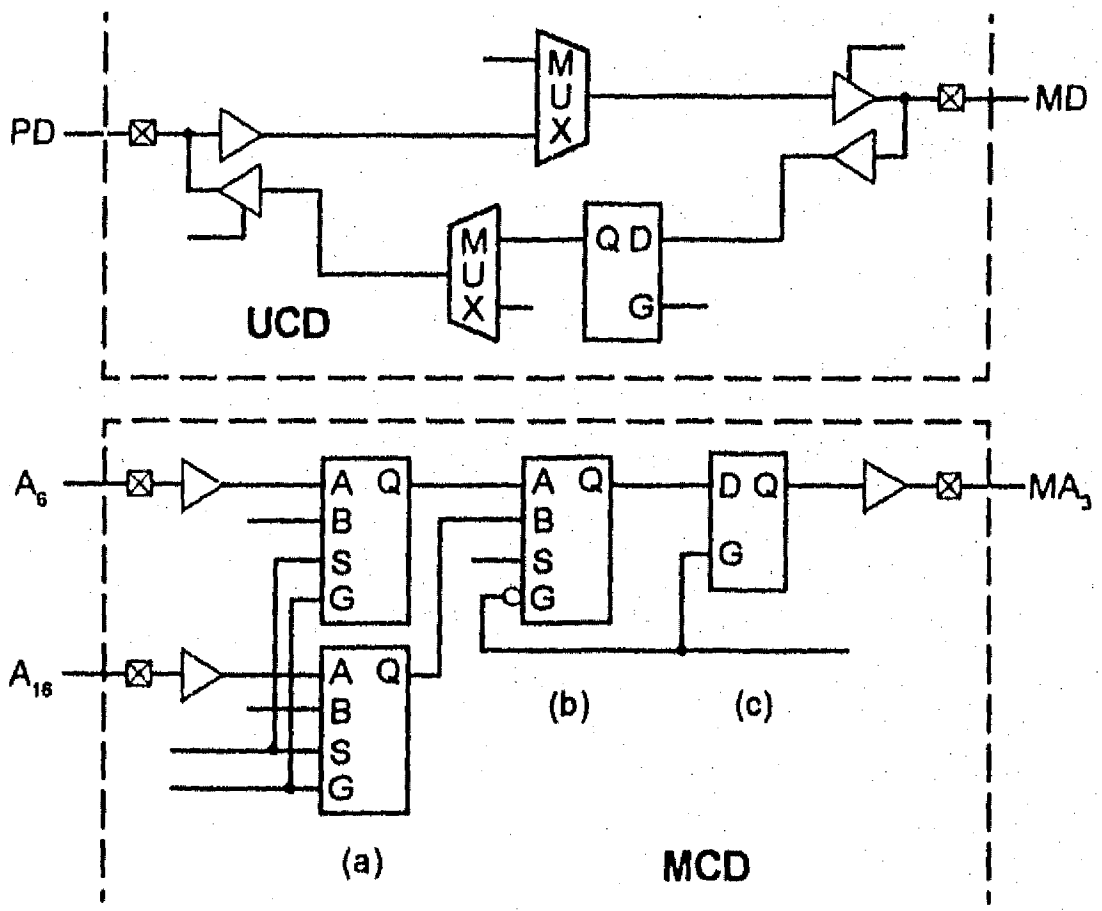


Figura 52: Trayectorias de datos y direcciones, para accesos locales.

El dispositivo MCD contiene un detector de identidad que compara la dirección actual, con la del acceso previo. Las transferencias son controladas por el dispositivo ADA y tratadas como accesos normales, o accesos en modo página, de acuerdo con el resultado de la comparación.

Supóngase que el dispositivo ADA inicia una transferencia solicitada por el procesador como consecuencia de la mayor prioridad de esta solicitud. El esquema de arbitraje se discute más adelante. El inicio de la transferencia es señalado por el dispositivo ADA al dispositivo MCD, por medio de la desactivación de la señal SYSQS (figura 53). El dispositivo MCD retiene el direccionamiento presente hasta antes de la desactivación de SYSQS, y transmite la dirección de fila (A_{16} en la figura 52) a la línea MD_3. Este valor es retenido durante 2.5 períodos de SNCLK, y posteriormente substituido por la dirección de columna (A_6). El dispositivo ADA genera las señales RAS# y CAS#, así como WE#, requeridas por la memoria.

La figura 53 muestra, aproximadamente a escala, las principales señales involucradas en una transferencia de lectura y escritura, para un período de reloj del procesador (CPUCLK) de 16 ns. Con relación al dispositivo de memoria utilizado (TM124BBK32-60, 4Mbytes x 32 bits, de Texas Instr. [69]), el diseño proporciona adecuados márgenes de seguridad para los tiempos de acceso, antelación y retención especificados. Durante los accesos de lectura, el dato es ubicado en el bus PD con una antelación de 12 ns, con respecto a la transición positiva de CPUCLK, con la cual BRDY# es detectado activo (al inicio de T_6). La antelación mínima requerida por el procesador es 6 ns. Para los accesos de escritura, el dato se encuentra ubicado en el bus MD al menos 20 ns previo a la activación de CAS#, esto es, con un margen del mismo valor.

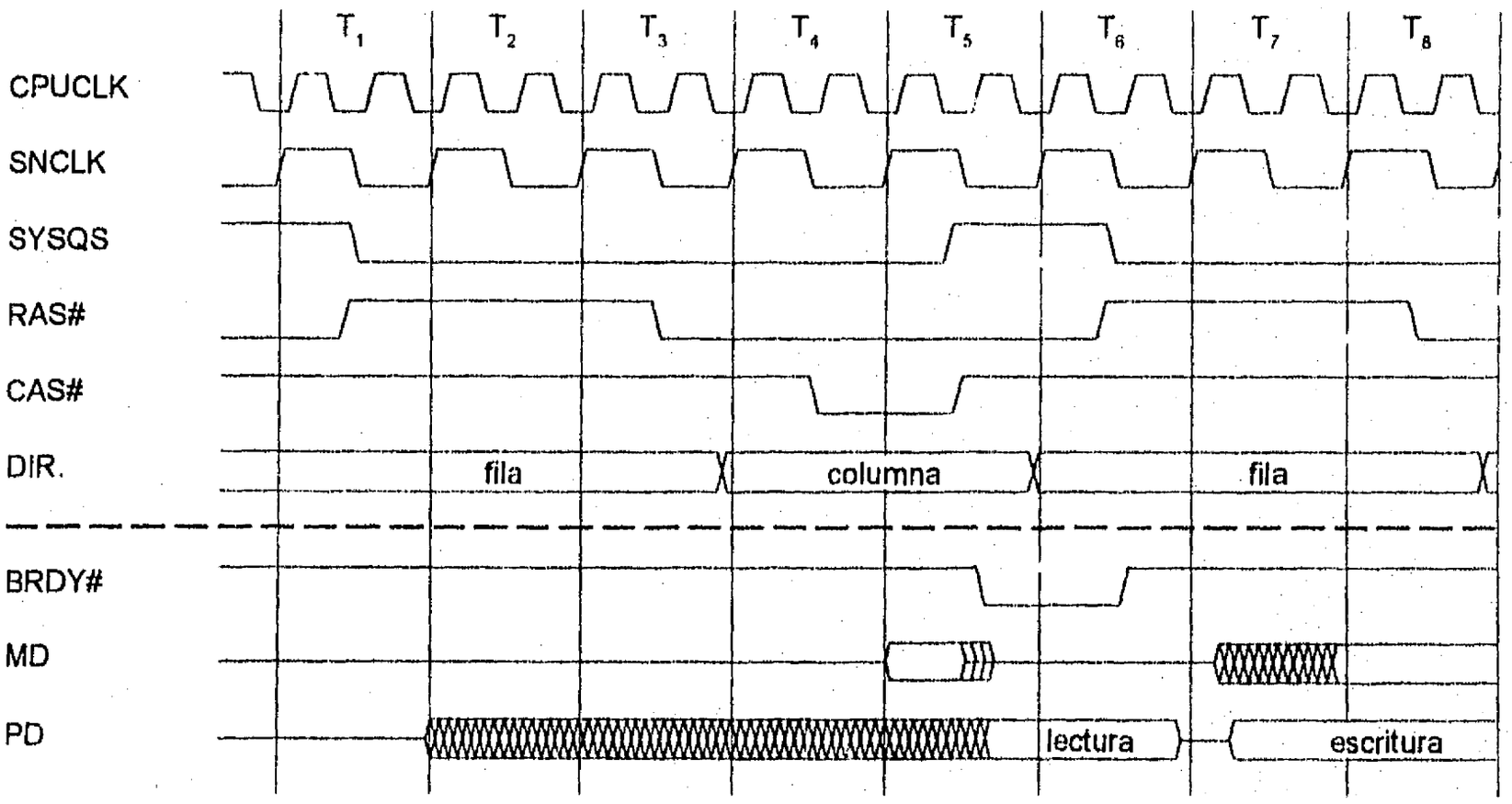


Figura 53: Accesos de lectura y escritura a memoria local

Cada acceso a memoria local requiere 5 ciclos de SNCLK. Para los accesos tipo ráfaga (transferencias de 256 bits), se producen los siguientes cambios con relación a la secuencia dada en la figura 53:

1.- Al concluir T_5, T_7 y T_9, el dispositivo MCD genera la siguiente dirección de columna, a partir de la dirección proporcionada en T_3 (véase [32]).
2.- El dispositivo ADA genera 3 pulsos adicionales de CAS# en T_7, T_9 y T_{11}, así como 3 activaciones de BRDY# en T_8, T_{10} y T_{12}. Una transferencia de línea de 256 bits se ejecuta en 12 ciclos de SNCLK.
3.- Cuando la dirección de fila del acceso previo coincide con la del acceso actual (*page hit*), el tiempo de pre-carga de RAS# es omitido, y solamente la dirección de columna es proporcionada por el dispositivo MDC. Esto es, T_1 y T_2 no son requeridos, y la transferencia de línea de 256 bits se ejecuta en 10 ciclos de SNCLK.

V.4 Transferencias entre procesador y memoria remota.

Para el diseño aquí descrito, los datos compartidos no son susceptibles de ser almacenados en memoria caché. En consecuencia, el formato de los datos transmitidos es de uno, hasta 4, bytes. Para transferencias de 16 o más bits, cuyas direcciones no estén alineadas sobre fronteras de 32 bits, el procesador genera un acceso adicional. Las trayectorias seguidas por las direcciones y los datos entre el procesador y la memoria remota, se muestran en la figura 54. La dirección generada por el procesador es ubicada en el bus BD por el dispositivo IDB. Uno de los elementos de conmutación asociados al procesador solicitante reconoce la dirección como correspondiente a la memoria con la que está interconectado, y captura la dirección en un registro transparente. El proceso de selección del elemento de conmutación se describe más adelante. El registro de direcciones es necesario, en virtud de que otro acceso remoto a la memoria seleccionada puede encontrarse en proceso, y, en consecuencia, el dispositivo MCD de esta memoria puede no estar en posibilidad de aceptar la nueva dirección.

El procesador, por medio del dispositivo ADA, genera una señal de solicitud de acceso remoto a los elementos de conmutación asociados al mismo. El elemento de conmutación seleccionado resuelve el arbitraje entre ésta, y otras solicitudes dirigidas a la misma memoria. Si la solicitud es la de mayor prioridad, activa una señal de solicitud al dispositivo ADA, y envía la dirección al dispositivo MCD, correspondientes a la memoria direccionada. El dispositivo ADA, a su vez, resuelve el arbitraje entre la solicitud remota, y las solicitudes locales y de refresco que pudieran estar activas. Si la solicitud remota es la de mayor prioridad, emite una señal de aceptación al elemento de conmutación, e inicia la transferencia deseada.

Para lectura, el dispositivo UCD de la memoria registra el dato como fue descrito en el punto anterior, y lo ubica en el bus RD. El dispositivo EC transmite el dato al bus BD, de donde es capturado por un registro transparente ubicado en el dispositivo UCD del procesador solicitante. El dispositivo EC

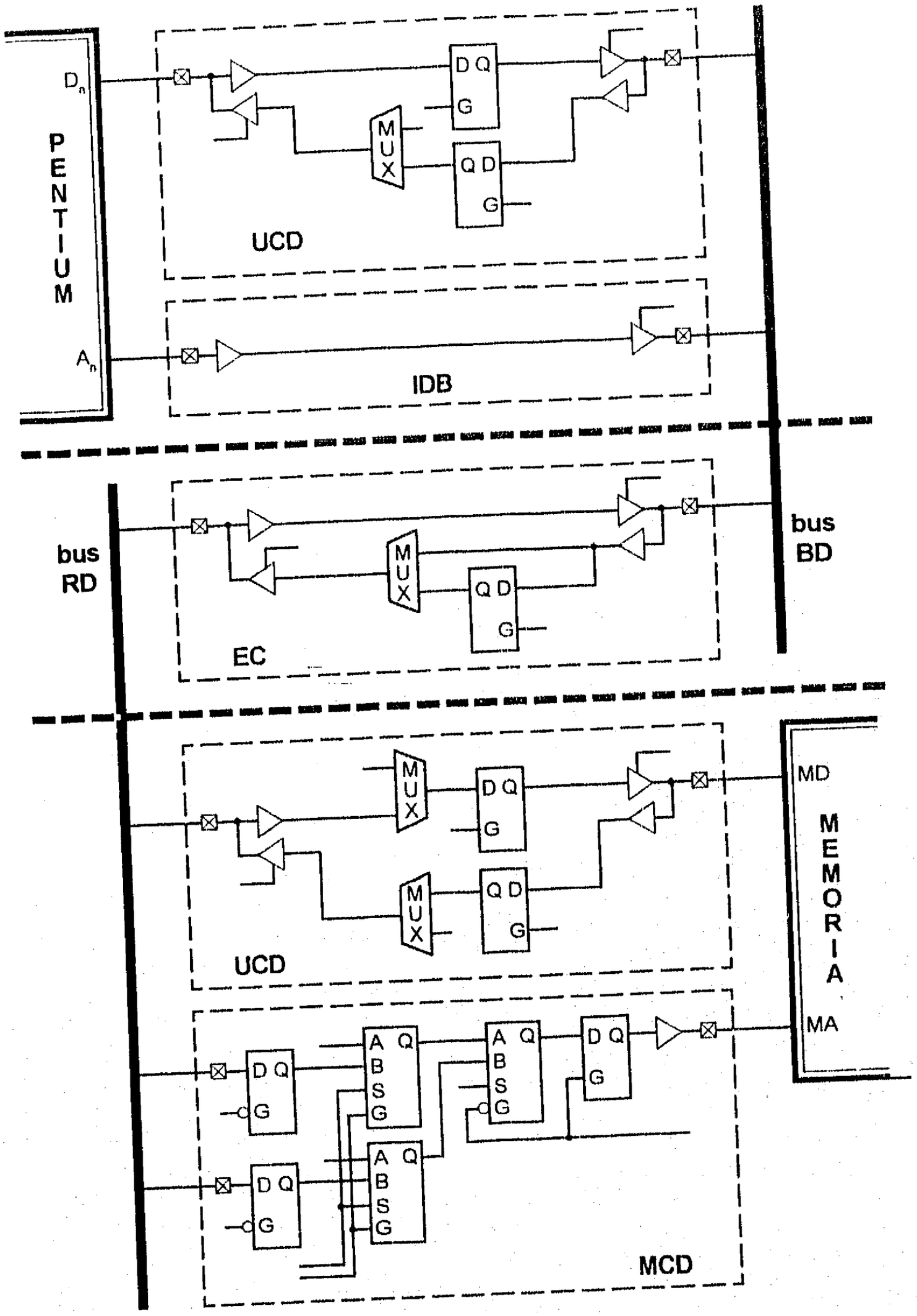


Figura 54: Trayectorias de datos y direcciones, para accesos remotos

señala al dispositivo ADA del procesador solicitante la disponibilidad del dato, que, en consecuencia, genera la señal BRDY# al procesador. Como la transferencia es asincrona, el esquema de doble registro libera a la memoria antes de que el procesador solicitante haya, en efecto, recibido el dato, para poder atender solicitudes subsecuentes.

Para los accesos de escritura, el dispositivo UCD del procesador solicitante registra el dato y lo ubica en el bus BD al ser habilitado por el dispositivo EC. El dato es transmitido de manera directa por este dispositivo al bus RD y, por el dispositivo UCD, a la memoria seleccionada. El dispositivo EC señala la conclusión de la transferencia al recibir la aceptación del dispositivo ADA asociado a la memoria, aún antes de que el dato fuera, efectivamente, escrito. El procesador solicitante puede, en consecuencia, emitir nuevas solicitudes de acceso. Sin embargo, si la solicitud es dirigida a la misma memoria remota, el dispositivo EC bloquea la solicitud hasta recibir la confirmación de la conclusión efectiva del acceso previo. Las señales de control de los registros, así como de habilitación de la salida de los dispositivos afectados por una transferencia remota, son generadas por el dispositivo EC. Como ya fue señalado, éste opera en forma síncrona con la memoria a la que está asociado. En consecuencia, la necesidad de sincronizar transferencias se limita a la interface entre los dispositivos EC y UCD del procesador solicitante, función realizada por los registros de datos de este último.

La figura 55 muestra, aproximadamente a escala, el diagrama de tiempos correspondiente a un acceso de lectura a memoria remota. El dispositivo ADA de la unidad de procesamiento solicitante genera BACC en T_1, para señalar la transferencia solicitada a los dispositivos de conmutación correspondientes. El direccionamiento es ubicado en el bus BD, por medio del dispositivo IDB, al final del ciclo anterior. El dispositivo EC seleccionado requiere 4 períodos de reloj para decidir la prioridad de la solicitud. En T_5 activa las señales SRDY# y RRQST#. La transición negativa de SRDY# inhibe la salida del dispositivo IDB, por lo que en T_6 el bus BD se encuentra en estado de alta impedancia (tercer estado). La transición negativa de RRQST# señala al dispositivo ADA de la memoria direccionada que una solicitud de acceso está pendiente. La misma transición causa que el direccionamiento sea registrado por el dispositivo MCD asociado a la memoria. Nótese que el direccionamiento se encuentra en el bus RD desde el inicio de T_5.

En T_7, la memoria inicia la transferencia propiamente dicha, y señala lo anterior activando DREM. El flanco positivo de DREM coincide, aproximadamente, con la desactivación de RAS# (véase la figura 53). La máquina de estados del dispositivo EC deja transcurrir 4 períodos de reloj, y activa los buses RD y BD, mediante las señales RDE# y BDE#, respectivamente. El dato leído es transferido al bus BD en T_{12}, y registrado por el dispositivo UCD de la unidad de procesamiento solicitante al ocurrir la transición positiva de BDE#. El elemento de conmutación desactiva SRDY# en T_{12}, para señalar la disponibilidad del dato. En respuesta, el dispositivo ADA genera BRDY# en T_{13}, y desactiva BACC en T_{14}.

Con objeto de simplificar la descripción anterior, no se señaló que el dispositivo EC activa una señal BDE# para las transferencias de los bits 0 al 31, y otra para los bits 32 al 63. Lo mismo se cumple para RDE#.

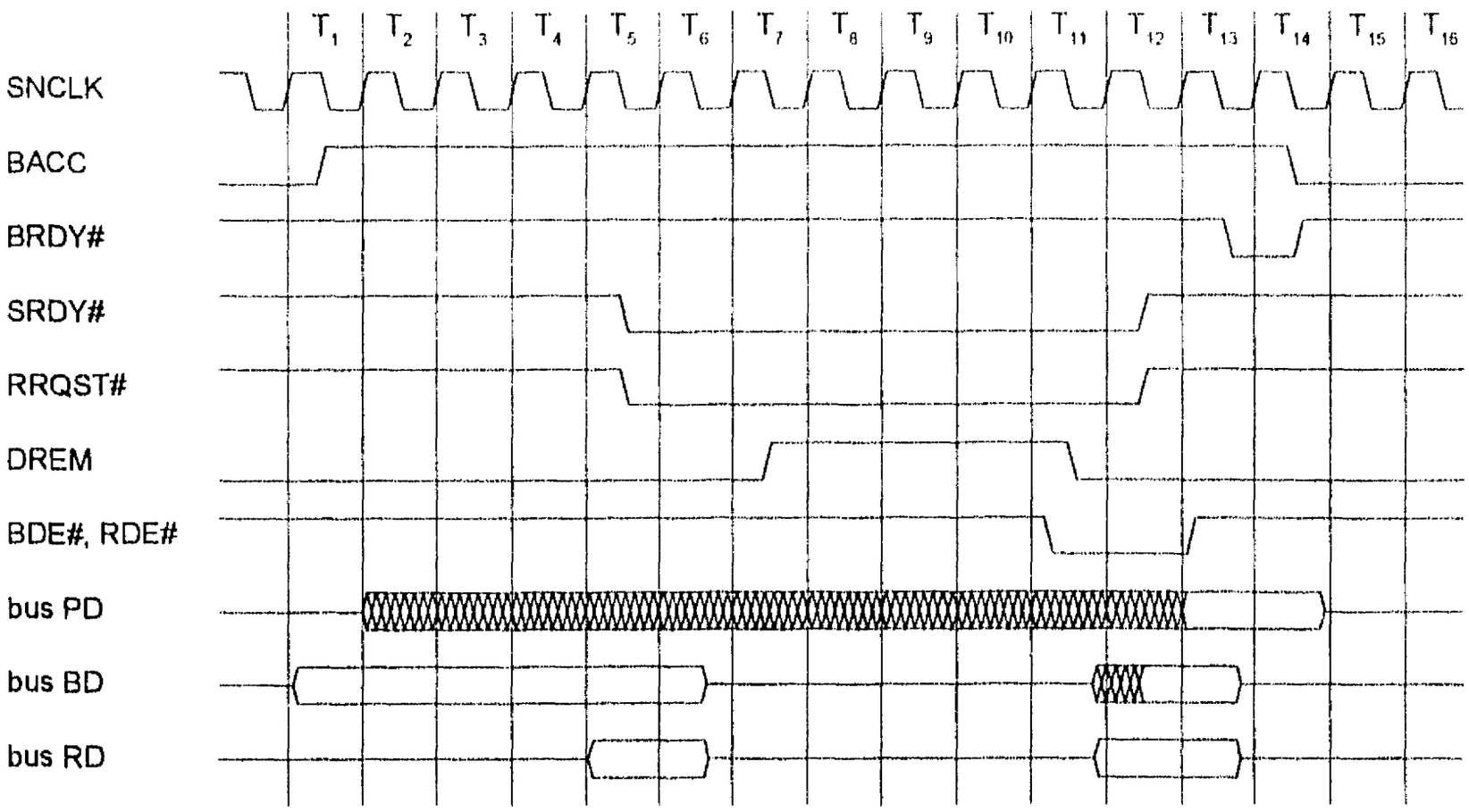


Figura 55: Acceso de lectura a memoria remota.

El diagrama de tiempos correspondiente a un acceso de escritura se muestra en la figura 56. El direccionamiento es transferido de manera similar a la descrita para los accesos de lectura. El dato proveniente del procesador se encuentra en el bus PD a partir de T_2. La señal BDE# generada por el dispositivo de conmutación habilita la salida del dispositivo UCD de la unidad de procesamiento, que ubica, entonces, el dato en el bus BD al concluir T_6. El dispositivo EC lo transfiere al bus RD, con un retardo de propagación de aproximadamente 21 ns (T_7). Con el flanco positivo de RDE# se desactiva el registro de entrada del dispositivo UCD asociado a la memoria, que retiene, así, el dato hasta que es transferido a la memoria. El dato es ubicado en el bus MD al inicio de T_8 (no mostrado), con una antelación de más de dos ciclos de SNCLK, con respecto a la activación de CAS#. La memoria concluye la escritura al finalizar T_{10}, y el dispositivo ADA señala lo anterior desactivando DREM en T_{11}.

A diferencia de los accesos de lectura, el dispositivo ADA de la unidad de procesamiento interpreta el flanco negativo de SRDY# como indicador de conclusión del acceso. En consecuencia, genera BRDY# en T_7, y desactiva BACC en T_8.

El elemento de conmutación desactiva RRQST# en T8. Por diseño, RRQST# no puede ser activado nuevamente (ya sea por el mismo dispositivo, u otro asociado a la misma memoria) sino hasta transcurridos dos ciclos, esto es, en T_{10}. Si la memoria está disponible, el dispositivo ADA reconoce la nueva solicitud en T_{12}, activando DREM.

Aunque no es estrictamente necesario para la presente aplicación, el elemento de conmutación puede realizar transferencias de 64 bits, como dos transferencias secuenciales de 32 bits cada una. Para la lectura, el tiempo de ejecución se incrementa en dos ciclos. Durante la escritura, la segunda

palabra de 32 bits es registrada por los dispositivos UCD al inicio de T_{10}, esto es, la escritura de palabras de 64 bits no requiere de tiempo adicional. La tabla 8 muestra el valor promedio, incluyendo la sincronización entre las unidades de procesamiento y los elementos de conmutación, de los tiempos requeridos para la lectura y escritura. Los valores corresponden a los primeros accesos (a), y a los accesos postergados por conflicto de memoria (b).

Tabla 8: Tiempo de ejecución de accesos remotos
(ciclos de SNCLK).

	a	b
Lectura, hasta 32 bits	14	10
Lectura, 64 bits	16	12
Escritura, hasta 64 bits	8	6

Nótese que la escritura a los módulos remotos es considerablemente más rápida que la lectura. Para el intercambio de información entre procesadores, lo anterior significa que es preferible que el emisor escriba a una memoria remota, y que el receptor tenga acceso a ese dato en memoria local. En estas condiciones, la transferencia, esto es, la escritura y lectura de un dato de 32 bits, requiere 13 ciclos de SNCLK.

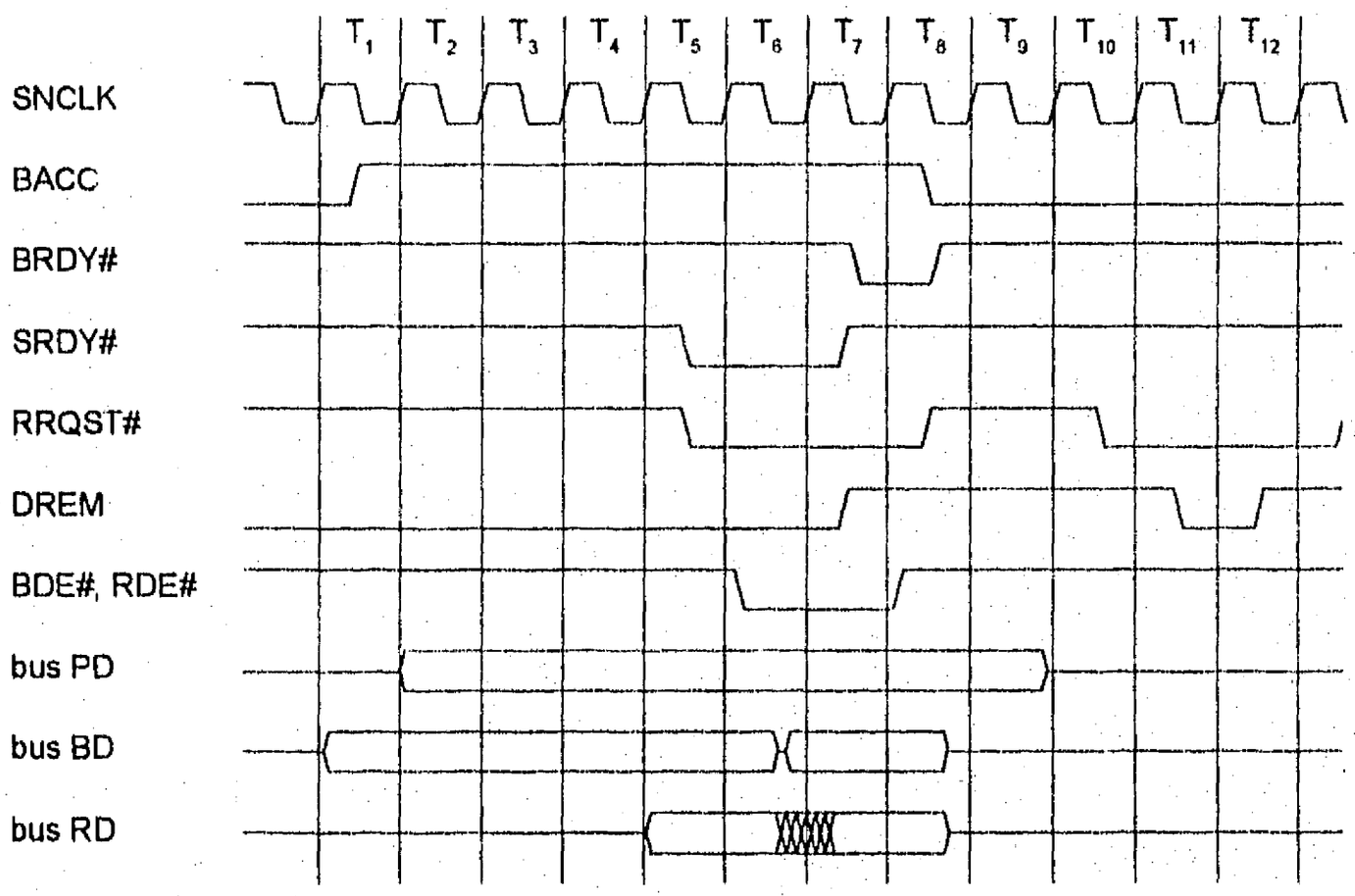


Figura 56: Acceso de escritura a memoria remota.

V.5 Transferencias entre las memorias y el procesador anfitrión.

El dispositivo PCI constituye la interface entre el multiprocesador y el bus del mismo nombre [68], controlado por el procesador anfitrión. El dispositivo efectúa la conversión de protocolos de comunicación y de sincronización entre ambos sistemas, e incluye funciones de monitoreo y control del esquema de multiproceso. La figura 57 muestra las funciones de comunicación entre los buses PCI y BD:

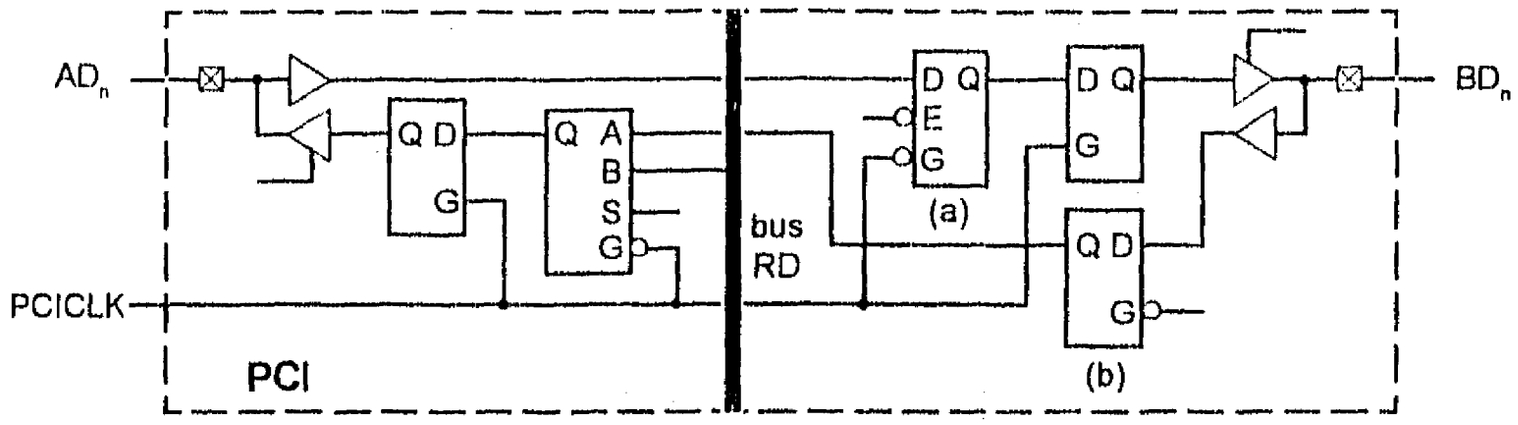


Figura 57: Trayectorias de datos del dispositivo PCI.

Como se observa, la comunicación entre los buses AD (anfitrión) y BD (multiprocesador) se establece por medio de registros tipo maestro-esclavo. La información proveniente del bus BD es, previamente, registrada en (b). El registro (a) posee una entrada de habilitación, esto es, los datos son transferidos a la salida si las entradas G y E están, ambas, activas, y retenidos en caso contrario. El bus RD permite la lectura de una serie de registros de estado internos al dispositivo. La operación es síncrona con relación al anfitrión (PCICLK) y asume que la frecuencia de PCICLK es igual o menor a SNCLK.

La figura 58 ilustra un acceso de escritura generado por el anfitrión. En T_1, activa la señal FRAME# y ubica información de control (tipo de acceso deseado) en las líneas C/BE# del bus PCI. También, ubica la dirección de la transferencia deseada en las líneas AD[0:31]. En T_2 genera las señales de habilitación de byte, BE[0:3]#, así como el dato a escribir. Esta información es retenida en el bus PCI hasta que la transferencia es concluida. El dispositivo PCI decodifica el tipo de acceso deseado durante T_2, y ubica la dirección, incluyendo la información relativa a él, o los bytes afectados, en el bus BD al concluir T_3. En T_4 activa BACC(n) al elemento de conmutación correspondiente a la memoria deseada. La escritura puede destinarse a una, o a todas las memorias, de acuerdo con el modo de operación programado para el dispositivo PCI. Durante este mismo ciclo, ubica el dato en el bus BD. A partir de este momento, la transferencia sigue el mismo curso descrito para los accesos de escritura a memorias remotas. El dispositivo EC señala la conclusión de la transferencia activando SRDY#. Debido a la operación asíncrona entre los dispositivos, la transición negativa de esta señal puede ser detectada por el dispositivo PCI en el mismo ciclo en que es generada (aquí, al concluir T_n), o un ciclo después. Dependiendo del modo de operación, el dispositivo PCI espera una señal SRDY#, o la activación de todas, para concluir el ciclo. En T_{n+1} activa TRDY#, para señalar la conclusión al procesador anfitrión. En el siguiente ciclo desactiva BACC(n), y en T_{n+3} libera al bus BD.

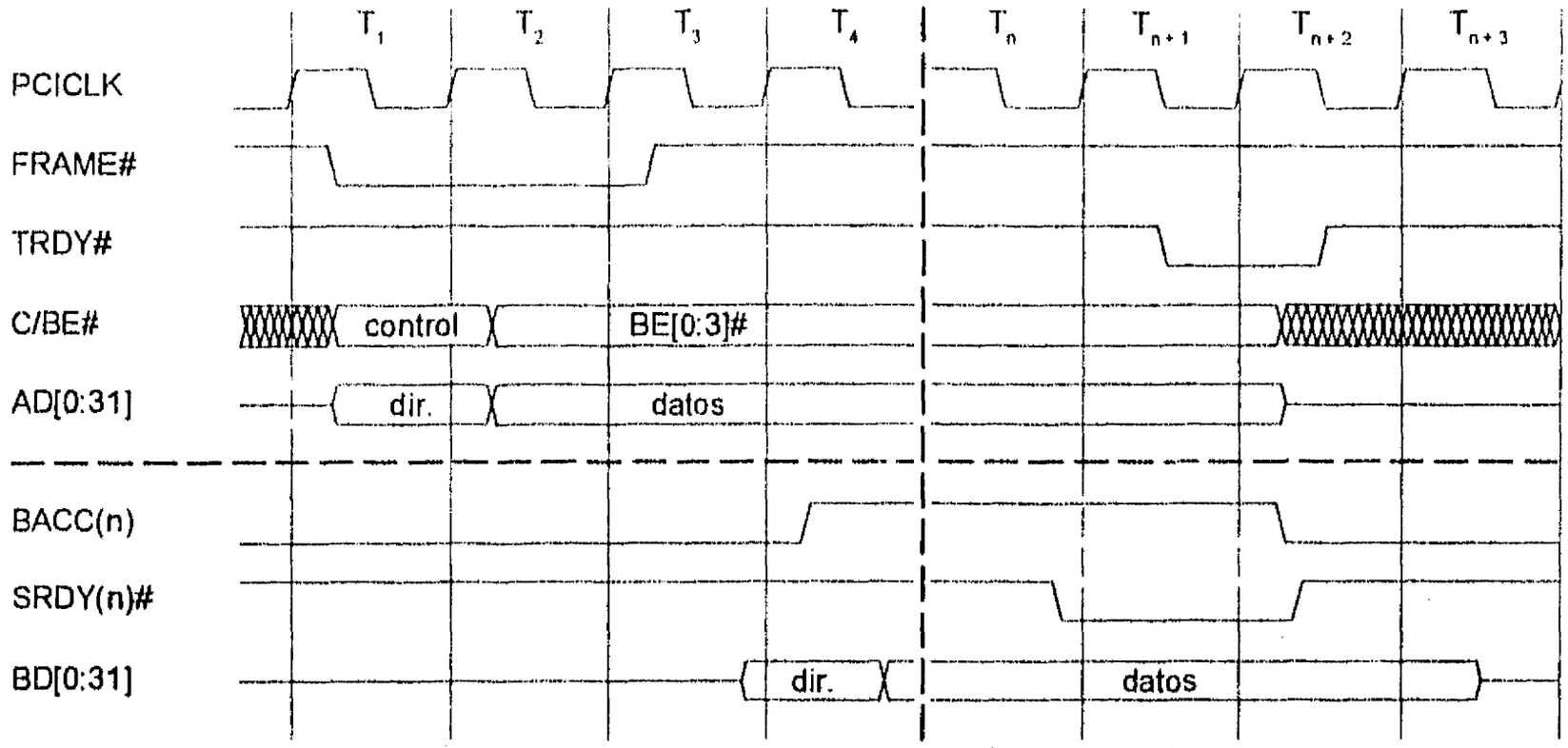


Figura 58: Acceso de escritura desde el procesador anfitrión.

Los accesos de lectura obedecen a un protocolo similar, con las siguientes diferencias (véase la figura 59). El procesador anfitrión libera las líneas AD[0:31] en T2. Posterior a la activación de BACC(n), el acceso es tratado por el dispositivo EC afectado, como fue descrito en el punto anterior. La transición positiva de SRDY(n)# causa que el dato ubicado por el dispositivo EC sea retenido en el registro (b) del dispositivo PCI (figura 57). Los retardos generados por la lógica de detección de SRDY# en el dispositivo garantizan la captura efectiva del dato. En el siguiente ciclo (T_{n+2}), el dispositivo genera TRDY# y transfiere el dato a las líneas AD[0:31]#.

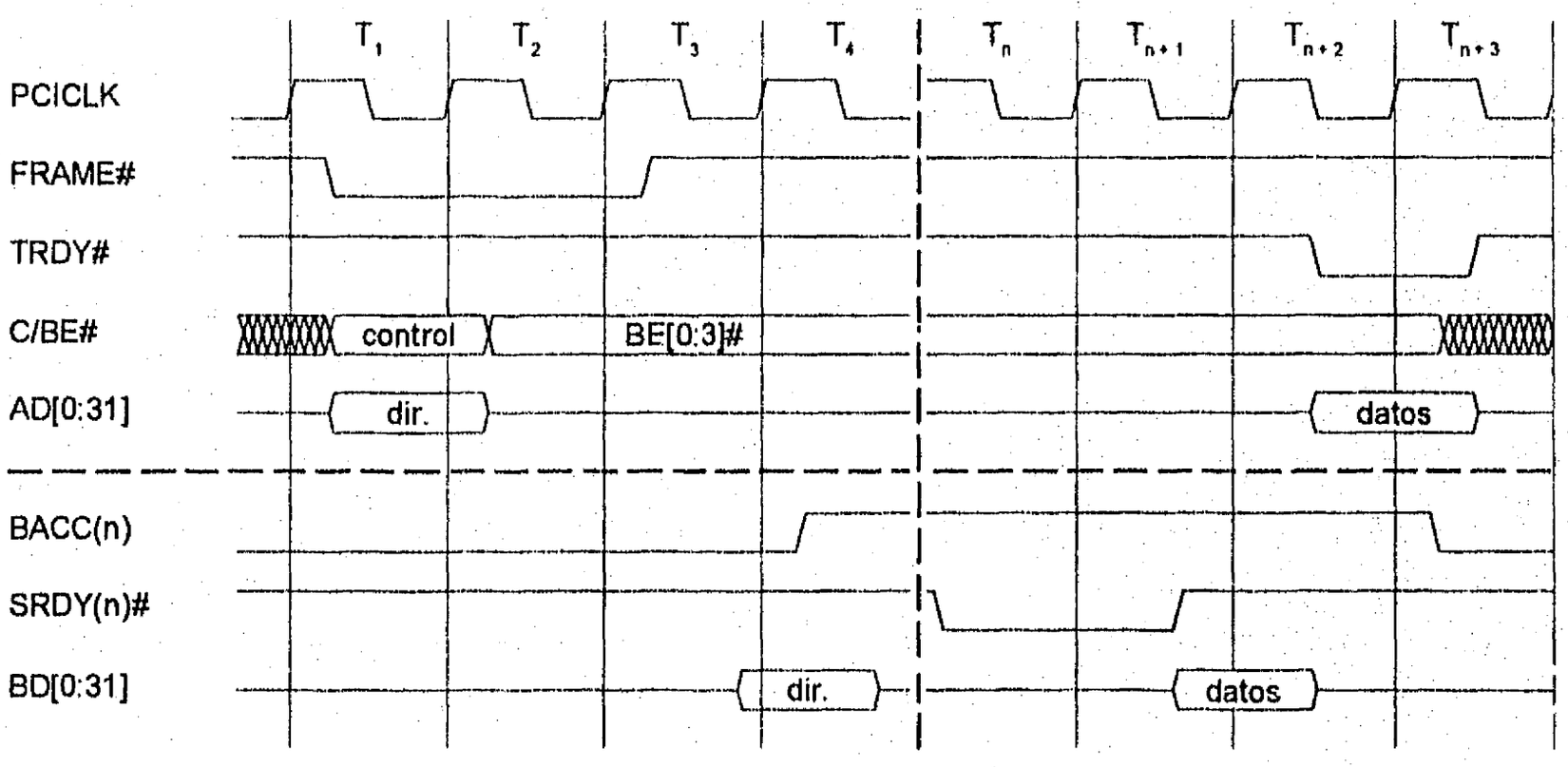


Figura 59: Acceso de lectura desde el procesador anfitrión.

V.6 Lógica de arbitraje y selección del elemento de conmutación.

En cada unidad de procesamiento, el arbitraje entre solicitudes de refresco, de acceso a la memoria del procesador local, o de algún procesador remoto, es efectuado por el dispositivo ADA (figura 49). La priorización es fija, y obedece al orden dado arriba. Sin embargo, el diseño garantiza que las solicitudes secuenciales de los accesos locales no conduzcan a bloqueos indefinidos de solicitudes remotas.

Un diagrama simplificado del esquema de arbitraje se muestra en la figura 60. Sea REFR la solicitud de refresco, internamente sincronizada por el dispositivo ADA. Similarmente, sean PLOC y PREM versiones internamente sincronizadas de una solicitud de acceso a memoria del procesador local, y de la señal RRQST#, activada por un elemento de conmutación (véase la descripción de transferencias remotas dada en el punto V.4). Entonces, el árbitro acepta la solicitud de refresco y genera AREF, si y sólo si, no se encuentra un acceso local (ALOC) o remoto (AREM) en proceso. De manera similar, el árbitro acepta PREM y genera AREM, si y sólo si, no existe un acceso local en proceso, ni solicitud de refresco, o de acceso local, pendientes.

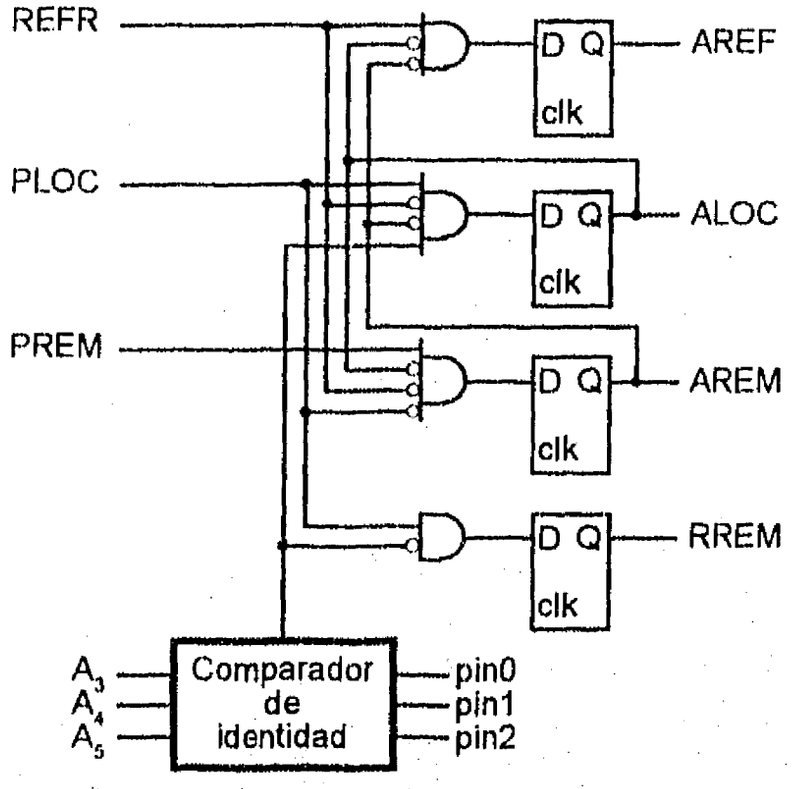


Figura 60: Arbitraje del dispositivo ADA.

Si la solicitud del procesador local corresponde a código o datos exclusivos, el árbitro la acepta y genera ALOC si no se encuentra en proceso un acceso remoto, ni solicitud de refresco pendiente. Para los datos compartidos, el diseño de la máquina emplea la división del espacio de direccionamiento descrita en la página 41. Un comparador de identidad establece si la dirección corresponde a la memoria local, o a una memoria remota, comparando los bits A_3-A_5 de la dirección con el número de identificación, PIN, del procesador. En caso de igualdad, el acceso es a la memoria local, y ALOC es activado sujeto a las condiciones dadas arriba. En caso contrario, se trata de un acceso a memoria remota, y RREM es activado.

Si REFR, o PREM, están activos mientras un acceso local se encuentra en proceso, la operación en modo *pipeline* del procesador es inhibida (véase la página 68). En consecuencia, PLOC no será activado sino, al menos, dos ciclos de reloj posteriores a la conclusión del acceso previo. Con ello se garantiza que la solicitud pendiente de mayor prioridad será atendida al concluir el acceso actual, antes de que una nueva solicitud del procesador pueda ser aceptada.

El arbitraje es efectuado en un ciclo de reloj. La decodificación de los distintos tipos de accesos locales posibles requiere un ciclo adicional. En general, el arbitraje es efectuado mientras una transferencia se encuentra en proceso, y no afecta significativamente la velocidad de ejecución.

El arbitraje entre las diferentes solicitudes remotas es efectuado en los elementos de conmutación, como ya fue descrito en el punto IV.2.1 (figura 43). Por sencillez se utilizó un esquema de prioridad fija, basado en la posición relativa de cada elemento de conmutación dentro de la red. Durante inicialización, cada unidad de procesamiento ubica su número de identificación (PIN) en los buses BD y RD. Sea el número recibido en el bus BD el número de fila, y el recibido en el bus RD el de columna, de un dispositivo EC en particular. Véase la figura 61. Los números de fila, f0 - f2, y de columna, c0 - c2, son almacenados en el dispositivo, y un circuito sumador calcula la prioridad del dispositivo como $pr = (c - f)\big|_{mod8}$. Nótese que el cálculo de la prioridad es efectuado durante inicialización, no durante operación normal del multiprocesador. Por otra parte, ningún dispositivo EC puede tener la prioridad 0 (véase, también, la página 68). Esta prioridad es reservada a los elementos de conmutación asociados a la interface anfitrión.

Un circuito decodificador genera 7 líneas de prioridad pr_1 - pr_7, de las cuales solamente una está activa. Por ejemplo, sea el dispositivo ubicado en la intersección de la fila 7 y columna 1. Su prioridad es 2, por lo que pr_2 está activa. Supóngase que BACC es activado. Un comparador de identidad efectúa la comparación entre las líneas de direccionamiento A_3 - A_5, y el número de columna en que se encuentra ubicado el dispositivo. Si la dirección corresponde a la columna, el dispositivo reconoce la solicitud y un biestable maestro-esclavo es activado en forma síncrona con SNCLK. Las líneas A_3 - A_5 toman el

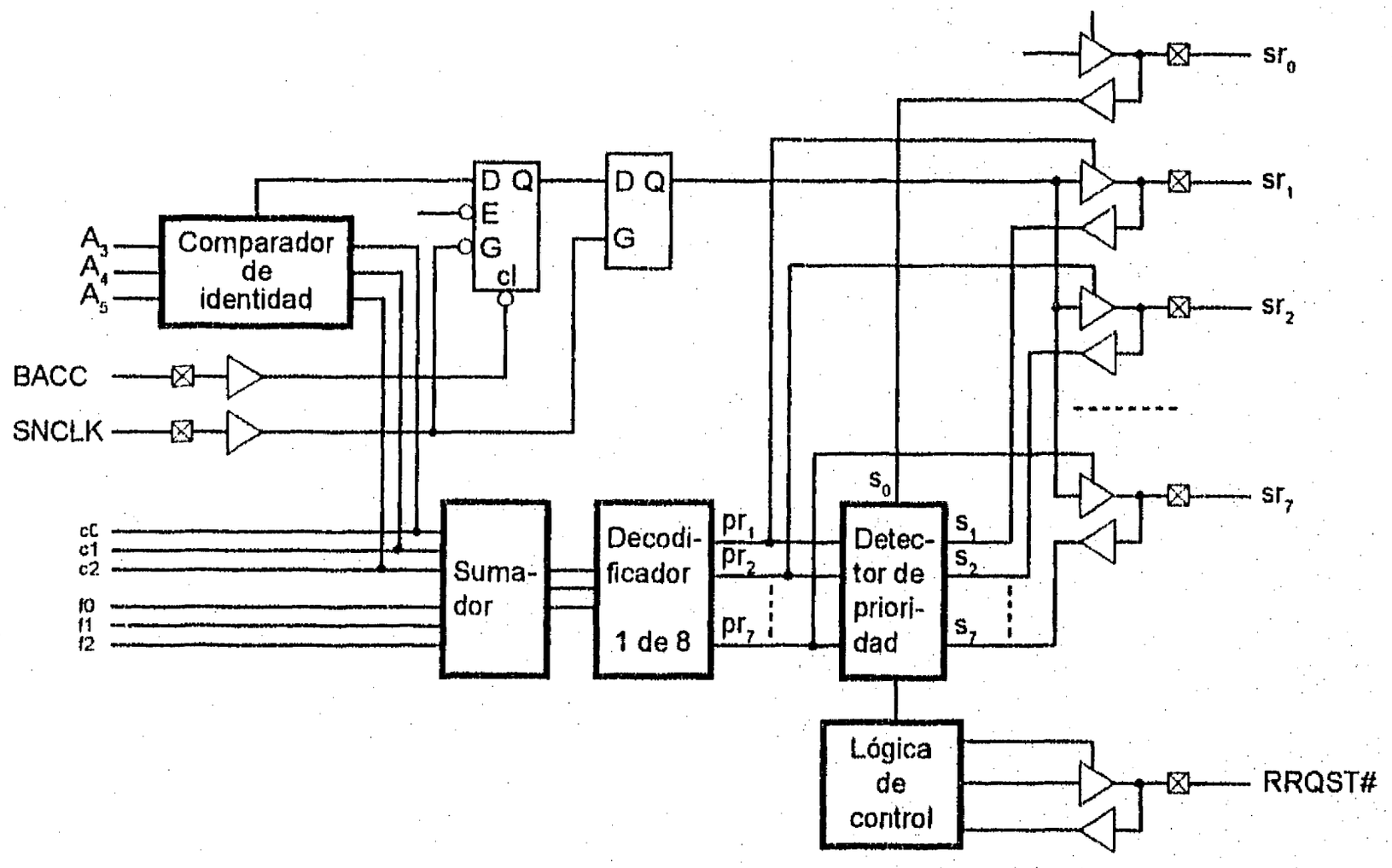


Figura 61: Arbitraje de accesos remotos.

valor de las líneas BD_3 - BD_5 durante el direccionamiento, que es ubicado por la unidad de procesamiento en ese bus un ciclo antes de la activación de BACC. La salida del biestable maestro-esclavo es suministrada a las compuertas reforzadoras (*buffers*) bidireccionales que conforman el bus SR, común a todos los elementos de conmutación de una columna dada. De éstas, solamente sr_2 está habilitada (por pr_2).

Puesto que las prioridades de estos dispositivos son todas diferentes, no puede existir contienda en el bus SR. Las solicitudes reflejadas por las señales sr_0 - sr_7 son suministradas, en conjunto con las líneas de priorización pr_1 - pr_7, al detector de prioridad. Este circuito resuelve la ecuación 64 (página 57), y presenta el resultado a la lógica de control. Si la solicitud es la de mayor prioridad (aquí, si $sr_0 = sr_1 = 0$), y si la línea RRQST# está inactiva (esto es, si no se encuentra otra solicitud en proceso), la lógica de control ubica la dirección en el bus RD y, un ciclo después, activa RRQST#. Estas acciones se efectúan en forma síncrona con SNCLK y, por lo tanto, en forma síncrona con la memoria direccionada.

El retardo generado por el control del bus SR, y por el detector de prioridad, es mayor a un período de SNCLK, operando a 33 MHz. En consecuencia, solicitudes son reconocidas únicamente en ciclos alternados. El biestable maestro posee una entrada de habilitación E, que permite el control en ese sentido. Obviamente, todos los dispositivos EC de una columna dada muestrean sus respectivas solicitudes (las señales BACC) durante la misma ventana de adquisición. El proceso de arbitraje requiere de 3 a 4 ciclos de reloj, dependiendo del instante de activación de BACC, con relación a la ventana de adquisición.

Como ya fue señalado, la prioridad 0 está destinada a la interface anfitrión. Los dispositivos EC poseen una entrada de control (no mostrada), que configura al dispositivo para activar la línea sr_0, bajo control del dispositivo PCI.

En la fecha de inicio del proyecto (1994), los dispositivos FPGA de mayor número de terminales, disponibles en empaque de plástico, contaban con 160 terminales. Por otra parte, con objeto de poder evaluar algunas de las alternativas analizadas en el presente proyecto, tales como el impacto en el rendimiento por reducción del número de líneas de datos, el diseño utiliza el número de líneas requerido por el mayor formato de datos manejado por el procesador para transferencias no almacenables en memoria caché (32 bits). En consecuencia, el diseño incorpora dos elementos de conmutación en cada dispositivo EC, que interconectan dos buses BD con un bus RD. Cada dispositivo incluye dos árbitros, que comparten el bus SR. Lo anterior no fue señalado en la descripción previa, para no complicar innecesariamente el texto. Finalmente, conviene mencionar que en esta fecha (1996) es factible incorporar cuatro elementos de conmutación, que interconectan 2 buses BD con otros tantos buses RD de 32 bits cada uno, en un sólo dispositivo de 208 terminales [65].

V.7 Funciones adicionales de la arquitectura.

El diseño proporciona mecanismos de sincronización y control de procesamiento, tanto entre los procesadores que conforman al multiprocesador, como entre éstos y el procesador anfitrión, con base en banderas programables. El control de estas funciones es efectuado por medio de registros, ubicados en el espacio de direccionamiento de puertos de entrada/salida del procesador. Los comandos de lectura y escritura a estos puertos son generados por el dispositivo ADA, a partir de las señales de control de acceso proporcionadas por el procesador.

V.7.1 Sincronización entre procesadores.

Las unidades de procesamiento están interconectadas, por medio del dispositivo IDB, a través de un bus bidireccional de banderas, pfl_0 - pfl_7. Cada unidad de procesamiento puede escribir el bit cuya posición corresponde a su número de identificación PIN, y leer todos los bits que conforman al bus. La encodificación del bit correspondiente a una unidad de procesamiento en particular es efectuada de manera similar a la descrita para las líneas de solicitud de acceso del esquema de arbitraje de los dispositivos EC. La bandera es controlada por un acceso de escritura a un puerto de control ubicado en el dispositivo IDB. El valor de las banderas puede ser indagado por un acceso de lectura a un registro dedicado del mismo dispositivo. Por otra parte, el hecho de que todas las banderas estén activas, se refleja en el estado de un bit del puerto de control . Este evento genera, también, una interrupción al procesador, enmascarable por este último.

La modificación del estado de una bandera por un procesador distinto al que efectúa la lectura del puerto correspondiente, durante el acceso de lectura, es inhibida por diseño, para evitar incertidumbre en protocolos de sincronización. La escritura y lectura a un puerto de entrada/salida requiere 4 ciclos de SNCLK.

V.7.2 Sincronización y control entre multiprocesador y procesador anfitrión.

Asociado a cada unidad de procesamiento existe una bandera programable, hf_n, que puede ser escrita y modificada tanto por este procesador, como por el procesador anfitrión. Físicamente, el control de esta bandera es efectuada en los dispositivos IDB, por una parte, y PCI, por la otra. Esta bandera permite establecer protocolos de control de proceso sencillos, y de rápida ejecución. Considérese el siguiente ejemplo:

1.- El procesador P_n no inicia ejecución, hasta que hf_n esté activa.
2.- El procesador anfitrión causa el inicio de ejecución, activando hf_n.
3.- El procesador P_n ejecuta el programa. Al terminar, desactiva hf_n.
4.- El procesador anfitrión conoce el estado de ejecución, leyendo hf_n.

La comunicación del estado de la bandera entre dispositivos es asincrona, con base en las señales HFD y HFS#, como se muestra en la figura 62. Normalmente, el anfitrión mantiene HFS# inactiva, lo que habilita la salida del dispositivo IDB (señal E) y permite al dispositivo PCI leer el estado de la bandera. Por otra parte, el estado de HFS# es muestreado en el dispositivo IDB por medio del biestable (b), cuya salida es utilizada para controlar la selección de entradas del biestable (c). El dispositivo IDB puede alterar el valor de la bandera, escribiendo a la entrada B del biestable (c).

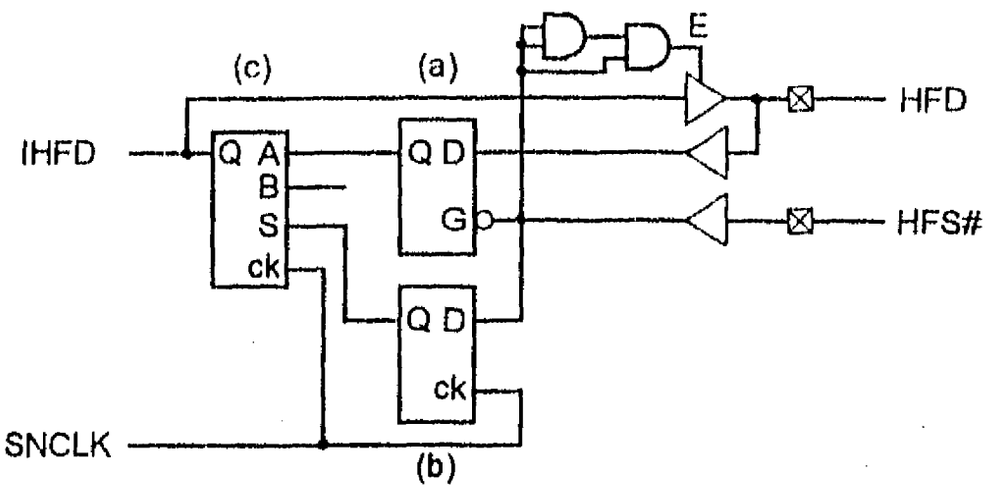


Figura 62: Control de la bandera de anfitrión en el dispositivo IDB.

Por su parte, el dispositivo PCI puede alterar el valor de la bandera, activando HFS# (figura 63) durante dos períodos de PCICLK, y activando la línea HFD durante el segundo período. El esquema es autoexplicativo, pero conviene señalar que la deshabilitación de la salida del dispositivo IDB es, por un retardo de compuerta, más rápida que la habilitación, para evitar contienda en la línea HFD (señal E).

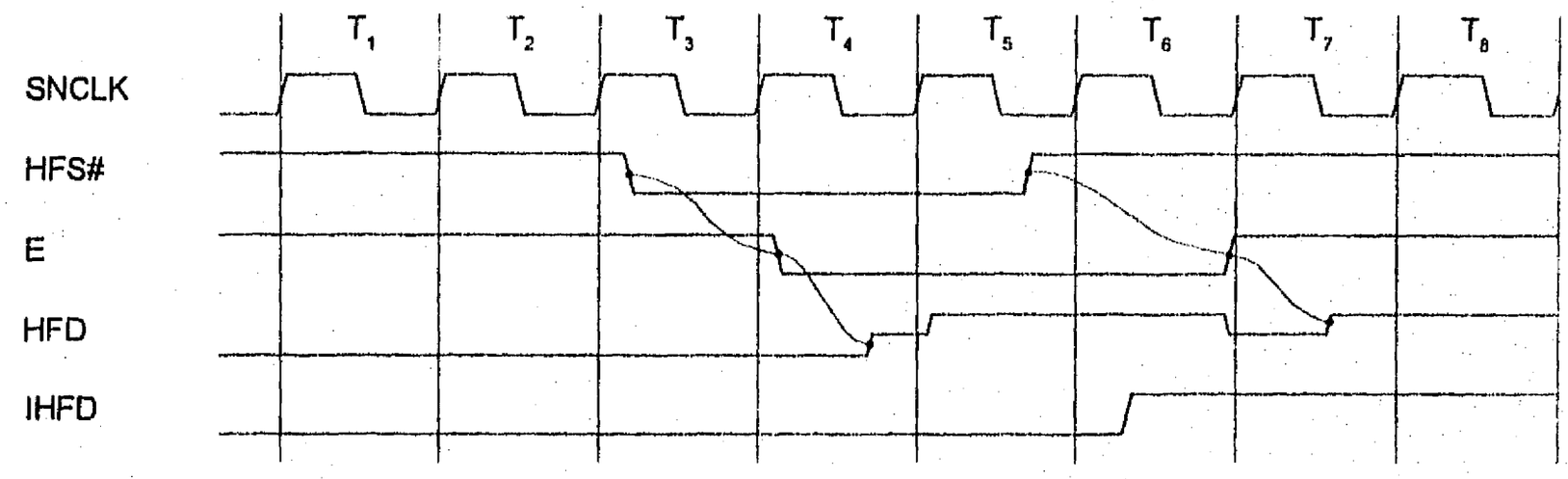


Figura 63: Escritura de la bandera de anfitrión desde el dispositivo PCI.

V.7.3 Funciones de control del dispositivo PCI.

Este dispositivo contiene 4 registros de lectura/escritura de 8 bits, ubicados en el espacio de puertos de entrada/salida del procesador anfitrión. A continuación se describe brevemente la función asociada a cada uno de ellos.

Registro de control (dirección 300H).

Para escritura, la función de cada bit es la siguiente:

bit 0 C/D. Define los accesos a memoria para código (=0) o datos (=1). Afecta el modo de direccionamiento de la interface (véase el direccionamiento de memoria).

bit 1 BCST. Habilita la escritura paralela a los módulos de memoria (=1) (modo *broadcast*). Afecta el modo de direccionamiento de la interface (véase el direccionamiento de memoria).

bit 2 TOEN. Habilita un contador de protección (=1). Dicho contador inicia cuenta regresiva al comenzar una transferencia a/de memoria remota. Si al llegar a cero, el dispositivo no ha recibido la señal SRDY(n)#, el contador genera la señal TRDY# al procesador anfitrión para concluir la transferencia, y activa la bandera TOFLAG (*timeout*).

bit 3 No usado.

bit 4 HOLD. Activa la señal de entrada del mismo nombre a todos los procesadores, deteniendo la ejecución de programa. Como resultado, los procesadores liberan los buses de control, direccionamiento y de datos.

bit 5 No usado.

bit 6 No usado.

bit 7 INIT. Comando de inicialización "tibia" (=1). El dispositivo genera, en respuesta a este bit, un pulso de dos períodos de PCICLK de duración, a todos los procesadores. El bit no es almacenado. Con inicialización "tibia", los procesadores ejecutan un brinco incondicional a la dirección inicial, pero preservan el contenido de las tablas asociados al direccionamiento.

Durante inicialización (PCIRST# activo), el bit 4 (HOLD) es activado. Los demás bits asumen el valor 0. Con ello es posible transferir el código de inicialización a los procesadores desde el anfitrión, antes de iniciar la ejecución. Durante la lectura, cada bit indica lo siguiente:

bit 0 C/D. Igual al valor escrito.

bit 1 BCST (*broadcast*). Igual al valor escrito.

bit 2 TOEN. Igual al valor escrito.

bit 3 TOFLAG. Si activo (=1), indica que no hubo respuesta a un comando de acceso de memoria de uno, o todos, los elementos de conmutación (SE's).

bit 4 HLDA. Si activo (=1), indica que todos los procesadores se encuentran en estado de detención (HOLD).

bit 5 HFLAG. Si activo (=1), indica que la bandera programable de todos los procesadores está activa, esto es, que HFD[0:7] = 1.

bit 6 HFLAG#. Si activo (=1), indica que ninguna de las banderas programables de los procesadores está activa, esto es, que HFD[0:7] = 0.

bit 7 TFLAG. Si activo (=1), indica que uno, o todos los elementos de conmutación (EC) afectados por una transferencia de memoria, han concluido el comando.

Los bits 3 (TOFLAG) y 7 (TFLAG) permiten la verificación, desde el procesador anfitrión, de transferencias a módulos de memoria, tanto para accesos a módulos individuales, como para escritura paralela a todos los módulos.

Registro TO/HLDA (dirección 301H).

Durante la escritura, registra el valor inicial del contador de protección. Este registro debe ser inicializado antes de habilitar la función de protección. Durante la lectura, refleja el estado de las señales HLDA (respuesta al comando de detención) de los procesadores.

Registro HFD (dirección 302H).

Durante la escritura, contiene el valor de cada una de las banderas hf(n) que el anfitrión desea escribir a los procesadores. Durante la lectura, contiene el valor de estas banderas existente en cada procesador. Véase la descripción de las funciones de estas banderas en el punto anterior.

HFS/PRDY (dirección 303H).

Durante la escritura, cada bit activo (=1) causa que el valor del correspondiente bit del registro HFD sea escrito al procesador cuyo número corresponde a la posición del bit activo. Durante la lectura, refleja el estado de las señales SRDY(n)#, esto es, de las respuestas de los dispositivos EC a comandos de acceso a memoria. Las direcciones de los registros HFD y HFS/PRDY permiten la escritura simultánea de ambos registros como una palabra de 16 bits. De esta forma, el procesador anfitrión puede activar cualquier combinación de banderas, con una sola operación de entrada/salida.

V.7.4 Direccionamiento de memoria del multiprocesador.

Desde el punto de vista del procesador anfitrión, los módulos de memoria constituyen un espacio de direccionamiento ubicado a continuación de la memoria propia de 16 MBytes, esto es a partir de la dirección 1000000H. Desde el anfitrión, las memorias son accesibles en los formatos de byte, palabra de 16 bits y palabra de 32 bits. El formato es definido, durante la fase de direccionamiento, por las líneas de control/habilitación de byte C/BE[0:3]#. Los modos de direccionamiento generados por el dispositivo PCI son los siguientes (las direcciones A[0:2] no son utilizadas):

1.- Acceso lineal, código y datos exclusivos (bits de control: C/D = 0, BCST = 0).

Cada módulo de memoria constituye un espacio de direccionamiento lineal, de 8 Mbytes, ubicados en secuencia a partir de la dirección 1000000H. Las direcciones A[23:26] definen al módulo seleccionado, como sigue:

m_0 = A23, m_1 = A24#, m_2 = A26 + A25*A24,

donde m_0, m_1, y m_2 son los bits correspondientes a la representación binaria del número de módulo m, y donde (#), (+), (*) denotan, respectivamente, la operación lógica de negación, la "O" y la "y".

El bus BD posee, durante direccionamiento, los siguientes valores:

BD0 = A3, BD1 = A4, BD2 = A5
BD[6:22] = A[6:22], BD[24:31]= BE[0:7]

Las líneas BD[3:5] no intervienen en este modo de direccionamiento.

2.- Acceso entrelazado, datos compartidos (bits de control: C/D = 1, BCST = 0).

Desde el punto de vista del procesador anfitrión, los módulos de memoria constituyen un espacio de direccionamiento lineal. Sea m, m = 0, 1, ..., 7, el número de identificación de un módulo de memoria. Entonces, el espacio de direccionamiento asociado a cada uno es $(k + m)2^3$, donde k = 0, 1, ... Las direcciones A[3:5] definen el número del módulo afectado. El bus BD posee, durante direccionamiento, los siguientes valores:

BD0 = A23, BD1 = A24#, BD2 = A26 + A25*A24, BD[6:22] = A[6:22], BD[24:31] = BE[0:7].

3.- Acceso de escritura paralelo, tipo *broadcast* (bits de control: C/D = 0, BCST = 1).

Los módulos de memoria constituyen un sólo espacio de direccionamiento lineal, de 8 Mbytes, ubicado a partir de la dirección 1000000H. Las líneas BD[3:5] no intervienen en este modo de direccionamiento. Las restantes poseen, durante direccionamiento, los siguientes valores:

BD0 = A3, BD1 = A4, BD2 = A5, BD[6:22]= A[6:22], BD[24:31] = BE[0:7].

En todos los modos de direccionamiento, la dirección A2 define cuál de las líneas de entre BD[24:27] y BD[28:31] son activadas (habilitación de byte).

V.7.5 Otras funciones del dispositivo PCI.

Adicionalmente a la inicialización "tibia", programable, el dispositivo reproduce una versión negada de la señal de inicialización "fría", RESET, del procesador anfitrión (PCIRST#). Finalmente, genera una señal de refresco periódica, REFRESH#, a los módulos de memoria. Esta señal es derivada de la señal de reloj PCICLK (25 MHz) y posee un período de 15.76 μs ($40x10^{-9}$ x 394).

VI.- UNA MEMORIA CACHE PARA DATOS COMPARTIDOS

Históricamente, la memoria caché es una memoria asociativa ubicada entre el procesador central y la memoria principal de un sistema de cómputo, que tiene por objetivo acelerar el acceso a información frecuentemente requerida por el procesador [60]. En principio, una solicitud de acceso generada por el procesador puede ser dirigida simultáneamente a la memoria caché y la memoria principal, o solamente a la memoria caché. En el segundo caso, únicamente si la dirección correspondiente no se encuentra en la memoria caché (*cache miss*), la solicitud es transmitida a la memoria principal. Por razones de índole comercial, el desarrollo de controladores y protocolos de comunicación de memorias caché se ha orientado hacia máquinas de uno, o pocos procesadores, basadas en la arquitectura de bus común. Para estos sistemas, un segundo objetivo de gran importancia, es la reducción de tráfico en el bus del sistema. Por lo tanto, la memoria caché es conectada en forma serial con el procesador y, en miroprocesadores modernos, se encuentra frecuentemente integrada a este último [67]. Dada la relativamente baja capacidad de almacenamiento de estas memorias, los sistemas de alto rendimiento utilizan memorias caché jerárquicamente escalonadas, correspondiendo el primer nivel a la memoria caché incorporada al procesador, y el segundo a una memoria caché externa al dispositivo, de mayor capacidad. En el caso particular de procesadores de la empresa INTEL, el controlador de memoria caché asegura el cumplimiento del principio de inclusión, esto es, el espacio de direccionamiento de la memoria caché primaria está, en todo momento, totalmente comprendido en el espacio de direccionamiento de la memoria caché secundaria. El tráfico en el bus de sistema es reducido a un mínimo, en virtud de que la gran mayoría de los accesos generados por el procesador puede ser satisfecha por las memorias caché.

Una desventaja de esta organización jerárquica consiste en que las solicitudes que deben ser atendidas por la memoria principal, sufren el retardo asociado a la indagación de contenido de las memorias caché. Por otra parte, aunque los accesos a la memoria caché desde dispositivos externos al procesador son, en principio, posibles, éstos obedecen a protocolos de comunicación complejos, y pueden interrumpir la operación del procesador.

En los esquemas de multiprocesamiento, y para datos compartidos, la memoria caché debe ser una extensión de la memoria compartida, y, obviamente, de mayor velocidad de acceso que esta última. Los protocolos que consideran las memorias caché como memorias privadas de los procesadores y, por lo tanto, requieren accesos de indagación para ubicar el dato mas recientemente modificado, claramente contradicen el objetivo de una "memoria compartida". Tal vez por esta razón, la máquina T3D de Cray Research, Inc., un sistema de paralelismo masivo basado en microprocesadores, no utiliza memorias caché secundarias [61].

Idealmente, la memoria caché para datos compartidos debe ser accesible por todos los procesadores, y contener, en todo momento, el dato más recientemente modificado. Para arquitecturas basadas en redes de conmutación, estos requisitos sitúan a la memoria caché en paralelo con la

memoria compartida, esto es, en paralelo con los módulos de memoria, cuando la memoria compartida está segmentada. Esta solución obvia, sin embargo, ofrece pocas ventajas en la práctica. Considérese el ejemplo de diseño presentado en el capítulo anterior, en particular, los valores de tiempos de acceso dados en la tabla 8. La lectura de un dato de hasta 32 bits requiere 14 periodos de reloj, de los cuales 5 corresponden al acceso de lectura propiamente dicho. Asumiendo que una memoria caché proporciona el dato deseado (en caso de disponer de él) en 2 ciclos, el acceso aún requiere 11 ciclos, esto es, 78.6% del tiempo requerido sin memoria caché. Evidentemente, entonces, una memoria caché efectiva para datos compartidos debe eliminar los retardos asociados al arbitraje de acceso, y de tránsito por la red.

El uso de memoria local en los elementos de conmutación de la red de interconexión, como un medio para incrementar la velocidad de transferencia, especialmente en redes escalonadas, ya se ha señalado. Sin embargo, la literatura no parece sugerir la posibilidad de ubicar, en los elementos de conmutación, memoria local que emule las funciones de memorias caché. Considérese la figura 64, que corresponde al diagrama de bloques simplificado del diseño descrito en el capítulo anterior (figura 48), con la inclusión de memorias caché para datos compartidos tanto en los elementos de conmutación (EC), como en los módulos de memoria. La figura muestra un acceso de escritura generado por el procesador p_0, al módulo de memoria m_1. Los buses involucrados en la transferencia están marcados en negro, y las memorias caché en forma sombreada. Nótese que el dato es escrito a todas las memorias caché asociadas a la memoria direccionada. Por lo tanto, el contenido de las memorias caché de una columna dada es, en todo momento, el mismo.

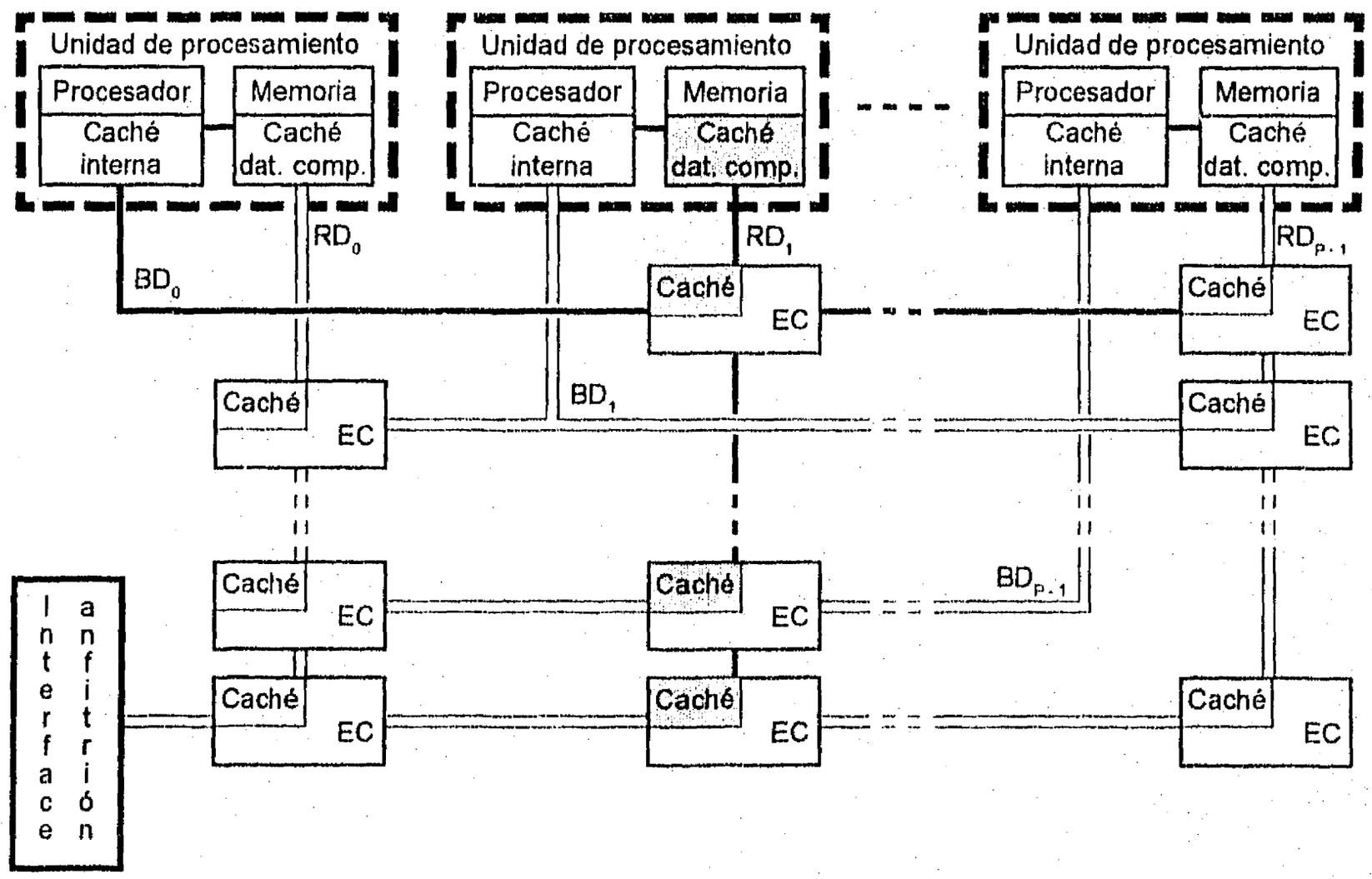


Figura 64: La arquitectura *crossbar* con memoria caché distribuida.

VI.1 Características del esquema de memorias caché propuesto.

El comportamiento del esquema es diferente para los accesos de escritura, y de lectura. Sea BD_j el bus asociado al procesador p_j. También, sea RD_k el bus correspondiente a la memoria M_k, y C_{jk} la memoria caché ubicada en el elemento de conmutación correspondiente a la intersección de los buses BD_j y RD_k. Finalmente, sea C_{kk} la memoria caché ubicada en el módulo M_k. Para los accesos de escritura, se tiene:

1.- El acceso desde el procesador p_j a C_{jj} no involucra arbitraje, ni genera contienda, en virtud de que el único bus involucrado es BD_j.

2.- La escritura a las restantes memorias caché, C_{jk}, $j \neq k$, requiere de arbitraje. Una vez otorgado el acceso, puede efectuarse en forma simultánea a todas las memorias.

3.- La escritura a la memoria M_k puede efectuarse, en principio, como una escritura transparente, o de actualización (véase, también, el punto IV.4). En ambos casos, el dato proviene de la memoria caché C_{jj}.

Desde el punto de vista del procesador, el tiempo de acceso a una memoria remota es el correspondiente a una memoria caché. Lo anterior se cumple aun para accesos simultáneos a un mismo módulo. Los accesos de lectura a este módulo, sin embargo, podrán requerir de un período de espera, hasta que se haya efectuado la escritura de las réplicas del dato, esto es, hasta que todas las memorias C_{jk}, para una j dada, hayan sido actualizadas. Este requerimiento, que garantiza la consistencia de memoria en todo momento, se presenta solamente si la dirección de lectura corresponde a la de escritura previa.

Los accesos de lectura, en caso de que el dato solicitado esté presente en memoria caché (*cache hit*), no enfrentan restricción alguna. Todos los procesadores pueden acceder a la misma memoria, y aun a la misma dirección física. El dato solicitado puede no encontrarse en la memoria caché, ya sea, porque una escritura de réplicas esté pendiente, o porque la dirección solicitada no forme parte del mapa de memoria de la memoria caché. En el primer caso, la lectura es postergada hasta que las memorias caché estén actualizadas. En el segundo caso, el acceso es dirigido al módulo de memoria M_k.

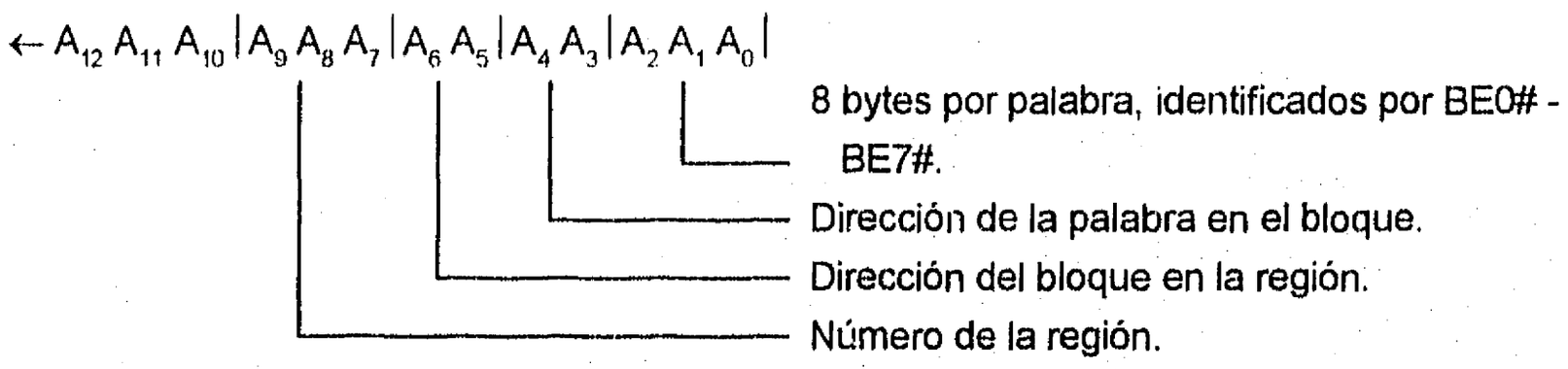
Resumiendo, el esquema propuesto satisface los requisitos de lectura y escritura concurrentes. El tiempo de acceso es el correspondiente a una memoria caché, y no involucra arbitraje. El tiempo de tránsito es reducido a los retardos asociados a un sólo bus (BD_j). Finalmente, escrituras simultáneas a una misma dirección física son posibles, registrándose el valor escrito al elemento de conmutación de mayor prioridad, de acuerdo con un criterio preestablecido. Esto es, el esquema se aproxima a una PRAM ideal, del tipo CRCW en sus variantes de *ganador arbitrario*, o de *modo prioritario* (véase el capítulo II).

VI.2 Complejidad de la memoria caché.

No es objetivo del presente trabajo profundizar en el estudio de las memorias caché. Se presenta, a continuación, un breve análisis de la complejidad de memorias caché del tipo asociativa por conjuntos (*set associative cache memory*), como ejemplo representativo de sistemas comercialmente disponibles [68]. El objetivo de esta presentación es evaluar la factibilidad de integrar físicamente memorias caché a elementos de conmutación.

Una memoria caché asociativa por conjuntos puede definirse como sigue. Sea {B} un bloque de direcciones consecutivas de la memoria principal, de 2^b elementos. El espacio de direccionamiento de la memoria caché es dividido en S regiones {R}, donde $S = 2^r$, $r > 0$, y cada región consta de 2^{r+b} direcciones. Por otra parte, el espacio de direccionamiento de la memoria principal es dividido en N bloques, $N = 2^L$. Entonces, un bloque {B(j)} de la memoria principal es asignado a la región {R(k)} de la memoria caché, si $j = k \mid_{\text{mod } S}$

Como ejemplo, supóngase que un bloque consta de 4 palabras de 64 bits, y que el espacio de direccionamiento de la memoria caché es dividido en 8 regiones, con capacidad de 4 bloques cada una. Esto es, la memoria caché cuenta con una capacidad de 1 Kbyte. Entonces, en términos de la memoria caché, la dirección de un dato en memoria principal asume el siguiente significado:



Supóngase que la memoria caché es organizada por bloques, esto es, para el ejemplo anterior, que cada localidad de la memoria caché almacena 256 bits (una *línea* en la literatura de INTEL). Asociado a cada bloque, la memoria caché almacena, también, las direcciones A_{10}, A_{11}, A_{12}, ... Por congruencia con el diseño presentado en el capítulo anterior, supóngase que se requiere poder acceder a formatos de datos variables, de 1, 2 y 4 bytes. Entonces, la presencia en memoria caché de un byte en particular no implica que la palabra de la cual forma parte se encuentre en su totalidad. Por lo tanto, se propone aquí almacenar, para cada byte de un bloque dado, un bit de "presencia", activo cuando la localidad correspondiente se encuentre ocupada, e inactivo en caso contrario. En estas condiciones, el protocolo de acceso a memoria caché toma la siguiente forma:

1.- Los bits A_7, A_8 y A_9 de la dirección suministrada identifican la región en la que se encuentra (posiblemente) la dirección deseada.
2.- Un número de comparadores de identidad igual al número de bloques por región (aquí, 4) comparan los bits A_{10}, A_{11}, A_{12}, ... de la dirección suministrada, con los mismos bits de las direcciones almacenadas.

Si el comparador correspondiente a los bits A_5 y A_6 de la dirección suministrada reporta coincidencia entre las direcciones suministrada y almacenada, el bloque está presente. En ese caso, y para los accesos de lectura:

3.- Un comparador verifica que las señales de habilitación de byte activas, suministradas como parte del direccionamiento, coincidan todas y cada una con los bits de presencia correspondientes a estos bytes. Las direcciones A_3 y A_4 suministradas identifican a la palabra deseada dentro del bloque.
4.- Si el, o los bytes solicitados se encuentran presentes, el acceso es concluido (*cache hit*), y la solicitud de lectura emitida al módulo de memoria asociado a la memoria caché es abortada. En caso contrario, o si el bloque deseado no está presente (*cache miss*), el acceso al módulo de memoria procede normalmente.

Nótese que un acceso de lectura puede coincidir con una solicitud de escritura de réplicas, generada por otra memoria caché conectada al mismo bus RD. En este caso, la escritura de réplicas debe ser ejecutada primero. Para los accesos de escritura:

5.- Si el bloque se encuentra presente, el dato es escrito a él, o los bytes especificados por las líneas de habilitación de bytes activas del direccionamiento, correspondientes a la palabra identificada por A_3 y A_4 del mismo. Por otra parte, los bits de presencia de los bytes modificados por la escritura son activados. Si el bloque no se encuentra presente, un bloque existente debe ser liberado, para poder ser ocupado por el nuevo dato. La escritura de este dato activa los bits de presencia de los bytes afectados, y desactiva los restantes de ese bloque.

Existen diversos criterios para liberar un bloque de la memoria caché y, en consecuencia, poder asignar un nuevo bloque a la localidad correspondiente. El más sencillo es, desde luego, el reemplazo del bloque más antiguo, esto es, conformar a los bloques de una región dada como una lista circular. En todo caso, cada bloque requiere de un bit de control, para especificar si el contenido del bloque ya ha sido escrito al módulo de memoria correspondiente y, por lo tanto, puede ser reemplazado. En caso contrario, una escritura de actualización debe ser efectuada, antes de que el nuevo dato pueda ser aceptado.

En general, una memoria caché almacena parte del direccionamiento, así como bits de control, adicionales al dato. Para el protocolo definido arriba, cada bloque de 32 bytes de datos requiere de 32 bits de presencia de byte, y un bit adicional para señalar la condición de reemplazo. El número de bits de direccionamiento almacenados depende de la cantidad de memoria total "asociada" a la caché. Supóngase que esta cantidad es 32 Mbytes. Entonces, los bits de direccionamiento A_{10} - A_{24} son almacenados, adicionalmente a los bits de control, para cada bloque. En este caso, entonces, un total de 6 bytes adicionales a 32 bytes de datos requieren ser almacenados (para formatos de datos más restrictivos, esta cantidad sería, obviamente, menor). Es interesante comparar la cantidad de almacenamiento requerida por una memoria caché, con la correspondiente cantidad para elementos

de conmutación empleados en redes escalonadas. De la tabla 5, punto IV.1.2, se desprende que una memoria caché de 1 Kbyte requiere poco más (1,216 bytes) que un elemento de conmutación para una red de 8 procesadores (1,008) y capacidad de transferencia de líneas de 256 bits.

El tiempo de acceso a la memoria caché está limitado por los retardos de propagación de la lógica encargada de la detección de acceso exitoso (*page hit*). Es fácil ver que los pasos 2 y 3 del protocolo antes descrito pueden efectuarse en forma concurrente, si el número de comparadores utilizado para el paso 3 es igual al número de bloques por región. Un análisis rígido demuestra que la condición de acceso exitoso puede ser determinada en un tiempo no mayor a 6 retardos unitarios de compuerta, si la magnitud del abanico de entrada (*fan-in*) de los operadores lógicos no constituye una limitante. En este sentido, la mayor demanda parece corresponder al comparador de identidad definido en el punto 2 del protocolo de acceso. Para el ejemplo considerado, el máximo número de entradas requerido a una compuerta es 15.

En síntesis, aunque el esquema no es realizable en términos de la tecnología ofrecida por dispositivos tipo FPGA, dado lo limitado del número de biestables ofrecido por empaque - máximo 1153 [65] - parece eminentemente factible en términos de tecnología C-MOS para dispositivos de aplicación específica.

VI.3 Impacto en rendimiento computacional.

Como una estimación del incremento en velocidad de ejecución posible con el esquema propuesto, supóngase que los algoritmos analizados en los puntos III.5 y III.6 son ejecutados en dos máquinas basadas en el diseño descrito en el capítulo V, de las cuales una cuenta con memorias caché para los datos compartidos. Para los tiempos de acceso a memoria se usarán los valores de la tabla 8 de ese capítulo (página 85). Por otra parte, supóngase que el tiempo medio de acceso a la memoria caché es de 3τ, donde τ es un período de SNCLK.

Transformada rápida de Fourier.

Un análisis del tiempo de ejecución de la operación mariposa por el procesador Pentium, con datos expresados en notación de punto flotante, representados en 32 bits, resulta en aproximadamente 24 períodos de CPUCLK, esto es, en 12τ. Por otra parte, para un arreglo de N datos (complejos), la ejecución del algoritmo involucra:

2N/p	lecturas de los datos de entrada,
(N/p)(logN - 1)	lecturas de los resultados parciales,
(N/2p)logN	operaciones mariposa y
(N/p)(logN + 1)	escrituras de los resultados parciales y finales.

Cada lectura de un dato de entrada se ejecuta en 14τ. La lectura de resultados parciales, ubicados en memorias locales a cada procesador, se ejecuta en 5τ, y la escritura de resultados a los módulos remotos requiere 8τ. Sea $N = 1024$ y $p = 8$. Entonces, el algoritmo se ejecuta en $29{,}568\tau$.

Para la máquina equipada con memorias caché, los tiempos de ejecución de la lectura de los datos de entrada y de la operación mariposa, permanecen iguales. La lectura y escritura de los resultados parciales se ejecutan en 3τ. Así, el tiempo de ejecución se reduce a $20{,}224\tau$, un 68.4%, aproximadamente, del tiempo requerido sin memorias caché. Por otra parte, la capacidad de almacenamiento requerida de estas memorias es, en este caso, 128 palabras de 32 bits, o sea, 512 bytes.

Ordenamiento.

Considérese la ecuación (52), para $N = 2^{16}$ y $p = 8$. Supóngase que los datos a ordenar son cantidad expresadas en 32 bits. El tiempo de ejecución está dado por:

$(N/2p)[\log^2 N + \log N]$ lecturas de datos,
$(N/4p)[\log^2 N + \log N]$ comparaciones de datos, y
$(N/4p)[\log^2 N + \log N]$ escrituras de datos.

Del total de las lecturas efectuadas, N/p son ejecutadas en 14τ, y las restantes se ejecutan en 5τ. Las comparaciones se ejecutan en la unidad de tiempo, y las escrituras requieren 8τ cada una. Substituyendo estos valores, y para N y p dados arriba, el tiempo de ejecución es $(10.658)10^6\tau$. Si la máquina contara con memorias caché, el tiempo se reduciría a $(5.661)10^6\tau$, esto es al 53.1%. La capacidad de almacenamiento de las memorias caché tendría que ser de 4,096 palabras de 32 bits, esto es, de 16 Kbytes. Para capacidades menores, el algoritmo generaría escrituras de actualización a los módulos de memoria, lo que incrementaría el tiempo de ejecución.

VII. CONCLUSIONES Y LINEAS DE INVESTIGACION FUTURAS

En los capítulos anteriores se demostró que, para multiprocesadores de hasta 64 procesadores, la red de interconexión conocida como *crossbar* es una elección competitiva con otro tipo de redes, no sólo en lo que a velocidad de transferencia de datos se refiere, sino también en términos del número de dispositivos requerido para su implementación, y costo. Para ello, se aportaron una serie de elementos, entre los que destacan:

1.- Una modificación a la arquitectura, consistente en asociar la memoria para datos compartidos a los procesadores, lo que permite, para la transferencia de datos entre procesadores, eliminar una de las dos travesías por la red requeridas (ya sea, el acceso de escritura, o el de lectura).

2.- Un esquema de arbitraje distribuido entre los elementos de conmutación, basado en el principio de redundancia, que compite en velocidad con los esquemas centralizados y elimina la necesidad de componentes adicionales. Por otra parte, simplifica, en gran medida, el enrutamiento de las líneas físicas asociadas al proceso de arbitraje.

3.- Un análisis cuidadoso de la complejidad de los elementos de conmutación, que demuestra que una reducción del ancho efectivo de los buses de interconexión (número de bits transmitidos en forma paralela) conduce a un número de dispositivos competitivo al correspondiente de redes escalonadas, conservando velocidades de transmisión iguales, o superiores, al de estas últimas.

Estas aportaciones, de naturaleza eminentemente ingenieril, son consecuencia, a su vez, de un modelo del comportamiento de multiprocesadores que considera los accesos simultáneos, o concentrados en un lapso pequeño dentro de un período de ejecución determinado. Este modelo parece más realista para aplicaciones de granularidad fina, en las que el tiempo esperado de conclusión de un cómputo es de mayor significado, que el rendimiento medio de la máquina para tareas estadísticamente independientes (*throughput* de la máquina). Como fue señalado, este modelo proporciona cotas máximas para el tiempo requerido para el intercambio de datos entre procesadores, por considerar implícitamente que la ejecución de un programa es sincronizada en todos los procesadores después de cada solicitud de acceso a memoria. Por lo tanto, una línea de investigación aún abierta consiste en el desarrollo de modelos más precisos, que permitan combinar una cota inferior, determinista, de tiempos de ejecución, con componentes aleatorias derivadas de retardos no determinísticamente predecibles.

Al momento de escribir estas líneas, la construcción de un prototipo de 8 procesadores, basado en la arquitectura descrita (figura 48), ha sido concluida. La operación correcta de los elementos de conmutación y, en particular, del esquema de arbitraje propuesto (figura 43) ha sido experimentalmente

verificada. Los tiempos requeridos para el intercambio de información entre procesadores corresponden a lo señalado en el capítulo V, por lo que, de manera preliminar, los objetivos planteados pueden considerarse como satisfechos. Una evaluación exhaustiva de la efectividad de la arquitectura, sin embargo, requiere todavía una considerable cantidad de trabajo en materia de programación, que será tarea para el futuro.

Una propiedad importante de la arquitectura propuesta consiste en la posibilidad de proporcionar memorias caché para datos compartidos, desligadas de los procesadores individuales, incorporadas a los elementos de conmutación. La complejidad resultante de estos dispositivos no es significativamente mayor a la correspondiente a los elementos de conmutación de redes escalonadas, que incorporan memoria propia para el manejo de colas de datos. El resultado de incorporar memorias caché en la forma descrita, es un multiprocesador que se aproxima al modelo ideal de máquina paralela de acceso aleatorio (PRAM), con las siguientes consecuencias:

1.- Tiempos de intercomunicación de datos más cortos, en los que retardos asociados al arbitraje de acceso, a los elementos de conmutación y a los buses de interconexión desempeñan un papel mucho menos importante.

2.- La realización física de la gráfica bipartita completa, que reduce la sensibilidad de un algoritmo a la arquitectura específica de una máquina en particular. Esto es, el factor determinante para la ejecución eficiente de un algoritmo paralelo ya no consiste en la geometría de la intercomunicación de datos requerida, sino solamente en la magnitud del problema, y de su partición adecuada para no exceder límites de capacidad de almacenamiento.

El problema de las memorias caché en los multiprocesadores es un tema de investigación de importancia en la actualidad, y no parece existir una solución aplicable a todos los tipos de redes de interconexión. En este sentido, la red *crossbar* posee la propiedad de que cualquier elemento de conmutación interconecta uno, y solamente un procesador con un módulo de memoria, por lo que el esquema de réplicas de datos aquí propuesto resulta mucho más sencillo de aplicar que en otro tipo de redes.

Un segundo campo de investigación que, tal vez y derivado del presente trabajo, ofrece mayores oportunidades para el futuro, es el de máquinas de paralelismo masivo de interconexión fija, que utilicen multiprocesadores como elementos de conmutación (nodos computacionales). Como antecedente, existe al menos una máquina comercialmente disponible (CRAY T3D, [61]), que utiliza dos procesadores en cada nodo de una malla cerrada en tres dimensiones (*toroide*, en la literatura de CRAY), además de proyectos de investigación ya antes citados. El multiprocesador con red tipo *crossbar* es trivialmente expandible para proporcionar las interfaces de interconexión requeridas. En la figura 65 (a) se muestra una solución basada en un procesador de comunicaciones dedicado, que establece la interface entre el multiprocesador, y tres buses externos X, Y, Z. En la figura 65 (b) se muestra la interconexión de multiprocesadores en forma de malla, de acuerdo con [61].

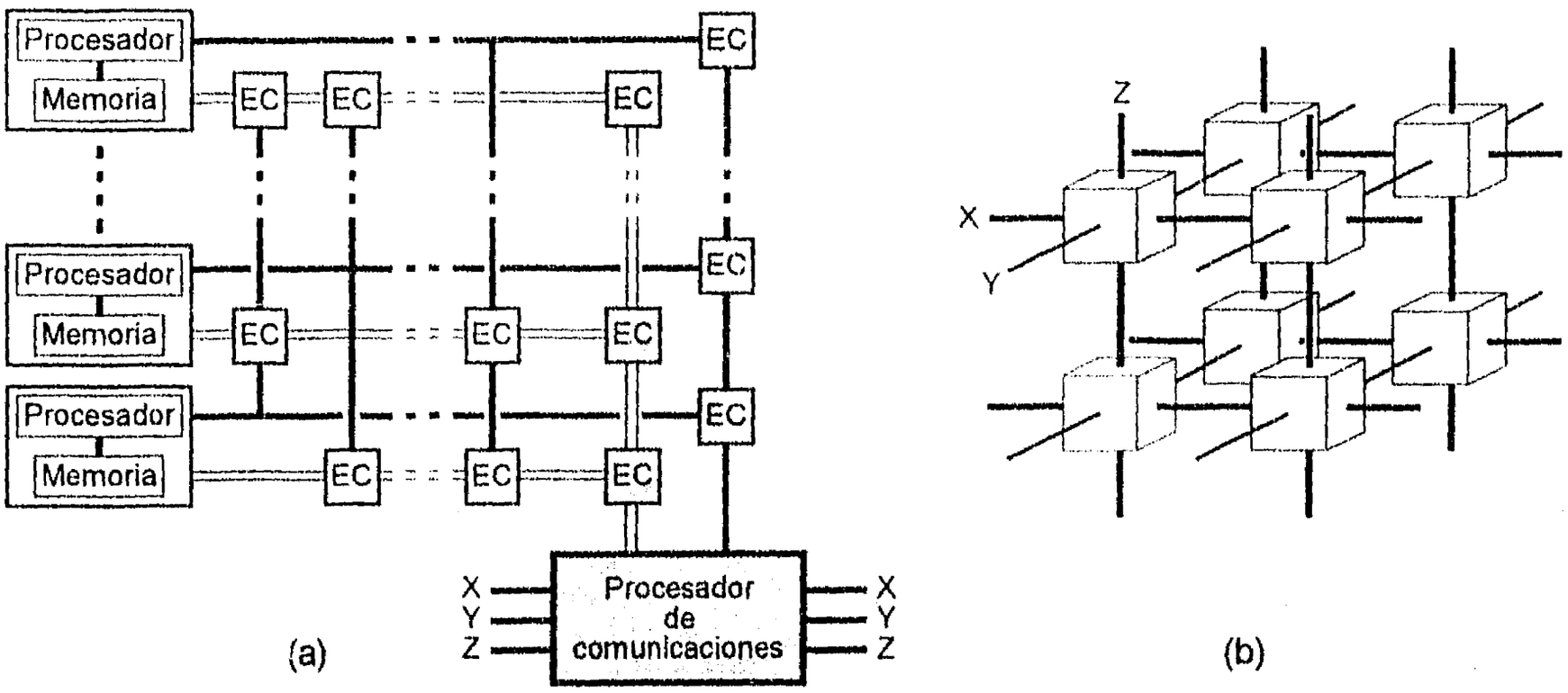


Figura 65: Extensión de la arquitectura (a), para uso en estructuras de interconexión fija (b).

El procesador de comunicaciones puede transmitir una solicitud de acceso a un nodo remoto desde cualquiera de los procesadores locales, por medio de un bus que, a través de elementos de conmutación, lo comunica con estos últimos (bus marcado en negro). Similarmente, cualquier acceso a memorias locales proveniente de nodos remotos, puede ser atendida por medio de un segundo bus que establece la comunicación requerida. Cabe subrayar los siguientes aspectos:

1.- Los elementos de conmutación adicionales requeridos (2p, donde p es el número de procesadores), proporcionan, adicionalmente al objetivo de comunicación señalado, un nivel de redundancia para la operación interna de la red de interconexión. Esto es, en caso de falla de un elemento de conmutación del multiprocesador, la conexión correspondiente puede ser establecida por medio de estos dispositivos adicionales.

2.- Desde el punto de vista de los nodos externos, los módulos de memoria del multiprocesador constituyen un sólo espacio de direccionamiento. En caso de que los elementos de conmutación cuenten con memorias caché, el acceso a este espacio no solamente es sumamente rápido, sino, y tal vez más importante, no interrumpe la operación del multiprocesador (para accesos exitosos).

3.- La velocidad de comunicación entre los nodos puede incrementarse, aumentando el número de buses entre el, o los, procesadores de comunicaciones, y el multiprocesador propiamente dicho. Para el ejemplo de la figura 65, hasta tres pares de buses, y tres procesadores de comunicaciones, pueden dar servicio a las interconexiones con otros nodos.

4.- Por la consistencia de memoria intrinseca al diseño, los protocolos de comunicación entre nodos son considerablemente simplificados. Para protocolos basados en directorios, como, por ejemplo, en la máquina "DASH" de la Universidad de Stanford [58], por lo menos un nivel puede ser eliminado.

Para los accesos provenientes de otros nodos, y por las características (2), (3) y (4) descritas arriba, el multiprocesador puede ser modelado como un banco de módulos de memoria, interconectados por una red de buses múltiples. Por otra parte, el acceso a un nodo remoto desde un procesador dado, tiene características similares a las encontradas en máquinas de bus común.

Una interrogante que requiere investigación futura es, entonces, la relativa al orden equivalente de una red cuyos nodos emplean multiprocesadores como elementos de procesamiento. Como ejemplo, considérese un hipercubo de orden N con 2^N procesadores, y un segundo hipercubo con multiprocesadores de orden M, de 2^M procesadores, donde $M < N$, como elementos computacionales en cada nodo. Evidentemente, si este segundo hipercubo es de orden N - M, tendrá el mismo número de procesadores que el primero. Por hipótesis, el segundo hipercubo no sólo es más fácil de construir, sino, además, posee un rendimiento computacional superior al primero. Esta hipótesis se fundamenta en dos argumentos: primero, la intercomunicación a nivel multiprocesador, esto es, para grupos de 2^M procesadores, es de mayor velocidad y, segundo, el diámetro de la red es menor (log(N-M), comparado con log N). Sin embargo, si más de un procesador de un multiprocesador dado genera accesos a un mismo nodo exterior a ese multiprocesador, se presenta la contienda por el bus que une a estos nodos. Luego, el nivel de incremento en rendimiento computacional no parece ser fácilmente cuantificable, y cabe la posibilidad de que, para un valor de N dado, exista un valor de M óptimo (sujeto a la restricción, claro, de que $M < N$), en términos de rendimiento computacional, cantidad de componentes requeridas, y costo. El resultado de estas evaluaciones puede tener profundas repercusiones para máquinas de paralelismo masivo, en particular, para protocolos de comunicación y modelos de programación.

BIBLIOGRAFIA

[1] M.J. Flynn: "Very High-Speed Computing Systems", *Proc. IEEE*, vol.54, pp. 1901-1909, diciembre de 1966.

[2] B. Codenotti, M. Leoncini.: *Introduction to Parallel Processing*, International Computer Science Series, Addison-Wesley, 1993.

[3] De Blasi, M.: *Computer Architecture*, Addison-Wesley, 1990.

[4] J. Von Neumann.: Collected Works (ed. A. H. Taub), 6 vols., Pergamon, New York, 1963; Vol. 5: *Design of Computers, Theory of Automata and Numerical Analysis*.

[5] G.S. Almasi y A. Gottlieb: "*Highly Parallel Computing*", Benjamin/Cummings, 1994.

[6] A.V. Aho, J. Hopcroft, J.D. Ullman: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[7] S. Fortune, J. Willy: "Parallelism in Random Access Machines", *Proc. of 10th Annual ACM Symp. on Theory of Computing*, 1978, pp. 114-118.

[8] D. Nassimi, S. Sahni: "Data Broadcasting in SIMD Computers", *IEEE Trans. on Computers*, C-30 (1981), pp. 101-107.

[9] L. Bhuyan, X. Zhang (ed.): *Multirpocessor Performance Measurement and Evaluation*, IEEE Computer Society Press, 1995.

[10] I. Sherson, A. Youssef (ed.): *Interconnection Networks for High-Performance Parallel Computers*, IEEE Computer Society Press, 1994.

[11] A. Varma, C.S. Raghavendra (ed): *Interconnection Networks for Mul;tiprocessors and Multicomputers: Theory and Practice*, IEEE Computer Society Press, 1993.

[12] G. Haring, G. Kotsis (ed.): Advances in Parallel Computing, Vol 7: *Performance Measurement and Visualization of Parallel Systems*, North-Holland, 1993.

[13] J. Dongarra, W. Gentzsch (ed.): Advances in Parallel Computing, Vol 8: *Computer Benchmarks*, North-Holland, 1993.

[14] M. Marsan, G. Balbo, G. Conte: *Performance Models of Multiprocessor Systems*, MIT Press, 1990.

[15] J. Medhi: *Stochastic Models in Queueing Theory*, Academic Press, 1991

[16] D. G. Kendall: "Some Problems in the Theory of Queues", *J. Roy. Stattist. Soc. Ser.* B 13(2) (1951), pp. 151-185.

[17] T. W. Keller: "Computer System Models with Passive Resources", Tesis doctoral, Departamento de Ciencias de la Computación, Universidad de Texas en Austin, 1976.

[18] K. M. Chandy, C. H. Sauer: "Approximate Methods for Analizing Queuing Network Models of Computer Systems", *ACM Computing Survey* 10(3)(septiembre de 1978), pp. 281-317.

[19] A. Marsan, M. Gerla: "Markov Models for Multiple Bus Multiprocessor Systems", *IEEE Transactions on Computers* C-31(3)(Marzo de 1982), pp. 239-248.

[20] C. Skinner, J. Asher: "Effect of Storage Contention on System Performance", *IBM System Journal* 8(4)(1969), pp. 319-333.

[21] D. P. Bhandarkar: "Analysis of Memory Interference in Multiprocessors", *IEEE Transactions on Computers* C-24(9)(Septiembre de 1975), pp. 897-908.

[22] C.H. Hoogendoorn: "A General Model for Memory Interference in Multiprocessors", *IEEE Transactions on Computers* C-26(10)(Octubre de 1977), pp. 988-1005.

[23] W. D. Strecker: "Analysis of the Instruction Execution Rate in Certain Computer Structures", *Tesis Doctoral*, Universidad Carnegie-Mellon, Pittsburgh, Pensylvania, 1970.

[24] T. N. Mudge, B. A. Makrucki: "Probabilistic Analysis of a Crossbar-Switch", *Proc. 9th Annual Symposium on Computer Architecture*, Austin, Texas, abril de 1982.

[25] L. N. Bhuyan, Q. Yang, D. P. Agrawal: "Performance of Multiprocessor Interconnection Networks", *Computer*, Vol.22, No.2, Feb. 1989, pp. 25-37.

[26] J.H. Patel: "Performance of Processor-Memory Interconnections for Multiprocessors", *IEEE Trans. Computers*, Vol. C-30, Oct. 1981, pp. 771-780.

[27] C.P. Kruskal, M. Snir: "The Performance of Multistage Interconnection Networks for Multiprocessors", *IEEE Trans. Computers*, Vol. C-32, Dic. 1983, pp. 1091-1098.

[28] C.P. Kruskal, M. Snir, A. Weiss: "The Distribution of Waiting Times in Clocked Multistage Interconnection Networks", *IEEE Trans. Computers*, Vol. 37, No. 11, Nov. 1988, pp. 1337-1352.

[29] L.R. Rabiner, B. Gold: *Theory and Application of Digital Signal Processing*, Prentice-Hall, 1975.

[30] W. Press, B. Flannery, S. Teukolsky, W. Vetterling: *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, 1986.

[31] F.T. Leighton: *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*, Morgan Kaufmann, 1992.

[32] Pentium™ Processor User's Manual, Vol. 1: *Pentium™ Processor Data Book*, INTEL Corp., 1994.

[33] K. Batcher: "Sorting networks and their applications", *Proceedings of the AFIPS Spring Joint Computing Conference*, Vol. 32, 1968, pp. 307-314.

[34] R. Cypher, G. Plaxton: "Deterministic sorting in nearly logarithmic time on the hypercube and related computers", *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, 1990, pp. 193 - 203.

[35] D. Knuth: *Searching and Sorting*, Vol. 3 de *The Art of Computer Programming*, Addison - Wesley, Reading, MA., 1973.

[36] A.A. Sawchuk, B.K. Jenkins, C.S. Raghavendra y A. Varma: "Optical Crossbar Networks", *Computer*, Junio 1987, pp. 50-60.

[37] R.C. Pearce, J.A. Field y W.D. Little: "Asynchronous Arbiter Module", *IEEE Trans. Computers*, Sept. 1975, pp. 931-932.

[38] T. Lang y M. Valero: "M - users B - servers Arbiter for Multiple-Buses Multiprocessors", *Microprocessing and Microprogramming*, 1982, pp.11-18.

[39] T.N. Mudge, J.P. Hayes y D.C. Winsor: "Multiple Bus Architectures", *Computer*, Junio 1987, pp. 42-48.

[40] W. Crowther, J. Goodhue, R. Gurwitz, R. Rettberg y R. Thomas: "The Butterfly Parallel Processor", *IEEE Computer Architecture Newsletter*, Sept. - Dic. 1985, pp. 18-45.

[41] W.J. Dally y C.L. Seitz: "The Torus Routing Chip", *Journal of Distributed Computing*, 1(4), 1986.

[42] R.M. Fujimoto: "VLSI Communication Components for Multicomputer Networks", *CS Division Report No. UCB/.CSD 83/136*, Universidad de California en Berkeley, C.A., 1983

[43] Y. Rimoni, I.Zisman, R. Ginosar, U. Weiser: "Communication Element for the Versatile MultiComputer", *15th IEEE Conference in Israel*, Abril 1987.

[44] Y. Tamir y G.L. Frazier: "High-Performance Multi-Queue Buffers for VLSI Communication Switches", *Proc. 15th Ann. Symp. Computer Architecture*, 1988, pp. 343-354.

[45] L. Censier y P. Feautrier: "A new solution to coherence problems in multicache systems", *IEEE Trans. Comput.*, vol. C-27, Dec. 1978, pp.1112-1118.

[46] M.S. Papamarcos, J.H. Patel: "A low overhead coherence solution for multiprocessors with private cache memories", *Proc. 11th Int. Symp. Comput. Architecture*, Mayo 1984, pp. 348-354.

[47] G. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta y J. Hennesy: "Memory consistency and event ordering in scalable shared-memory multiprocessors", *Proc. 17th Int. Symp. Comput. Architecture*, Mayo 1990, pp. 15-26.

[48] A. Gupta, W.D. Weber y T. Mowry: "Reducing memory and traffic rquirements for scalable directory-based cache coherence schemes", *Proc. 1990 Int. Conf. Parallel Processing*, Agosto 1990, pp. 1:312-321.

[49] K. Chen: "Study on the cache memory miss ratio issue in multiprocessor systems", *Inst. Nat. Recherche Inf. Autom.*, Le Chesnay, France, 1990.

[50] A. Agarwal, B.H. Lim, D. Kranz y J. Kubiatowics: "LimitLESS directories: A scalable cache coherence scheme", *Proc. Fourth Int. Conf. Architectural Support Programming Languages and Oper. Syst.*, Abril 1991, pp. 224-234.

[51] A. Leff, P.S. Yu, J.L. Wolf: "Policies for efficient memory utilization in a remote caching architecture", *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, IEEE Comput. Soc. Press, 1991.

[52] A. Choudhary, S. Krishnamoorthy: "Experimental evaluation of multilevel caches for shared memory multiprocessors", *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, IEEE Comput. Soc. Press, 1991.

[53] Z. Fang, M. Lu, H. Lin: "An approach to solve the cache thrashing problem", *Proceedings of the Fifth International Parallel Processing Symposium*, IEEE Comput. Soc. Press, 1991.

[54] J. Fang, M. Lu: "An iteration partition approach for cache or local memory thrashing on parallel processing", *Languages and Compilers dor Parallel Computing. Fourth International Workshop*, Springer Verlag, Berlin, Alemania, 1992.

[55] D. Lenoski, J. Laudon, G. Gharachorloo, W.D. Weber, A. Gupta, J. Hennesy, M. Morowitz y M. Lam.: "The Stanford Dash Multiprocessor", *IEEE Computer*, vol.25, no.3, marzo de 1992, pp. 63-79.

[56] E. Hagersten, A.Landin y S. Haridi: "DDM - A Cache-Only Memory Architecture", *Computer*, Sept. 1992, pp. 44-54.

[57] Zima, H.P. (ed.): *Parallel Computation*. Proceedings of the First International ACPC Conference, Springer Verlag, Berlin, Alemania, 1992.

[58] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta y J. Hennesy: "The Dash Prototype: Logic Overhead and Performance", *IEEE Trans. Parallel and Distributed Systems*, Vol.4, No.1, enero de 1993, pp. 41-61.

[59] M.S. Lam: "Locality optimizations for parallel machines", *Parallel Processing: CONPAR 94 - VAPP VI. Third Joint International Conference on Vector and Parallel Processing Proseedings*, Springer-Verlag, Berlin, Alemania, 1994.

[60] J.P. Hayes: *Computer Architecture and Organization*, McGraw-Hill, 1978.

[61] Cray Research, Inc.: "Cray T3D: Technical Summary", Septiembre 1993.

[62] PRO Series Reference Manual (tres volúmenes), *Viewlogic Systems, Inc.*, 1993.

[63] Hiwire II User's Guide, *Wintek Corp.*, 1992.

[64] FPGA Data Manual, *Texas Instruments,* 1994.

[65] FPGA Data Book and Design Guide, *Actel Corp.*, 1995.

[66] Programmable Logic Devices Data Book and Design Guide, *National Semiconductor*, 1995.

[67] Pentium™ Processor User's Guide (tres volúmenes), *INTEL Corp.*, 1994.

[68] Cache Tutorial. *INTEL Corp.*,1991.

[69] MOS Memory Data Book, *Texas Instruments*, 1993.

[70] 82420/82430 PCIset ISA and EISA Bridges, *INTEL Corp.*, 1994.

APENDICE I

Tiempo medio de acceso de la red *crossbar*

La probabilidad de que un acceso a memoria dado involucre contienda por módulos depende de la naturaleza del algoritmo, esto es, de la simetría de la gráfica del problema. Considérese la situación en la cual un acceso desde p procesadores involucra a los módulos de memoria j, j = 1, 2, ..., m, tal que:

1.- La ocurrencia de una referencia al módulo j es igualmente probable para toda j.
2.- La ocurrencia de una referencia al módulo j es independiente de la ocurrencia de una referencia al módulo k ($k \neq j$).

En lo subsecuente, se deriva una forma de calcular el tiempo medio de acceso $\tau(p,m)$ de p procesadores a m módulos de memoria, en un multiprocesador de memoria compartida, basado en la red de conmutación *crossbar*. Supóngase que una configuración de acceso de p procesadores a m módulos de memoria es representada por una matriz {0,1} de p x m elementos, $A = \{a_{ij}\}$, tal que:

$\sum_j a_{ij} = 1$, para i = 1, 2, ..., p.

Aquí, $a_{ij} = 1$ si y sólo si el procesador i genera una solicitud de acceso al módulo de memoria j. Sea $c(A) = (c_1, c_2, ..., c_m)$, donde $c_j = \sum_i a_{ij}$ is definida como la cantidad de accesos simultáneos (colisiones) al módulo de memoria j, j = 1, 2, ..., m. Finalmente, supóngase que el tiempo de acceso C(A) de la configuración de acceso A está dado por $\max_j\{c_j\}$.

Ejemplo. Considérese la siguiente configuración de acceso, que involucra a 8 procesadores y 8 módulos de memoria:

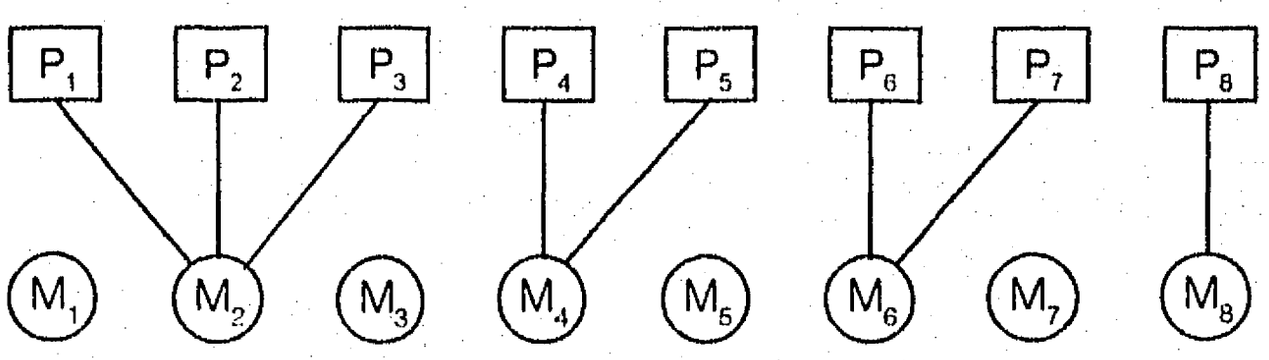


Figura 1

Esta configuración de acceso queda, entonces, representada por la siguiente matriz:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

De la matriz anterior se obtiene: c(A) = (0, 3, 0, 2, 0, 2, 0, 1) y C(A) = 3. Ahora bien, si $\mathbf{A}_{pm}$ es el conjunto de configuraciones de acceso que involucra p procesadores y m módulos de memoria (este conjunto tiene m^p elementos), y si se denota por $\alpha[k,p,m]$ el número de matrices $A \in \mathbf{A}_{pm}$ tal que C(A) = k (k es un entero positivo), entonces:

$$\tau(p,m) = \frac{1}{m^p} \sum_{k=1}^{p} k\alpha[k,p,m] \qquad \text{..... 1}$$

Parece difícil obtener una expresión cerrada para $\alpha[k,p,m]$. Sin embargo, para $p \geq m$, los siguientes casos son fácilmente verificados:

$$\alpha[1,p,m] = m!/(m - p)!$$

$$\alpha[2,p,m] = \sum_{j=1}^{\lfloor p/2 \rfloor} \frac{m!p!}{(m-p+j)!(2)^j j!(p-2j)!}$$

$$a[p-1,p,m] = (m-1)mp$$

$$a[p,p,m] = p$$

A cada $A \in \mathbf{A}_{pm}$ se asocian los vectores $b(A) = (b_1, b_2, ..., b_s)$ y $d(A) = (d_1^{e_1}, d_2^{e_2}, ..., d_r^{e_r})$. El vector b(A) es obtenido ordenando en forma decreciente los elementos positivos de c(A), mientras que d(A) satisface:

(i) $d_1 > d_2 > ... > d_r > 0$, y

(ii) para cada $j = 1, 2, ..., r$ existe una i correspondiente ($1 \leq i \leq s$) tal que $d_j = b_i$, y $e_j > 0$ es el número de elementos de b(A) iguales a d_j.

Para el ejemplo dado arriba, b(A) = (3, 2, 2, 1) y $d(A) = (3, 2^2, 1)$.

Sea $V_{pm} = \{b(A) \mid A \in \mathbf{A}_{pm}\}$. Entonces:

Proposición 1.

$$\alpha[k,p,m] = \sum_{\substack{(b_1,\ldots,b_s)\in V_{pm} \\ b_1 = k}} \frac{p!m!}{(m-s)!\prod_{j=1}^{s} b_j!\prod_{k=1}^{r} e_k!} \qquad \ldots\ldots 2$$

Prueba. Sea $(b_1, \ldots, b_s)\in V_{pm}$ y sea $\gamma[(b_1, \ldots, b_s), p, m]$ el número de matrices $A\in \mathbf{A}_{pm}$ tal que $b(A) = (b_1, \ldots, b_s)$. Entonces, obviamente,

$$\alpha[k,p,m] = \sum_{\substack{(b_1,\ldots,b_s)\in V_{pm} \\ b_1 = k}} \gamma[(b_1, \ldots, b_s), p, m]$$

Como, por otra parte,

$$\beta = \frac{p!}{\prod_{j=1}^{s} b_j!\prod_{k=1}^{r} e_k!} \qquad \ldots\ldots 3$$

es la cantidad de particiones posibles de un conjunto de p filas en s subconjuntos de cardinalidad $b_1, \ldots, b_s$ (véase [1], por ejemplo), entonces $\gamma[(b_1, \ldots, b_s), p, m]$ puede ser calculado como el producto de β por m!/(m-s)! (cantidad de elecciones posibles de s columnas de entre m columnas). Por lo tanto:

$$\gamma[(b_1, \ldots, b_s), p, m] = \frac{p!m!}{(m-s)!\prod_{j=1}^{s} b_j!\prod_{k=1}^{r} e_k!} \qquad \ldots\ldots 4$$

y la proposión sigue. □

Ejemplo. Usando (2) y (4) para calcular $\alpha[k,p,m]$ for k = 3, p = m = 8, se tiene:

$\gamma[(3, 1^5),8, 8]$ = $(8!)^2/(8-6)!\{3!\}5!$ = 1,128,960
$\gamma[(3, 2, 1^3), 8, 8]$ = $(8!)^2/(8-5)!\{3!2!\}3!$ = 3,763,200
$\gamma[(3, 2^2, 1), 8, 8]$ = $(8!)^2/(8-4)!\{3!2!2!\}2!$ = 1,411,200
$\gamma[(3^2, 1^2), 8, 8]$ = $(8!)^2/(8-4)!\{3!3!\}2!2!$ = 470,400
$\gamma[(3^2, 2), 8, 8]$ = $(8!)^2/(8-3)!\{3!3!2!\}2!$ = 94,080

Y, por lo tanto, $\alpha[3,8,8]$ = 6,867,840. Esto es, de 8^8 = 16,777,216 configuraciones de acceso posibles, aproximadamente el 40.9% involucra colisiones de orden 3.

Con base en las ecuaciones (1), (2) y (4) para p = m, se han calculado las tablas I y II:

Tabla I: tiempo medio de acceso

p	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
τ(p,m)	1.00	1.50	1.89	2.125	2.29	2.41	2.51	2.60	2.68	2.75	2.82	2.88	2.93	2.99	3.03

Tabla II: tiempo medio de acceso

p	1	2	4	8	16	32	64	128	256
$\tau(p,m)$	1.00	1.50	2.125	2.60	3.08	3.53	3.97	4.39	4.75

Se mostrará, a continuación, que el tiempo medio de acceso puede ser expresado en términos de una función de distribución de densidad de probabilidad. Sea $x = (x_1, ..., x_m)$ un vector columna, y denótese por P(p,m) la expansión polinomial de $(x_1 + ... + x_m)^p$ sin reagrupar términos, y sea $\Im(x)$ el producto:

$$\Im(x) = \prod_{j=1}^{m} x_j$$

Proposición 2. El número de términos en P(p,m) cuyo exponente máximo es k (k = 1, ..., p), está dado por $\alpha[k,p,m]$.

Prueba. Dado que cada matriz $A \in \mathbf{A}_{pm}$ corresponde a un vector único Ax, tenemos:

$$P(p,m) = \sum_{A \in \mathbf{A}_{pm}} \Im(Ax)$$

Además, cada matriz $A \in \mathbf{A}_{pm}$ corresponde a un término $\Im(Ax)$ cuyo máximo exponente es claramente igual a C(A). □

El tiempo medio de acceso definido arriba considera que un conjunto de p accesos no ha sido atendido sino hasta que la colisión de mayor orden haya sido ejecutada (secuencialmente) por el módulo afectado. Una segunda medida para el tiempo de acceso puede consistir en el valor medio experimentado por cada procesador, esto es, sea el tiempo de acceso $\tau(A)$ de la configuración de acceso A dado por:

$$\tau(A) = \frac{1}{s} \sum_{j=1}^{p} c_j \qquad \text{..... 5}$$

Donde s es el número de elementos del vector b(A). Como ejemplo, para c(A) = (0, 3, 0, 2, 0, 2, 0, 1), se obtiene $\tau(A) = 2$. El tiempo medio de acceso puede ser calculado, entonces, asociando a cada γ un factor $\tau(A)$, dado por (5).

REFERENCIA

[1] Graham, R., Knuth, D., Patashnik, O.: "Concrete Mathematics", *Addison-Wesley Publishing Co.*, 1992.

APENDICE II

Simulación de la red mariposa

Lo subsecuente está basado en la representación tipo delta de la red mariposa. Esta representación posee la siguiente propiedad [1]: A cada pareja de nodos de salida de un elemento de conmutación de dos interruptores se le asigna la etiqueta 0,1. Entonces, cualquier trayectoria de entrada/salida puede ser especificada por la secuencia de etiquetas que corresponde a la representación binaria del nodo destino de la trayectoria. Por ejemplo, la trayectoria desde la entrada 2 a la salida 6 posee la secuencia de etiquetas $110 = 6_2$.

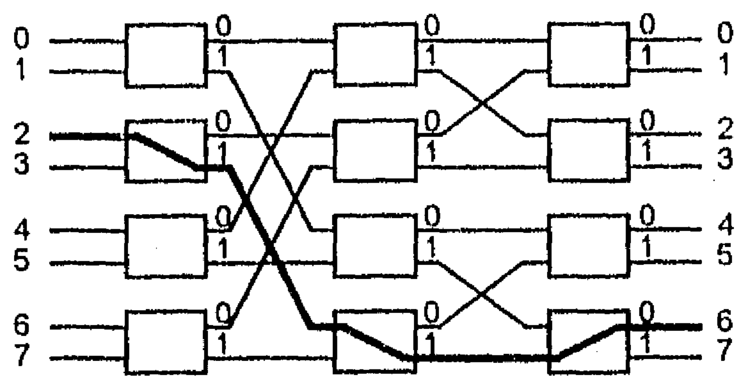


Figura 1: La trayectoria entre la entrada 2 y la salida 6.

Considérese una red de $N=2^r$ nodos de entrada, donde r es un entero positivo. Denótese cada interruptor sw, y su correspondiente entrada, s, por la pareja i,j, donde i es la fila ($0 \le i \le N-1$), y j la columna ($0 \le j \le \log N-1$) de su ubicación en la red. Para una trayectoria dada, el número de entrada i del interruptor sw_{ij} es una función del número de columna en la cual se encuentra ubicado.

Sea $i^* = i$, para i par, y sea $i^* = i-1$, para i impar. Por otra parte, sea $L = N/2^{j+1}$. Denótese por $l_{i,j}$ la etiqueta del interruptor $sw_{i,j}$, y por $l^c_{i,j}$ el complemento lógico (negación) de $l_{i,j}$. Entonces, para $kL \le i<(k+1)L$, k = 0, 1, ..., la salida del interruptor $sw_{i,j}$ está conectada a la entrada del interruptor $s_{i,j+1}$, donde i(j+1) está dada por:

$$i(j+1) = i^*(j)+Ll_{i,j}, \quad \text{para k par y}$$
$$i(j+1) = i^*(j)+1-Ll^c_{i,j}, \quad \text{para k impar} \qquad \ldots\ldots 1$$

Para la trayectoria entre la entrada 2 y la salida 6 y utilizando (1), se obtiene la secuencia de entradas de interruptores $s_{2,0}$, $s_{6,1}$, $s_{7,2}$. De manera similar, para la trayectoria entre la entrada 2 y la salida 4, la secuencia de entradas es 2, 6, 5 (para, respectivamente, las columnas 0, 1 y 2). Supóngase que las N entradas a la red son representadas por la secuencia <0, 1, ..., N-1>. Entonces, se puede construir una tabla con N filas, donde cada fila contiene la secuencia de las entradas de los interruptores para una de las trayectorias de entrada/salida especificadas. La tabla I es un ejemplo para la secuencia de salidas < 0, 5, 7, 0, 1, 2, 0, 2 >, donde d_i denota a la salida del interruptor $sw_{i,\log N-1}$.

Sea k el número de fila de la tabla I, $0 \le k \le \log N-1$. Si un número de entrada dado se encuentra en más de una fila de una columna dada, entonces se presenta una contienda por ese interruptor, esto es, más de una trayectoria pasa por el mismo interruptor. El algoritmo que se describe a continuación

Tabla I: Una secuencia de entradas a interruptores.

s_{i0}	s_{i1}	s_{i2}	d_i
0	0	0	0
1	4	4	5
2	6	7	7
3	2	1	0
4	1	0	1
5	1	2	2
6	3	1	0
7	3	3	2

está basado en la suposición de que cada interruptor posee memoria propia en la forma de una memoria de primer arribo - primera salida (FIFO), de suficiente capacidad para contener colas de longitud arbitraria.

Supóngase que a cada entrada de un interruptor se asocia un apuntador $p_{i,j}$, cuyo valor indica el orden en que los datos son transferidos a la entrada $s_{i,j}$. Entonces, para una columna dada y para toda k, cualquier valor de $p_{i,j}$ puede aparecer, a lo más, en una ocasión.

El algoritmo emplea dos contadores, uno para el número de iteración ejecutado, y un segundo contador, que acumula la cantidad de datos que han sido transferidos a la salida de la red. La cuenta inicial para ambos contadores es cero. Las condiciones iniciales del algoritmo son $p_{i,j}=1$ para $j=0$, y $p_{i,j}=0$ para $j\neq 0$. Esto es, los datos han sido escritos a todas las entradas de la columna cero. Los datos se supone serán transferidos de la columna j a la columna j+1, un paso a la vez. El algoritmo mide el número de pasos requerido para transferir todos los datos desde las entradas $s_{i,0}$ a las salidas d_i, como sigue.

Para i=0, 2, ..., N-2, j=logN-1, logN-2, ..., 0:

INICIO: Iguala j=logN-1 e incrementa al contador de iteraciones.

1.- Iguala i=0.

2.- Desde k=0 hasta N-1, encuentra $s_{i,j}$ y $s_{i+1,j}$ tales que $p_{i,j}\neq 0$ y $p_{i,j}=\min(p_{i,j})$, $p_{i+1,j}\neq 0$ y $p_{i+1,j}=\min(p_{i+1,j})$. Esto es, localiza a la pareja de entradas que contiene los primeros datos aún no transferidos.

3.- Si no se encontró entrada alguna, procede al siguiente valor de i.
Si una entrada ha sido encontrada, ejecuta la rutina de transferencia. Después, procede al siguiente valor de i.
Si dos entradas han sido encontradas, entonces para $s_{i,j}$ y $s_{i+1,j}$, compara con $s_{i,j+1}$ y $s_{i+1,j+1}$, respectivamente. Si son iguales, ejecuta la rutina de transferencia para $s_{i,j}$. En caso contrario, ejecuta la rutina de transferencia para ambos $s_{i,j}$ y $s_{i+1,j}$. Después, procede al siguiente valor de i.

Siguiente valor de i:
Iguala i=i+2. Si i < N, procede a (2). En caso contrario, procede al siguiente valor de j.

Siguiente valor de i:
Iguala j=j-1. Si $j\geq 0$, procede a (1). En caso contrario:

4.- Verifica que todas las entradas han llegado a sus respectivas salidas:
Si la cuenta del contador de salida es igual a N, procede a FIN. En caso contrario, procede a INICIO.

FIN: La cuenta del contador de iteraciones es el resultado.

Rutina de transferencia:

1.- Para $s_{i,j}$ dado, iguala $p_{i,j}=0$.
Si $s_{i\,j+1}= d_i$:

2.- Incrementa al contador de salida.
Si $s_{i,j}\neq d_i$:

2.- Para $s_{i,j}$ dado, y desde k=0 hasta N-1, encuentra $\max(p_{i,j+1})$.

3.- Iguala $p_{i,j+1}$ con $\max(p_{i,j+1})+,1$ y regresa.

La tabla II ejemplifica el algoritmo para la secuencia de salida especificada en la tabla I. Los valores de $p_{i,j}$ son mostrados entre paréntesis. Para mayor claridad, los valores modificados de $p_{i,j}$ en una iteración dada son mostrados en negritas. El signo "+" en la columna d_i indica que un dato de entrada ha sido transferido a la salida.

Tabla II:
Un ejemplo del algoritmo

Condiciones iniciales:

$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	d_i
0 (1)	0	0	0
1 (1)	4	4	5
2 (1)	6	7	7
3 (1)	2	1	0
4 (1)	1	0	1
5 (1)	1	2	2
6 (1)	3	1	0
7 (1)	3	3	2

Primera iteración:

$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	d_i
0 (0)	0 (1)	0	0
1 (0)	4 (1)	4	5
2 (0)	6 (1)	7	7
3 (0)	2 (1)	1	0
4 (0)	1 (1)	0	1
5 (1)	1	2	2
6 (0)	3 (1)	1	0
7 (1)	3	3	2

Segunda iteración:

$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	d_i
0	0 (0)	0 (1)	0
1	4 (0)	4 (1)	5
2	6 (0)	7 (1)	7
3	2 (0)	1 (1)	0
4	1 1	0	1
5 (0)	1 (2)	2	2
6	3 1	1	0
7 (0)	3 (2)	3	2

Tercera iteración:

$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	d_i
0	0	0 (0)	0 +
1	4	4 (0)	5 +
2	6	7 (0)	7 +
3	2	1 (1)	0
4	1 (0)	0 (1)	1
5 (0)	1 (2)	2	2
6	3 (0)	1 (2)	0
7 (0)	3 (2)	3	2

Cuarta iteración:

$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	d_i
0	0	0	0
1	4	4	5
2	6	7	7
3	2	1 (0)	0 +
4	1	0 (0)	1 +
5	1 (0)	2 (1)	2
6	3	1 (2)	0
7	3 (0)	3 (1)	2

Quinta iteración:

$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	d_i
0	0	0	0
1	4	4	5
2	6	7	7
3	2	1	0
4	1	0	1
5	1	2 (0)	2 +
6	3	1 (0)	0 +
7	3	3 (1)	2