PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 12/08 — A2

(11) International Publication Number: **WO 95/25306**

(43) International Publication Date: 21 September 1995 (21.09.95)

(21) International Application Number: PCT/US95/02594

(22) International Filing Date: 3 March 1995 (03.03.95)

(30) Priority Data:
08/213,222 14 March 1994 (14.03.94) US

(71) Applicant: STANFORD UNIVERSITY [US/US]; Stanford, CA 94305-4055 (US).

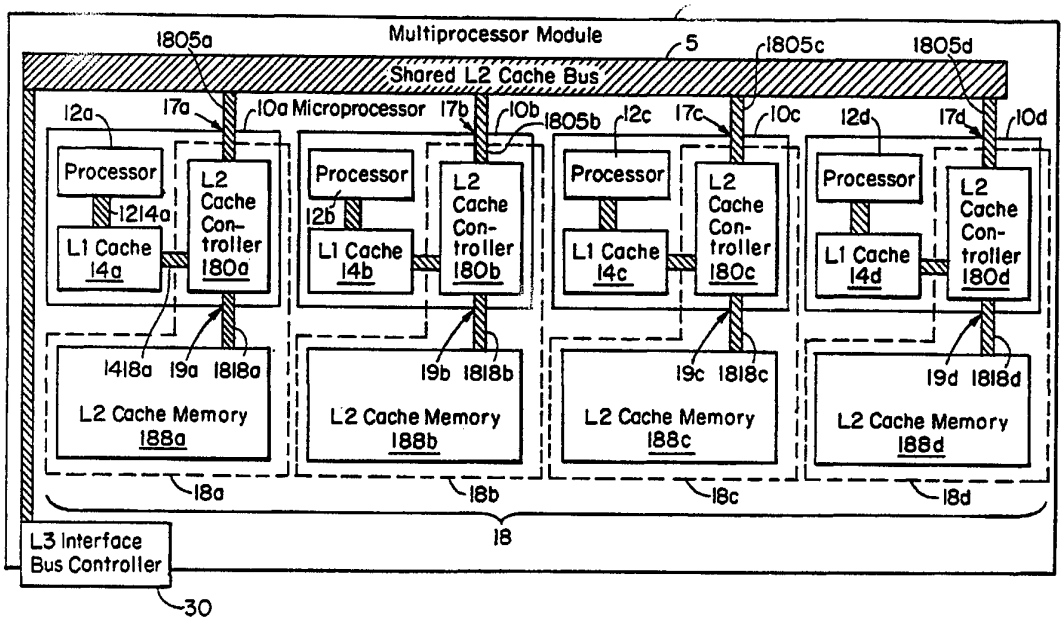
(72) Inventor: CHERITON, David, R.; 131 Couper Street, Palo Alto, CA 94301 (US).

(74) Agents: SMITH, James, M. et al.; Hamilton, Brook, Smith & Reynolds, Two Militia Drive, Lexington, MA 02173 (US).

(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: DISTRIBUTED SHARED-CACHE FOR MULTI-PROCESSORS



(57) Abstract

A distributed-shared cache operates at the level of a second-level memory cache, at the third level of the memory system, and at the purely software-managed page cache level. On a cache miss that is local to a processor, an attempt is made to locate the data in a cache memory block on a peer memory level, before explicitly requesting the data from more distant memory. Communication support is integrated into the memory system to piggyback communication performance improvements on improvements to the memory system. In particular, the cache lines can operate in a message mode to deliver message data to interested receivers to support networking and devices. Embodiments of the invention works across all memory levels with only modest changes in detail.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

DISTRIBUTED SHARED-CACHE FOR MULTI-PROCESSORS

Background of the Invention

Multiple-memory hierarchy levels have been used in cache-based shared memory multiprocessors to provide statistically fast memory access at a cost far below requiring all memory to be fast memory. Using the current generation of microprocessors, there is a private on-chip cache per processor that absorbs a large percent of the processor memory references. A second level or broad-level cache is provided to handle first-level cache misses quickly, compared to accessing the referenced data in main memory. In some current generation multiprocessors, the second-level cache interfaces to a memory bus, which provides access to memory as well as carrying consistency actions between the second-level caches.

Shared caching has been proposed as a means of reducing the per-processor cost of cache hardware, reducing the cost of interprocessor data sharing, and reducing the miss rate when the shared-cache is large enough. For example, shared second-level caches have been implemented in multiprocessors. In effect, current multiprocessors may treat main memory as a shared-cache, but use private second-level caches.

Communication facilities are a performance-critical aspect of operating systems and supporting hardware platforms, influencing the modularity with which sophisticated applications can be constructed. Passing messages between programs using shared memory is a technique for efficient communication that takes advantage of memory system performance.

Memory-based messaging uses a shared memory communication area between processes by creating a shared segment to act as a communication channel. Source and destination processes bind the shared segment into their respective address spaces, messages are written to the shared segment and, after notification at the destination(s), messages are read from the shaved segment.

Summary of the Invention

Shared-caching has a number of significant advantages. The per-processor cost of hardware is reduced, the cost of data sharing between processors is reduced, and the cost of cache misses to the next lower (and slower) level of the memory hierarchy (i.e., away from the processor) is amortized over multiple processors if there is significant sharing taking place in the multiprocessors. However, shared-caching also introduces several concerns.

There are three major concerns involving shared-caching. First, shared-caching can result in replacement interference between processors sharing the same cache. That is, if one processor accesses a large amount of data, data required by a second processor may have to be removed from the cache to make space for the data of the first processor. Second, shared-caching with a shared-cache controller can result in increased queuing time for a cache access because of competing requests to the cache controller. Finally, the communication and arbitration cost of accessing a cache through a shared bus or channel to a shared-cache controller can introduce extra overhead. As a specific example at the hardware level, the Rambus™ memory interface from Rambus, Inc. provides very high data rates, but severely limits the interconnection distance from a processor to memory to roughly four inches. It is not feasible to make a sizeable second cache from this technology and have the cache connect directly to multiple processors within the Rambus™ wire length constraint.

A preferred embodiment of the invention is a shared-cache multiprocessor system in which a shared-cache is partitioned or distributed among several cache controllers. The partitioning of the cache memory limits the replacement interference that can occur in the system. Multiple cooperating cache controllers reduce the probability of one processor waiting for another processor at a cache controller. The partitioning tightly couples processors to at least one portion of the cache. In particular, distributed shared-caching is implemented at the second-level in a memory hierarchy, assuming a sufficiently large first-level private cache for each processor.

A preferred embodiment of the invention attempts to locate desired data at a peer level from closely-coupled peer caches before, and preferably instead of, going to the next level down (i.e., toward main memory) in the memory hierarchy. The mechanisms for doing this peer access changes between the different levels of the memory hierarchy. In a preferred embodiment of the invention, a hierarchical memory structure is provided both for memory (or cache) and interconnect. A preferred structure allows for heterogeneous (or different) forms of interconnect (i.e., bus, backplane network, and fiber optics) at different levels.

A preferred embodiment of the invention is a distributed shared cache memory. A common level of cache memory is distributed into cache memory areas. Each cache memory area is locally coupled to a respective data accessor, such as a data processor that accesses memory. A shared cache interconnect couples cache memory areas together to form a cache memory cluster. In particular, there are four cache memory areas per cache memory cluster and the interconnect is a shared bus or an interconnection network.

In each cache memory cluster, each cache memory area is remotely coupled to the locally-coupled data accessor of each other cache memory area. Local cache controllers are coupled between each data accessor and the locally-coupled cache memory area. Each local cache controller requests, on behalf of the data accessor, a cache line from the locally-coupled cache memory area. A cache miss may be returned from the locally-coupled cache memory area. Upon a cache miss from the locally-coupled cache memory area, the cache line is requested from the remotely-coupled cache memory areas.

A cache miss may also be returned from each of the remotely-coupled cache memory areas, in which case the cache line is requested from more distant memory. The more distant memory can be a lower-level memory or a peer level cache memory area in another cache memory cluster. Once the request is satisfied, the cache line is provided to the requesting data accessor. The ownership of the cache line is not invalidated or changed in the memory area holding the data relative to memory more distant from the data accessor than the holding memory.

A private cache can be coupled between each data accessor and the respective cache memory cluster. Each private cache memory area provides a private memory level relative to the respective data accessor. Furthermore, an interface is coupled to the shared cache interconnect for interconnecting each cache memory cluster to a respective main memory area. The main memory area attempts to satisfy cache misses that occur in each cache memory cluster.

In a preferred embodiment of the invention, the interface between the first-level and the second-level of the memory hierarchy provides a combination of private caching and shared caching behavior for second-level cache organization that meshes well with the directions of multiprocessor design and the possibilities of using Dynamic Random Access Memory (DRAM) in place of Static Random Access Memory (SRAM) as the second-level cache, thereby reducing the cost of computing systems.

In a preferred embodiment of the invention, the third-level of the memory hierarchy focuses on main memory and uses a cache directory that allows replication of cache block units at the third level. In particular, a preferred embodiment of the invention interconnects a small number of third-level memory areas with directories and caches. Thus, it is feasible to contact each of the other third-level modules in an interconnected cluster when there is a miss in the local third-level directory. A preferred embodiment of the invention at the third-level exploits a directory cache to allow the total physical address space to exceed the actual memory space.

There are preferably a plurality of main memory areas. Each main memory area is coupled to at least one other main memory area to form a group of main memory areas. The main memory areas can be coupled by a fiber optic coupling system or by a shared bus.

In a preferred embodiment of the invention, a fourth-level mechanism uses cache information to attempt to locate data at the peer level. A preferred mechanism provides dissemination of information about which nodes hold which pages.

A preferred embodiment of the invention is an integrated circuit for a multiple-level hierarchical memory computing system. The integrated circuit is adapted to a distributed shared cache. A data accessor requires memory references, typically due to execution of software instructions in a data processor from memory. A first cache memory provides a private memory level relative to the data accessor for storing a copy of frequently referenced memory. A first cache controller has an input bus and an output bus and interfaces the data accessor to the first cache memory, and a second cache controller is coupled to the input and output bus. The second cache controller is cooperatively shared with at least one remote data accessor and controls access to a local second cache memory and requests memory references from remote second cache memories and lower level memory. A remote data accessor can satisfy a memory reference from the local second cache memory.

Typical computer system architecture focus more on optimizing the memory system than the communication facilities. In a preferred embodiment of the invention, communication support is integrated into the memory system, both at the operating system and hardware level, effectively piggybacking communication performance improvements on the improvement of the memory system by optimizing the basic memory-based messaging model. In a particular preferred embodiment, the cache line contains a message mode for delivering message data between data accessors.

In a preferred embodiment of the invention, an external data accessor can be coupled to the cache memory cluster such that the external data accessor can access the cache memory areas. Each cache memory area in the cache memory cluster is remotely coupled to the external data accessor. The external data accessor can include a device controller operating as a peer with the local cache controllers. The device controller can be a disk controller, a network interface, or a controller for other computer peripherals.

Brief Description of the Drawings

The above and other features of the invention, including various novel details of construction and combination of parts, will now be more particularly described with reference to the accompanying drawings and pointed out in the claims. It will be understood that the particular distributed shared-cache for multi-processors embodying the invention is shown by way of illustration only and not as a limitation of the invention. The principles and features of this invention may be employed in varied and numerous embodiments, without departing from the scope of the invention.

FIG. 1 is a block diagram of a distributed shared second-level cache memory according to a preferred embodiment of the invention.

FIG. 2A is a flowchart illustrating the processing steps for a cache reference in the first-level cache 14 of FIG. 1.

FIG. 2B is a flowchart illustrating the processing steps for an invalidation from the second-level cache 18 in the first-level cache 14 of FIG. 1.

FIG. 2C is a flowchart illustrating the processing step for a downgrade operation from the second-level cache 18 in the first-level cache 14 of FIG. 1.

FIG. 3A is a flowchart illustrating the processing steps for a cache reference from a first-level cache 14 in a second level cache 18 of FIG. 1.

FIG. 3B is a flowchart illustrating the processing steps for an invalidation from memory in the second-level cache 18 of FIG. 1.

FIG. 3C is a flowchart illustrating the processing steps for a downgrade operation from memory in the second-level cache 18 of FIG. 1.

FIG. 4 is a high-level block diagram of a preferred embodiment of a processor chip 10 of FIG. 1.

FIG. 5 is a block diagram of a distributed shared third-level cache memory according to a preferred embodiment of the invention.

FIG. 6 is a block diagram of a bus-based distributed shared third-level cache memory according to a preferred embodiment of the invention.

FIG. 7 is a block diagram of a distributed shared page cache memory according to a preferred embodiment of the invention.

FIG. 8 is a block diagram of a multiprocessor module interfaced to an external device controller according to a preferred embodiment of the invention.

FIG. 9 is a block diagram illustrating address-valued signaling for message notification.

FIGs. 10A-10B are block diagrams illustrating the benefits of a single message delivery transaction.

Detailed Description of Preferred Embodiments of the Invention

Distributed shared-caching, according to preferred embodiments of the invention, is a general technique for building shared-caching systems. A preferred distributed shared-cache system achieves the benefits of shared-caches while minimizing some of the disadvantages associated with a shared central cache controller or manager. In preferred embodiments of the invention, a distributed shared-cache can be implemented at a second-level (2L) memory cache, at a third-level (3L) of the memory system, and at a purely software-managed page cache level. A preferred embodiment of a distributed shared-cache accommodates each of these levels well, with different approaches to locating data and maintaining consistency.

In general, distributed shared-caching limits replacement interference between processors sharing the same cache, reduces contention for shared-cache access, and reduces the cache access time for access to local portions of the cache. At the same time, distributed shared-caching retains the established benefits of shared-caches previously achieved with a shared central cache controller.

In general, distributed shared-caching maximizes the benefit of fast caches at various levels in the memory hierarchy, and minimizes the demands placed on the comparatively slower next level on the memory system. A preferred embodiment of

the invention functions across all memory levels with only modest changes in details.

Second-Level Distributed Shared-Caching

Distributed shared-caching can be implemented at the second-level in a memory hierarchy, assuming a sufficiently large first-level private cache for each processor. The following description of a preferred embodiment of a second-level distributed shared-cache mechanism illustrates a non-limiting example of a second-level distributed shared-caching system using current technology.

FIG. 1 is a block diagram of a distributed shared second-level cache memory according to a preferred embodiment of the invention. As illustrated, four processors 12a,...,12d implement and share a second-level cache 18 as part of a single multiprocessor module 1. The processors 12 are specific examples of data accessors, which access memory but do not necessarily process the data. The second-level cache 18 contains second-level cache controllers 180 and respective second-level cache memory 188.

Each microprocessor chip 10 contains a processor 12, a private first-level cache 14, and a local second-level cache controller 180. The processor 12 is coupled to the respective private first-level cache 14 by a private first-level cache bus 1214. The private first-level cache 14 is coupled to the local second-level cache 18 by a local second-level cache bus 1418. Each microprocessor 10 also includes two ports 17, 19. A shared bus port 17 connects the local second-level cache controller 180 to a shared second-level cache bus 5. A cache memory port 19 connects the local second-level cache controller 180 to a respective second-level cache memory 188.

A preferred embodiment of the invention utilizes a fast but narrow memory interconnection technology, so the two ports 17, 19 can be fast and independent, yet fit within reasonable chip pin limitations. One such memory interconnection technology is that of Rambus™. It being understood that other suitable memory interconnection technologies can be used in place of Rambus™.

The first-level cache 14 includes two bits per cache entry that identify a processor 12 associated with each cache line, in addition to the normal valid, writable, modified cache tags. Each second-level cache controller 180 manages the respective directly attached second-level cache memory 188 as a two-way set-associative cache memory. The degree of associativity can be an implementation decision, potentially ranging from direct-mapped to higher degrees of associativity. The cache tag memory, stored as part of the second-level cache memory 188, includes standard second-level cache tags.

When a processor 12a experiences a first-level cache miss, the respective local second-level cache controller 180a checks local second-level cache tags for the presence of the desired cache line. If the desired cache line is present, the cache line is delivered to the first-level cache 14a immediately across the local second-level cache bus 1418a from the respective second-level cache memory 188a. The two processor bits for this first-level entry are then set to indicate this requesting processor 12a. FIGs. 2A-2C illustrate the main processing functions of the first-level cache 14.

FIG. 2A is a flowchart illustrating the processing steps for a cache reference in a first-level cache 14a of FIG. 1. The first-level cache 14a receives an address for a cache reference from the respective private processor 12a. At step 1102, the first-level cache 14a uses index bits in the address from the processor 12a as an address to access the first-level cache 14a. Control then flows to step 1104, where the tag bits of the first-level cache line are checked to determine whether the tag bits are equal to the tag part of the address, and whether the first-level cache line is valid. If both conditions are satisfied, control flows to step 1106. If both conditions are not satisfied, control flows to step 1108.

Step 1106 determines whether the cache reference is a write operation to the cache. If the cache reference is a write operation, control flows to step 1110. If the cache reference is not a write operation, control flows to step 1116. At step 1116, the first-level cache 14a has a read hit. The required cache line is returned to the processor 12a.

Step 1110 determines whether the first-level cache 14a has write permission. If there is write permission, control flows to step 1118, indicating a write hit. At step 1118, the word is modified and the state of the cache line is updated by setting a "dirty" bit. If there is no write permission, control flows to step 1120. At step 1120, the first-level cache 14a transmits a read and Block-Move-In-Private (BMIP) signal and the address to the local second-level cache 18a over the local second-level cache bus 1418a. Control then flows to step 1122, where the first-level cache 14a waits for an assert-ownership signal from the second-level cache 18a over the local second-level cache bus 1418a. Upon receiving the assert-ownership signal, control flows to step 1128. At step 1128, the first-level cache 14a obtains the word from the second-level cache 18a over the local second-level cache bus 1418a and modifies the word. The state of the first-level cache line is updated to write permission and "dirty." The cache line is then returned to the processor 12a.

If there is no tag match at step 1104, then a check is performed at step 1108 to determine whether there is an empty line (i.e., a cache line that is not valid). If there is an empty line, control flows to step 1112. At step 1112, the empty line is selected as a victim in which to load the required cache line from the second-level cache 18a. Control then flows to step 1130. If there is not an empty line, control flows from step 1108 to step 1114.

At step 1114, a victim cache line is chosen. Preferably a Least Recently Used (LRU) algorithm is used to select the victim cache line. After the victim cache line is chosen, control flows to step 1124. At step 1124, a check is made to determine whether the victim line is "dirty." If the victim line is "dirty," control flows to step 1126. At step 1126, the "dirty" victim line is written back to the second-level cache 18a over the local second-level cache bus 1418a and control flows to step 1130. If the victim cache line is not "dirty," control flows from step 1124 to step 1130.

At step 1130, a read signal and address is transmitted to the second-level cache 18a over the local second-level cache bus 1418a. Control then flows to step 1132, where the first-level cache 14a waits for a response from the second-level

cache 18a. After a response is received, control flows to step 1134 where the first-level cache 14a updates the cache line, and returns the cache line to the requesting processor 12a, over the private first-level cache bus 1214a, if the operation is a read operation, or modifies the word if the operation is a write operation.

FIG. 2B is a flowchart illustrating processing steps for an invalidation operation from a second-level cache 18a in the first-level cache 14a of FIG. 1. The first-level cache 14a receives an address over the local second-level cache bus 1418a from the second-level cache 18a. At step 1202, index bits are used as an address to access the first-level cache 14a. Control flows to step 1204, where tag bits of the first-level cache line are checked to determine whether the tag bits are equal to the tag part of the address and whether the first-level cache line is valid. If the check is false, control flows to step 1212. If the check is true, control flows to step 1206.

At step 1206, a check is performed to determine whether this first-level cache line is "dirty." If the cache line is "dirty," control flows to step 1208, where the "dirty" line is written back to the second-level cache 18a (step 1208) and control flows to step 1210. If the cache line is not "dirty," control flows to step 1210.

At step 1210, the state of the first-level cache 14a is changed to invalid by clearing the valid bit. Control flows to step 1212.

At step 1212, an acknowledgement signal is transmitted to the second-level cache 18a over the local second-level cache bus 1418a. The acknowledgement signal indicates that the invalidation operation has completed.

FIG. 2C is a flowchart illustrating the processing steps for a downgrade operation in a first-level cache 14a of FIG. 1. A downgrade operation changes the first-level cache 14a to shared mode. An address is received from the second-level cache 18a over the local second-level cache bus 1418a. At step 1302, index bits are used as an address to access the first-level cache 14a. Control flows to step 1304, where a check is performed to determine whether tag bits of the first-level cache line are equal to the tag part of the address and whether the first-level cache line is valid. If the check is false, control flows to step 1312. If the check is true, control flows to step 1306.

At step 1306, a check is performed to determine whether this first-level cache line is "dirty." If the cache line is "dirty," processing flows to step 1308, where the "dirty" line is written back to the second-level cache 18a over the local second-level cache bus 1418a and the "dirty" bit is cleared. After step 1308, or if the cache line is not "dirty," control flows to step 1310. At step 1310, the state of the first-level cache 14a is changed to shared mode by clearing a write permission bit. Control then flows to step 1312.

At step 1312, an acknowledgement signal is transmitted back to the second-level cache 18a over the local second-level cache bus 1418a. The acknowledgement signal indicates that the downgrade operation has completed.

If the desired cache line is not present locally, the local second-level cache controller 180a broadcasts a request for the cache line on the shared second-level cache bus 5. Each other (i.e., remote) second-level cache controller 180b, 180c, 180d attached to the shared second-level cache bus 5, as well as a third-level interface bus controller 30, receives the broadcast request.

If the cache line is present in one of the remote caches 186, the remote second-level cache controller 180b controlling that cache line responds with the requested cache line. The remote second-level cache controller 180b records in a directory entry for this cache line the processor 12a holding this cache line. In particular, a preferred embodiment of the invention follows the directory scheme used in the ParaDiGM multicomputer architecture, as described by D.R. Cheriton et al. in "Paradigm: A Highly Scalable Shared-Memory Multicomputer Architecture," IEEE Computer, 24(2) (Feb. 1991). In that scheme, there are N bits in each directory entry, each bit corresponding to a respective processor. The requesting first-level cache 14a also records the responding (i.e., remote) second-level cache 18b so the requesting first-level cache 14a can write-back the cache line to the correct remote processor 12b and remote second-level cache controller 180b combination, if necessary. The response from the remote second-level cache controller 180b also disables the request at the third-level interface bus controller 30.

If a remote second-level cache controller 180b does not respond to a request that was broadcast over the shared second-level cache bus 5, the third-level interface bus controller 30 obtains the requested data from more distant memory by re-requesting the request over a third-level shared-cache bus (not shown). In this case, the cache line is loaded into the local second-level cache memory 188a managed by the requesting processor 12a. That is, a new second-level cache data block is loaded into the portion of local second-level cache memory 188a associated with the requesting processor 12a. If necessary, each processor 12 has at least one cache line of write buffer so the data in the cache line chosen for write-back can be stored in the write buffer pending write-back, making space for new cache line data. The act of replacing a cache line can also involve forcing a write-back of this cache line from the first-level cache 14b of a remote processor 12b.

Each second-level cache controller 180 can respond to an external second-level cache request concurrently with the respective processor 12 executing out of the respective private first-level cache 14, as long as the processor 12 does not access the local second-level cache controller 180 at the same time. If a processor 12 does access the respective second-level cache controller 180 concurrently with an external second-level cache request, the second requestor accessing the second-level cache controller 180 is forced to wait until the first request is completed. In a preferred embodiment of the invention, arbitration logic is used to rank the requesters. In a preferred embodiment of the invention, a private first-level cache controller 140a can broadcast a request to remote second-level cache controllers 180b, 180c, 180d at the same time as the local second-level cache controller 180a is handling an external second-level cache request.

The shared second-level cache memory 188 is thus partitioned among the four portions 188a,...,188d (as illustrated in FIG. 1) managed by each respective processor 12a,...,12d. Because a first-level cache miss in any one of processors 12 is satisfied if the data is stored in any one of the memory portions 188a,...,188d, the overall effect is that of a single shared cache. The effective single shared-cache does have a slightly higher penalty for accessing another processor's second-level

memory portion 188b, 188c, 188d than the local portion of the second-level cache 188a. At this level, data is not stored multiple times between the portions of the second-level cache memory 188a...,188d, which furthers the benefits as a shared-cache. FIGs. 3A-3C illustrate the main processing functions of the second-level cache 18.

FIG. 3A is a flowchart illustrating the processing steps in a second-level cache 18a for a cache reference from a first-level cache 14a of FIG. 1. The second-level cache 18a receives an address for a cache reference over the local second-level cache bus 1418a. At step 2102, the second-level cache 18a uses index bits in the address as an address to access the cache 18a.

Steps 2104, 2108, 2114 determine whether the tag bits of the second-level cache line are equal to the tag part of the address and whether the second-level cache line is valid. Step 2104 checks the local second-level cache 18a. If there was no local tag match, control flows to step 2108. At step 2108, a request signal and address are sent to other (i.e., remote) second-level caches 18b, 18c, 18d over the shared second-level cache bus 5. Control then flows to step 2114. Step 2114 determines whether there is a remote tag match. If both tag match conditions are satisfied, control flows to step 2106. If both tag match conditions are not satisfied, control flows to step 2120.

Step 2106 determines whether the cache reference is a write operation to cache from the processor 12a. If the cache reference is a write operation, control flows to step 2110. If the cache reference is not a write operation, control flows to step 2112.

Step 2112 determines whether the requesting first-level cache 14a is the owner of this second-level cache 18. If the owner is the requesting first-level cache 14a, control flows to step 2130. If the requesting first-level cache 14a is not the owner, control flows to step 2118.

Step 2118 determines whether the second-level cache 18 is in private mode. If the second-level cache 18 is not in private mode, control flows to step 2130. If the second-level cache 18 is in private mode, control flows to step 2128. At step

2128, a downgrade signal is transmitted over the shared second-level cache bus 5 to first-level caches 14 that are current owners of this second-level cache line, except the requesting first-level cache 14a. Control then flows to step 2130.

If the cache reference is a write operation, step 2110 determines whether the second-level cache line has the exclusive copy of the memory line. If the copy is not the exclusive cluster copy, control flows to step 2120. If this is the exclusive cluster copy, control flows to step 2116.

Step 2116 determines whether the second-level cache line is in private mode. If the second-level cache line is not in private mode, control flows to step 2126. If the second-level cache line is in private mode, control flows to step 2122. Step 2122 determines whether the requesting first-level cache 14a is the owner of this second-level cache line. If the requesting first-level cache 14a is the owner, control flows to step 2130. If the requesting first-level cache 14a is not the owner, control flows to step 2126. At step 2126, an invalidation signal is transmitted over the shared second-level cache bus 5 to first-level caches 14 that are current owners of the second-level cache line, except the requesting first-level cache 14a. After step 2126, control flows to step 2130.

At step 2120, a request signal and address are sent to memory. At step 2124, the second-level cache 18 waits for memory to respond to the request signal of step 2120. Upon receiving a response, control flows to step 2130.

At step 2130, the state of the second-level cache line is updated. The cache line is returned to the requesting first-level cache 14a.

In a preferred embodiment as shown in FIG. 3A, a memory reference is satisfied by searching the local second-level cache at step 2102, the remote second-level caches (if necessary) at step 2108, and then more distant memory (if necessary) at step 2120. The method as shown separates searches to the remote second-level caches from searches to more distant memory. That approach is particularly appropriate where the second-level memory is capable of providing memory references with a quicker time than the third-level memory. That approach is also well-suited to memory systems where it may be difficult to cancel a memory

request to more distant memory. In another preferred embodiment, which is further described below, the requests to remote second-level caches and more distant memory are done concurrently and the requests to more distant memory are cancelled when a request to remote second-level caches succeeds.

FIG. 3B is a flowchart illustrating processing steps for an invalidation operation from memory in a second-level cache 18 of FIG. 1. The second-level cache 18 receives an address from memory over the shared second-level cache bus 1418. At step 2202, index bits are used as an address to access the second-level cache 18. Control flows to step 2204, where tag bits of the second-level cache line are checked to determine whether the tag bits are equal to the tag part of the address and whether the second-level cache line is valid. If the check is false, control flows to step 2216. If the check is true, control flows to step 2206.

At step 2206, an invalidation signal is sent to the first-level caches 14 that are owners of this second-level cache line. Control flows to step 2208, where the second-level cache 18 waits for a first-level cache response signal acknowledgement over the shared second-level cache bus 5. After the acknowledgement is received, control flows to step 2210.

Step 2210 determines whether this second-level cache line is "dirty." If this second-level cache line is "dirty," control flows to step 2212. At step 2212, the "dirty" line is written back to memory and control flows to step 2214. If the second-level cache line is not "dirty," control flows to step 2214.

At step 2214, the state of the second-level cache 18 is changed and the valid bit is cleared. Control then flows to step 2216.

At step 2216, an acknowledgement signal is sent back to memory to indicate that the second-level cache 18 has finished the invalidation operation.

FIG. 3C is a flowchart illustrating the processing steps for a downgrade operation in a second-level cache 18 of FIG. 1. A downgrade operation changes a second-level cache line to shared mode. An address is received from memory over the shared second-level cache bus 5. At step 2302, index bits are used as an address to access the second-level cache 18. Control flows to step 2304, where a check is

performed to determine whether tag bits of the second-level cache line are equal to the tag part of the address and whether the second-level cache line is valid. If the check is false, control flows to step 2316. If the check is true, control flows to step 2306.

At step 2306, a downgrade signal is sent to the first-level caches 14 that are owners of this second-level cache line. Control then flows to step 2308, where the second-level cache 18 waits for the first-level cache 14 to respond with a signal acknowledgement. After the signal acknowledgment response is received, control flows to step 2310.

Step 2310 checks whether the second-level cache line is "dirty." If the second-level cache line is not "dirty," control flows to step 2314. If the second-level cache line is "dirty," control flows to step 2312. At step 2312, the "dirty" line is written back to memory and the "dirty" bit is cleared. Control then flows to step 2314.

At step 2314, the state of the second-level cache line is changed to shared mode. In addition, the exclusive and private mode bits are cleared. Control then flows to step 2316.

At step 2316, an acknowledgement signal is sent back to memory to indicate the completion of the downgrade operation.

FIG. 4 is a high-level block diagram of a preferred embodiment of a processor chip 10, focusing on the second-level cache controller 180. The processor core 12 is connected to a private first-level cache controller 140 by a processor bus 1214. The private first-level cache controller 140 is connected to the private first-level cache memory 148 by a first-level cache memory bus 1414. The private first-level cache controller 140 is also connected to the local second-level cache controller 180 by a local input bus 1418_{IN} and a local output bus 1418_{OUT}. The local second-level cache controller 180 is also connected to the shared second-level cache bus 5 by the shared bus port 17 and the respective second-level cache memory 188 by the cache memory port 19. The local second-level cache controller 180 has a set of input request slots 184 that hold requests to write data, read data, and invalidate data

in the second-level cache memory 188. FIG. 4 illustrates five entries 184-1, 184-2, 184-3, 184-4, and 184-5 in the input request slots 184, with one request slot 184 assigned to each processor 12 and private first-level cache 14 pair and the associated local second-level cache controller 180, and one for the third-level. These slots are processed in round-robin order.

When the private first-level cache memory 148 has a miss, a request is queued for the local second-level cache controller 180. If the local second-level cache controller 180 fails to locate the requested cache line, the local second-level cache controller 180 transmits the request on the shared second-level cache bus 5, possibly signals the private first-level cache controller 140 to wait, and continues with the next request. Each remote second-level cache controller 180 receives the request from the shared second-level cache bus 5, as does the third-level interface bus controller 30. The first controller to respond positively cancels the corresponding request at the other controllers. In general, the second-level cache controllers 180 can respond much faster than the third-level interface bus controller 30 (unless there are no other second-level cache controllers 180).

When the local second-level cache controller 180 receives the cache line in response to a cache line request, the local second-level cache controller 180 provides the data to the private first-level cache controller 140 with an indication of the storage site for the data. The storage site of the data is local to the microprocessor 10 (i.e., the respective second-level controller 180) or at one of the remote second-level cache controllers 180. In particular, if the request was satisfied by the third-level interface bus controller 30, the local second-level cache controller indicates that the cache line is local and writes the cache line into the respective second-level cache memory 188 (at least the tags) concurrently with providing the cache line to the private first-level cache controller 140. The private first-level cache controller 140 then adds the cache line to the private first-level cache memory 148 and allows the processor 12 to continue. Otherwise, the storage site is the second-level cache controller 180 that responded to the first-level cache controller 140. A request for exclusive ownership of the cache line is handled similarly.

On write-back, the private first-level cache controller 140 writes the cache line to either the local second-level cache controller 180 or over the shared second-level cache bus 5 to a peer second-level cache controller 180. The choice between writing back to the local second-level cache controller 180 or a peer second-level cache controller 180 depends on the processor tag bits associated with the cache line, which specify the storage site.

The second-level cache controller 180 processes the request slots 184 in round-robin order. A write request causes the data in the request to be written to the respective second-level cache memory 188. The write to the respective second-level cache memory 188 possibly updates second-level cache flags. A read request causes the data to be returned to the requesting unit. The data is returned via the shared bus port 17 if the requesting unit is off-chip, and otherwise via the first-level local input bus 1418_{IN} to the first-level cache controller 140. A read or invalidate request may require that the second-level cache controller 180 invalidate, or force to a non-exclusive state, cache lines in the local or remote private first-level cache memory 148. Local invalidation is handled by signaling the private first-level cache controller 140 over the first-level local input bus 1418_{IN}. Remote invalidation is handled by sending request over the shared second-level cache bus 5.

When a controller generates a controller request, the request is acknowledged by each second-level cache controller 180 that has a corresponding request slot 184 free at the time of the request, indicating the request is stored in the corresponding request slot 184. Otherwise, the issuing controller is signaled to retry the operation. If a second-level cache controller 180 is asked to retry a request, the second-level cache controller 180 signals to the unit whose request is being acted on to retry the request as well, thereby avoiding deadlock. A write request does not have to be retried, so a first-level cache controller 140 can safely continue after having a write request acknowledged as being stored in a request slot 184.

In a preferred embodiment of the invention, access to each second-level cache controller 180 is shared by the local processor 12, the other second-level cache controllers 180, and the third-level interface bus controller 30. A miss at the

first level does not block the local second-level cache controller 180 longer than handling a request to the respective local cache even if this miss has to be handled by a remote second-level cache controller 180 or the third-level interface bus controller 30.

With further advances in Very Large Scale Integration (VLSI) technology, it is expected to be feasible to put multiple processors 12 on a single chip or to use multi-chip modules. With these advances, a preferred embodiment of the invention has the multiple processors on the same chip share the on-chip second-level cache controller 180, assuming the number of processors remains at four or less. It is expected that the demands on integrated circuit real estate for fast floating point, first-level cache, and multiple pipeline units in support of superscalar execution will preclude a larger number of processors per chip in the near future. It is also anticipated that processors may have multiple levels of cache on the chip, but these multiple levels are expected to be sized and used effectively as levels of first-level or private cache, in the terminology used herein. With these advances in VLSI technology, a preferred embodiment of the invention provides multiple second-level cache controllers 180 on the microprocessor chip 10, with the shared second-level cache bus 5 interconnecting the processors 12, which are also contained on the microprocessor chip 10. In that embodiment, the number of processors 12 is expected to be limited by the pin-out of the chip to connect to second-level cache memory 188.

An advantage of the distributed structure is that the distributed structure provides a very fast and simple connection of the microprocessor to the second-level cache memory. In particular, a preferred embodiment of the invention is directed to Rambus^{TM}-like interconnection of the microprocessor to the second-level cache, providing a very fast data access within a small area, with a relatively small pin count. A higher first-level miss rate on data than on software instructions is expected. A higher degree of software sharing than data sharing is also expected. Thus, most first-level misses should go to the local portion of the second-level cache because the cache line placement policy preferably places private software and data

locally. Also, access to shared software and data is expected to be less frequent. Moreover, retrieving shared data from another portion of the second-level cache is substantially faster than accessing the data from the third-level, because of intrinsically longer access latency as well as potentially substantial queuing delay at that level. More generally, the established advantages of shared-caches are retained in distributed shared-caching.

Another advantage of a distributed structure is that distributed structures are compatible with the industry trend to put a second-level cache controller on a microprocessor chip. This trend is motivated by the desire to:

1. minimize board logic for adding a second-level cache;
2. facilitate fast second-level cache access; and
3. utilize available on-chip real estate.

Moreover, the approach of using a cache memory port 19 to interface to second-level cache memory 188 and a shared bus port 17 to interconnect to the rest of the system is attractive even for uniprocessor systems. The shared bus port 17 provides a route to the third-level and I/O, which is required even for uniprocessors. In fact, providing a consistency protocol on the shared second-level port supports cache invalidations and updates arising from I/O in a uniprocessor, further justifying the functionality of preferred embodiments of the invention, independent of multiprocessor requirements.

In addition, preferred embodiments of the invention provide a higher degree of concurrency in the cache controller than is feasible with one shared-cache controller. In particular, a processor is only blocked significantly if another processor is accessing the same one (out of four) second-level cache controller 180 that the first processor is needing to access. Thus, the probability of two processors interfering on a second-level cache load is lowered by a factor of four, with four processors. A plurality of cooperating cache controllers allows the cache associativity to increase with increasing processors sharing the cache.

In contrast to a snooping cache, a cache fill in the first level does not generate a copy of the data in the local second-level portion if the data is retrieved

from another processor's second-level portion, thereby better utilizing the cache space. On write-back from the first level, the cache line is written back to the owner of the cache line, which may be a portion of the second level cache belonging to another processor. In addition, broadcasting is only used to locate a missing second-level cache line among the other second-level peers on the same shared second-level cache bus. Broadcast is not used on write-back, which is the most difficult case to handle in snooping cache schemes.

Third-Level Distributed Shared Caching

In a preferred embodiment of the invention, third-level caching is the DRAM memory pool that serves as backing storage and as a staging area for the second level caches. In a preferred embodiment of the invention, a system has a substantial amount of second-level cache in the range of about 8 megabytes or more per processor cluster. The DRAM memory pool is typically in the hundreds of megabytes.

FIG. 5 is a block diagram illustrating a third-level distributed shared-cache memory system. Each third-level memory module 40 contains a third-level directory cache 42, a memory controller 44 and a third-level memory area 46. The directory cache 42 contains entries describing the cache lines present in the second-level caches 18A,...,18M. The third-level directory entry contains various consistency flags per physical address; similar to the second-level cache directory entries, except that M identification bits refer to second-level or multiprocessor modules 1, not individual processors 12. The third-level directory cache 42 holds cache information about the state of cache lines from page descriptors. Each third-level cache directory entry contains an address for the cache line at the third-level. The memory controller 44 manages access to the cache directory 42 and the third-level memory area 46. The third-level memory area 46 contains page descriptors, data pages, and software-maintained physical-address-to-page-descriptor translation tables. A page is the unit of mapping to virtual addresses, as in conventional virtual

memory systems. There is a page descriptor for each page represented by at least one cache line in an second-level cache.

In a preferred embodiment of the invention, the interconnection between a second-level and the third-level is a fiber optic connection between the multiprocessor modules 1A,...,1M and the third-level modules 40a...,40m. Preferably, each third-level module 40 has a four or eight port connector, allowing up to four and eight multiprocessor modules 1 and third-level modules 40, respectively.

In another preferred embodiment, the interconnection is a shared bus between third-level modules 40. The shared bus can also be used to interconnect the multiprocessor modules 1A,...,1M to the third-level modules 40a,...,40m. In that case, each multiprocessor module 1 is coupled to one third-level module 40, although multiple multiprocessor modules 1 can share a single third-level module 40. FIG. 6 is a block diagram of a third-level bus-based distributed shared-cache memory system.

On a second-level cache miss going to the third-level, as previously described, the third-level interface first checks with the third-level directory 42 to see if the cache line is present in another second-level cache 18 in a different cluster. If so, the request is directed to a processor cluster containing the cache line, which returns the data to satisfy the cache miss. The necessary consistency actions, such as transferring ownership, are performed at the same time. If the third-level directory cache 42 indicates the data is in the third-level memory area 46, the data is transferred from the third-level memory area 46, with cache tags in the cache directory entry being updated accordingly.

If the requested cache line is not present in the third-level directory cache 42, a software trap is taken by the processor 12 and a software look-up is performed using the translation table, mapping the physical address to the page descriptor. If the page descriptor is found, then the page descriptor describes the location of the data in DRAM memory. This software translation table is global across the third-level modules 40a,...,40m, even though FIGs. 5 and 6 illustrate separate portions of

third-level memory 46 per third-level module 40. If the data is retrieved from memory, the page descriptor is updated according to the consistency protocol. Also, a cache line entry is created in the third-level directory cache 42 for this cache line, possibly writing back an entry from this directory cache 42 to the associated page descriptor to make room for this new entry. The processor 12 then restarts the memory operation that caused the miss.

The look-up may determine that required data is in another of the third-level modules 40 from that associated with the faulting multiprocessor module 1. In that case, the third-level directory cache 42 in that third-level module 40 is loaded accordingly.

If the data is not present at the third-level at all, software allocates memory in the third-level module 40 associated with the faulting multiprocessor module 1 and transfers the data into this area of memory from a high-speed network connection, and then performs the actions described above. Thus, the software mimics the shared operation of the second level, where cache loads go to the portion of the cache of the requesting unit. Because the transfer from the network to the third level is done in software and it is advantageous to move this data immediately into the highest portion of the requesting units memory hierarchy, the network connection resides on the shared second-level cache bus 5. By placing the network connection on the shared second-level cache bus 5, the processor 12 is allowed to copy the data directly from the network interface to cache lines residing at the second-level.

There is a special form of a software page descriptor, referred to as a pageless page descriptor, which does not have a corresponding data page portion in DRAM memory. The page pool can convert a page descriptor into a pageless page descriptor and reclaim the associated data page area when the data of the page is contained in the second-level cache area 20. A scavenger process, similar to that used in conventional virtual memory systems, checks whether there have been any recent accesses to the data page and, if not, reclaims the data page and updates the directory cache 42 as appropriate. A processor may have to handle a write-back

fault, which arises when a write-back occurs for a cache line having no third-level location recorded in the third-level directory cache 42.

The third-level memory area 46 is directly addressable as a portion of the physical address space. A reference to the third-level memory area 46 is detected at the point of trap to a software process after a directory cache miss. A direct mapping to the memory for the cache line then takes place. Otherwise, access to this third-level memory area 46 is transparent to the processor 12 and handled in the directory and consistency mechanism, the same as other portions of memory. Each multiprocessor module 1 contains a local memory containing the software instructions for third-level miss handling, so that a processor 12 never faults at the third-level on software instructions to handle third-level faults. Alternatively, software instructions to handle this miss handling can be provided in a memory module at the third level.

The directory cache 42 allows most third-level misses to be handled fast, by retrieving the data either from another processor cluster or from the third-level memory area 46. By making the directory a cache, the cache directory 42 can support a very large physical address space without actually having that amount of physical memory. For example, a machine with less than a few hundred megabytes of data can support a physical address space addressed by 64 bits. Providing a very large physical or shared address space allows the operating system to maximize the time between reuse of addresses. By maximizing the time between reuse of addresses, these addresses are meaningful over a longer period of time and confusion that can arise from reusing addresses quickly is minimized. Providing a directory large enough to hold entries covering the entire 64-bit physical address space would be expensive.

In addition, preferred embodiments of the invention efficiently allow software control of the memory system at the cache line unit level. In particular, if a cache line of a page is not present locally, the software gains control and can retrieve the required cache line, even if the rest of the page is present locally.

Furthermore, the software-managed page descriptor structure allows a more sophisticated management approach than is feasible with a pure hardware solution. In particular, a data page for every page descriptor is not required in preferred embodiments of the invention. Data pages can be used more for a prefetching and write-behind I/O storage, rather than primarily duplicating the second-level cache data entirely. Also, page descriptors can be used to refer to remote copies of the page. In addition, the global page descriptor data structure is used to ensure that there is only one copy of a page across all local third-level modules 40.

In another preferred embodiment of the invention, each third-level module 40 handles only one subset of the total processors in the node. For example, a directory cache entry may allow for only eight multiprocessor modules 1, by allowing eight M bits for multiprocessor module identification. However, the saving in M bits is not regarded as a significant benefit as a percentage of third-level directory cache entry size, and transferring ownership of cache lines between the third-level appears complicated.

In an alternative embodiment of the invention, the third-level modules 40 are divided based on an address range. While requiring no interaction between third-level modules 40, this embodiment leads to very uneven loads between modules depending on the distribution of the load on memory. This embodiment also requires every third-level module 40 to contain the maximum number of M bits per directory entry that any machine configuration would ever require.

In another preferred embodiment of the invention, the third-level modules 40 are divided based on the low-order bits of the page numbers. For example, with two third-level modules 40, the first module might handle all page addresses that are less than 4 modulo 8 whereas the other module handles all that are greater than or equal to 4 modulo 8. This embodiment distributes the load more evenly, but is more complicated and still limits the number of multiprocessor modules 1 in a configuration to the number of M bits in a third-level module 40.

Page-Level Shared Cache System

FIG. 7 is a block diagram of a preferred distributed shared page cache memory system. A page cache is memory managed by the virtual memory system, largely in software but with some hardware assistance for address translation. In a preferred embodiment of the invention, the machine environment is a cluster of nodes 110a,...,110k connected by a high-speed network 100. Each node has a respective local DRAM memory module of several hundreds of megabytes or more. Pages are retrieved from a shared distributed file system, which stores pages on disk 200 connected to the high-speed network 100.

Each node 110 maintains a page frame pool 116 similar to that in conventional virtual memory systems. Each page is identified as a portion of a segment. Preferably, the pages are identified as in the V++ distributed system virtual memory system. This identification is global and uniquely identifying across the set of nodes sharing the file system. Each page frame pool 116 also maintains a list of pages resident at the other nodes in the network. Each such page is recorded as a special form of a standard page descriptor, referred to as a remote page descriptor. The page descriptors contain consistency information as required to support the third-level operation previously described, as well as for internode operation. The page descriptors are a natural extension of this level, and they are similar to that used in various distributed shared memory implementations.

Each page frame pool manager periodically multicasts the list of resident pages in the corresponding pool to the other nodes 110. On receiving such an update, a node 110 updates the list of pages at the sending node. The update can include specific mention of pages that have been discarded. In an alternative preferred embodiment, the page frame pool manager discards remote page descriptors after some time-out period without reconfirmation, or after requesting the page from a node 110 and discovering that the requested page is no longer present in that node.

On page look-up, the local process either finds a local page descriptor, a remote page descriptor, or no page descriptor. In the former case, the page data

may be local and is satisfied immediately. However, because the cache line is the unit of consistency, that portion may have to be accessed from a remote node, which is described in a remote page descriptor 126. If a remote page descriptor 126 is found, the process requests the data from the node 110 identified by the remote page descriptor 126. Otherwise, the page is requested from an appropriate disk subsystem 200 or file server 150, incurring the multi-millisecond cost of accessing the disk subsystems 200.

It is possible that the remote node from which the page is requested in the second case above has discarded the page. In that case, the remote node responds negatively and the requesting node can remove the remote page descriptor 126 and then retry the operation.

The file server or disk server can have some page buffering and advertise the contents of associated caches. However, in a loaded system better performance can be obtained by limiting traffic to the file server 150 primarily to request requiring disk I/O. By so limiting the traffic, page requests that can be satisfied from other nodes do not unnecessarily interfere with page requests that have to be handled by the server 150.

A page requestor may request a page from the file server 150 even though the page is present in another node's page frame pool 116, because the presence of that page has not been propagated from the remote node to the requesting node. With a consistent file page caching protocol in place, it is necessary for the file server 150 to forward the request to a node with exclusive ownership of the cache line or the page. This type of redirect is feasible to handle in software and should not represent a performance problem provided that the redirection does not occur frequently.

A preferred embodiment of a page-level shared-cache system provides the effect of having the totality of the page frame pools 116 forming a large shared-cache, which is much larger than the memory available on each node. In general, a page-level shared-cache system provides the advantages of the shared-cache among a

set of network nodes, assuming that the network transfer time is substantially less than the disk transfer time, including latency.

An advantage of a preferred embodiment of a page-level shared-cache system is that the system allows pages to be duplicated at different nodes. The system recognizes that even the network latency is significant for repeated second-level cache misses to the same page.

In a preferred embodiment of the invention, the page fault load is distributed from the file servers to other nodes. This distribution is important with point-to-point networks that appear to be a key aspect of future high-speed networking. Rather than have all page requests congest the path from client nodes to server nodes, some of the requests are routed to clients, making better use of network resources and reducing congestion-induced delay. In a particular preferred embodiment of the invention, the requesting node takes the location and route to the node described by the remote page descriptor into consideration in selecting the node from which to request when there are multiple nodes holding a page.

In addition, the dissemination of page frame pool 116 contents by best-efforts multicast avoids extra network loading, processing overhead, and extra latency in case of total miss of trying to apply the broadcast approach of the second-level cache to locating a page. With point-to-point networking, transmitting a multicast request to every node as a result of a page fault can be a significant load during phases of significant page faulting. Because the page frame pool 116 is handled in software, each node incurs the cost of processing such a request. Moreover, because a page can be duplicated at several nodes, a multicast request can result in duplicate responses. In the case of a node requesting a page not present in any page frame pool 116, the request must time-out before directing a request to the file server. Because time-outs with network and software processes have to be large to avoid unnecessary re-transmission, such a time-out adds significant delay to a page fault that requires going to disk.

A preferred embodiment of a page-level shared-cache system also fits with the provision of extra page descriptors that have no corresponding data page, as introduced above with the third-level caching support.

Memory-based Messaging to Support Networking and Devices

Parallel applications that use shared memory can benefit from high-performance communication. For example, if a program is structured such that processes allocate work from a shared work queue, it is more efficient in memory traffic for the processors to communicate by messages with a processor running this allocator than trapping the code and data associated with the allocator into its cache and then executing the allocator itself, assuming that this processor is unlikely to have been the last to execute this allocator. That is, function shipping is intrinsically more efficient than data shipping in this case. Fast communication allows an application to effectively optimize for these situations rather than incur the delay and memory overload of conventional shared memory techniques.

Large-scale memory systems can reduce communication performance because the cost of copying data in these machines increases, due to generally poor cache behavior of copying and the increasing ratio of processor speed to average memory access. Furthermore, maintaining shared-memory consistency semantics in large-scale memory systems is increasingly costly due to increased memory latency. The cost of remapping data in multiprocessor systems also appears to be significantly greater than in uniprocessors because of the need to update or invalidate the TLBs, or page tables, in all processors. Moreover, the costs of queuing and notification become more significant in large-scale parallel machines. Consequently, memory-based messaging techniques in these machines cause large amounts of memory contention on highly shared data structures, such as shared messages and message queues.

FIG. 8 is a block diagram of a multiprocessor module interfaced to an external device controller according to a preferred embodiment of the invention. A

distributed shared cache can support high-performance device communication by allowing a device controller 160 to participate in the inter-cache protocol as a peer to the other local cache controllers 180 associated with data processors 12. Particular examples of appropriate device controllers 160 include, but are not limited to, high-performance disk controllers for interfacing the processor cluster to a disk subsystem 200' and high-speed network interfaces for interfacing the processor cluster to other processor clusters 100' or a switch (not shown). It should be understood that other device controller can be used to interface with other computer peripherals.

As illustrated, the microprocessor module 1 includes an external interface port 165 that provides the device controller 160 with access to the second-level cache memory 188 via the shared bus 5. In particular, on read access to data at a given location, a device controller 160 requests the data from the peer-level local cache controllers 180 the same as if the device controller were a cache controller whose associated processor 12 had missed an access to this data. On write to data, the device can similarly request ownership of the data from the peer-level cache controllers and update the data accordingly, directly updating the next level of memory if the data is not present in the peer-level cache memories.

In a particular embodiment, the distributed shared caches support a memory-based messaging mode for cache line units. In particular, each cache line can be set in message mode as an alternative to one of the standard consistent shared memory modes of invalid, shared, or exclusively owned. A particular memory-based messaging technique is described in the above-referenced ParaDiGM architecture, which uses cache lines of shared memory as message buffers.

In a preferred embodiment, a memory segment is created to act as a communication channel, processes bind this segment into their respective address spaces, and messages are copied into and out of this segment to effect communication. A preferred memory-based message model is optimized over prior art models by providing address-valued signals, message-oriented memory consistency, and automatic signal-on-write. These refinements represent a

significant simplification of the kernel interface over that provided by other systems for memory-based messaging. They also allow for a significantly faster and simpler implementation, especially for scalable shared memory multiprocessor architectures.

FIG. 9 is a block diagram illustrating a preferred address-valued signal mechanism, which is a notification that is optimized for memory-based messaging. With address-valued signals, a process sends a signal S_x with a single virtual address parameter. The virtual address is mapped from a shared message region 113x (i.e., a range of virtual address space) in the address space 112x containing the virtual address to a shared segment 115, and the signal S_x is delivered to all processes that bind in that shared segment 115. The signal S_x is delivered as a call to a signal handler with a single parameter, which is the virtual address of the signaled area 113y,113z in the recipient address space 112y,112z.

A process sending a message writes the message data into a free area of the message region associated with the destination process(es) and then signals using the virtual address of this free area. The system delivers the signal to each recipient process, calling the process signal handler with the virtual address of the new message in the process address space. The signal handler uses this address to access the new message and deliver the message to the application.

Hardware provides a per-processor First-In-First-Out (FIFO) buffer in which memory addresses representing signals delivered to this processor (but not yet processed) are stored. The signaled processors 12x,12y are determined from the physical address of the signal S_x (translated from the virtual address from the sender) combined with the cache directory entry associated with the cache line specified by the physical address. In particular, each cache line has a cache directory entry containing a 3-bit mode field and various other tag bits describing the state of the cache line. The FIFO buffer eliminates the need for software delivery of the address values and synchronized software queues to hold those addresses.

The mode field encodes the conventional shared, private and invalid states of an ownership-based consistency protocol as well as a special message mode. Tag

bits include a set of processor identification bits, one per processor sharing the cache, that normally indicate which processors have copies of the cache line. In message mode, the processor identification bits indicate the processors to be signaled for this cache line. The memory system architecture uses a hierarchical cache structure, with a "global" bit to indicate notification to the next lower level of the memory hierarchy. The lower level maintains its own cache directory and further propagates the signal to other clusters of processors, as indicated by the cache tags. The tag bits are stored in software-maintained page frame descriptors and fetched when the memory is cached.

When a process writes a message cache line, hardware generates a signal at that virtual address (referred to as automatic signal-on-write) and maps the referenced virtual address to a physical address, using a standard virtual-to-physical address translation mechanism. The physical address is then mapped in hardware to a cache directory entry. A cache controller generates signals to all processors indicated by the cache tags as recipients, possibly including the next lower level cache (such as the third level cache), which then propagates the signal further as appropriate. The signal is transmitted over each bus as a special bus transaction specifying the address and control lines indicating the affected processors. With direct cache-to-cache transfer, the cache line can be transmitted as part of the same bus transaction.

When it receives a signal bus transaction, each processor's bus interface stores the address in the processor's FIFO buffer and interrupts the processor. When a processor receives the signal interrupt, the processor takes the next physical address from the FIFO buffer, maps the physical address to each virtual address and delivers the signal to each process associated with this signal area that is also assigned to this processor. In delivery of the signal, the kernel saves the process context and creates a stack frame to call the signal handler associated with the signaled memory region, passing the translated signal address as the parameter. On return from the signal handler, the process resumes without reentering the kernel.

On systems without hardware support for automatic signal-on-write, a signal can be generated by a kernel operation.

Address-valued signaling provides a notification mechanism for memory-based messaging that is simple, yet efficient. The translated signal address provides a direct, immediate and asynchronous specification to the recipient(s) of the message, allowing each recipient to immediately locate the message within the segment. In particular, the same signal handler procedure can be used for several different segments and still immediately locate the signaled message using the supplied virtual address. Furthermore, different signal handlers can be bound to different memory regions. The particular signal handler is selected automatically based on the region of memory in which the signal occurs.

The additional hardware to support address-valued signaling is a small percentage of the overall hardware cost, and arguably close to zero cost in large configurations. In particular, a FIFO buffer is required for interprocessor and device interrupts on large-scale systems because the conventional approach of having a dedicated bus line from interrupter to interruptee is not feasible. The FIFO buffer stores the interrupt, allowing the sending processor or device to send the interrupt across the interconnection network and not hold a connection. Storing a single address, rather than the entire message, is essentially the same cost as storing a potentially smaller value such as processor identifier or device identifier. In a preferred embodiment, all device interrupts in the system are handled as address-valued signals, simplifying the hardware and operating system software.

Message-oriented memory consistency is a consistency mode for a segment of memory in which the reader of the segment is only guaranteed to see the last write to this memory after the reader has received an address-valued signal on this segment. In fact, a message and signal can be lost and the receiver is not guaranteed to see the update at all. A process can only expect a new packet to be available after an interrupt from the interface, not at the time the packet is written. Also, if a packet is not received, or is received in a corrupted form, the data is not available at all.

Memory segments with message-oriented memory consistency are preferably used in a unidirectional fashion as part of memory-based messaging. One process binds the segment as writable and others bind it as read-only. Consequently, there is generally a single writer for a set of addresses within the segment. However, a shared channel may have multiple writers, similar to a citizen-band type radio channel. For example, clients can use a well-known channel to multicast to locate a server.

The message-oriented consistency uses an additional mode for each cache line, as described above. Because there are extra code values available in a particular preferred embodiment beyond those used by the conventional shared memory states, the additional message mode does not increase the cost per cache directory entry. The worst-case would be the addition of an extra bit per cache directory entry, still a small percentage space overhead.

In a preferred embodiment of the invention, there is logic in the cache controller to handle message mode. This logic is relatively simple because message mode modifies actions already performed by the cache controller to comply with the shared memory consistency protocol, and does not require new types of actions. In particular, message mode requires the cache controller to generate an invalidation to each recipient processor after the cache line has been written, rather than before as with consistent shared memory. In a particular preferred embodiment, it is even faster (and simpler) to have the cache controller perform no invalidations, leaving the signaled processors to invalidate their respective caches.

By supporting direct cache-to-cache transfers in hardware, the source processor's cache can transfer the cache line to the recipient processors' caches, providing data transfer and notification in a single bus transaction. This avoids the cache line transfer that normally follows the invalidation after signal reception.

Message-oriented memory consistency reduces the number of bus transactions required to send a message compared with conventional memory consistency. FIGs. 10A-10B are block diagrams illustrating that message-oriented memory consistency reduces the receiver's message invalidation (61), the signaling interrupt (63), the

message transfer (64) and corresponding acknowledgements (62,65) into a single message delivery transaction (69). Rather than having to invalidate (61) the receivers' copies of each cache line being written by the sender, as in conventional ownership protocols (FIG. 10A), message-oriented memory consistency allows the memory system to simply transmit the update (69) when the message unit has been written (FIG. 10B). Transmitting the update (69) at this time also allows the signal notification to be piggybacked on the data transfer, or vice versa, rather than requiring a separate bus transaction for notification. Thus, the update traffic matches in behavior and efficiency that of a specialized communication facility. In contrast to write-update consistency protocols, message-oriented memory consistency allows updates after a full cache line, page or segment (depending on the consistency unit) rather than on each word write. In that vein, message-oriented memory consistency on single cache-line messages allows the sender to ensure that the receiver receives either all or none of the message, rather than receiving word-level updates. This property is exploited in a preferred embodiment of higher-level protocols.

The message-oriented memory consistency semantics also provides a simple network embodiment of a shared channel segment between two or more network nodes. In particular, an update generated at one node can be transmitted as a datagram to the set of nodes that also bind the affected shared segment. The best-efforts, but unreliable, update semantics of message-oriented consistency obviates the complexity of handling retransmission, timeout and error reporting at the memory level. A large-scale multiprocessor configuration can similarly afford to discard such bus and interconnection network traffic under load or error conditions without violating the consistency semantics.

Finally, message-oriented consistency minimizes the interference between the source and destination processors. The sending processor is not delayed to gain exclusive ownership of the cache line and the reading processors are not delayed by cache line flushes to provide consistent data. In contrast to other relaxed memory consistency models such as Stanford's DASH release consistency, message-oriented

consistency reduces the base number of bus transactions and invalidations required, rather than just reordering them or allowing them to execute asynchronously.

The automatic signaling on write is on a per-cache line basis (or other consistency unit) so that writing the last byte of such a cache line causes the hardware to generate a signal without any further action by the writing processor. It is understood that the hardware can signal on any other portion of the cache line, but in a particular preferred embodiment the last byte is the easiest convention to use.

The cache controller monitors each write operation to the cache. If a write operation is to a cache line in message mode and the address is the last address of the cache line, the cache controller generates a signal bus transaction after allowing the write to complete.

In a preferred embodiment, the on-chip, or private, cache 14 (FIG. 1) does not support this logic. Consequently, the sender writes through the private cache 14 so that the second level cache controller 180 can detect last-byte writes, and generate the signal. Ideally, the first-level cache controller 140 (FIG. 4) supports this mechanism. This support adds very little complexity to an on-chip cache controller 140.

Automatic signal-on-write reduces the processing overhead on a sending processor when no process on this processor will receive the signal. Reduced sender overhead, in turn, shortens the latency of the message. In the absence of this facility, the sending process must explicitly execute a kernel call, map the signal virtual address to a physical address and then presumably access hardware facilities to generate the signal, or at least an interprocessor interrupt of some form. Providing kernel-level access to the cache controller to explicitly generate a signal is more expensive in hardware than providing the logic in the cache controller to detect and act on writes to the end of a cache line in message mode.

In automatic signal-on-write, the channel segment specifies the unit of signaling, transparent to the sending process. For example, a receiver can pass the sender a channel which signals on every cache line or every page, depending on the

receiver's choice. Without automatic signal-on-write, the sender needs to explicitly signal and thus needs to be coded explicitly for each form of behavior, perhaps switching between different behaviors based on the type of channel.

In message mode, a holding memory for the cache line is notified of the update after the cache line has been written, rather than before as occurs with conventional consistent shared memory. When a cache line in message mode is written, the cache line is transferred directly to the set of cache controllers that have indicated an interest in this cache line using the cache directory tags normally used to indicate those memories holding a shared copy when the cache line is in shared mode. These tags are cached in the cache partition when the cache line is loaded into the local cache partition. The cache line can be invalidated in all caches when these tags are modified, thereby preserving consistency of the tags.

This transfer uses the same direct cache-to-cache transfer mechanism that is used to writeback a cache line in exclusive mode from one cache partition to the peer-level owning cache partition. In the message-oriented consistency mode, the writing process overwrites the cache line and transmits the cache line to these other caches without first acquiring exclusive ownership of the cache line. In contrast, conventional shared memory consistency mechanisms require the writer of a cache line to first acquire exclusive ownership of the cache line, invalidating all copies of the cache line that might occur in other caches.

Returning to FIG. 8, a device controller 160 participates in this protocol as follows. When data is to be written from the device to an address in memory that is set in message-oriented consistency mode, the data is transferred directly to the peer-level cache controllers 180 and other device controllers according to the cache tag bits described above. In a particular preferred embodiment, a direct transfer only occurs if the transfer is to a single other controller, and otherwise, the data is transferred to a next-level memory area and the cache line is invalidated in the peer-level caches, notifying them of the revised cache line. A device controller 160 loads the tags to the affected area of memory as part of establishing access to this memory area, either at the time a mapping of device data to this memory is setup or

at the point the data is to be written to the area of memory. A device controller also participates by receiving direct transfers of cache lines in message mode from other cache controllers and device controllers, as occurs on processor and device writing to these cache lines.

Equivalents

Those skilled in art will know, or be able to ascertain using no more than routine experimentation, many equivalents to the specific embodiments of the invention described herein.

These and all other equivalents are intended to be encompassed by the following claims.

CLAIMS

The invention claimed is:

1. In a computer system having a plurality of data accessors, a cache memory for providing stored cache lines to be accessed by the data accessors, the cache memory comprising:

a plurality of cache memory areas at a common level of cache memory, each cache memory area locally coupled to a respective data accessor, the cache memory areas coupled together to form a cache memory cluster; and

a plurality of local cache controllers, each local cache controller coupled between a respective data accessor and a respective locally-coupled cache memory area to request a cache line on behalf of the data accessor from the respective locally-coupled cache memory area and from remaining, remotely-coupled cache memory areas of the cache memory cluster in response to an unsatisfied cache line request at the locally-coupled cache memory area.

2. The cache memory according to Claim 1 further comprising a private cache coupled between each data accessor and the respective local cache controller, each data accessor having exclusive access to the respective private cache, and wherein the private cache requests a cache line from the local cache controller only upon the cache line request being unsatisfied by the private cache.

3. The cache memory according to Claim 1 wherein the second plurality is less than the first plurality.

4. The cache memory according to Claim 1 wherein the cache memory areas are coupled by a shared cache interconnect.

5. The cache memory according to Claim 1 wherein there are less than or equal to four cache memory areas in the cache memory cluster.

6. The cache memory according to Claim 1 further comprising an external data accessor coupled to the cache memory cluster for accessing a cache line, each cache memory area in the cache memory cluster is remotely coupled to the external data accessor.

7. The cache memory according to Claim 1 wherein data has an associated ownership in a holder memory area, the satisfaction of a memory reference from data in the holder memory area does not affect the ownership of the data in the holder memory area relative to more distant memory.

8. The cache memory according to Claim 1 wherein the cache line is accessed for message data between data accessors.

9. A distributed shared cache computing system having a plurality of data accessors for referencing data in memory, each memory reference being satisfied from a hierarchical memory structure having a plurality of memory levels relative to the data accessors, comprising:

a first memory level of private cache memory in the hierarchical memory structure for satisfying memory references at the first memory level, each private cache memory providing exclusive access to a respective data accessor; and

a second memory level of at least one cluster of shared cache memory in the hierarchical memory structure for satisfying memory references at the second memory level, each cluster of shared cache memory being coupled to a respective group of data accessors and distributed such that each area of the cluster of shared cache memory is a local cache memory to a respective data accessor from the group of data accessors and a remote cache memory to the remaining at least one data accessor from the group of data accessors.

10. The distributed shared cache computing system according to Claim 9 wherein there are four areas per cluster of shared cache memory and four data accessor per group of data accessors.

11. The distributed shared cache computing system according to Claim 9 wherein each data accessor is coupled to the remote cache memory such that a cache miss for a memory reference request to the local cache memory is re-requested to the remote cache memory.

12. The distributed shared cache computing system according to Claim 11 wherein each data accessor is further coupled to at least one lower memory level, such that a cache miss for a memory reference to the respective cluster of shared cache memory is satisfied by a more distant memory unit.

13. The distributed shared cache computing system according to Claim 9 wherein the data has an associated ownership in a holder memory area, the satisfaction of a memory reference from data in the holder memory area does not affect the ownership of the data in the holder memory area relative to more distant memory.

14. The distributed shared cache computing system according to Claim 9 wherein the data is message data between data accessors.

15. The distributed shared cache computing system according to Claim 9 further comprising an external data accessor coupled to a cluster of shared cache memory in the second memory level for accessing a cache line, each area of the cluster is remotely coupled to the external data accessor.

16. A computing system having a plurality of data accessors and a multiple level hierarchical memory structure, comprising:

a plurality of first cache memory areas, each first cache memory area coupled to a respective data accessor for providing a private memory relative to the respective data accessor;

a plurality of second cache memory areas, each second cache memory area coupled to a respective private memory level, each second cache memory area providing a local cache memory relative to the respective data accessor, a local cache memory being searched to satisfy a first cache miss in the respective private memory; and

at least one shared cache interconnect to couple at least one cluster of second cache memory areas together such that a second cache miss in a local cache memory causes a search at the remaining second cache memory areas from the cluster of second cache memory areas to satisfy the second cache miss.

17. The computing system according to Claim 16 further comprising a plurality of cache controller, each cache controller coupling a respective second cache memory area to the respective private memory and the shared cache interconnect.

18. The computing system according to Claim 16 wherein each cluster of second cache memory areas is a shared cache memory, each second cache memory area in the cluster of second cache memory areas being an equal subset of the shared cache memory.

19. The computing system according to Claim 16 wherein there are less than five second cache memory areas in each cluster of second cache memory areas.

20. The computing system according to Claim 19 wherein there are four second cache memory areas in each cluster of second cache memory areas.

21. The computing system according to Claim 16 further comprising an external data accessor coupled to a cluster of second memory areas to search the second cache memory areas to satisfy a memory reference by the external data accessor.

22. The computing system according to Claim 21 wherein the external data accessor includes a device controller.

23. The computing system according to Claim 16 further comprising an interface coupled to each shared cache interconnect for coupling each respective cluster of second cache memory areas to a respective main memory area, the main memory area being searched to satisfy a third cache miss from the cluster of second cache memory areas.

24. The computing system according to Claim 23 wherein each cluster of second cache memory areas and the respective coupled data accessors, first cache memory areas, and interface are integrated as a multiprocessor module.

25. The computing system according to Claim 23 wherein there are a plurality of main memory areas, each main memory area coupled to at least one other main memory area to form a cluster of main memory areas, each main memory area serving as a local main memory to the respective cluster of second cache memory areas and as a remote main memory area to the at least one cluster of second cache memory areas coupled to respective remaining main memory areas of the plurality of main memory areas.

26. The computing system according to Claim 25 wherein the cluster of main memory areas are coupled by a fiber optic coupling system.

27. The computing system according to Claim 25 wherein the cluster of main memory areas are coupled by a shared bus.

28. The computing system according to Claim 16 wherein the shared cache interconnect is a shared bus.

29. The computing system according to Claim 16 wherein the shared cache interconnect is an interconnection network.

30. The computing system according to Claim 16 wherein the cache memory is accessed for message data between data accessors.

31. An integrated circuit for a multiple level hierarchical memory computing system, comprising:

a data processor for requiring memory references;

a first cache memory for providing a private memory level relative to the data processor for storing a copy of data to satisfy memory references at the first cache memory; and

a local cache controller coupled to the private memory level for controlling access to a respective local second cache memory and to a cluster of remote second cache memory for requesting memory references from the remote second cache memory.

32. The integrated circuit according to Claim 31 wherein the local cache controller is cooperatively shared with at least one remote data processor, such that the remote data processor can satisfy a memory reference from the respective local second cache memory.

33. The integrated circuit according to Claim 31 further comprising a private cache controller coupled between the data processor, the first cache memory, and the local cache controller for controlling access to the private memory level and requesting memory references from the local cache controller upon a cache miss in the respective first cache memory.

34. In a computer system having a plurality of data accessors, a method of providing stored cache lines from a cache memory to be accessed by the data accessors, comprising the steps of:

distributing the cache memory into a common level of cache memory areas, each cache memory area coupled to a respective data accessor;

forming the cache memory areas into at least one cache memory cluster, the forming steps comprising:

a) associating each cache memory area with a respective data accessor to form a local cache memory area for each data accessor;

b) controlling access to each cache memory area such that a memory reference from a data accessor is requested at the respective local cache memory area and the remaining cache memory areas in the cache memory cluster are searched to satisfy a cache miss in a local cache memory area; and

returning data from memory to the data accessor in satisfaction of the memory reference.

35. The method according to Claim 34 further comprising, before the step of returning, the step of satisfying a cache miss in a cache memory cluster from memory external to the cache memory cluster.

36. The method according to Claim 34 wherein there is an external data accessor not locally coupled to a cache memory area and the step of controlling access includes searching the cache memory areas in the cache memory cluster to satisfy a memory access by the external data accessor.

37. The method according to Claim 34 wherein the cache lines are accessed for message data between data accessors.

38. A method for locating a memory reference on a multiple-level hierarchial memory computing system having at least one data accessor for requesting memory references to be obtained from memory, each memory level having at least one memory area for storing data to satisfy memory references, comprising the steps of:

at a memory level having at least one peer memory area, locating the memory reference comprising the locating steps of:

a) searching a memory area that is locally coupled to the data accessor requiring the memory reference;

b) upon a memory miss at the locally-coupled memory area, searching at least one peer memory area that is remotely-coupled to the data accessor requiring the memory reference; and

returning the located memory reference to the data accessor.

39. The method according to Claim 38 further comprising, before the step of returning, the step of searching more distant memory relative to the data accessor to locate the memory reference at a more distant memory.

40. The method according to Claim 38 wherein there is at least one data accessor not having a locally-coupled memory area in the memory level.

41. The method according to Claim 40 wherein the data accessor not having a locally-coupled memory area includes a device controller.

42. The method according to Claim 38 wherein the referenced memory contains message data between data accessors.

43. A method of communicating between data accessors in a cache memory computing system, comprising the steps of:

providing a cache line in a cache memory area shared between the data accessors;

binding at least one receiver data accessor to receive message data from the cache line;

from a sender data accessor, writing message data to the cache line; and

delivering the message data from the cache line to the receiver data accessor.

44. The method according to Claim 43 wherein the shared cache line is in a multi-level memory hierarchy.

45. The method according to Claim 43 wherein the cache line has a message mode and the step of binding comprises the step of setting tag bits associated with the cache line to indicate the data accessors to receive message data from the cache line.

46. The method according to Claim 43 wherein at least one data accessor includes a device controller.

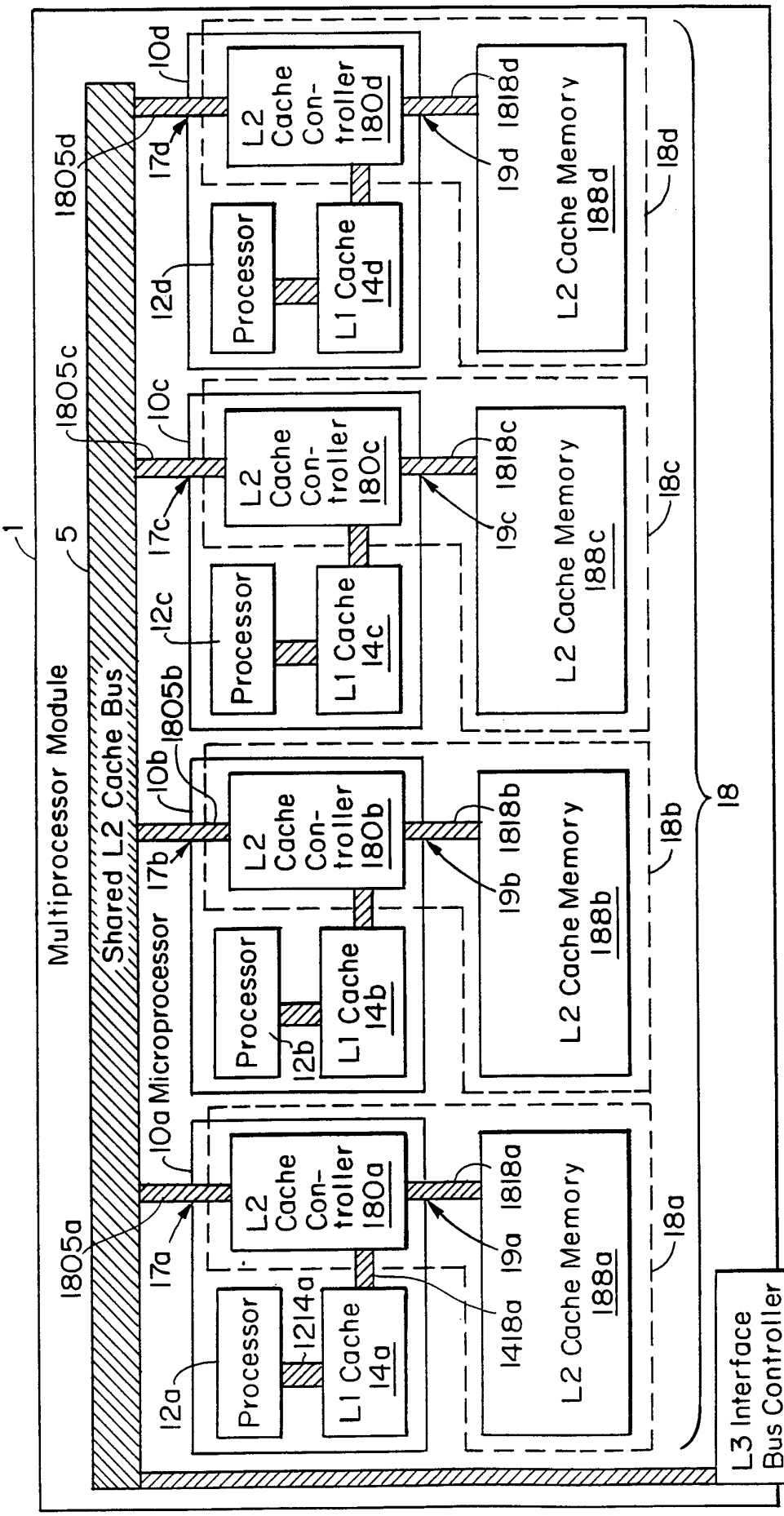

1
Multiprocessor Module
5
Shared L2 Cache Bus
10a Microprocessor
10b
10c
10d
1805a
1805b
1805c
1805d
17a
17b
17c
17d
12a
12b
12c
12d
1214a
14a
14b
14c
14d
L1 Cache
Processor
L2 Cache Controller 180a
L2 Cache Controller 180b
L2 Cache Controller 180c
L2 Cache Controller 180d
1418a 19a 1818a
19b 1818b
19c 1818c
19d 1818d
L2 Cache Memory 188a
L2 Cache Memory 188b
L2 Cache Memory 188c
L2 Cache Memory 188d
18a
18b
18c
18d
18
30
L3 Interface Bus Controller

FIG. 1

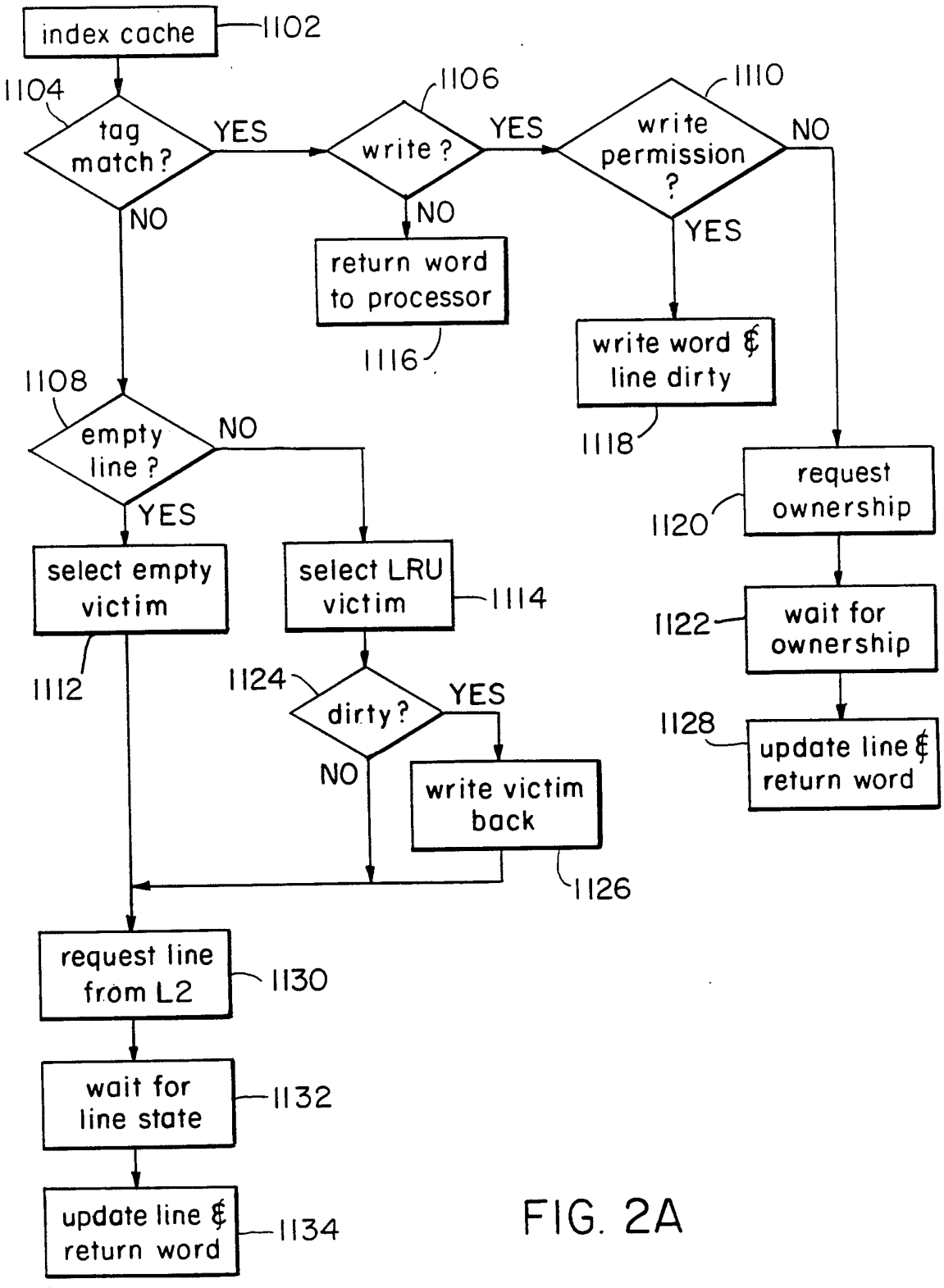

index cache
1102
1104
tag match?
YES
NO
1106
write ?
YES
NO
return word to processor
1116
1110
write permission ?
NO
YES
write word & line dirty
1118
request ownership
1120
wait for ownership
1122
update line & return word
1128
1108
empty line ?
NO
YES
select empty victim
1112
select LRU victim
1114
1124
dirty?
YES
NO
write victim back
1126
request line from L2
1130
wait for line state
1132
update line & return word
1134

FIG. 2A

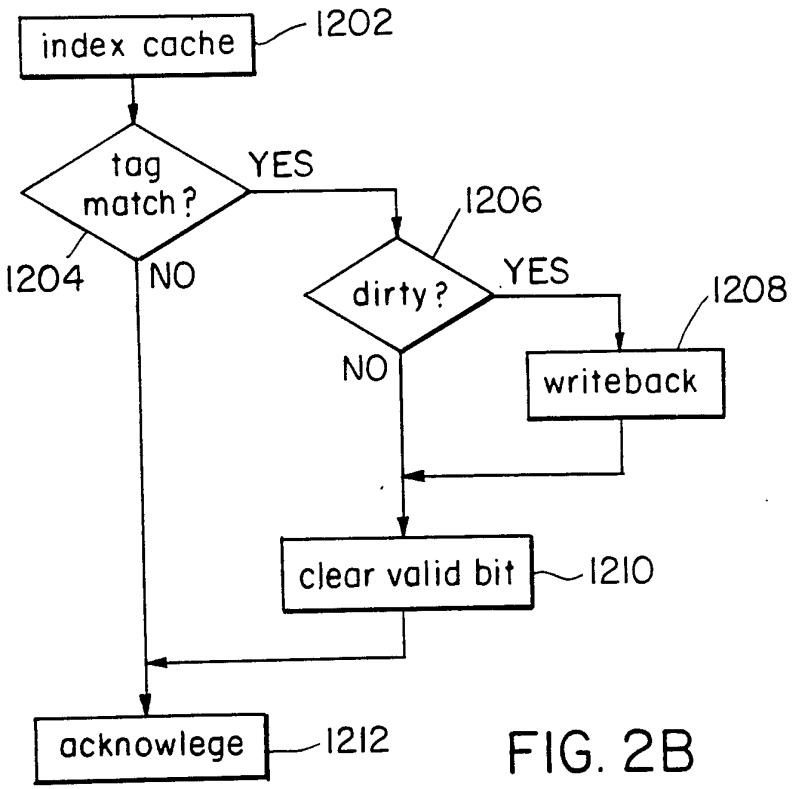

index cache
1202
tag match?
YES
1206
1204
NO
dirty?
YES
1208
NO
writeback
clear valid bit
1210
acknowlege
1212

FIG. 2B

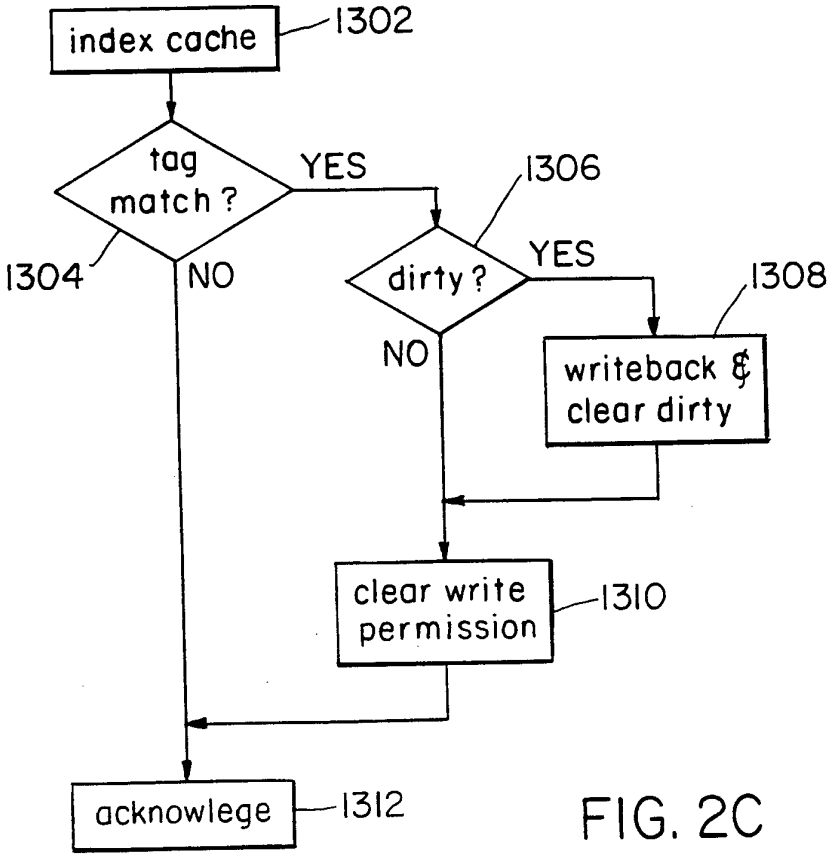

index cache
1302
tag match?
YES
1306
1304
NO
dirty?
YES
1308
NO
writeback & clear dirty
clear write permission
1310
acknowlege
1312

FIG. 2C

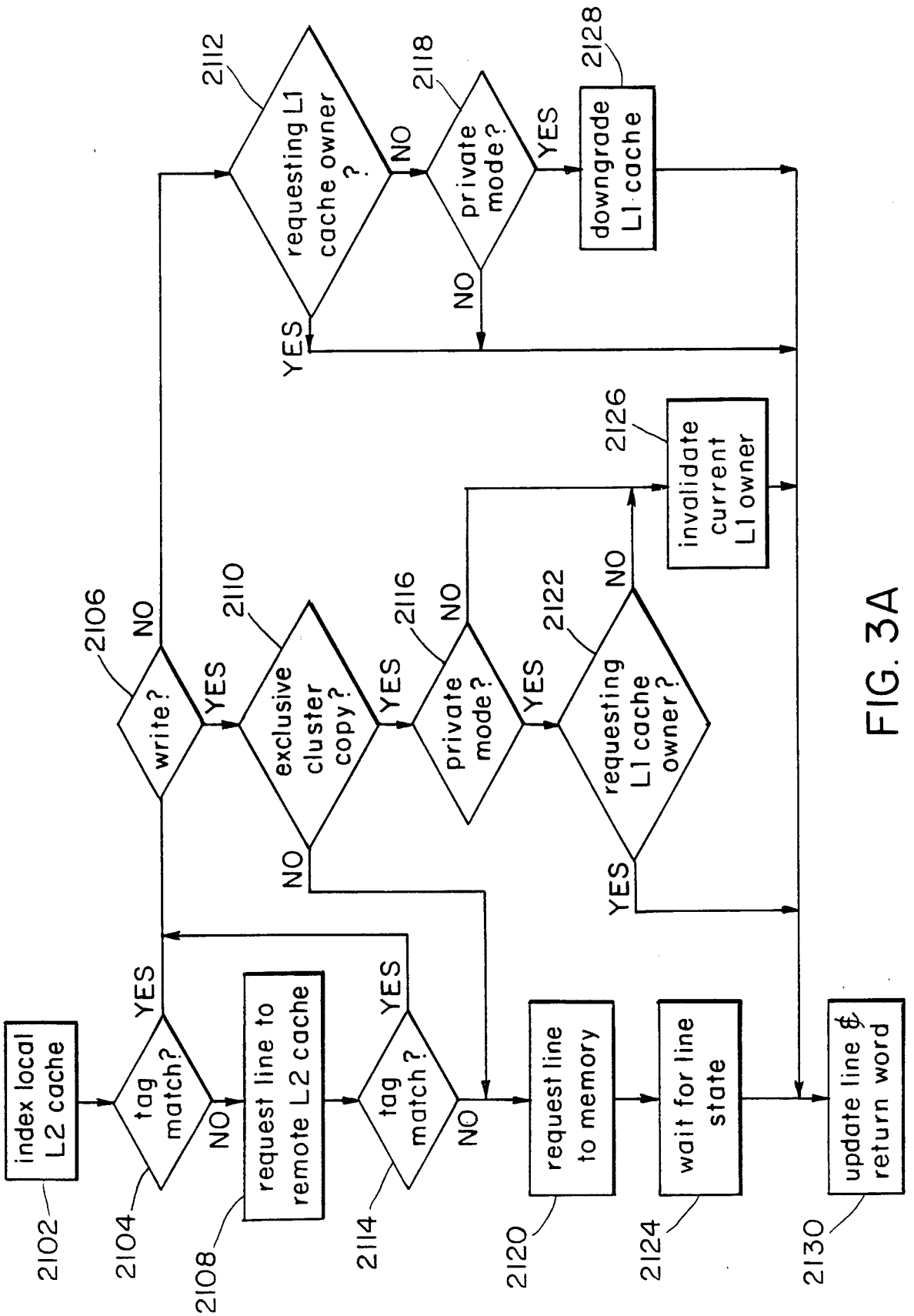

index local L2 cache
2102
tag match?
2104
YES
NO
request line to remote L2 cache
2108
tag match?
2114
YES
NO
request line to memory
2120
wait for line state
2124
update line & return word
2130
write?
2106
NO
YES
exclusive cluster copy?
2110
YES
NO
private mode?
2116
NO
YES
requesting L1 cache owner?
2122
NO
YES
invalidate current L1 owner
2126
requesting L1 cache owner?
2112
YES
NO
private mode?
2118
NO
YES
downgrade L1 cache
2128

FIG. 3A

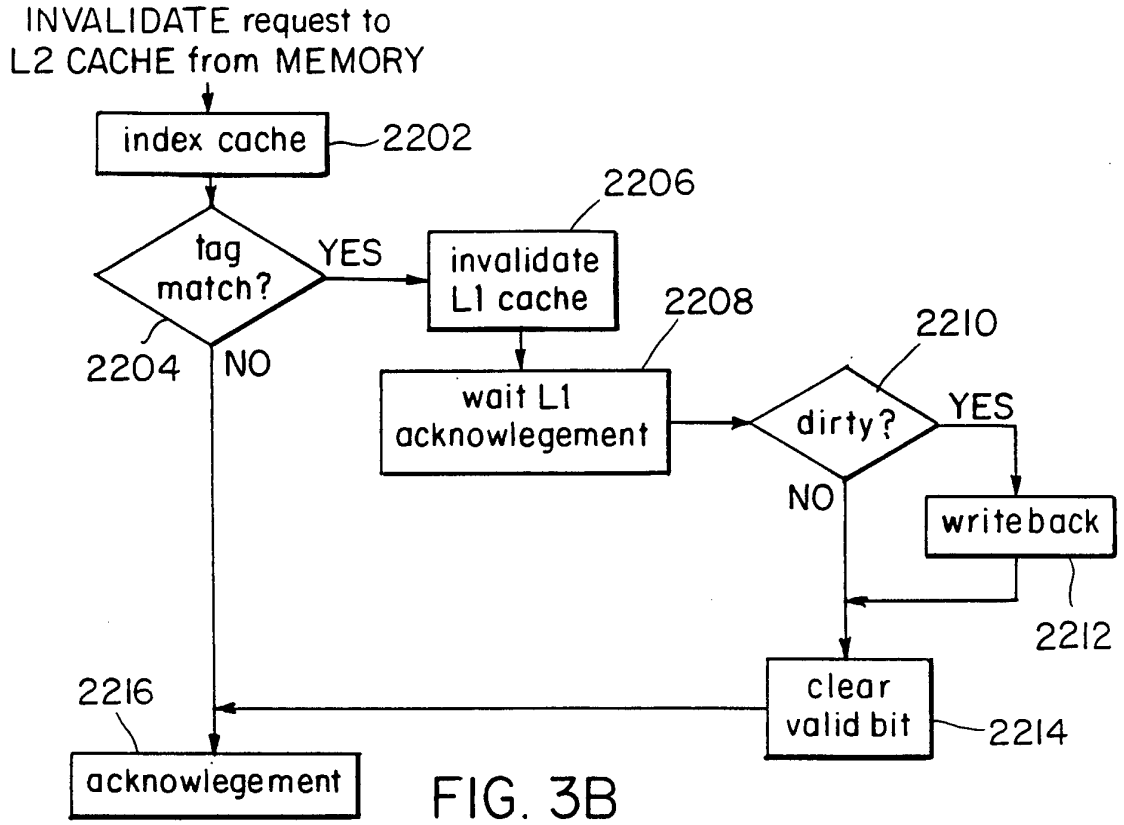

INVALIDATE request to
L2 CACHE from MEMORY
index cache
2202
tag match?
YES
2204
NO
2206
invalidate L1 cache
2208
wait L1 acknowlegement
2210
dirty?
YES
NO
writeback
2212
clear valid bit
2214
2216
acknowlegement

FIG. 3B

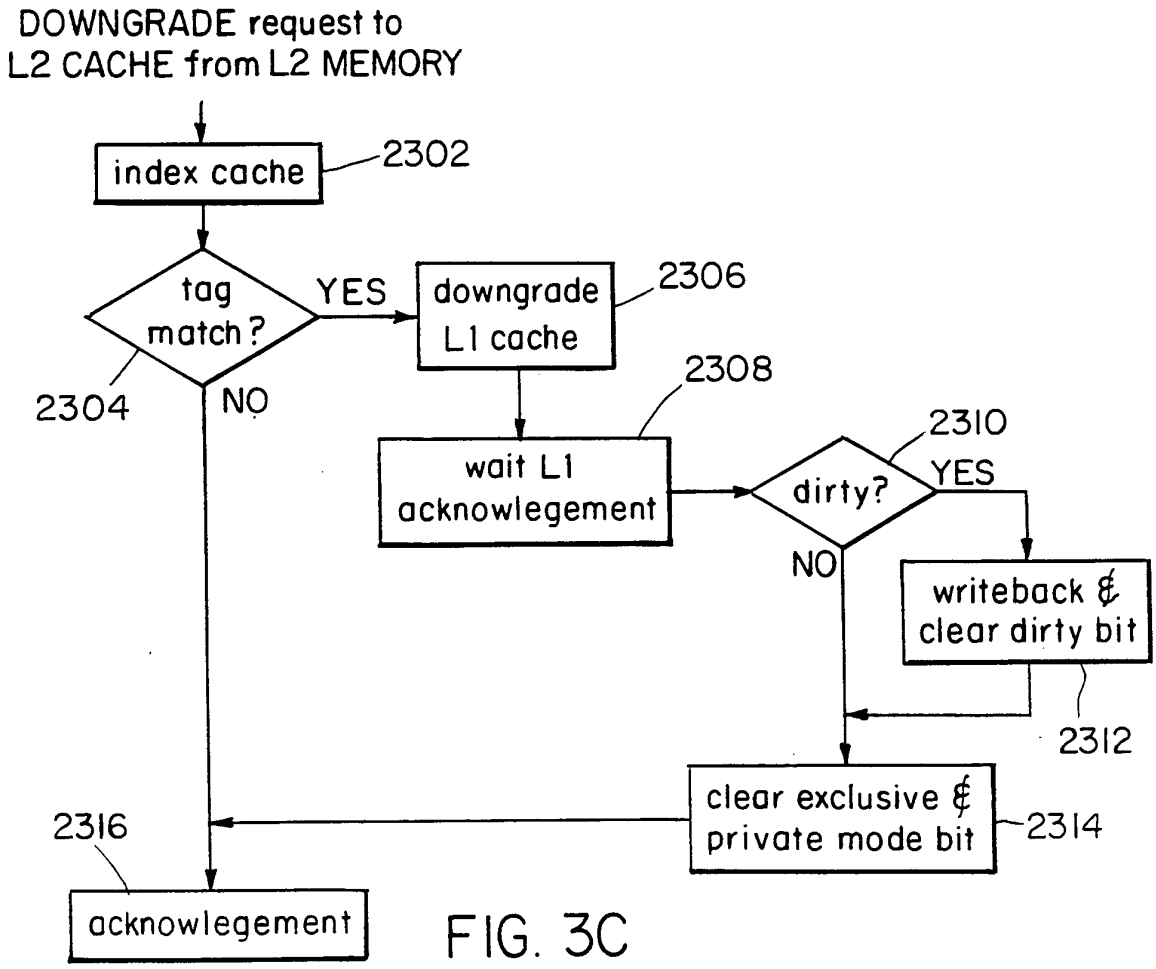

DOWNGRADE request to
L2 CACHE from L2 MEMORY
index cache
2302
tag match?
YES
2304
NO
downgrade L1 cache
2306
2308
wait L1 acknowlegement
2310
dirty?
YES
NO
writeback & clear dirty bit
2312
clear exclusive & private mode bit
2314
2316
acknowlegement

FIG. 3C

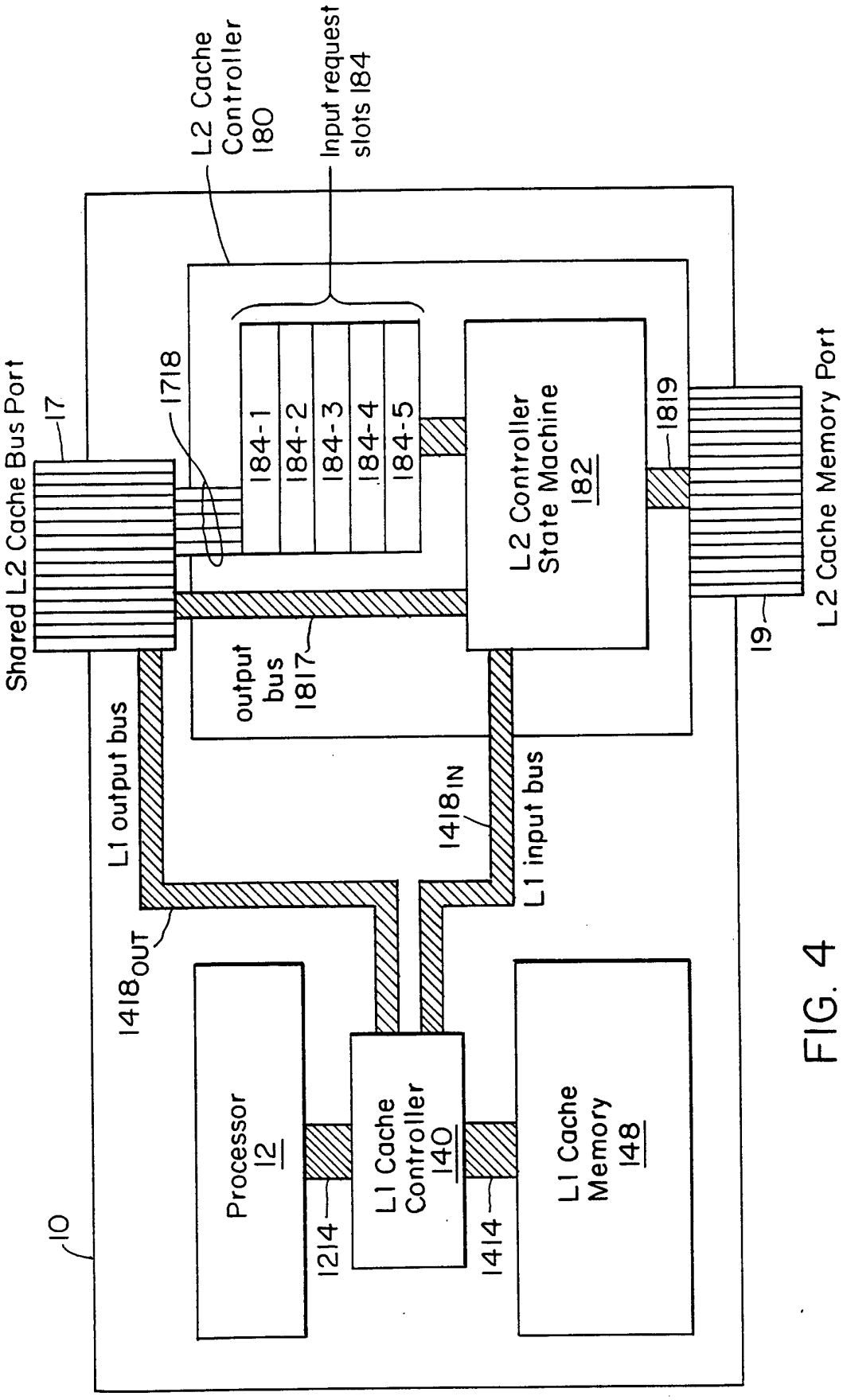

L2 Cache Controller 180
Input request slots 184
Shared L2 Cache Bus Port
17
1718
184-1
184-2
184-3
184-4
184-5
L2 Controller State Machine 182
1819
L2 Cache Memory Port
19
output bus 1817
L1 output bus
L1 input bus
1418IN
1418OUT
Processor 12
L1 Cache Controller 140
L1 Cache Memory 148
1214
1414
10

FIG. 4

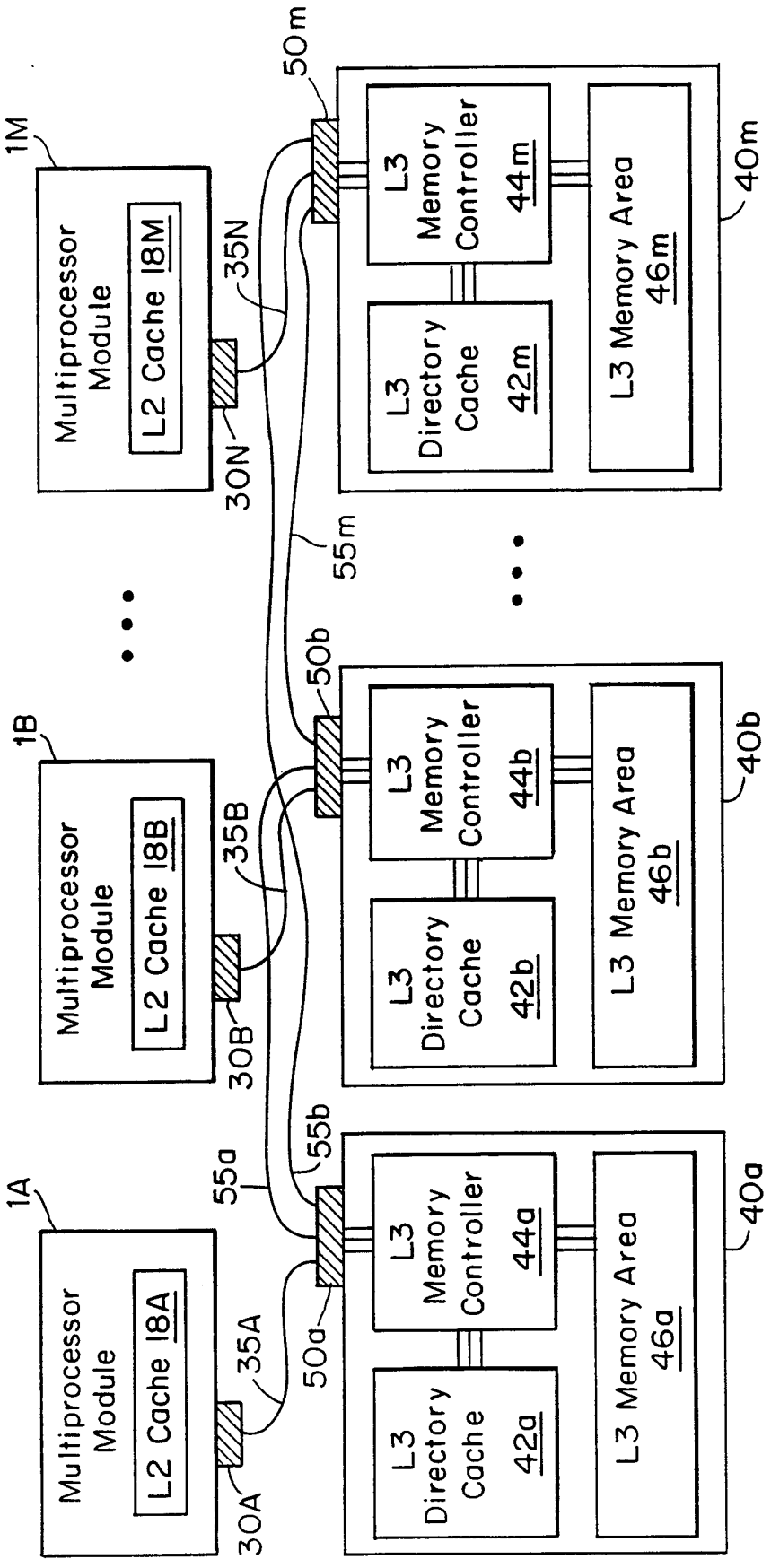

1A
Multiprocessor Module
L2 Cache 18A
30A
35A
50a
55a
55b
L3 Memory Controller 44a
L3 Directory Cache 42a
L3 Memory Area 46a
40a
1B
Multiprocessor Module
L2 Cache 18B
30B
35B
50b
L3 Memory Controller 44b
L3 Directory Cache 42b
L3 Memory Area 46b
40b
1M
Multiprocessor Module
L2 Cache 18M
30N
35N
50m
55m
L3 Memory Controller 44m
L3 Directory Cache 42m
L3 Memory Area 46m
40m

FIG. 5

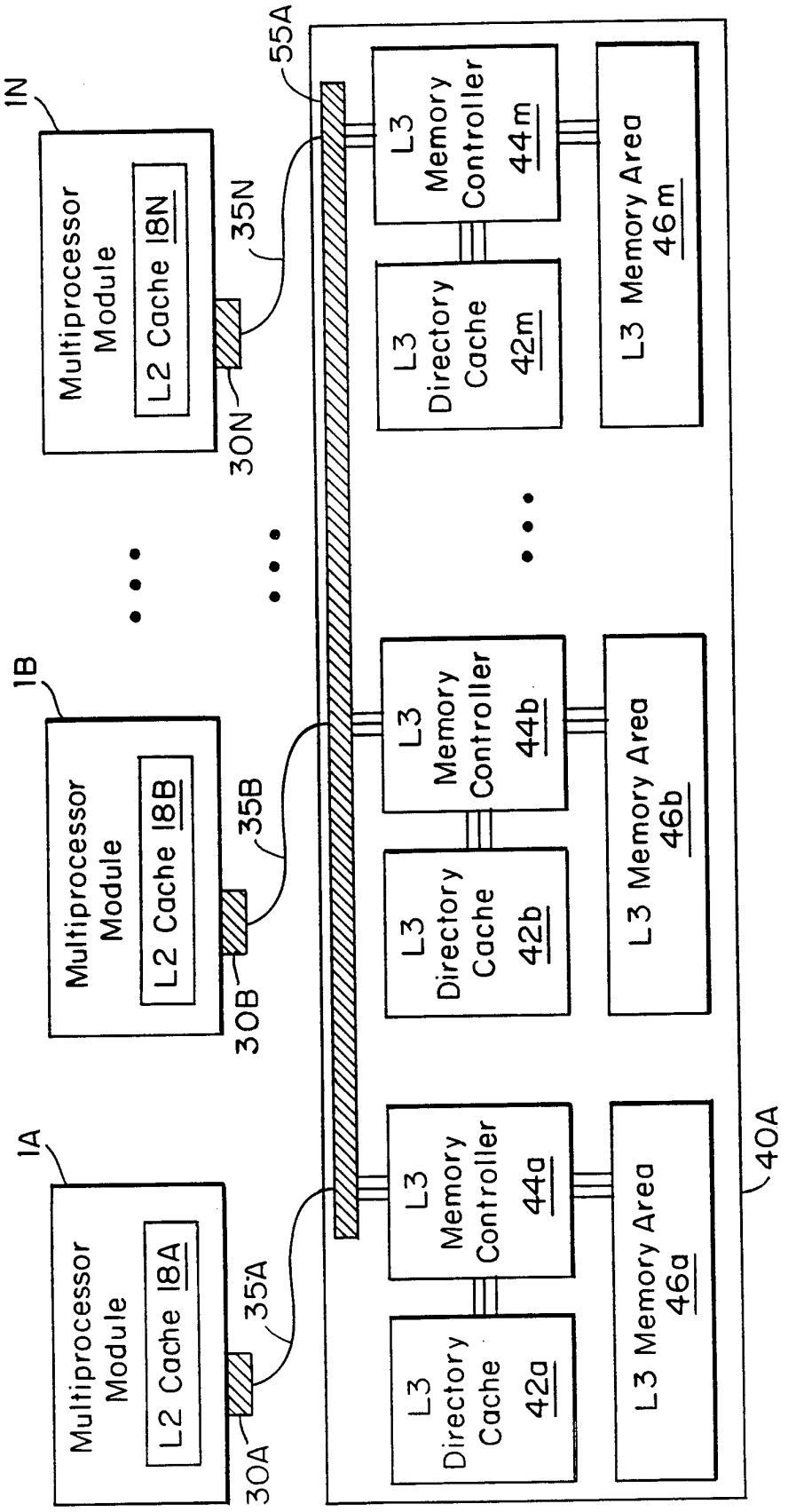

1N
Multiprocessor Module
L2 Cache 18N
30N
35N
55A
L3 Memory Controller 44m
L3 Directory Cache 42m
L3 Memory Area 46m
1B
Multiprocessor Module
L2 Cache 18B
30B
35B
L3 Memory Controller 44b
L3 Directory Cache 42b
L3 Memory Area 46b
1A
Multiprocessor Module
L2 Cache 18A
30A
35A
L3 Memory Controller 44a
L3 Directory Cache 42a
L3 Memory Area 46a
40A

FIG. 6

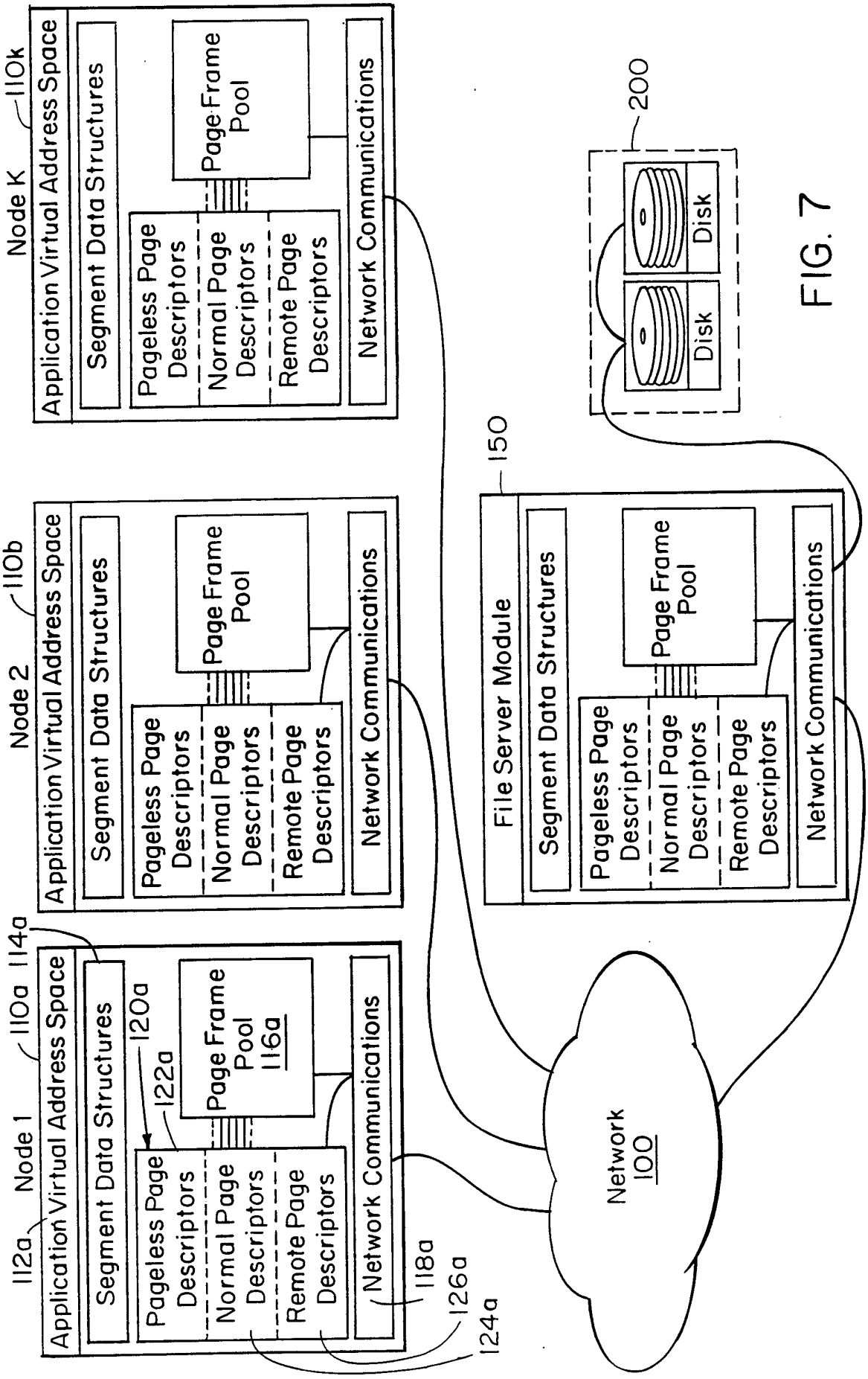

Node K
110k
Application Virtual Address Space
Segment Data Structures
Pageless Page Descriptors
Normal Page Descriptors
Remote Page Descriptors
Page Frame Pool
Network Communications
Node 2
110b
Application Virtual Address Space
Segment Data Structures
Pageless Page Descriptors
Normal Page Descriptors
Remote Page Descriptors
Page Frame Pool
Network Communications
Node 1
110a
114a
112a
Application Virtual Address Space
Segment Data Structures
120a
122a
Pageless Page Descriptors
Normal Page Descriptors
Remote Page Descriptors
Page Frame Pool 116a
Network Communications
118a
126a
124a
Network 100
File Server Module
150
Segment Data Structures
Pageless Page Descriptors
Normal Page Descriptors
Remote Page Descriptors
Page Frame Pool
Network Communications
200
Disk
Disk
Disk

FIG. 7

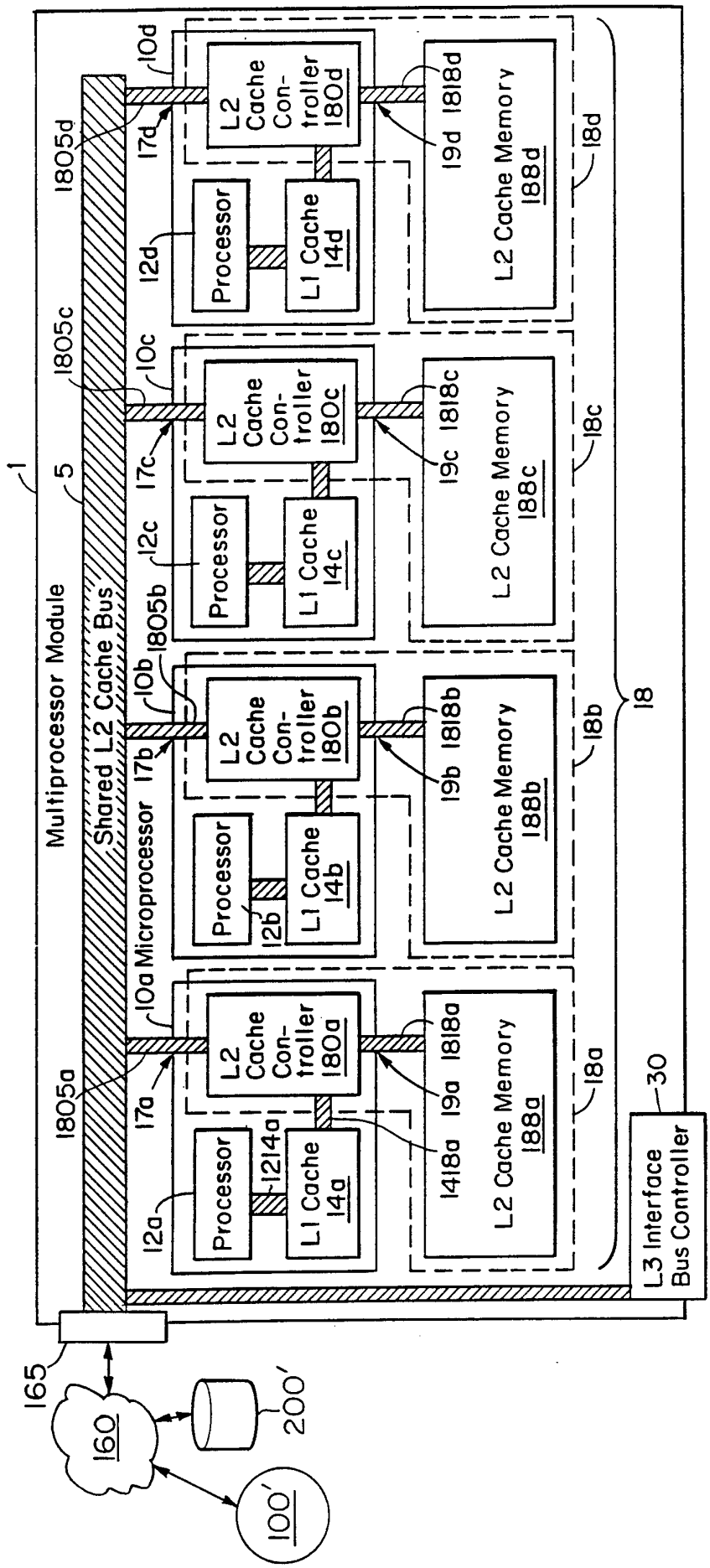

1
Multiprocessor Module
5
Shared L2 Cache Bus
Microprocessor
10a
10b
10c
10d
1805a
1805b
1805c
1805d
17a
17b
17c
17d
12a
12b
12c
12d
1214a
Processor
L1 Cache 14a
L1 Cache 14b
L1 Cache 14c
L1 Cache 14d
1418a
L2 Cache Controller 180a
L2 Cache Controller 180b
L2 Cache Controller 180c
L2 Cache Controller 180d
19a 1818a
19b 1818b
19c 1818c
19d 1818d
L2 Cache Memory 188a
L2 Cache Memory 188b
L2 Cache Memory 188c
L2 Cache Memory 188d
18a
18b
18c
18d
18
30
L3 Interface Bus Controller
165
160
200'
100'

FIG. 8

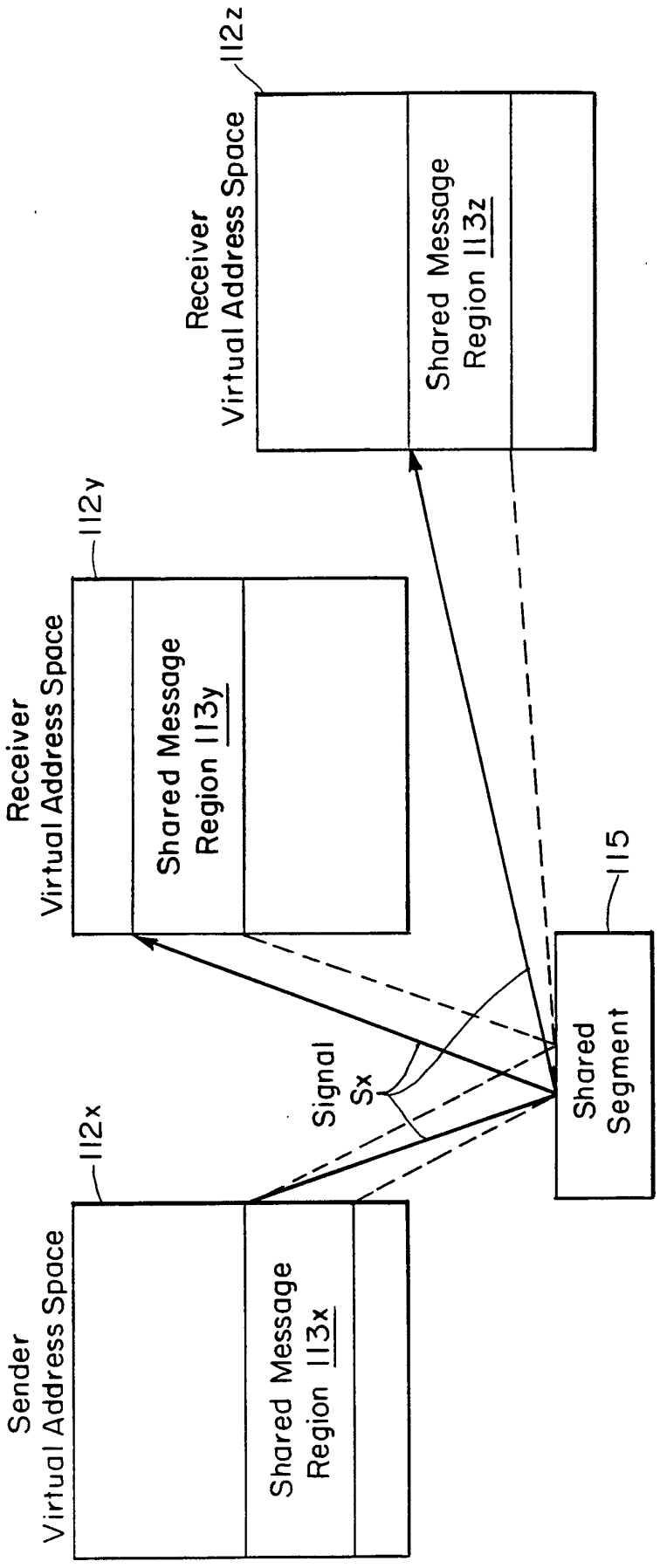

Receiver
Virtual Address Space
112z
Shared Message
Region 113z
Receiver
Virtual Address Space
112y
Shared Message
Region 113y
Signal
Sx
Shared
Segment
115
Sender
Virtual Address Space
112x
Shared Message
Region 113x

FIG. 9

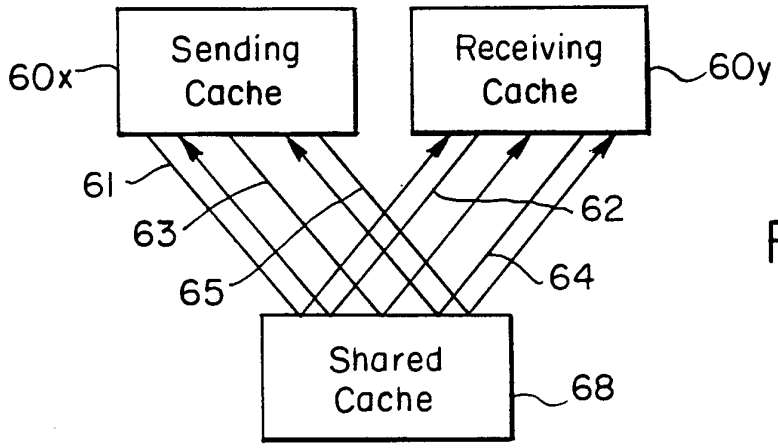


FIG. 10A

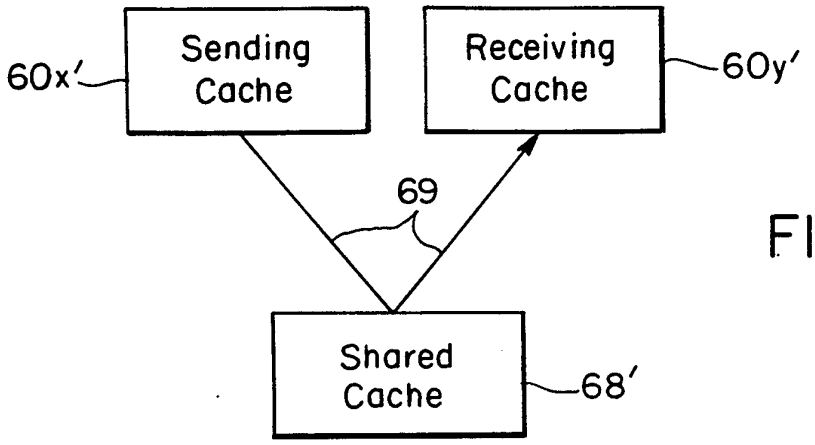


FIG. 10B