EXHIBIT A

US007296121B2

(12) **United States Patent**
Morton et al.

(10) **Patent No.: US 7,296,121 B2**
(45) **Date of Patent: Nov. 13, 2007**

(54) **REDUCING PROBE TRAFFIC IN MULTIPROCESSOR SYSTEMS**

(75) Inventors: **Eric Morton**, Austin, TX (US); **Rajesh Kota**, Austin, TX (US); **Adnan Khaleel**, Austin, TX (US); **David B. Glasco**, Austin, TX (US)

(73) Assignee: **Newisys, Inc.**, Austin, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 250 days.

(21) Appl. No.: **10/966,161**

(22) Filed: **Oct. 15, 2004**
(Under 37 CFR 1.47)

(65) **Prior Publication Data**

US 2007/0055826 A1 Mar. 8, 2007

Related U.S. Application Data

(63) Continuation-in-part of application No. 10/288,347, filed on Nov. 4, 2002, now Pat. No. 7,003,633.

(51) **Int. Cl.**
G06F 12/00 (2006.01)

(52) **U.S. Cl.** **711/148**; 711/141

(58) **Field of Classification Search** 711/141, 711/148, 131, 144, 145, 146; 709/206, 213, 709/216, 217, 218, 219
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,195,089 A 3/1993 Sindhu et al.
5,394,555 A 2/1995 Hunter et al.
5,524,212 A 6/1996 Somani et al.
5,692,123 A 11/1997 Logghe
5,751,995 A 5/1998 Sarangdhar
5,829,032 A 10/1998 Komuro et al.
5,893,151 A 4/1999 Merchant
6,018,791 A 1/2000 Arimilli et al.
6,038,652 A 3/2000 Phillips et al.
6,052,769 A 4/2000 Huff et al.
6,067,603 A 5/2000 Carpenter et al.
6,073,210 A 6/2000 Palanca et al.
6,085,295 A 7/2000 Ekanadham et al.
6,108,737 A 8/2000 Sharma et al.
6,122,715 A 9/2000 Palanca et al.

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 0239242 5/2002

OTHER PUBLICATIONS

Guo, et al., "A Probe-Based Server Selection Protocol for Differentiated Service Networks", © 2002 IEEE, p. 2353-2357.*

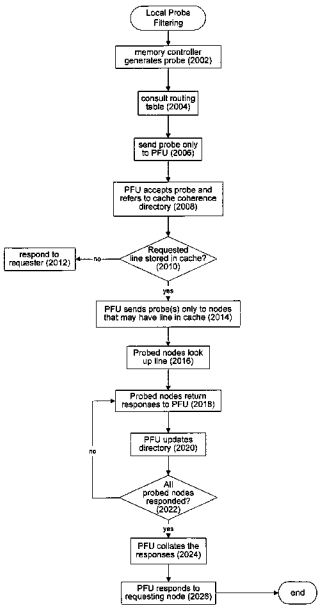
(Continued)

Primary Examiner—Brian R. Peugh
(74) *Attorney, Agent, or Firm*—Beyer Weaver LLP

(57) **ABSTRACT**

A computer system having a plurality of processing nodes interconnected by a first point-to-point architecture is described. Each processing node has a cache memory associated therewith. A probe filtering unit is operable to receive probes corresponding to memory lines from the processing nodes and to transmit the probes only to selected ones of the processing nodes with reference to probe filtering information. The probe filtering information is representative of states associated with selected ones of the cache memories.

25 Claims, 25 Drawing Sheets



U.S. PATENT DOCUMENTS

6,148,378	A	11/2000	Bordaz et al.	
6,167,492	A	12/2000	Keller et al.	
6,173,393	B1	1/2001	Palanca et al.	
6,189,078	B1	2/2001	Bauman et al.	
6,192,451	B1	2/2001	Arimilli et al.	
6,205,520	B1	3/2001	Palanca et al.	
6,209,065	B1	3/2001	Durham et al.	
6,292,705	B1	9/2001	Wang et al.	
6,292,906	B1	9/2001	Fu et al.	
6,330,643	B1	12/2001	Arimilli et al.	
6,334,172	B1	12/2001	Arimilli et al.	
6,338,122	B1	1/2002	Baumgartner et al.	
6,343,347	B1	1/2002	Arimilli et al.	
6,385,705	B1	5/2002	Keller et al.	
6,405,289	B1	6/2002	Arimilli et al.	
6,463,529	B1	10/2002	Miller et al.	
6,467,007	B1	10/2002	Armstrong et al.	
6,490,661	B1	12/2002	Keller et al.	
6,542,926	B2	4/2003	Zalewski et al.	
6,615,319	B2	9/2003	Khare et al.	
6,631,447	B1	10/2003	Morioka et al.	
6,633,945	B1	10/2003	Fu et al.	
6,633,960	B1	10/2003	Kessler et al.	
6,636,906	B1	10/2003	Sharma et al.	
6,640,287	B2	10/2003	Gharachorloo et al.	
6,658,526	B2	12/2003	Nguyen et al.	
6,665,767	B1	12/2003	Comisky et al.	
6,704,842	B1	3/2004	Janakiraman et al.	
6,738,870	B2	5/2004	Van Huben et al.	
6,738,871	B2	5/2004	Van Huben et al.	
6,751,698	B1	6/2004	Deneroff et al.	
6,751,721	B1	6/2004	Webb et al.	
6,754,782	B2	6/2004	Arimilli et al.	
6,760,809	B2	7/2004	Arimilli et al.	
6,760,819	B2	7/2004	Dhong et al.	
6,775,749	B1 *	8/2004	Mudgett et al.	711/146
6,799,252	B1	9/2004	Bauman	
6,865,595	B2	3/2005	Glasco	
6,892,282	B2	5/2005	Hass et al.	
7,003,633	B2	2/2006	Glasco	
2001/0013089	A1	8/2001	Weber	
2001/0029574	A1 *	10/2001	Razdan et al.	711/130
2001/0037435	A1	11/2001	Van Doren	
2002/0007463	A1	1/2002	Fung	
2002/0046327	A1	4/2002	Gharachorloo et al.	
2002/0052914	A1	5/2002	Zalewski et al.	
2002/0083149	A1	6/2002	Van Huben et al.	
2002/0083243	A1	6/2002	Van Huben	
2002/0087807	A1	7/2002	Gharachorloo et al.	
2002/0087811	A1	7/2002	Khare et al.	
2003/0009623	A1	1/2003	Arimilli et al.	
2003/0182508	A1	9/2003	Glasco	
2003/0182509	A1	9/2003	Glasco	
2003/0182514	A1	9/2003	Glasco	
2003/0195939	A1	10/2003	Edirisooriya et al.	
2003/0196047	A1	10/2003	Kessler et al.	
2003/0210655	A1	11/2003	Giasco	
2003/0212741	A1	11/2003	Giasco	
2003/0233388	A1	12/2003	Giasco et al.	
2004/0024836	A1 *	2/2004	Keller et al.	709/213
2004/0073755	A1	4/2004	Webb et al.	
2004/0088492	A1	5/2004	Glasco	
2004/0088493	A1	5/2004	Glasco	
2004/0088494	A1	5/2004	Giasco	
2004/0117559	A1	6/2004	Glasco	
2004/0255002	A1	12/2004	Kota et al.	

OTHER PUBLICATIONS

HyperTransport™ I/O Link Specification Revision 1.03, HyperTransport™ Consortium, Oct. 10, 2001, Copyright © 2001 HyperTransport Technology Consortium.

PCT Search Report PCT/US03/34756, Int'l filing date Oct. 30, 2003, Search report Mailed Dec. 16, 2004.

Bilir et al., "Multicast Snooping: A New Coherence Method Using a Multicast Address Network", Computer Architecture, 1999. Proceedings of the 26th International Symposium on, May 2-4, 1999.

Martin et al., "Bandwidth Adaptive Snooping", Proceedings of the Eighth International Symposium on High-Performance Computer Architecture on Feb. 2-6, 2002; pp. 251-262.

Sorin et al., "Specifying and Verifying a Broadcast and a Multicast Snooping Cache Coherence Protocol", IEEE Transactions on Parallel and Distributed Systems, vol. 13, No. 6, Jun. 2002.

U.S. Appl. No.: 10/288,347 (Now U.S. Pat. No. 7,003,633), Notice of Allowance, dated Sep. 12, 2005.

U.S. Appl. No. 10/288,347 (Now U.S. Pat. No. 7,003,633), First Office Action, dated Nov. 18, 2004.

Kim et al., "Power-aware Partitioned Cache Architectures", 2001 ACM p. 6467.

Powell et al., "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping" 2001 IEEE, p. 54-65.

Culler, D. E., J. P. Singh, A. Gupta, "Parallel Computer Architecture", 1999 Morgan Kaufmann, San Francisco, CA USA XP002277658.

Tanenbaum, Andrew, "Computer Networks", Computer Networks, London: Prentice Hall International, GB, 1996, pp. 345-403, XP002155220.

U.S. Appl. No.: 10/288,347 (Now U.S. Pat. No. 7,003,633). Final Office Action, dated May 12, 2005.

U.S. Office Action mailed Sep. 22, 2004, from U.S. Appl. No. 10/106,426 [NWISP002].

U.S. Office Action mailed Mar. 7, 2005, form U.S. Appl. No. 10/106,426 [NWISP002].

U.S. Office Action mailed Jul. 21, 2005, from U.S. Appl. No. 10/106,426 [NWISP002].

U.S. Office Action mailed Sep. 23, 2004, from U.S. Appl. No. 10/106,430 [NWISP003].

U.S. Office Action mailed Mar. 10, 2005, form U.S. Appl. No. 10/106,430 [NWISP003].

U.S. Office Action mailed Jul. 21, 2005, from U.S. Appl. No. 10/106,430 [NWISP003].

U.S. Office Action mailed Sep. 22, 2004, from U.S. Appl. No. 10/106,299 [NWISP004].

U.S. Office Action mailed Mar. 10, 2005, from U.S. Appl. No. 10/106,299 [NWISP004].

U.S. Office Action mailed Jul. 21, 2005, from U.S. Appl. No. 10/106,299 [NWISP004].

U.S. Office Action mailed Jul. 20, 2005, from U.S. Appl. No. 10/608,846 [NWISP030].

U.S. Office Action mailed Sep. 9, 2005, from U.S. Appl. No. 10/462,015 [NWISP040].

U.S. Office Action mailed Sep. 9, 2005, from U.S. Appl. No. 10/426,084 [NWISP033].

U.S. Office Action mailed Nov. 2, 2005, from U.S. Appl. No. 10/106/430 [NWISP003].

U.S. Office Action mailed Oct. 5, 2005, from U.S. Appl. No. 10/635,703 [NWISP036].

* cited by examiner

Figure 1A

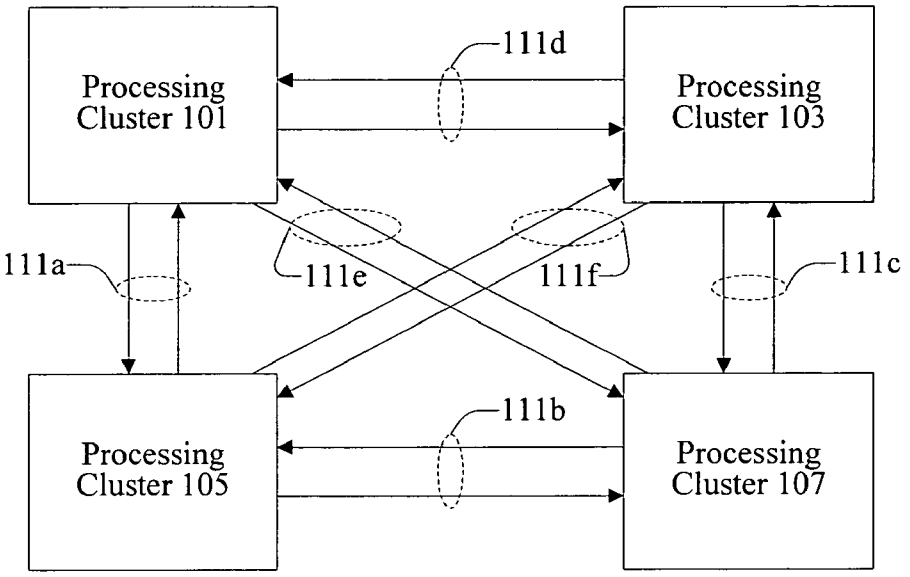


Figure 1B

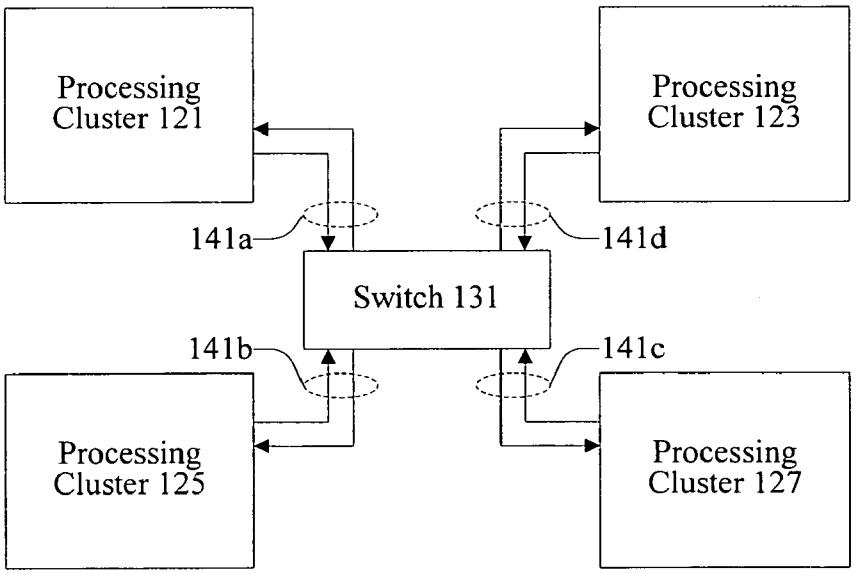


Figure 2

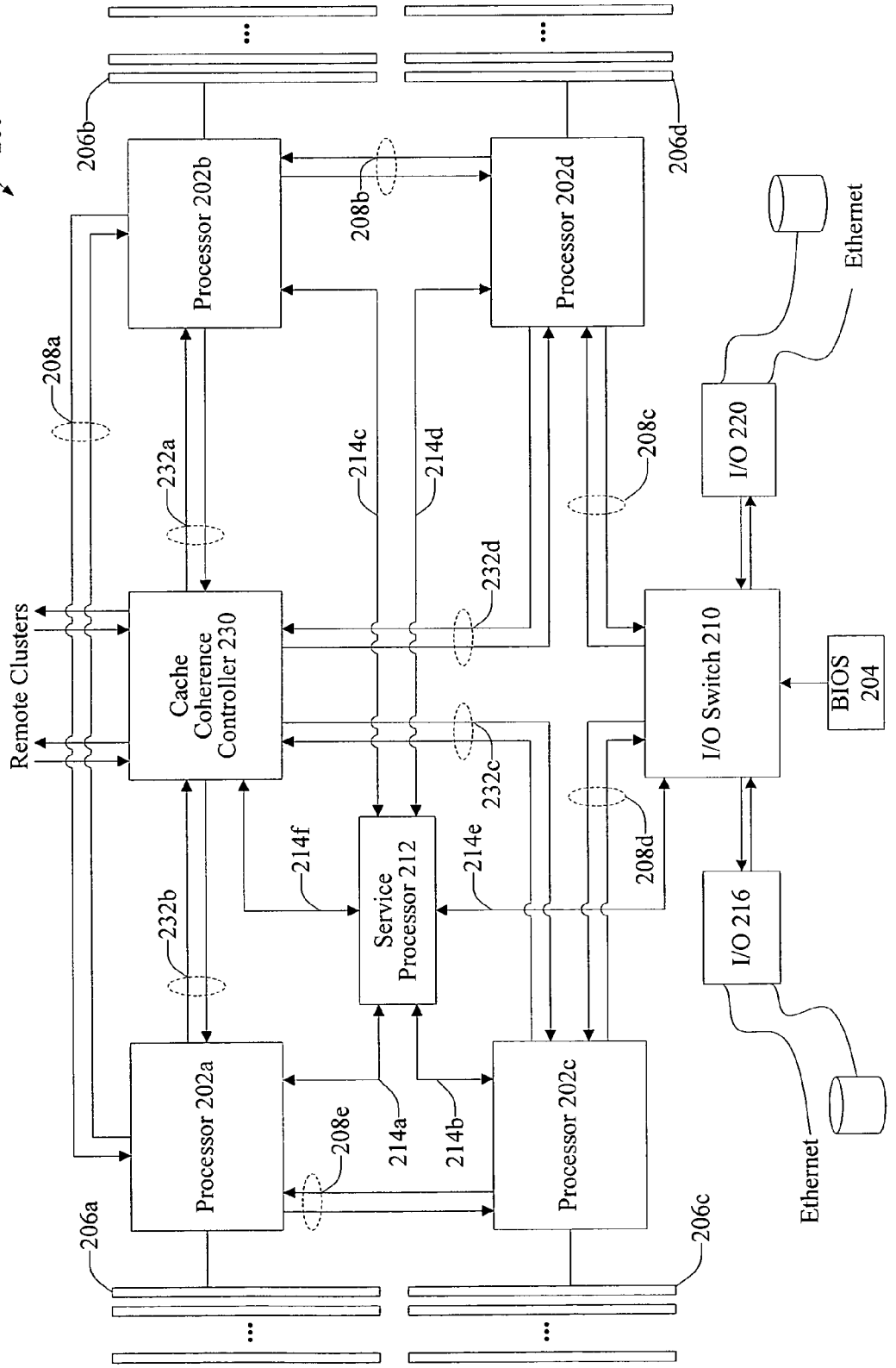
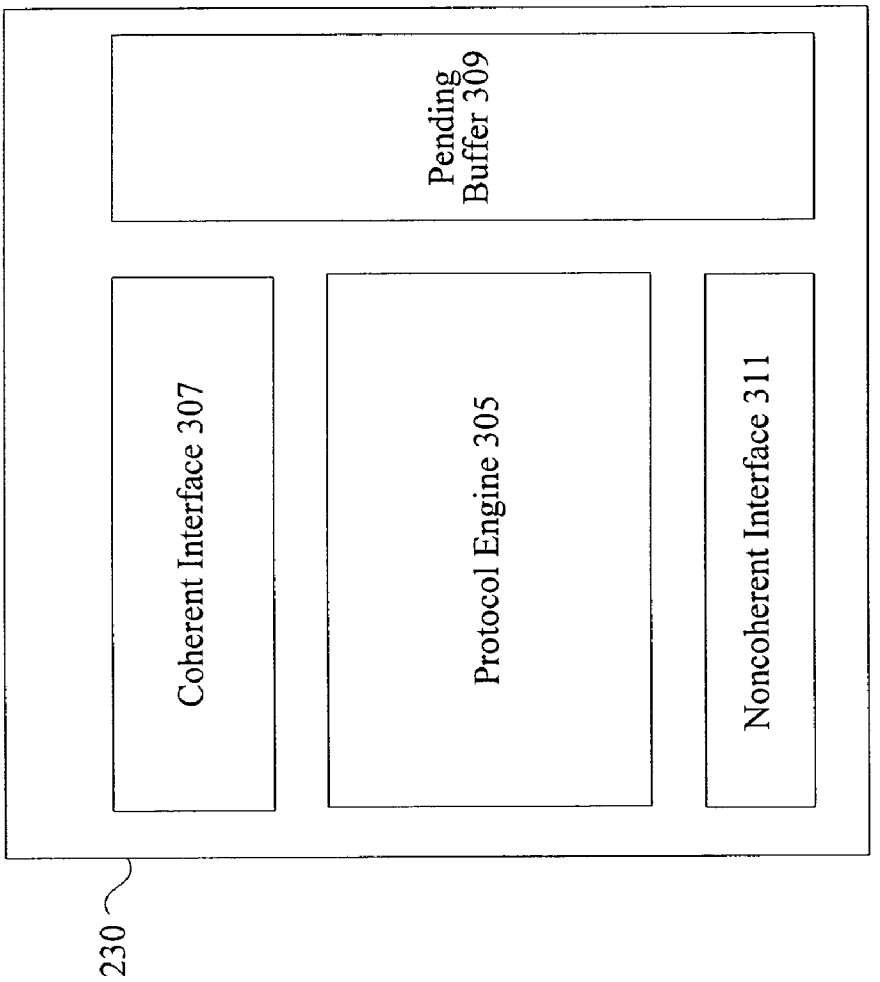

200
206b
206d
Processor 202b
Processor 202d
208b
208a
232a
214c
214d
208c
Ethernet
I/O 220
Remote Clusters
Cache Coherence Controller 230
232d
I/O Switch 210
BIOS 204
232c
214f
Service Processor 212
214e
208d
232b
I/O 216
Processor 202a
Processor 202c
208e
214a
214b
Ethernet
206a
206c

Figure 3


Pending Buffer 309
Coherent Interface 307
Protocol Engine 305
Noncoherent Interface 311
230

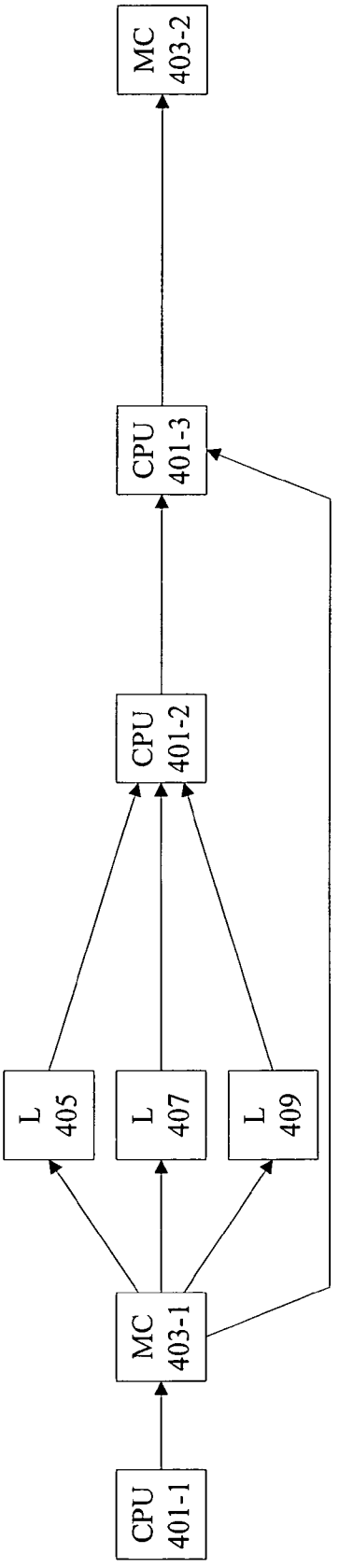


Figure 4

Figure 5A

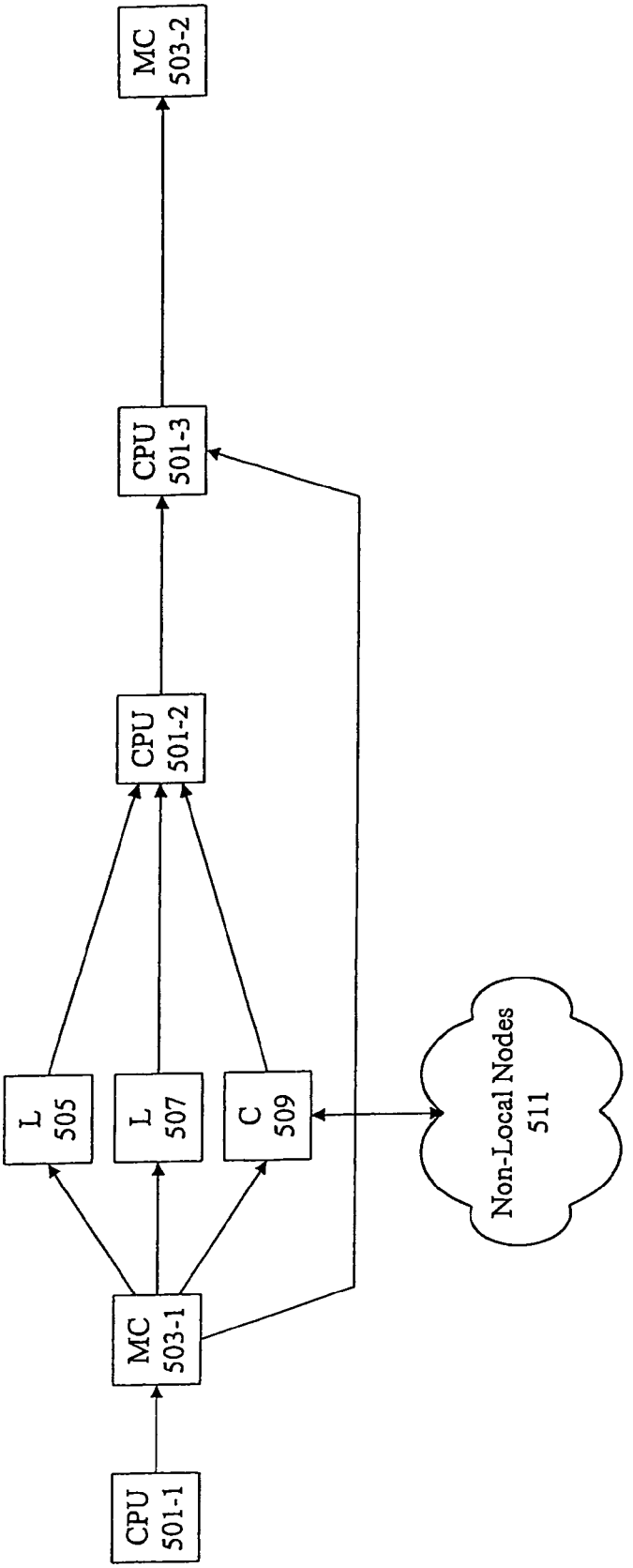
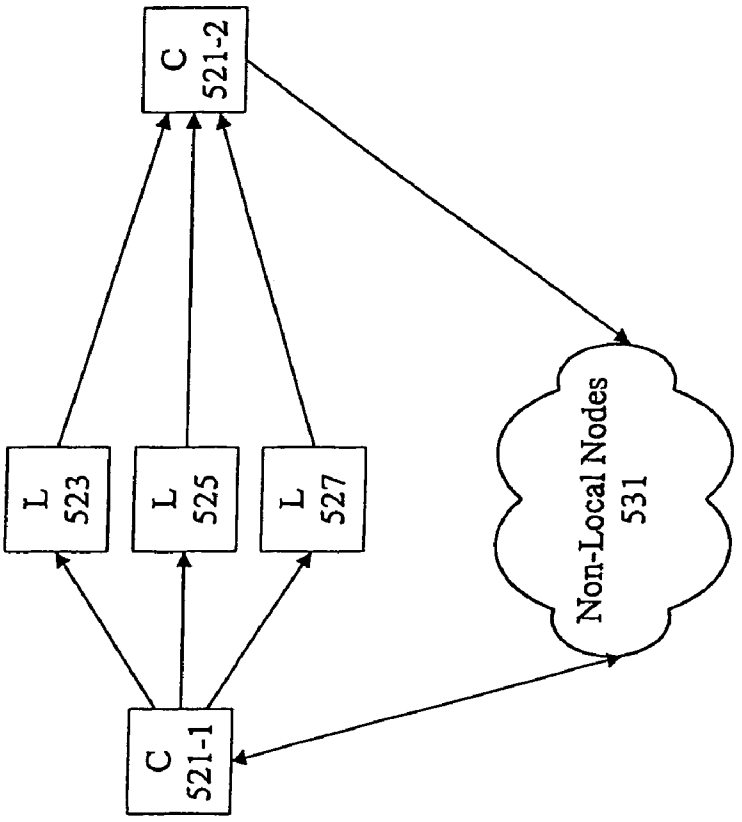

CPU 501-1
MC 503-1
L 505
L 507
C 509
Non-Local Nodes 511
CPU 501-2
CPU 501-3
MC 503-2

Figure 5B



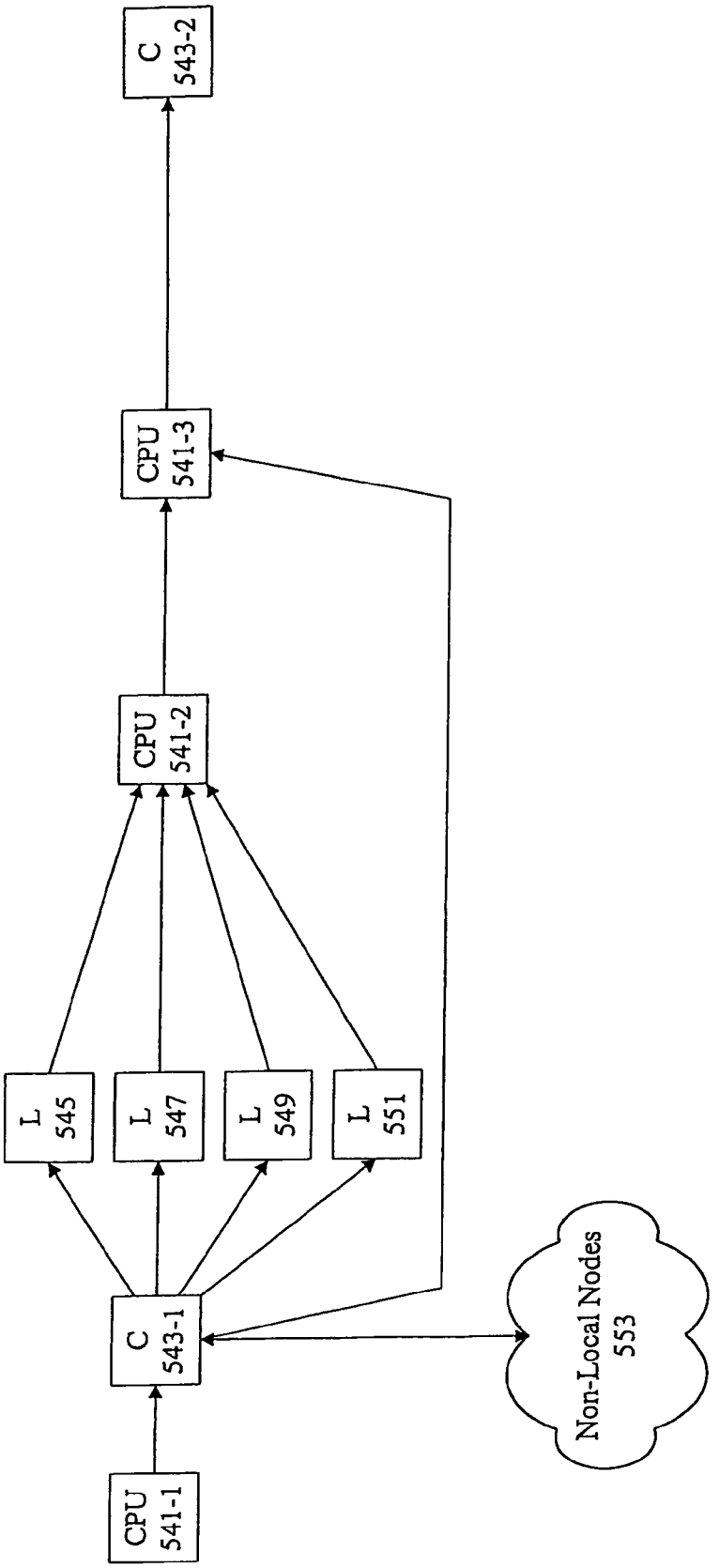

C 543-2
CPU 541-3
CPU 541-2
L 545
L 547
L 549
L 551
C 543-1
Non-Local Nodes 553
CPU 541-1

Figure 5C

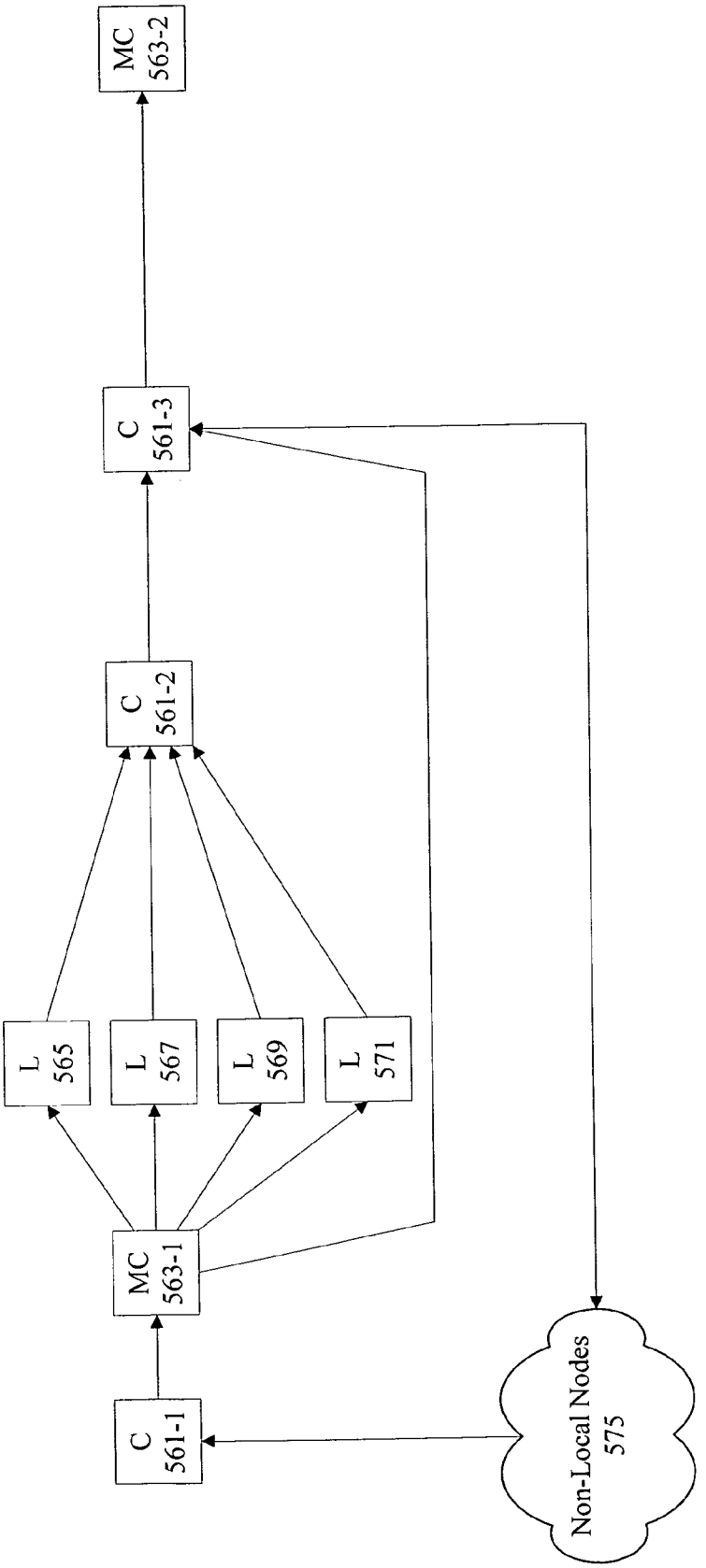

MC 563-2
C 561-3
C 561-2
L 565
L 567
L 569
L 571
MC 563-1
C 561-1
Non-Local Nodes 575

Figure 5D

Figure 6

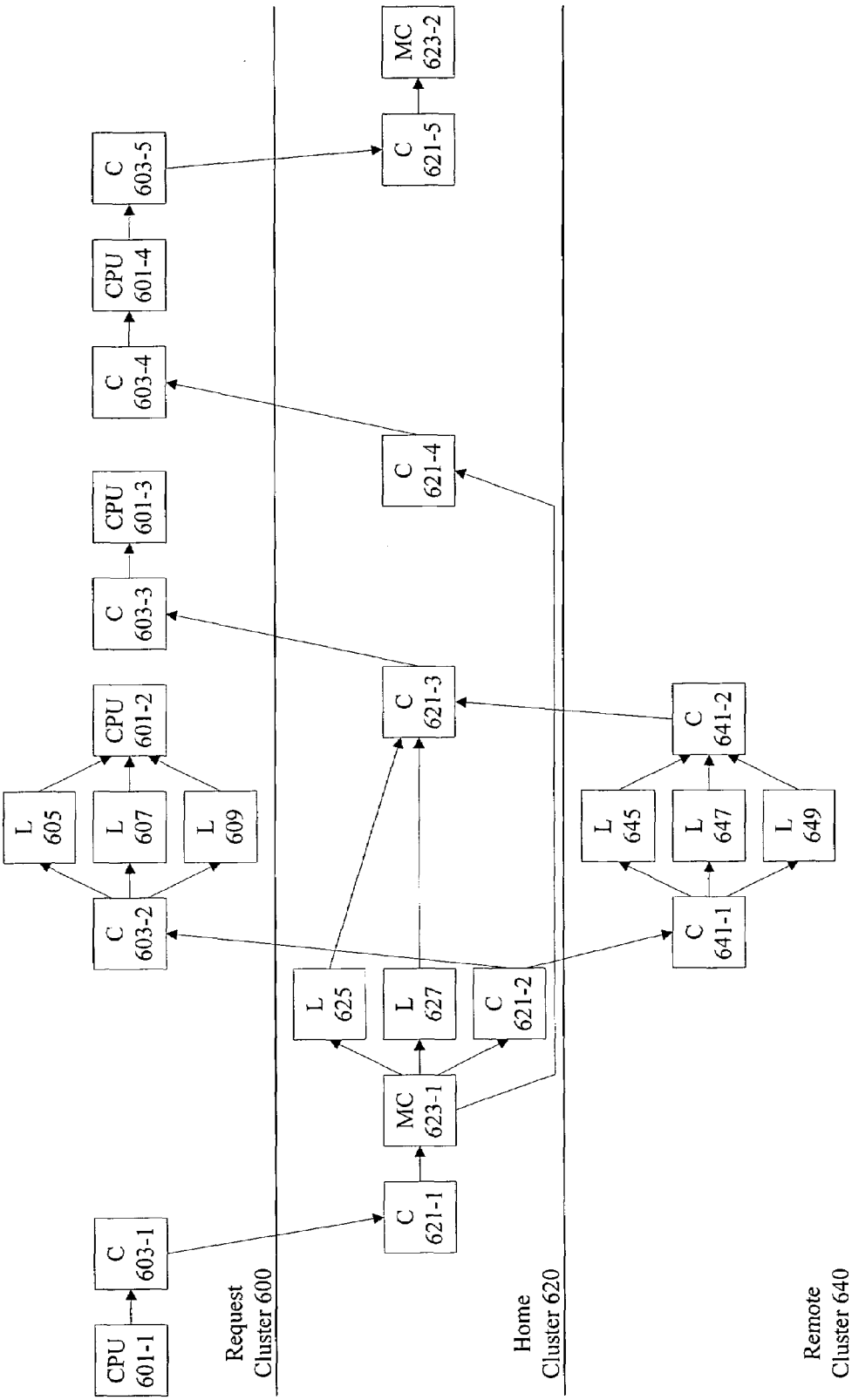

CPU 601-1
C 603-1
C 621-1
MC 623-1
L 625
L 627
C 621-2
C 603-2
L 605
L 607
L 609
CPU 601-2
C 621-3
C 603-3
CPU 601-3
C 603-4
CPU 601-4
C 603-5
C 621-5
MC 623-2
C 621-4
C 641-1
L 645
L 647
L 649
C 641-2
Request Cluster 600
Home Cluster 620
Remote Cluster 640

Figure 7

Coherence Directory 701			
Memory Line 711	State 713	Dirty Data Owner 715	Occupancy Vector 717
Address 721	Invalid	N/A	N/A
Address 731	Invalid	N/A	N/A
Address 741	Shared	N/A	Clusters 1,3
Address 751	Shared	N/A	Clusters 1, 2, 3, 4
Address 761	Owned	Cluster 4	Cluster 2, 3, 4
Address 771	Owned	Cluster 2	Cluster 2, 4
Address 781	Modified	Cluster 2	N/A
Address 791	Modified	Cluster 3	N/A
...	...	...	...

Figure 8

Probe Filter Information 821	Read Block (Read) 823	Read Block Modify (Read/Write) 825
Invalid 831	Can use completion bit. Probe home cluster. (801)	Can use completion bit. Probe home cluster. (809)
Shared 833	Can use completion bit. Probe home cluster. (803)	N/A (811)
Owned 835	Can use completion bit. Probe remote cluster with line cached in owned state. (805)	N/A (813)
Modified 837	Can use completion bit. Probe remote cluster with line cached in modified state. (807)	Can use completion bit. Probe remote cluster. (815)

Figure 9

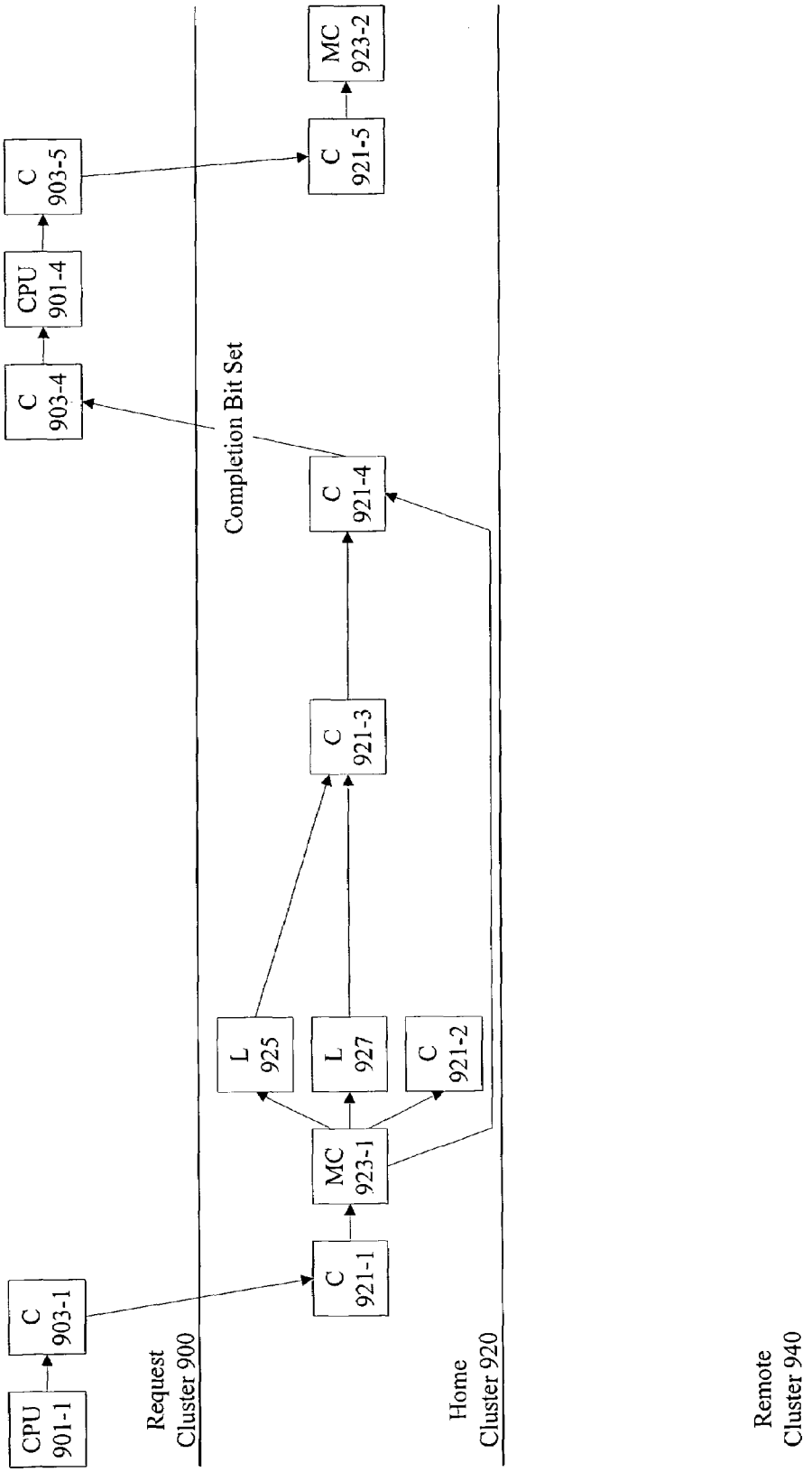

CPU 901-1
C 903-1
C 903-4
CPU 901-4
C 903-5
C 921-1
MC 923-1
L 925
L 927
C 921-2
C 921-3
C 921-4
C 921-5
MC 923-2
Completion Bit Set
Request Cluster 900
Home Cluster 920
Remote Cluster 940

Figure 10

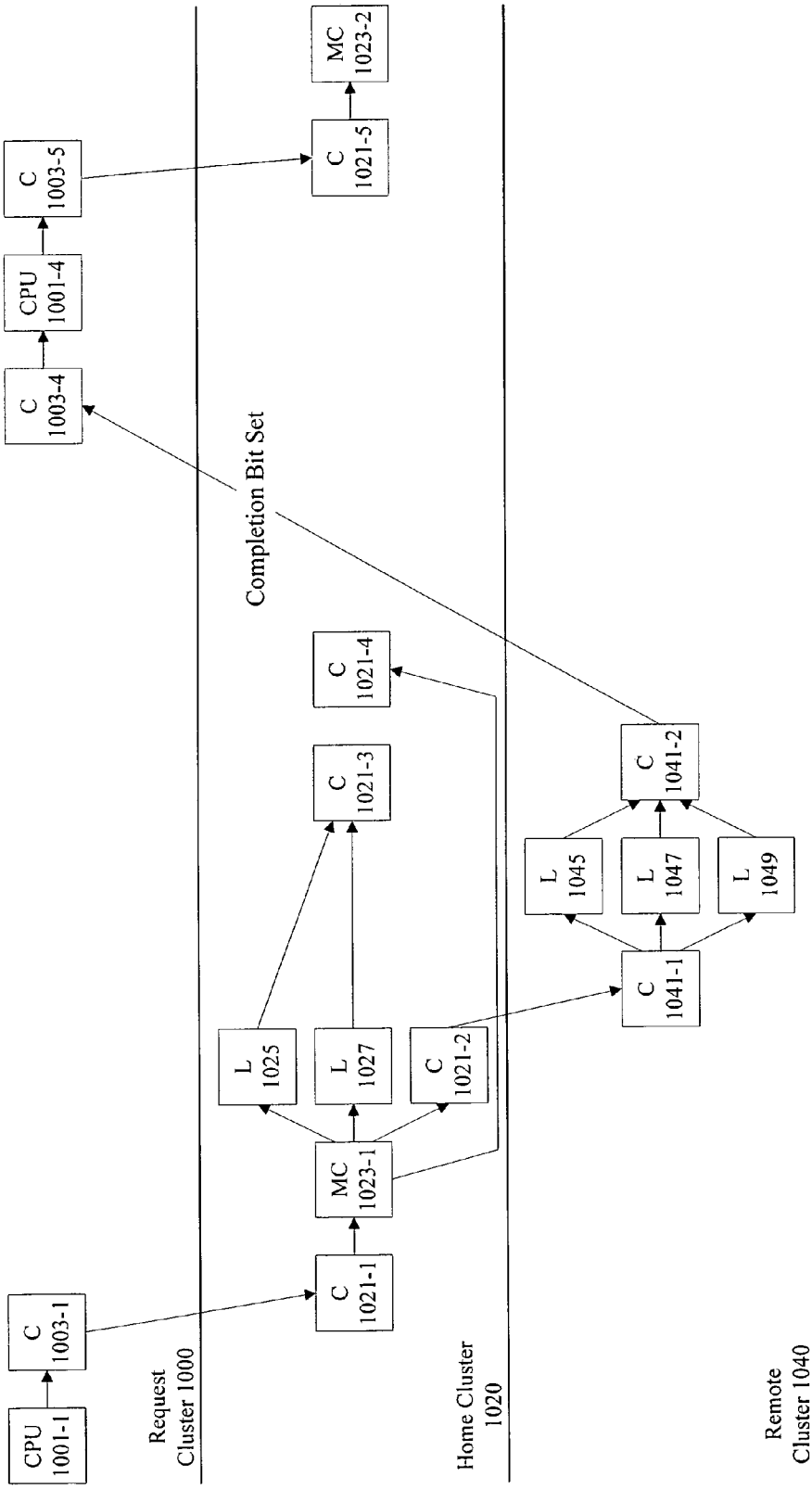

CPU 1001-1
C 1003-1
C 1021-1
MC 1023-1
L 1025
L 1027
C 1021-3
C 1021-4
C 1021-2
C 1003-4
CPU 1001-4
C 1003-5
C 1021-5
MC 1023-2
Completion Bit Set
C 1041-1
L 1045
L 1047
L 1049
C 1041-2
Request Cluster 1000
Home Cluster 1020
Remote Cluster 1040

Figure 11

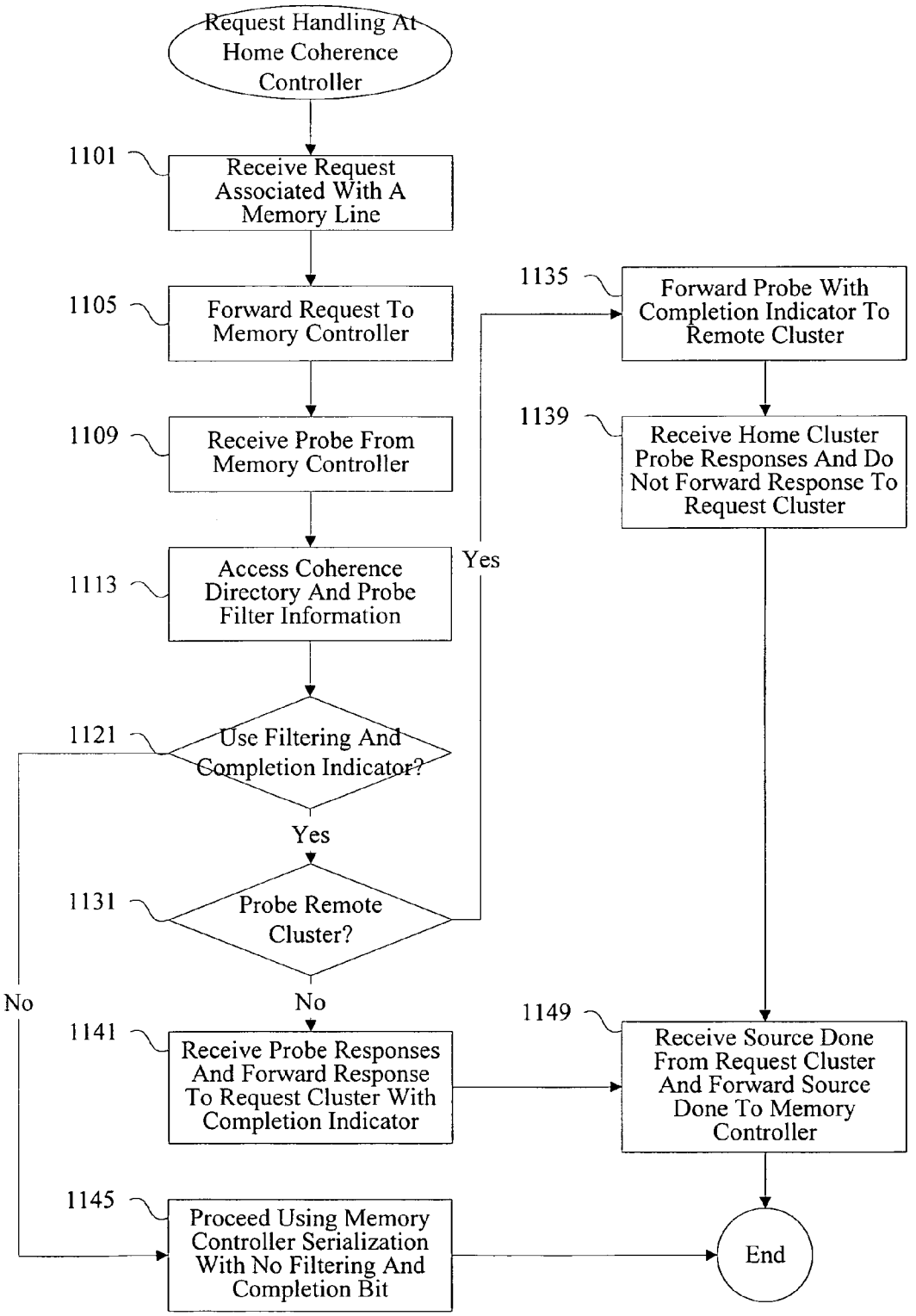

Request Handling At Home Coherence Controller
1101
Receive Request Associated With A Memory Line
1105
Forward Request To Memory Controller
1109
Receive Probe From Memory Controller
1113
Access Coherence Directory And Probe Filter Information
1121
Use Filtering And Completion Indicator?
Yes
No
1131
Probe Remote Cluster?
Yes
No
1135
Forward Probe With Completion Indicator To Remote Cluster
1139
Receive Home Cluster Probe Responses And Do Not Forward Response To Request Cluster
1141
Receive Probe Responses And Forward Response To Request Cluster With Completion Indicator
1149
Receive Source Done From Request Cluster And Forward Source Done To Memory Controller
1145
Proceed Using Memory Controller Serialization With No Filtering And Completion Bit
End

Figure 12

Memory Controller Filter Information 1221		
	Read Block [Read] 1223	Read Block Modify [Read/Write] 1225
Invalid 1231	Send request to target. (1201)	Send request to target. (1209)
Shared 1233	Send request to target. (1203)	Send request to target. (1211)
Owned 1235	Forward Probe To Owning Cluster. (1205)	Send request to target. (1213)
Modified 1237	Forward Probe To Modified Cluster. (1207)	Forward Probe To Modified Cluster. (1215)

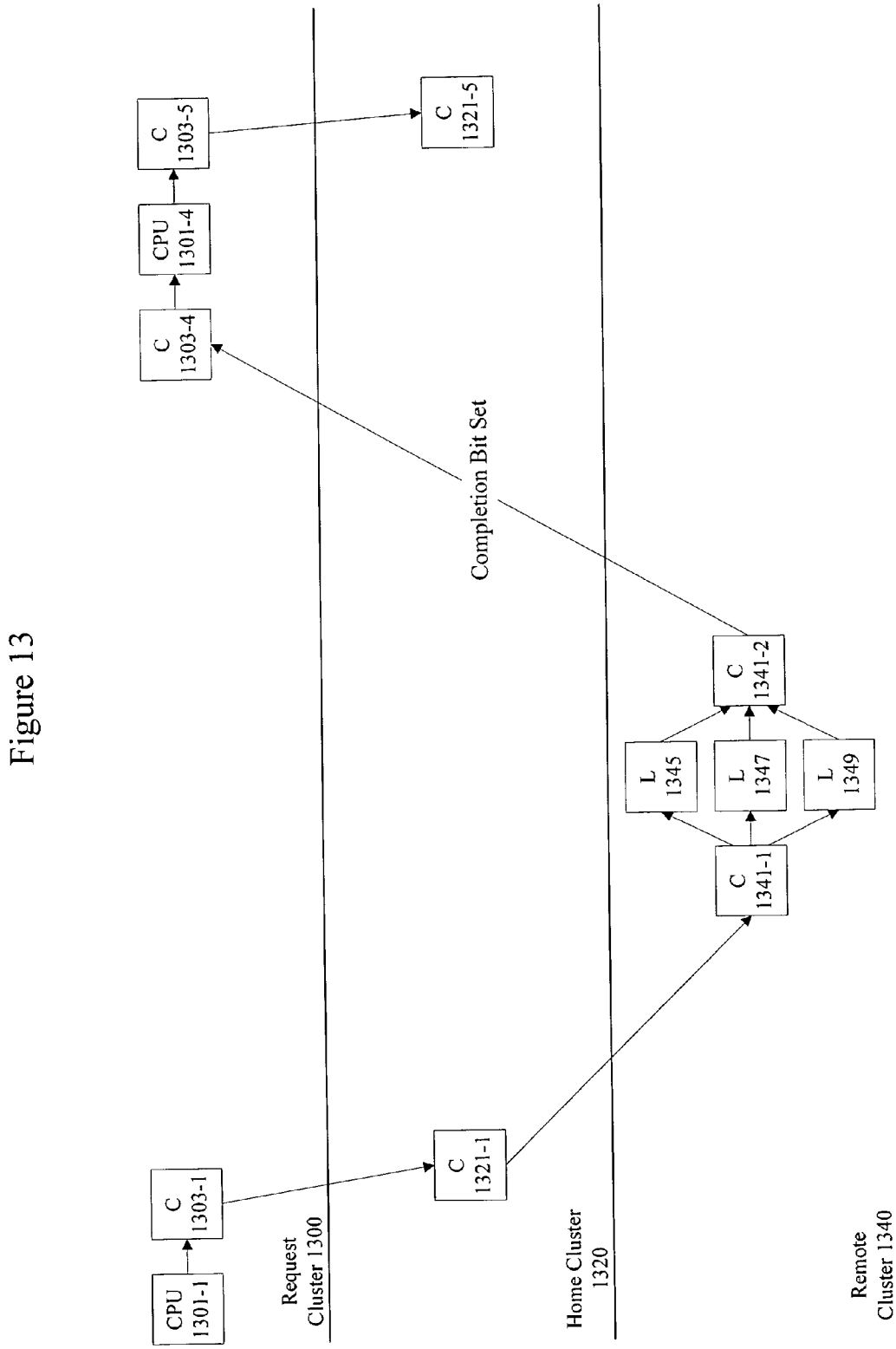
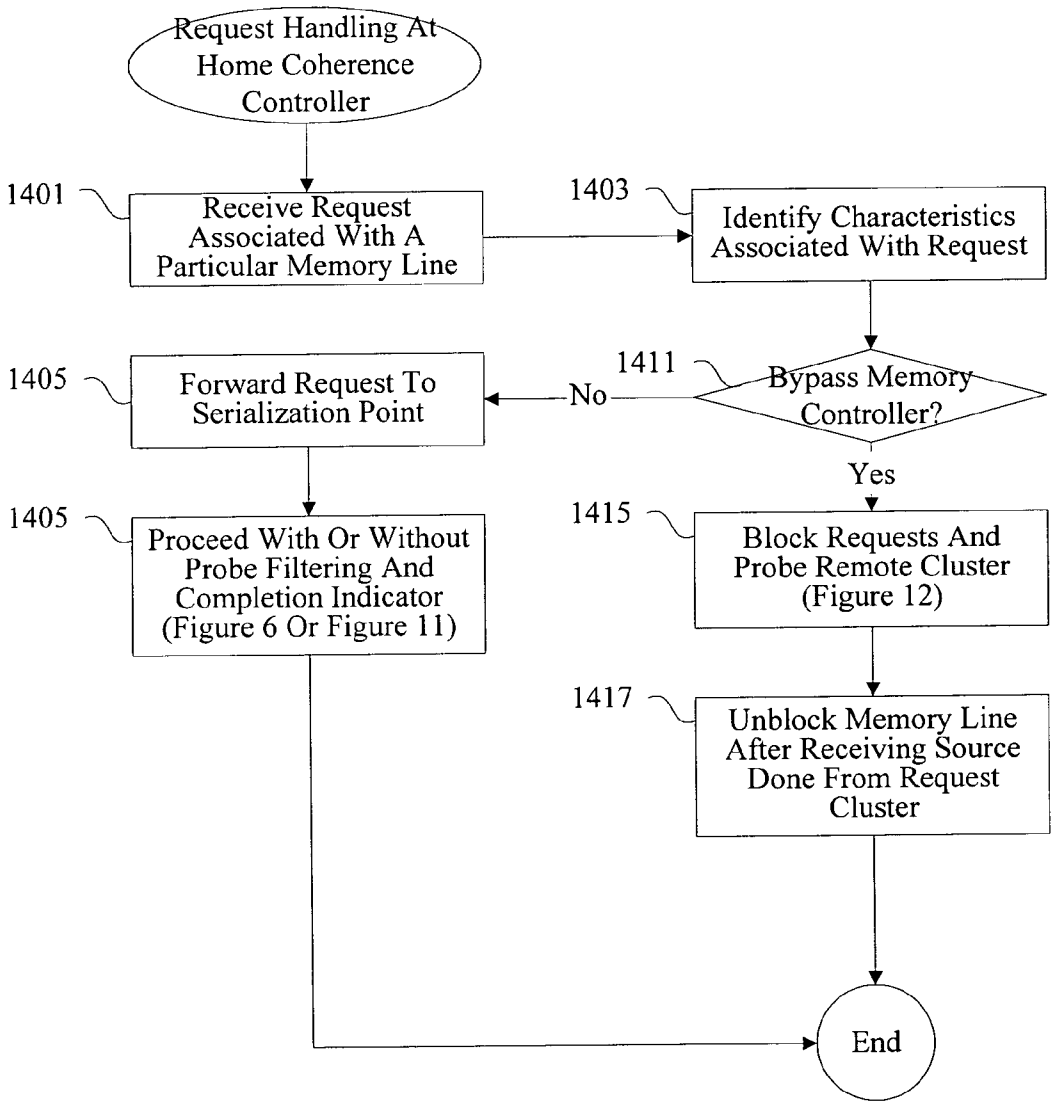

CPU 1301-1
C 1303-1
C 1321-1
C 1303-4
CPU 1301-4
C 1303-5
C 1321-5
Completion Bit Set
C 1341-1
L 1345
L 1347
L 1349
C 1341-2
Request Cluster 1300
Home Cluster 1320
Remote Cluster 1340

Figure 13

Figure 14


Request Handling At Home Coherence Controller
1401
Receive Request Associated With A Particular Memory Line
1403
Identify Characteristics Associated With Request
1411
Bypass Memory Controller?
No
Yes
1405
Forward Request To Serialization Point
1405
Proceed With Or Without Probe Filtering And Completion Indicator (Figure 6 Or Figure 11)
1415
Block Requests And Probe Remote Cluster (Figure 12)
1417
Unblock Memory Line After Receiving Source Done From Request Cluster
End

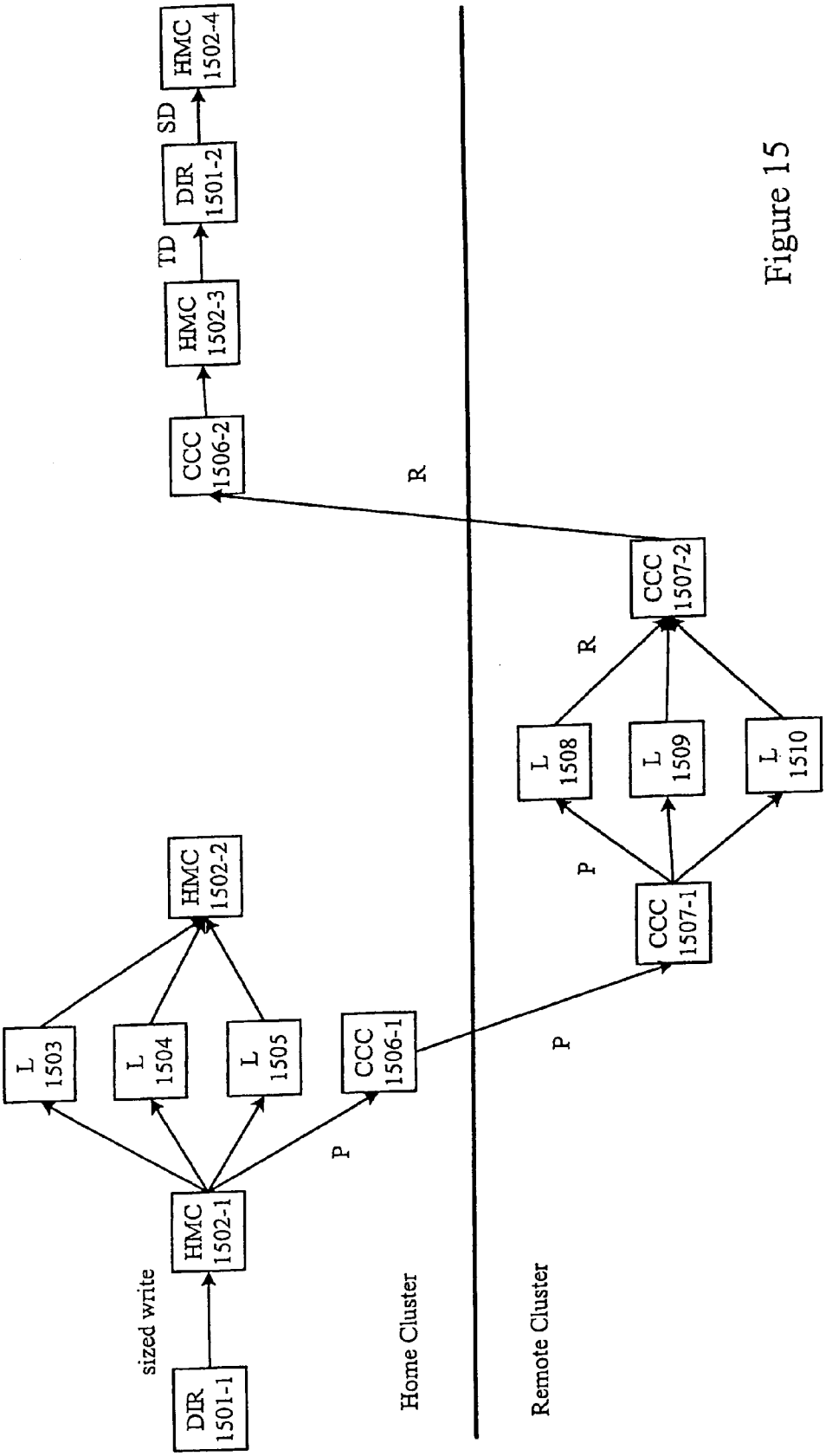

DIR 1501-1
HMC 1502-1
HMC 1502-2
HMC 1502-3
HMC 1502-4
DIR 1501-2
L 1503
L 1504
L 1505
L 1508
L 1509
L 1510
CCC 1506-1
CCC 1506-2
CCC 1507-1
CCC 1507-2
sized write
TD
SD
P
P
R
R
Home Cluster
Remote Cluster

Figure 15

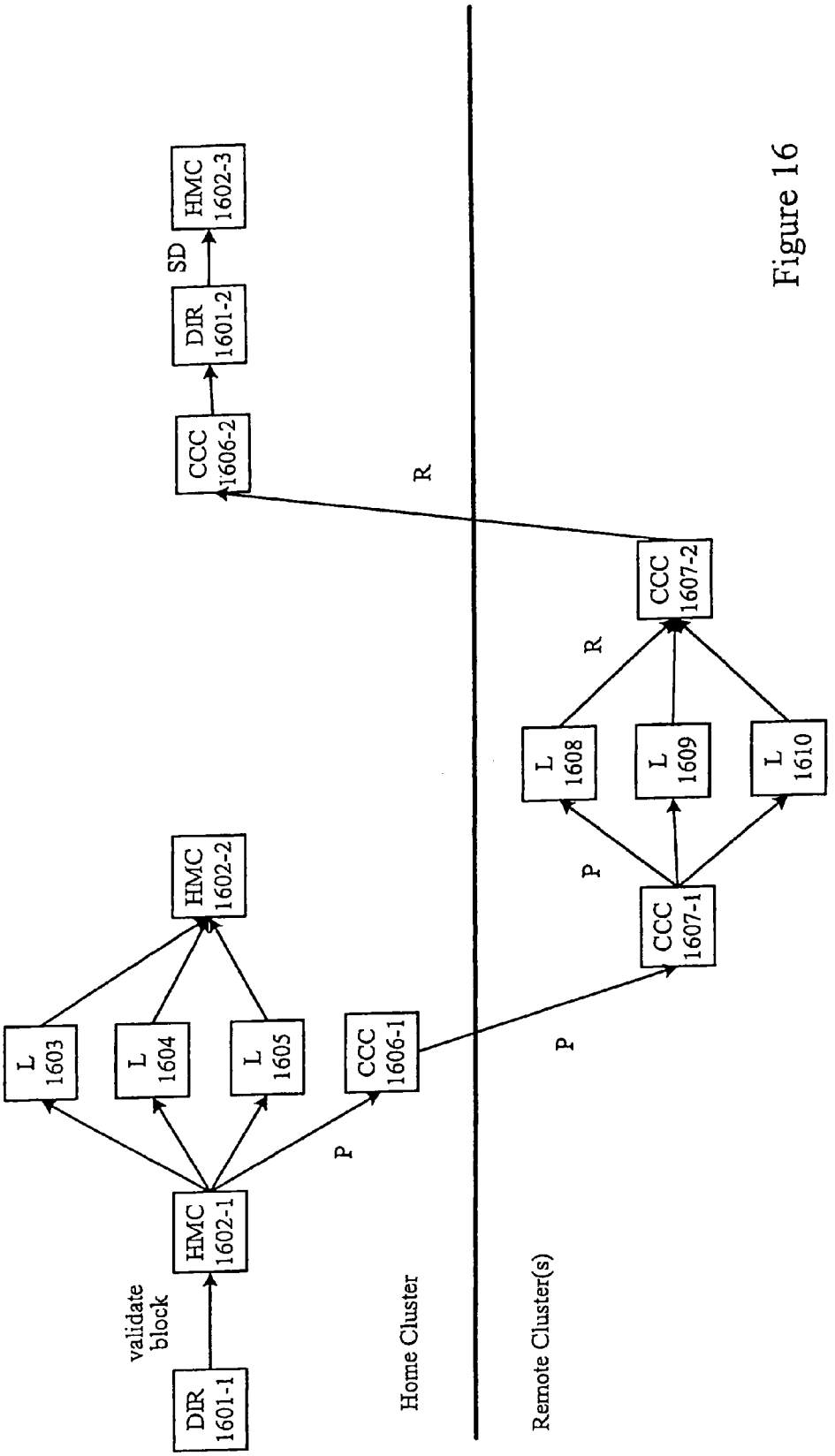

DIR 1601-1
validate block
HMC 1602-1
L 1603
L 1604
L 1605
HMC 1602-2
CCC 1606-1
P
Home Cluster
Remote Cluster(s)
P
CCC 1607-1
L 1608
L 1609
L 1610
R
CCC 1607-2
R
CCC 1606-2
DIR 1601-2
SD
HMC 1602-3

Figure 16

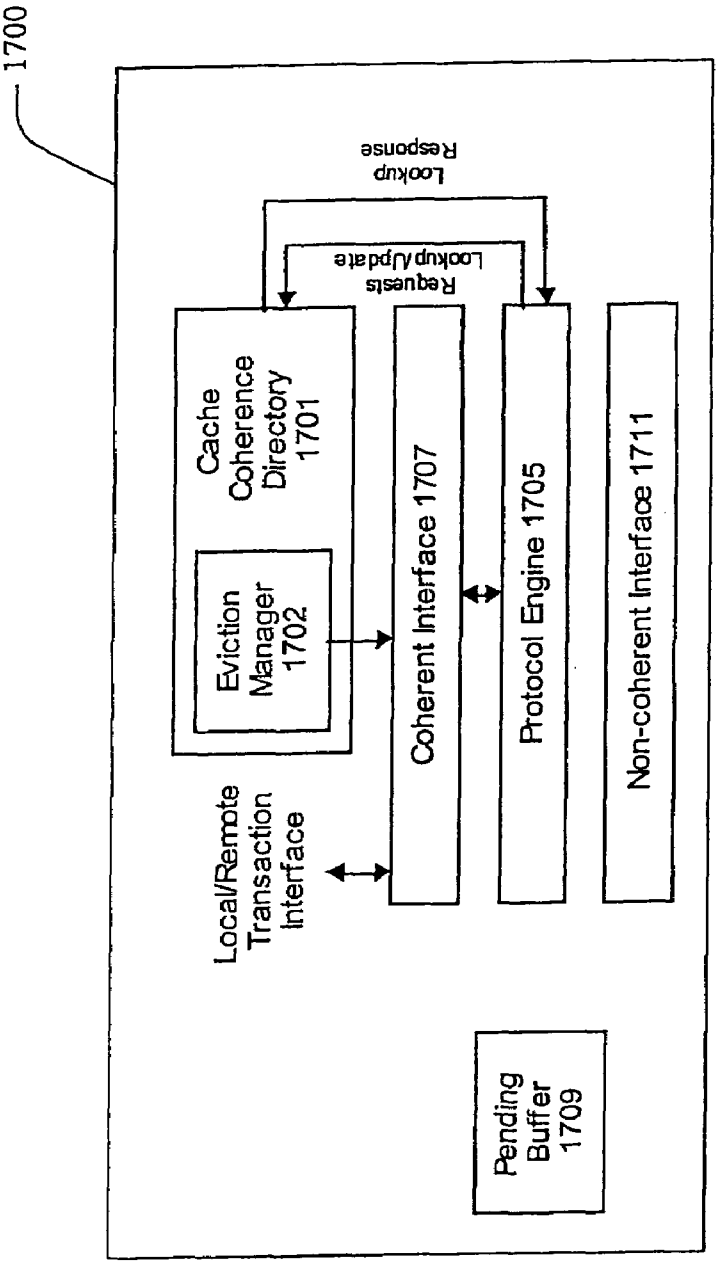

1700
Lookup Response
Requests Lookup/Update
Cache Coherence Directory 1701
Eviction Manager 1702
Coherent Interface 1707
Protocol Engine 1705
Non-coherent Interface 1711
Local/Remote Transaction Interface
Pending Buffer 1709

Figure 17

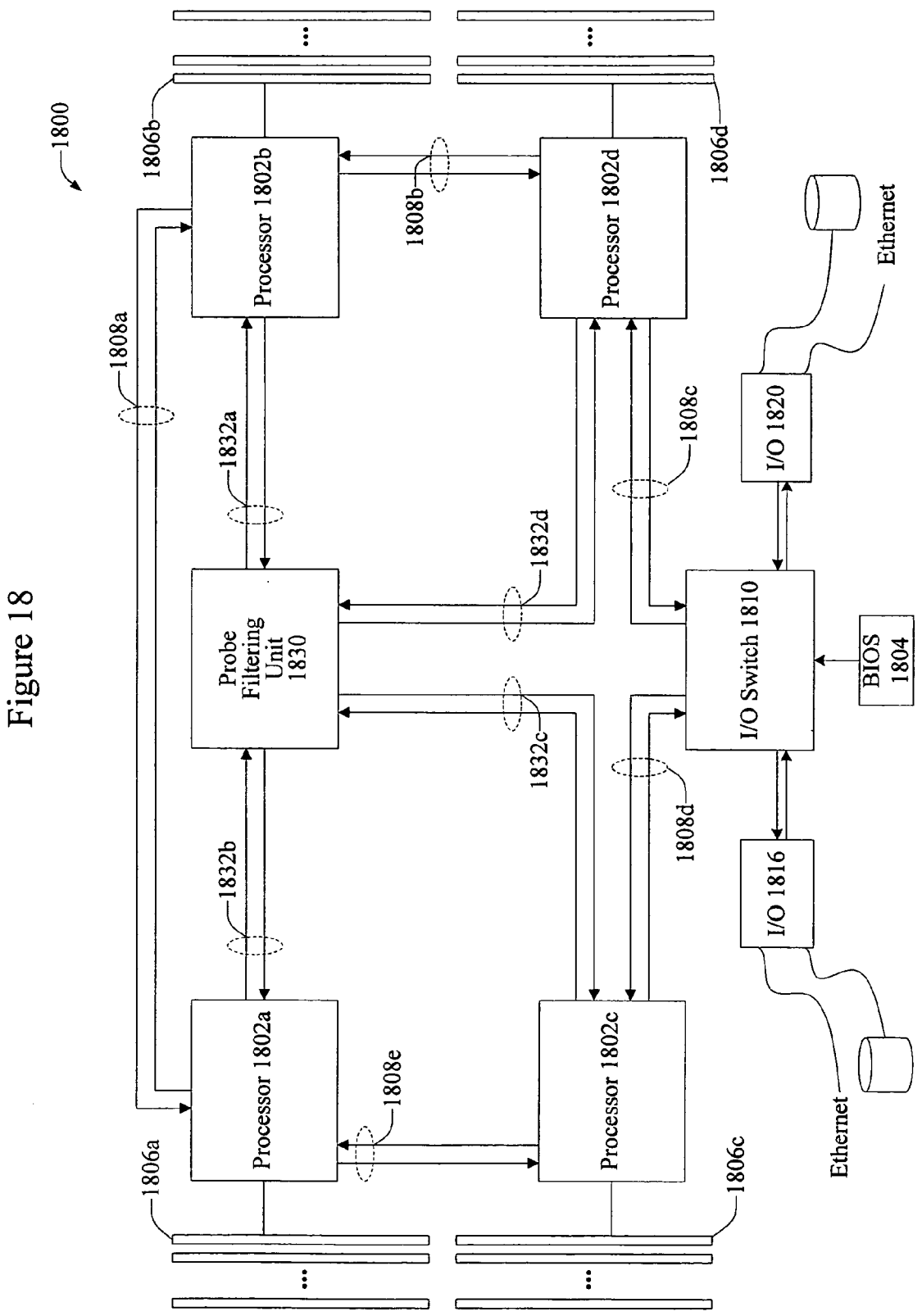

1800
1806b
1806d
Processor 1802b
Processor 1802d
1808b
1808a
1832a
1808c
Ethernet
I/O 1820
Probe Filtering Unit 1830
1832d
I/O Switch 1810
BIOS 1804
1832c
1808d
1832b
I/O 1816
Processor 1802a
Processor 1802c
1808e
1806a
1806c
Ethernet

Figure 18

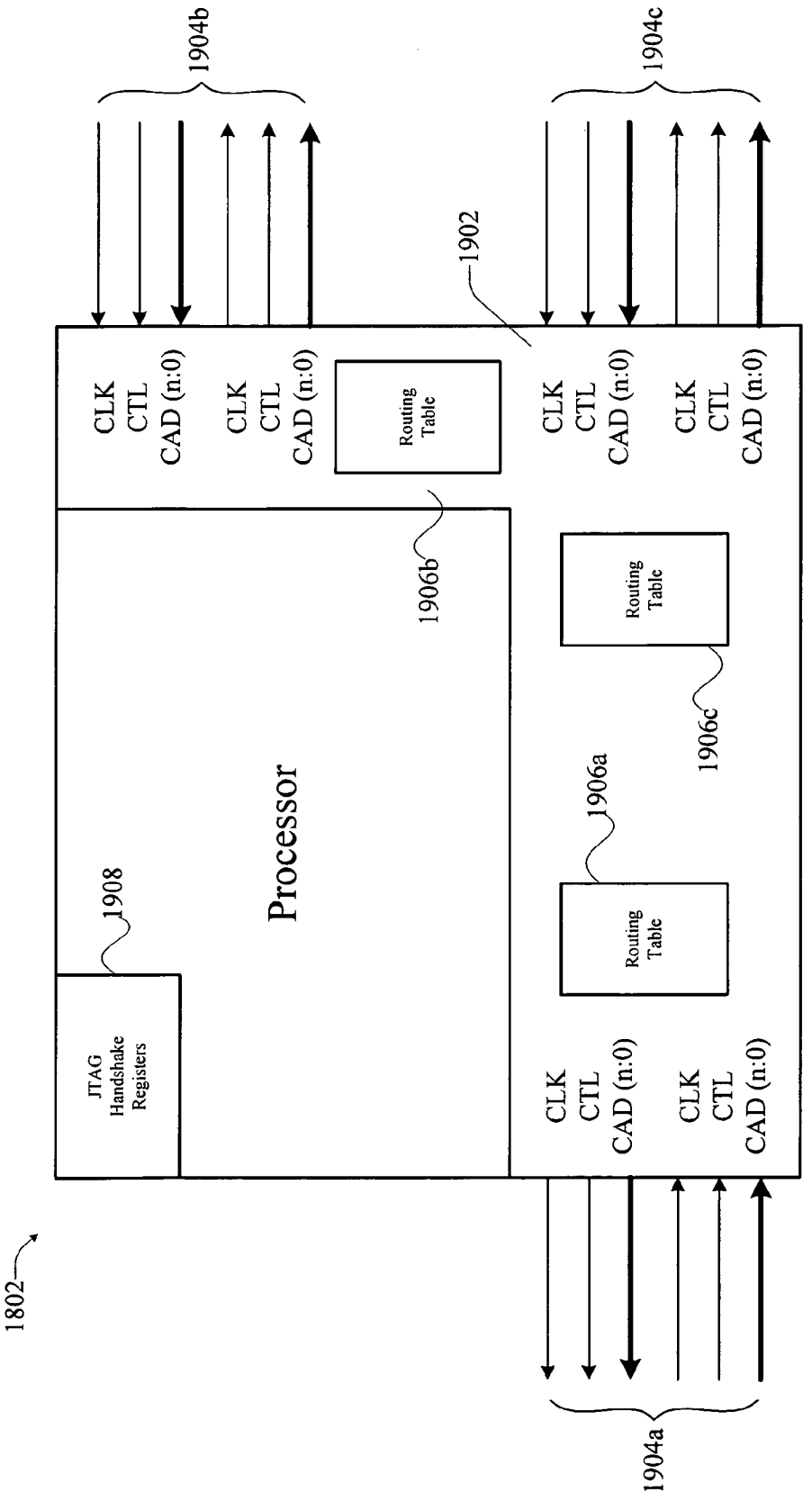

1802
1904b
1904c
1902
CLK
CTL
CAD (n:0)
CLK
CTL
CAD (n:0)
Routing Table
CLK
CTL
CAD (n:0)
CLK
CTL
CAD (n:0)
1906b
Routing Table
1906c
1906a
Routing Table
Processor
1908
JTAG Handshake Registers
CLK
CTL
CAD (n:0)
CLK
CTL
CAD (n:0)
1904a

Figure 19

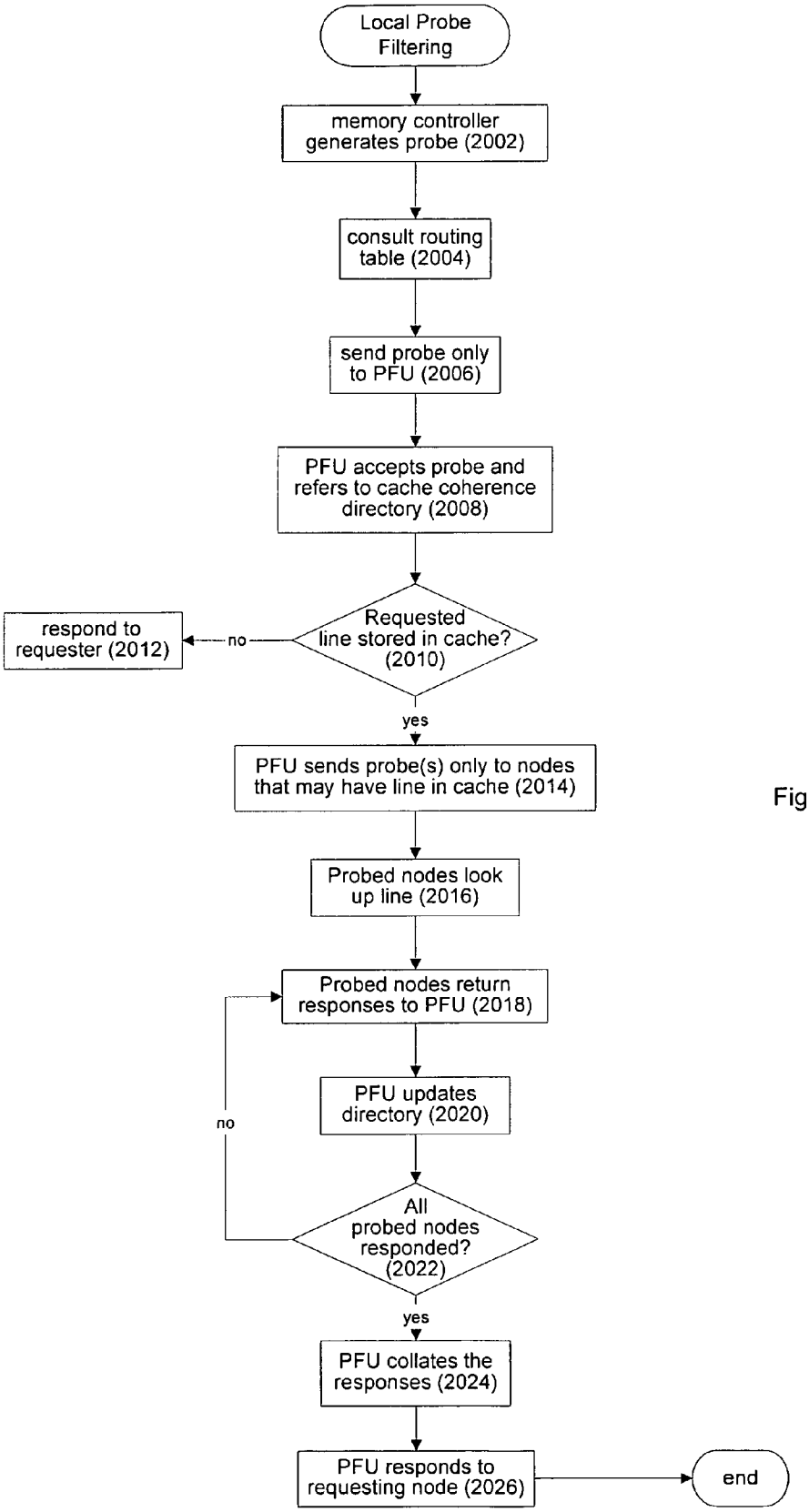

Local Probe Filtering
memory controller generates probe (2002)
consult routing table (2004)
send probe only to PFU (2006)
PFU accepts probe and refers to cache coherence directory (2008)
Requested line stored in cache? (2010)
no
respond to requester (2012)
yes
PFU sends probe(s) only to nodes that may have line in cache (2014)
Probed nodes look up line (2016)
Probed nodes return responses to PFU (2018)
PFU updates directory (2020)
All probed nodes responded? (2022)
no
yes
PFU collates the responses (2024)
PFU responds to requesting node (2026)
end

Figure 20

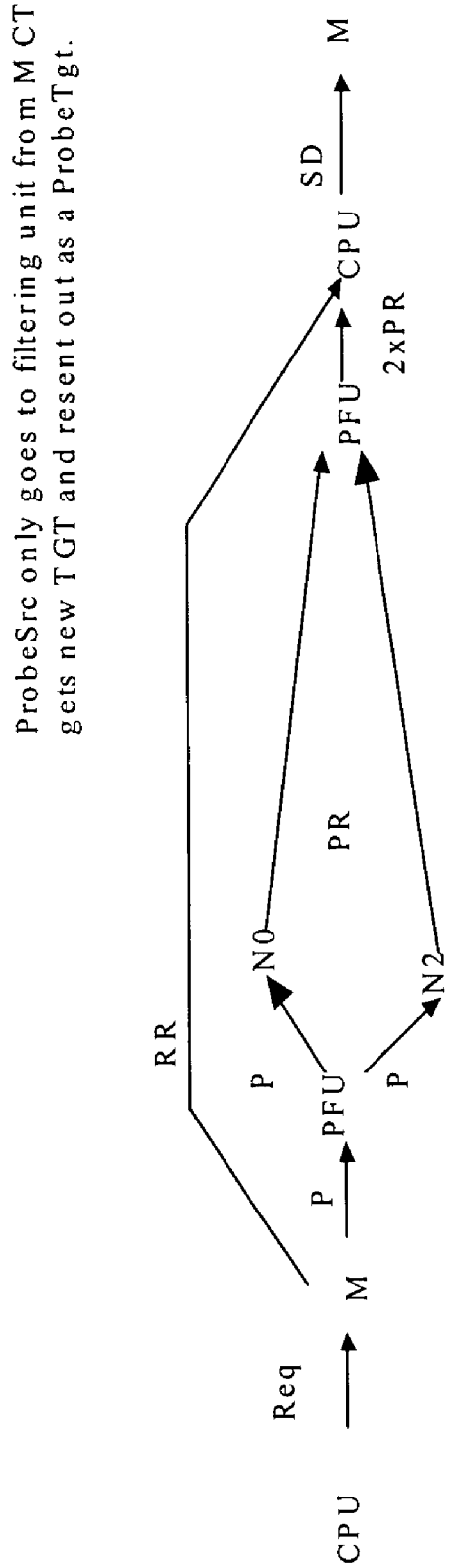

ProbeSrc only goes to filtering unit from M CT, gets new TGT and resent out as a ProbeTgt.
CPU
Req
M
P
PFU
RR
P
N0
PR
P
N2
PFU
2xPR
CPU
SD
M
N1
N3
Non probed nodes due to probe filtering are not aware of the transacto in.

Figure 21

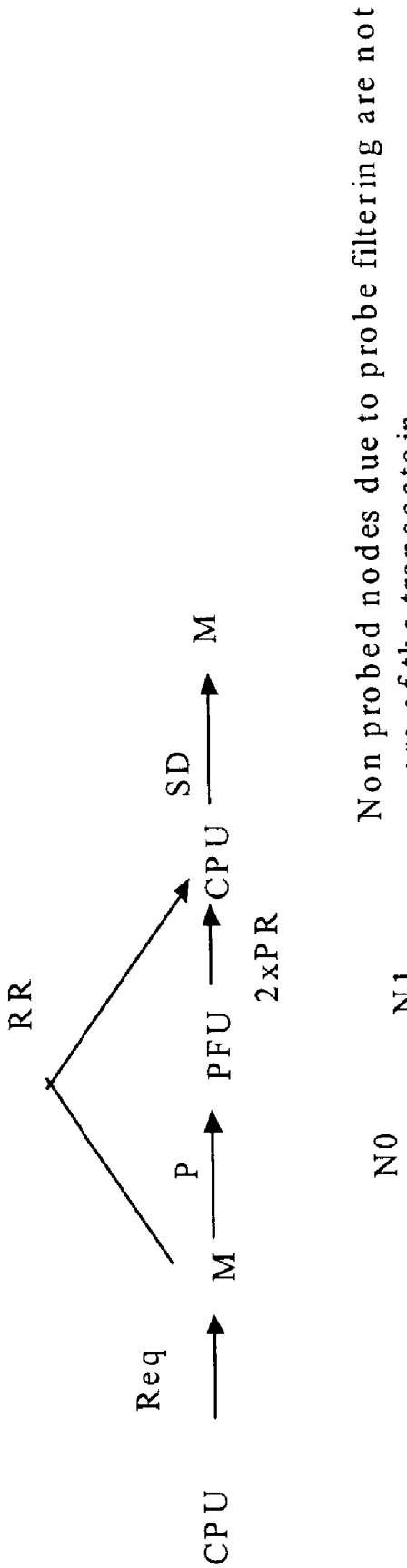

CPU
Req
M
P
PFU
2xPR
RR
CPU
SD
M
N0
N1
N2
N3
Non probed nodes due to probe filtering are not aware of the transactoin.

Figure 22

REDUCING PROBE TRAFFIC IN MULTIPROCESSOR SYSTEMS

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation-in-part of and claims priority under 35 U.S.C. 120 to U.S. patent application Ser. No. 10/288,347 now U.S. Pat. No. 7,003,633 for METHODS AND APPARATUS FOR MANAGING PROBE REQUESTS filed on Nov. 4, 2002 the entire disclosure of which is incorporated herein by reference for all purposes. The subject matter described in the present application is also related to U.S. patent application Ser. No. 10/288,399 now U.S. Pat. No. 7,103,726 for METHODS AND APPARATUS FOR MANAGING PROBE REQUESTS filed on Nov. 4, 2002, the entire disclosure of which is incorporated herein by reference for all purposes.

BACKGROUND OF THE INVENTION

The present invention generally relates to accessing data in a multiple processor system. More specifically, the present invention provides techniques for reducing memory transaction traffic in a multiple processor system.

Data access in multiple processor systems can raise issues relating to cache coherency. Conventional multiple processor computer systems have processors coupled to a system memory through a shared bus. In order to optimize access to data in the system memory, individual processors are typically designed to work with cache memory. In one example, each processor has a cache that is loaded with data that the processor frequently accesses. The cache is read or written by a processor. However, cache coherency problems arise because multiple copies of the same data can co-exist in systems having multiple processors and multiple cache memories. For example, a frequently accessed data block corresponding to a memory line may be loaded into the cache of two different processors. In one example, if both processors attempt to write new values into the data block at the same time, different data values may result. One value may be written into the first cache while a different value is written into the second cache. A system might then be unable to determine what value to write through to system memory.

A variety of cache coherency mechanisms have been developed to address such problems in multiprocessor systems. One solution is to simply force all processor writes to go through to memory immediately and bypass the associated cache. The write requests can then be serialized before overwriting a system memory line. However, bypassing the cache significantly decreases efficiency gained by using a cache. Other cache coherency mechanisms have been developed for specific architectures. In a shared bus architecture, each processor checks or snoops on the bus to determine whether it can read or write a shared cache block. In one example, a processor only writes an object when it owns or has exclusive access to the object. Each corresponding cache object is then updated to allow processors access to the most recent version of the object.

Bus arbitration is used when both processors attempt to write the same shared data block in the same clock cycle. Bus arbitration logic decides which processor gets the bus first. Although, cache coherency mechanisms such as bus arbitration are effective, using a shared bus limits the number of processors that can be implemented in a single system with a single memory space.

Other multiprocessor schemes involve individual processor, cache, and memory systems connected to other processors, cache, and memory systems using a network backbone such as Ethernet or Token Ring. Multiprocessor schemes involving separate computer systems each with its own address space can avoid many cache coherency problems because each processor has its own associated memory and cache. When one processor wishes to access data on a remote computing system, communication is explicit. Messages are sent to move data to another processor and messages are received to accept data from another processor using standard network protocols such as TCP/IP. Multiprocessor systems using explicit communication including transactions such as sends and receives are referred to as systems using multiple private memories. By contrast, multiprocessor system using implicit communication including transactions such as loads and stores are referred to herein as using a single address space.

Multiprocessor schemes using separate computer systems allow more processors to be interconnected while minimizing cache coherency problems. However, it would take substantially more time to access data held by a remote processor using a network infrastructure than it would take to access data held by a processor coupled to a system bus. Furthermore, valuable network bandwidth would be consumed moving data to the proper processors. This can negatively impact both processor and network performance.

Performance limitations have led to the development of a point-to-point architecture for connecting processors in a system with a single memory space. In one example, individual processors can be directly connected to each other through a plurality of point-to-point links to form a cluster of processors. Separate clusters of processors can also be connected. The point-to-point links significantly increase the bandwidth for coprocessing and multiprocessing functions. However, using a point-to-point architecture to connect multiple processors in a multiple cluster system sharing a single memory space presents its own problems.

Consequently, it is desirable to provide techniques for improving data access and cache coherency in systems having multiple processors connected using point-to-point links.

SUMMARY OF THE INVENTION

According to the present invention, various techniques are provided for reducing traffic relating to memory transactions in multi-processor systems. According to various specific embodiments, a computer system having a plurality of processing nodes interconnected by a first point-to-point architecture is provided. Each processing node has a cache memory associated therewith. A probe filtering unit is operable to receive probes corresponding to memory lines from the processing nodes and to transmit the probes only to selected ones of the processing nodes with reference to probe filtering information. The probe filtering information is representative of states associated with selected ones of the cache memories.

According to other embodiments, methods and apparatus are provided for reducing probe traffic in a computer system comprising a plurality of processing nodes interconnected by a first point-to-point architecture. A probe corresponding to a memory line is transmitted from a first one of the processing nodes only to a probe filtering unit. The probe is evaluated with the probe filtering unit to determine whether a valid copy of the memory line is in any of the cache memories. The evaluation is done with reference to probe

filtering information associated with the probe filtering unit and representative of states associated with selected ones of the cache memories. The probe is transmitted from the probe filtering unit only to selected ones of the processing nodes identified by the evaluating. Probe responses from the selected processing nodes are accumulated by the probe filtering unit. Only the probe filtering unit responds to the first processing node.

A further understanding of the nature and advantages of the present invention may be realized by reference to the remaining portions of the specification and the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention may best be understood by reference to the following description taken in conjunction with the accompanying drawings, which are illustrative of specific embodiments of the present invention.

FIGS. **1**A and **1**B are diagrammatic representation depicting a system having multiple clusters.

FIG. **2** is a diagrammatic representation of a cluster having a plurality of processors.

FIG. **3** is a diagrammatic representation of a cache coherence controller.

FIG. **4** is a diagrammatic representation showing a transaction flow for a data access request from a processor in a single cluster.

FIG. **5**A-**5**D are diagrammatic representations showing cache coherence controller functionality.

FIG. **6** is a diagrammatic representation depicting a transaction flow for a request with multiple probe responses.

FIG. **7** is a diagrammatic representation showing a cache coherence directory.

FIG. **8** is a diagrammatic representation showing probe filter information that can be used to reduce the number of probes transmitted to various clusters.

FIG. **9** is a diagrammatic representation showing a transaction flow for probing of a home cluster without probing of other clusters.

FIG. **10** is a diagrammatic representation showing a transaction flow for probing of a single remote cluster.

FIG. **11** is a flow process diagram showing the handling of a request with probe filter information.

FIG. **12** is a diagrammatic representation showing memory controller filter information.

FIG. **13** is a diagrammatic representation showing a transaction flow for probing a single remote cluster without probing a home cluster.

FIG. **14** is a flow process diagram showing the handling of a request at a home cluster cache coherence controller using memory controller filter information.

FIG. **15** is a diagrammatic representation showing a transaction flow for a cache coherence directory eviction of an entry corresponding to a dirty memory line.

FIG. **16** is a diagrammatic representation showing a transaction flow for a cache coherence directory eviction of an entry corresponding to a clean memory line.

FIG. **17** is a diagrammatic representation of a cache coherence controller according to a specific embodiment of the invention.

FIG. **18** is a diagrammatic representation of a cluster having a plurality of processing nodes and a probe filtering unit.

FIG. **19** is an exemplary representation of a processing node.

FIG. **20** is a flowchart illustrating local probe filtering according to a specific embodiment of the invention.

FIG. **21** is a diagrammatic representation of a transaction flow in which local probe filtering is facilitated according to a specific embodiment of the invention.

FIG. **22** is a diagrammatic representation of another transaction flow in which local probe filtering is facilitated according to a specific embodiment of the invention.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Reference will now be made in detail to some specific embodiments of the invention including the best modes contemplated by the inventors for carrying out the invention. Examples of these specific embodiments are illustrated in the accompanying drawings. While the invention is described in conjunction with these specific embodiments, it will be understood that it is not intended to limit the invention to the described embodiments. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims. Multi-processor architectures having point-to-point communication among their processors are suitable for implementing specific embodiments of the present invention. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. Well-known process operations have not been described in detail in order not to unnecessarily obscure the present invention. Furthermore, the present application's reference to a particular singular entity includes that possibility that the methods and apparatus of the present invention can be implemented using more than one entity, unless the context clearly dictates otherwise.

According to various embodiments, techniques are provided for increasing data access efficiency in a multiple processor system. In a point-to-point architecture, a cluster of processors includes multiple processors directly connected to each other through point-to-point links. By using point-to-point links instead of a conventional shared bus or external network, multiple processors are used efficiently in a system sharing the same memory space. Processing and network efficiency are also improved by avoiding many of the bandwidth and latency limitations of conventional bus and external network based multiprocessor architectures. According to various embodiments, however, linearly increasing the number of processors in a point-to-point architecture leads to an exponential increase in the number of links used to connect the multiple processors. In order to reduce the number of links used and to further modularize a multiprocessor system using a point-to-point architecture, multiple clusters may be used.

According to some embodiments, multiple processor clusters are interconnected using a point-to-point architecture. Each cluster of processors includes a cache coherence controller used to handle communications between clusters. In one embodiment, the point-to-point architecture used to connect processors are used to connect clusters as well.

By using a cache coherence controller, multiple cluster systems can be built using processors that may not necessarily support multiple clusters. Such a multiple cluster system can be built by using a cache coherence controller to represent non-local nodes in local transactions so that local nodes do not need to be aware of the existence of nodes outside of the local cluster. More detail on the cache coherence controller will be provided below.

In a single cluster system, cache coherency can be maintained by sending all data access requests through a serialization point. Any mechanism for ordering data access requests (also referred to herein as requests and memory requests) is referred to herein as a serialization point. One example of a serialization point is a memory controller. Various processors in the single cluster system send data access requests to one or more memory controllers. In one example, each memory controller is configured to serialize or lock the data access requests so that only one data access request for a given memory line is allowed at any particular time. If another processor attempts to access the same memory line, the data access attempt is blocked until the memory line is unlocked. The memory controller allows cache coherency to be maintained in a multiple processor, single cluster system.

A serialization point can also be used in a multiple processor, multiple cluster system where the processors in the various clusters share a single address space. By using a single address space, internal point-to-point links can be used to significantly improve intercluster communication over traditional external network based multiple cluster systems. Various processors in various clusters send data access requests to a memory controller associated with a particular cluster such as a home cluster. The memory controller can similarly serialize all data requests from the different clusters. However, a serialization point in a multiple processor, multiple cluster system may not be as efficient as a serialization point in a multiple processor, single cluster system. That is, delay resulting from factors such as latency from transmitting between clusters can adversely affect the response times for various data access requests. It should be noted that delay also results from the use of probes in a multiple processor environment.

Although delay in intercluster transactions in an architecture using a shared memory space is significantly less than the delay in conventional message passing environments using external networks such as Ethernet or Token Ring, even minimal delay is a significant factor. In some applications, there may be millions of data access requests from a processor in a fraction of a second. Any delay can adversely impact processor performance.

According to various embodiments, probe management is used to increase the efficiency of accessing data in a multiple processor, multiple cluster system. A mechanism for eliciting a response from a node to maintain cache coherency in a system is referred to herein as a probe. In one example, a mechanism for snooping a cache is referred to as a probe. A response to a probe can be directed to the source or target of the initiating request. Any mechanism for filtering or reducing the number of probes and requests transmitted to various nodes is referred to herein as managing probes. In one example, managing probes entails characterizing a request to determine if a probe can be transmitted to a reduced number of entities.

In typical implementations, requests are sent to a memory controller that broadcasts probes to various nodes in a system. In such a system, no knowledge of the cache line state needs to be maintained by the memory controller. All nodes in the system are probed and the request cluster receives a response from each node. In a system with a coherence directory, state information associated with various memory lines can be used to reduce the number of transactions. Any mechanism for maintaining state information associated with various memory lines is referred to herein as a coherence directory. According to some embodiments, a coherence directory includes information for memory lines in a local cluster that are cached in a remote cluster. According to others, such a directory includes information for locally cached lines. According to various embodiments, a coherence directory is used to reduce the number of probes to remote quads by inferring the state of local caches. According to some embodiments, such a directory mechanism is used in a single cluster system or within a cluster in a multi-cluster system to reduce the number of probes within a cluster.

FIG. **1**A is a diagrammatic representation of one example of a multiple cluster, multiple processor system that can use the techniques of the present invention. Each processing cluster **101**, **103**, **105**, and **107** can include a plurality of processors. The processing clusters **101**, **103**, **105**, and **107** are connected to each other through point-to-point links **111***a-f*. In one embodiment, the multiple processors in the multiple cluster architecture shown in FIG. **1**A share the same memory space. In this example, the point-to-point links **111***a-f* are internal system connections that are used in place of a traditional front-side bus to connect the multiple processors in the multiple clusters **101**, **103**, **105**, and **107**. The point-to-point links may support any point-to-point protocol.

FIG. **1**B is a diagrammatic representation of another example of a multiple cluster, multiple processor system that can use the techniques of the present invention. Each processing cluster **121**, **123**, **125**, and **127** can be coupled to a switch **131** through point-to-point links **141***a-d*. It should be noted that using a switch and point-to-point links allows implementation with fewer point-to-point links when connecting multiple clusters in the system. A switch **131** can include a processor with a coherence protocol interface. According to various implementations, a multicluster system shown in FIG. **1**A is expanded using a switch **131** as shown in FIG. **1**B.

FIG. **2** is a diagrammatic representation of a multiple processor cluster, such as the cluster **101** shown in FIG. **1**A. Cluster **200** includes processors **202***a*-**202***d*, one or more Basic I/O systems (BIOS) **204**, a memory subsystem comprising memory banks **206***a*-**206***d*, point-to-point communication links **208***a*-**208***e*, and a service processor **212**. The point-to-point communication links are configured to allow interconnections between processors **202***a*-**202***d*, I/O switch **210**, and cache coherence controller **230**. The service processor **212** is configured to allow communications with processors **202***a*-**202***d*, I/O switch **210**, and cache coherence controller **230** via a JTAG interface represented in FIG. **2** by links **214***a*-**214***f*. It should be noted that other interfaces are supported. It should also be noted that in some implementations, a service processor is not included in multiple processor clusters. I/O switch **210** connects the rest of the system to I/O adapters **216** and **220**. It should further be noted that the terms node and processor are often used interchangeably herein. However, it should be understood that according to various implementations, a node (e.g., processors **202***a*-**202***d*) may comprise multiple sub-units, e.g., CPUs, memory controllers, I/O bridges, etc.

According to specific embodiments, the service processor of the present invention has the intelligence to partition system resources according to a previously specified partitioning schema. The partitioning can be achieved through direct manipulation of routing tables associated with the system processors by the service processor which is made possible by the point-to-point communication infrastructure. The routing tables are used to control and isolate various system resources, the connections between which are defined therein.

The processors **202***a-d* are also coupled to a cache coherence controller **230** through point-to-point links **232***a-d*. Any mechanism or apparatus that can be used to provide communication between multiple processor clusters while maintaining cache coherence is referred to herein as a cache coherence controller. The cache coherence controller **230** can be coupled to cache coherence controllers associated with other multiprocessor clusters. It should be noted that there can be more than one cache coherence controller in one cluster. The cache coherence controller **230** communicates with both processors **202***a-d* as well as remote clusters using a point-to-point protocol.

More generally, it should be understood that the specific architecture shown in FIG. **2** is merely exemplary and that embodiments of the present invention are contemplated having different configurations and resource interconnections, and a variety of alternatives for each of the system resources shown. However, for purpose of illustration, specific details of server **200** will be assumed. For example, most of the resources shown in FIG. **2** are assumed to reside on a single electronic assembly. In addition, memory banks **206***a*-**206***d* may comprise double data rate (DDR) memory which is physically provided as dual in-line memory modules (DIMMs). I/O adapter **216** may be, for example, an ultra direct memory access (UDMA) controller or a small computer system interface (SCSI) controller which provides access to a permanent storage device. I/O adapter **220** may be an Ethernet card adapted to provide communications with a network such as, for example, a local area network (LAN) or the Internet.

According to a specific embodiment and as shown in FIG. **2**, both of I/O adapters **216** and **220** provide symmetric **1/0** access. That is, each provides access to equivalent sets of I/O. As will be understood, such a configuration would facilitate a partitioning scheme in which multiple partitions have access to the same types of I/O. However, it should also be understood that embodiments are envisioned in which partitions without I/O are created. For example, a partition including one or more processors and associated memory resources, i.e., a memory complex, could be created for the purpose of testing the memory complex.

According to one embodiment, service processor **212** is a Motorola MPC855T microprocessor which includes integrated chipset functions. The cache coherence controller **230** is an Application Specific Integrated Circuit (ASIC) supporting the local point-to-point coherence protocol. The cache coherence controller **230** can also be configured to handle a non-coherent protocol to allow communication with I/O devices. In one embodiment, the cache coherence controller **230** is a specially configured programmable chip such as a programmable logic device or a field programmable gate array.

FIG. **3** is a diagrammatic representation of one example of a cache coherence controller **230**. According to various embodiments, the cache coherence controller includes a protocol engine **305** configured to handle packets such as probes and requests received from processors in various clusters of a multiprocessor system. The functionality of the protocol engine **305** can be partitioned across several engines to improve performance. In one example, partitioning is done based on packet type (request, probe and response), direction (incoming and outgoing), or transaction flow (request flows, probe flows, etc).

The protocol engine **305** has access to a pending buffer **309** that allows the cache coherence controller to track transactions such as recent requests and probes and associate the transactions with specific processors. Transaction information maintained in the pending buffer **309** can include transaction destination nodes, the addresses of requests for subsequent collision detection and protocol optimizations, response information, tags, and state information.

The cache coherence controller has an interface such as a coherent protocol interface **307** that allows the cache coherence controller to communicate with other processors in the cluster as well as external processor clusters. The cache coherence controller can also include other interfaces such as a non-coherent protocol interface **311** for communicating with I/O devices. According to various embodiments, each interface **307** and **311** is implemented either as a full crossbar or as separate receive and transmit units using components such as multiplexers and buffers. It should be noted, however, that the cache coherence controller **230** does not necessarily need to provide both coherent and non-coherent interfaces. It should also be noted that a cache coherence controller in one cluster can communicate with a cache coherence controller in another cluster.

FIG. **4** is a diagrammatic representation showing the transactions for a cache request from a processor in a system having a single cluster without using a cache coherence controller or other probe management mechanism. A processor **401-1** sends an access request such as a read memory line request to a memory controller **403-1**. The memory controller **403-1** may be associated with this processor, another processor in the single cluster or may be a separate component such as an ASIC or specially configured Programmable Logic Device (PLD). To preserve cache coherence, only one processor is typically allowed to access a memory line corresponding to a shared address space at anyone given time. To prevent other processors from attempting to access the same memory line, the memory line can be locked by the memory controller **403-1**. All other requests to the same memory line are blocked or queued. Access by another processor is typically only allowed when the memory controller **403-1** unlocks the memory line.

The memory controller **403-1** then sends probes to the local cache memories **405**, **407**, and **409** to determine cache states. The local cache memories **405**, **407**, and **409** then in turn send probe responses to the same processor **401-2**. The memory controller **403-1** also sends an access response such as a read response to the same processor **401-3**. The processor **401-3** can then send a done response to the memory controller **403-2** to allow the memory controller **403-2** to unlock the memory line for subsequent requests. It should be noted that CPU **401-1**, CPU **401-2**, and CPU **401-3** refer to the same processor.

FIGS. **5**A-**5**D are diagrammatic representations depicting cache coherence controller operation. The use of a cache coherence controller in multiprocessor clusters allows the creation of a multiprocessor, multicluster coherent domain without affecting the functionality of local nodes in each cluster. In some instances, processors may only support a protocol that allows for a limited number of processors in a single cluster without allowing for multiple clusters. The cache coherence controller can be used to allow multiple clusters by making local processors believe that the non-local nodes are merely a one or more local nodes embodied in the cache coherence controller. In one example, the processors in a cluster do not need to be aware of processors in other clusters. Instead, the processors in the cluster communicate with the cache coherence controller as though the cache coherence controller were representing all non-local nodes. In addition, although generally a node may correspond to one or a plurality of resources (including, for example, a processor), it should be noted that the terms node

and processor are often used interchangeably herein. According to a particular implementation, a node comprises multiple sub-units, e.g., CPUs, memory controllers, I/O bridges, etc.

It should be noted that nodes in a remote cluster will be referred to herein as non-local nodes or as remote nodes. However, non-local nodes refer to nodes not in a request cluster generally and includes nodes in both a remote cluster and nodes in a home cluster. A cluster from which a data access or cache access request originates is referred to herein as a request cluster. A cluster containing a serialization point is referred to herein as a home cluster. Other clusters are referred to as remote clusters. The home cluster and the remote cluster are also referred to herein as non-local clusters.

FIG. **5**A shows the cache coherence controller acting as an aggregate remote cache. When a processor **501-1** generates a data access request to a local memory controller **503-1**, the cache coherence controller **509** accepts the probe from the local memory controller **503-1** and forwards it to non-local node portion **511**. It should be noted that a coherence protocol can contain several types of messages. In one example, a coherence protocol includes four types of messages; data or cache access requests, probes, responses or probe responses, and data packets. Data or cache access requests usually target the home node memory controller. Probes are used to query each cache in the system. The probe packet can carry information that allows the caches to properly transition the cache state for a specified line. Responses are used to carry probe response information and to allow nodes to inform other nodes of the state of a given transaction. Data packets carry request data for both write requests and read responses.

According to various embodiments, the memory address resides at the local memory controller. As noted above, nodes including processors and cache coherence controllers outside of a local cluster are referred to herein as non-local nodes. The cache coherence controller **509** then accumulates the response from the non-local nodes and sends a single response in the same manner that local nodes associated with cache blocks **505** and **507** send a single response to processor **501-2**. Local processors may expect a single probe response for every local node probed. The use of a cache coherence controller allows the local processors to operate without concern as to whether non-local nodes exist.

It should also be noted that components such as processor **501-1** and processor **501-2** refer herein to the same component at different points in time during a transaction sequence. For example, processor **501-1** can initiate a data access request and the same processor **501-2** can later receive probe responses resulting from the request.

FIG. **5**B shows the cache coherence controller acting as a probing agent pair. When the cache coherence controller **521-1** receives a probe from non-local nodes **531**, the cache coherence controller **521-1** accepts the probe and forwards the probe to local nodes associated with cache blocks **523**, **525**, and **527**. The cache coherence controller **521-2** then forwards a final response to the non-local node portion **531**. In this example, the cache coherence controller is both the source and the destination of the probes. The local nodes associated with cache blocks **523**, **525**, and **527** behave as if the cache coherence controller were a local processor with a local memory request.

FIG. **5**C shows the cache coherence controller acting as a remote memory. When a local processor **541-1** generates an access request that targets remote memory, the cache coherence controller **543-1** forwards the request to the non-local nodes **553**. When the remote request specifies local probing, the cache coherence controller **543-1** generates probes to local nodes and the probed nodes provide responses to the processor **541-2**. Once the cache coherence controller **543-1** has received data from the non-local node portion **553**, it forwards a read response to the processor **541-3**. The cache coherence controller also forwards the final response to the remote memory controller associated with non-local nodes **553**.

FIG. **5**D shows the cache coherence controller acting as a remote processor. When the cache coherence controller **561-1** at a first cluster receives a request from a processor in a second cluster, the cache coherence controller acts as a first cluster processor on behalf of the second cluster processor. The cache coherence controller **561-1** accepts the request from portion **575** and forwards it to a memory controller **563-1**. The cache coherence controller **561-2** then accumulates all probe responses as well as the data fetched and forwards the final response to the memory controller **563-2** as well as to non-local nodes **575**.

By allowing the cache coherence controller to act as an aggregate remote cache, probing agent pair, remote memory, and remote processor, multiple cluster systems can be built using processors that may not necessarily support multiple clusters. The cache coherence controller can be used to represent non-local nodes in local transactions so that local nodes do not need to be aware of the existence of nodes outside of the local cluster.

FIG. **6** is a diagrammatic representation depicting the transactions for a data request from a local processor sent to a non-local cluster using a cache coherence controller. The multicluster system includes a request cluster **600**, a home cluster **620**, and a remote cluster **640**. As noted above, the home cluster **620** and the remote cluster **640** as well as any other clusters excluding the request cluster **600** are referred to herein as non-local clusters. Processors and cache coherence controllers associated with local and non-local clusters are similarly referred to herein as local processors, local cache coherence controllers, non-local processors, and non-local cache coherence controllers, respectively.

According to various embodiments, processor **601-1** in a local cluster **600** sends a data access request such as a read request to a cache coherence controller **603-1**. The cache coherence controller **603-1** tracks the transaction in the pending buffer of FIG. **3** and forwards the request to a cache coherence controller **621 -1** in a home cluster **620**. The cache coherence controller **621-1** at the home cluster **620** receives the access request and tracks the request in its pending buffer. In one example, information associated with the requests are stored in the pending buffer. The cache coherence controller **621-1** forwards the access request to a memory controller **623-1** also associated with the home cluster **620**. At this point, the memory controller **623-1** locks the memory line associated with the request. In one example, the memory line is a unique address in the memory space shared by the multiple processors in the request cluster **600**, home cluster **620**, and the remote cluster **640**. The memory controller **623-1** generates a probe associated with the data access request and forwards the probe to local nodes associated with cache blocks **625** and **627** as well as to cache coherence controller **621-2**.

It should be noted that although messages associated with requests, probes, responses, and data are described as forwarded from one node to another, the messages themselves may contain variations. In one example, alterations are made to the messages to allow the multiple cluster architecture to be transparent to various local nodes. It should be noted that

write requests can be handled as well. In write requests, the targeted memory controller gathers responses and sends the responses to the processor when gathering is complete.

The cache coherence controller **641-1** associated with the remote cluster **640** receives a probe from cache coherence controller **621-2** and probes local nodes associated with cache blocks **645**, **647**, and **649**. Similarly, the cache coherence controller **603-2** associated with the request cluster **600** receives a probe and forwards the probe to local nodes associated with cache blocks **605**, **607**, and **609** to probe the cache blocks in the request cluster **600**. Processor **601-2** receives probe responses from the local nodes associated with cache blocks **605**, **607**, and **609**.

According to various embodiments, cache coherence controller **621-3** accumulates probe responses and sends the probe responses to cache coherence controller **603-3**, which in turn forwards the probe responses to the processor **601-3**. Cache coherence controller **621-4** also sends a read response to cache coherence controller **603-4**, which forwards the read response to processor **601-4**. While probes and probe responses carry information for maintaining cache coherency in the system, read responses can carry actual fetched data. After receiving the fetched data, processor **601-4** may send a source done response to cache coherence controller **603-5**. According to various embodiments, the transaction is now complete at the requesting cluster **600**. Cache coherence controller **603-5** forwards the source done message to cache coherence controller **621-5**. Cache coherence controller **621-5** in turn sends a source done message to memory controller **623-2**. Upon receiving the source done message, the memory controller **623-2** can unlock the memory line and the transaction at the home cluster **620** is now complete. Another processor can now access the unlocked memory line.

It should be noted that because the cache coherence controller **621-3** waits for remote cluster probe responses before sending a probe response to cache coherence controller **603-3**, delay is introduced into the system. According to various embodiments, probe responses are gathered at cache coherence controller **603-3**. By having remote clusters send probe responses through a home cluster, both home cluster probe responses and remote cluster probe responses can be delayed at the home cache coherence controller. In one example, remote cluster probe responses have to travel an additional hop in order to reach a request cluster. The latency for transmission of a probe response between a remote cluster and a request cluster may be substantially less than the latency for transmission of a probe response between a remote cluster and a request cluster through a home cluster. Home cluster probe responses are also delayed as a result of this added hop.

As will be appreciated by one of skill in the art, the specific transaction sequences involving requests, probes, and response messages can vary depending on the specific implementation. In one example, a cache coherence controller **621-3** may wait to receive a read response message from a memory controller **623-1** before transmitting both a probe response message and a read response message to a cache coherence controller **603-3**. In other examples, a cache coherence controller may be the actual processor generating the request. Some processors may operate as both a processor and as a cache coherence controller. Furthermore, various data access request messages, probes, and responses associated with reads and writes are contemplated. As noted above, any message for snooping a cache can be referred to as a probe. Similarly, any message for indicating to the memory controller that a memory line should be unlocked can be referred to as a source done message.

It should be noted that the transactions shown in FIG. **6** show examples of cache coherence controllers performing many different functions, including functions of remote processors, aggregate local caches, probing agent pairs, and remote memory as described with reference to FIGS. **5**A-**5**D.

The cache coherence controller **621-1** at the home cluster **620** is acting as a remote processor. When the cache coherence controller receives a request from a request cluster processor, the cache coherence controller is directed to act as the requesting processor on behalf of the request cluster processor. In this case, the cache coherence controller **621-1** accepts a forwarded request from processor **601-1** and sends it to the memory controller **623-1**, accumulates responses from all local nodes and the memory controller **623-1**, and forwards the accumulated responses and data back to the requesting processor **601-3**. The cache coherence controller **621-5** also forwards a source done to the local memory controller **623-2**.

The cache coherence controller **603-1** at the request cluster **600** is acting as a remote memory. As remote memory, the cache coherence controller is designed to forward a request from a processor to a proper remote cluster and ensure that local nodes are probed. In this case, the cache coherence controller **603-1** forwards a probe to cache coherence controller **621-1** at a home cluster **620**. Cache coherence controller **603-2** also probes local nodes **605**, **607**, and **609**.

The cache coherence controller **641-1** at the request cluster **640** is acting as a probing agent pair. As noted above, when a cache coherence controller acting as a probing agent pair receives a probe from a remote cluster, the cache coherence controller accepts the probe and forwards it to all local nodes. The cache coherence controller accumulates the responses and sends a final response back to the request cluster. Here, the cache coherence controller **641-1** sends a probe to local nodes associated with cache blocks **645**, **647**, and **649**, gathers probe responses and sends the probe responses to cache coherence controller **621-3** at home cluster **620**. Similarly, cache coherence controller **603-2** also acts as a probing agent pair at a request cluster **600**. The cache coherence controller **603-2** forwards probes to local nodes including local nodes associated with cache blocks **605**, **607**, and **609**.

The cache coherence controller **621-2** and **621-3** is also acting as an aggregate remote cache. The cache coherence controller **621-2** is responsible for accepting the probe from the memory controller **623-1** and forwarding the probe to the other processor clusters **600** and **640**. More specifically, the cache coherence controller **621-2** forwards the probe to cache coherence controller **603-2** corresponding to request cluster **600** and to cache coherence controller **641-1** corresponding to remote cluster **640**. As noted above, using a multiple cluster architecture may introduce delay as well as other undesirable elements such as increased traffic and processing overhead.

Probes are transmitted to all clusters in the multiple cluster system even though not all clusters need to be probed. For example, if a memory line associated with a request is invalid or absent from cache, it may not be necessary to probe all of the caches associated with the various clusters. In a system without a coherence directory, it is typically necessary to snoop all clusters. However, by using a coherence directory, the number of transactions in

the system can be reduced by probing only a subset of the clusters (or nodes) in a system in order to minimize traffic and processing overhead.

By using a coherence directory, global memory line state information (with respect to each cluster) can be maintained and accessed by a memory controller or a cache coherence controller in a particular cluster. According to various embodiments, the coherence directory tracks and manages the distribution of probes as well as the receipt of responses. If coherence directory information indicates that probing of a specific cluster is not required, the probe to the specific cluster can be eliminated. In one example, a coherence directory indicates that probing of requesting and remote clusters is not necessary. A cache coherence controller in a home cluster probes local nodes without forwarding probes to the request and remote clusters. The cache coherence controller in the home cluster then sends a response to the request cluster after probe responses are received. However, in typical multiple cluster systems, a requesting cluster expects a predetermined number of responses from the various probed clusters. In one example, if the multiple cluster system includes four clusters, a request cluster would expect probe responses associated with nodes in all four clusters.

According to various embodiments, the techniques of the present invention provide a completion bit associated with a probe response. The completion bit indicates to the requesting cluster that no other probe responses from other clusters should be expected. Any mechanisms for notifying a request cluster that no other probe responses should be expected from other clusters is referred to herein as a completion indicator. In one example, a completion indicator is a completion bit included in the response sent to a request cluster after local nodes are probed. In another example, a completion indicator is separate data transmitted to a request cluster. By using a coherence directory and a completion indicator, the number of transactions associated with probing various clusters can be reduced. For example, with reference to FIG. **6**, probes to cache coherence controller **603-2** and cache coherence controller **641-1** can be eliminated. A single response with a completion indicator can be transmitted by cache coherence controller **621-4** to the request cluster **600**.

FIG. **7** is one example of a coherence directory that can be used to allow management and filtering of probes. Various coherence directories are available. In one example, a full directory provides an entry for every memory line in a system. In this example, the coherence directory is maintained at the memory controller and is accessible by a cache coherence controller. However, in a system with a large amount of system memory, a full directory may not be efficient or practical. According to various embodiments, a sparse directory is provided with a limited number of entries associated with a selected set of memory lines. In one example, the coherence directory **701** includes state information **713**, dirty data owner information **715**, and an occupancy vector **717** associated with the memory lines **711**. In some embodiments, the memory line states are modified, owned, shared, and invalid.

In the invalid state, a memory line is not currently available in cache associated with any remote cluster. In the shared state, a memory line may be present in more than one cache, but the memory line has not been modified in any of these caches. When a memory line is in the shared state, an occupancy vector **717** can be checked to determine what caches share the relevant data. An occupancy vector **717** may be implemented as an N-bit string, where each bit represents the availability of the data in the cache of N clusters. Any mechanism for tracking what clusters hold a copy of the relevant memory line in cache is referred to herein as an occupancy vector. The memory line with address **741** is in the shared state, and the occupancy vector **717** indicates that clusters **1** and **3** each have a copy of the shared memory line in cache.

In the modified state, a memory line has been modified and the modified copy exists in cache associated with a particular cluster. When a memory line is modified, dirty data owner information field **715** can be checked to determine the owner of the dirty data. Any mechanism for indicating what cluster owns a modified copy of the memory line in cache is referred to herein as a dirty data owner information field. In one example, the memory line associated with address **781** is modified, and the dirty data owner field **715** indicates that cluster **2** owns the memory line.

In the owned state, a dirty memory line is owned by a single cache but may be resident in multiple caches. In this case, the copy held in memory is stale. If the memory line is in the owned state, dirty data owner field **715** can be accessed to determine which cluster owns the dirty data. In one example, the memory line associated with address **761** is in the owned state and is owned by cluster **4**. The occupancy vector **717** can also be checked to determine what other caches may have the relevant data. In this example, the occupancy vector **717** indicates that clusters **2**, **3**, and **4** each have a copy of the data associated with the memory line in cache.

Although the coherence directory **701** includes the four states of modified, owned, shared, and invalid, it should be noted that particular implementations may use a different set of states. In one example, a system may have the five states of modified, exclusive, owned, shared, and invalid. The techniques of the present invention can be used with a variety of different possible memory line states.

The coherence directory tracks the various transactions such as requests and responses in a multiple cluster system to determine when memory lines are added to the coherence directory, when memory lines are removed from the directory, and when information associated with each memory line is updated. By using the coherence directory, specific embodiments of the present invention recognize that the number of transactions such as probes can be reduced by managing or filtering probes that do not need to be sent to specific clusters. In addition, some embodiments employ this notion to manage or filter probes within a single cluster.

FIG. **8** is a diagrammatic representation showing probe filter information that can be used to reduce the number of transactions in a multiple or single cluster system. Any criterion that can be used to reduce the number of clusters or nodes probed is referred to herein as probe filter information. Transactions such as probes can have a variety of characteristics. Characteristics of the probe include the next state of the memory line associated with the probe which indicates the type of the associated request for instance whether the probe is a read block (read) **823** or a read block modify (read/write) **825**. According to various embodiments, a coherence directory maintains information for memory lines in the local cluster that are cached in non-local clusters, where non-local clusters can include request and remote clusters. According to other embodiments, such a directory includes information about locally cached lines.

If the state of the memory line associated with a probe is invalid **831** as indicated in the coherence directory, no copies of the memory line reside in other clusters (or other nodes for single cluster embodiments). Consequently, only the

home cluster needs to be probed and a completion bit can be used to indicate to a request cluster that the request cluster should expect only a single response from home cluster instead of a response from each of the clusters. If the memory line associated with the probe is in the shared state **833**, and the transaction is a read transaction, only the home cluster needs to be probed and a completion bit can again be used to indicate to the request cluster that only a single response from home cluster should be expected (**803**).

For read transactions on owned memory lines, only the remote cluster with the line cached in the owned state needs to be probed. The remote cluster can transmit the response with a completion bit back to a request cluster. For transactions on modified memory lines, the probe can be sent to the remote cluster with the line cached in the modified state. Although transactions such as read block (read) and read block modify (read/write) are described, it should be noted that other transactions such as test and test and set are contemplated.

FIG. **9** is a diagrammatic representation depicting one example of transactions for probing only a home cluster as indicated in entries **801**, **809**, and **803** in FIG. **8**. According to various embodiments, processor **901-1** in a local cluster **900** sends a data access request such as a read request to a cache coherence controller **903-1**. The cache coherence controller **903-1** forwards the request to a cache coherence controller **921-1** in a home cluster **920**. The cache coherence controller **921-1** at the home cluster **920** receives the access request and forwards the access request to a memory controller **923-1**, which then probes local nodes **925**, **927**, and cache coherence controller **921-2**. It should be noted that a cache coherence controller **921-1** is typically responsible for updating the coherence directory during various transactions. The cache coherence controller **921-2** determines characteristics associated with the probe from the memory controller **923-1** to determine whether remote probes are needed and whether a completion bit can be used. Here, the cache coherence controller **921-2** determines that no remote probes are needed and does not forward probes to the remote cluster **940** or to request cluster **900**.

After cache coherence controller **921-4** receives the probe responses from local nodes as well as the read response from the memory controller **923-1**, the response message with a completion indicator is transmitted to the request cluster. With the completion indicator, the request cluster does not wait for additional responses from other clusters. The coherence controller **903-4** forwards the response with the completion bit set to CPU **901-4**. After receiving the response with the completion bit set, the CPU does not wait for additional responses from the local caches. CPU **901-4** forwards a source done message to cache coherence controller **903-5** to home cluster cache coherence controller **921-5**, which can then perform updates of its coherence directory. The source done is then forwarded to memory controller **923-1**.

FIG. **9** shows one example of a sequence where only the home cluster needs to be probed. FIG. **10** shows one example of a sequence where only a single remote cluster needs to be probed. FIG. **10** is a diagrammatic representation depicting an example of transactions for probing a remote cluster as indicated in entries **805**, **807**, and **815** in FIG. **8**. According to various embodiments, processor **1001-1** in a local cluster **1000** sends a data access request such as a read request to a cache coherence controller **1003-1**. The cache coherence controller **1003-1** forwards the request to a cache coherence controller **1021-1** in a home cluster **1020**. The cache coherence controller **1021-1** at the home cluster **1020** receives the access request and forwards the access request to a memory controller **1023-1**, which then probes local nodes **1025**, **1027**, and cache coherence controller **1021-2**. The cache coherence controller **1021-2** determines characteristics associated with the probe from the memory controller **1023-1** to determine whether remote probes are needed and whether a completion bit can be used. Here, the cache coherence controller **1021-2** determines that only a remote cluster needs to be probed and does not forward a probe to request cluster **1000**.

After cache coherence controller **1021-4** receives the probes from local nodes as well as the read response from the memory controller **1023-1**, a response message is not transmitted to the request cluster because the remote cluster is sending a response message with a completion indicator is transmitted to the request cluster. With the completion indicator, the request cluster does not wait for additional responses from other clusters. The response is forwarded to CPU **1001-4** and a source done message is sent from cache coherence controller **1003-5** to home cluster cache coherence controller **1021-5**. With the completion bit set in the response to CPU **1001-4**, it does not wait for any other local responses. After all responses from local nodes are received, the source done is then forwarded to memory controller **1023-1**, which can then perform updates of its coherence directory.

FIG. **11** is a process flow diagram showing one example of a technique for handling requests at a home cache coherence controller. At **1101**, a request associated with a memory line is received. At **1105**, the cache coherence controller forwards the request to the memory controller. At **1109**, the cache coherence controller receives a probe from the memory controller and accesses a coherence directory and probe filter information at **1113** to determine whether the number of probes to various clusters in the system can be reduced. At **1121**, it is determined whether filtering and a completion indicator can be used. In one example, it is determined the filtering and a completion indicator can be used by identifying the criteria specified in FIG. **8** and by accessing a coherence directory as shown in FIG. **7**.

If a completion indicator cannot be used, probes are broadcast to the various nodes with no filtering and no completion bit **1145**. If filtering and a completion indicator can be used, it is determined at **1131** if a remote cluster should be probed. If a single remote cluster is the cluster that should be probed, the probe is forwarded with the completion indicator to the remote cluster at **1135**. At **1139**, home cluster probe responses are received but are not forwarded to the request cluster. The response is not sent to the request cluster from home cluster because a remote cluster is sending a response with a completion indicator to the request cluster.

At **1149**, source done information is received from the request cluster and forwarded to the memory controller. If it is determined at **1131** that only the home cluster needs to be probed, then the cache coherence controller at **1141** does not send probes to any request or remote clusters and instead sends a response to the request cluster with a completion indicator. The cache coherence controller sends the response with the completion indicator after receiving home cluster probe responses. At **1149**, the cache coherence controller at the home cluster receives source done information from the request cluster and forwards the source done information to the memory controller.

According to various embodiments, when the only cluster that needs to be probed is the home cluster, only the nodes in the home cluster are probed. No probes are transmitted to

any request or remote cluster. However, when the only cluster that needs to be probed is a remote or request cluster, not only are the nodes in the remote cluster probed, but the nodes in the home cluster are probed as well. As will be seen, in some embodiments, the nodes within a home cluster may be filtered using probe filter information corresponding to locally cached lines.

According to various embodiments, the techniques of the present invention provide that when only a remote or request cluster needs to be probed, the memory controller can sometimes be bypassed to allow probing of only the remote or request cluster. In one example, a probe is not forwarded within the home cluster and a probe is forwarded directly to the remote cluster from the home cluster cache coherence controller.

FIG. **12** is a diagrammatic representation showing exemplary memory controller filter information. Any criterion used to reduce the number of requests forwarded to a memory controller is referred to herein as memory controller filter information. Characteristics of a request can again be analyzed when a cache coherence controller receives the request from a request cluster. Requests can have a variety of characteristics. Some characteristics include whether the request is a read block (read) **1223** or a read block modify (read/write) **1225**. When the state of the memory line associated with the request is invalid **1231**, no remote probes are required because no remote clusters have a copy of the memory line in cache. In some embodiments, the cache coherence controller does not maintain knowledge of the home cluster cache state. In such cases, the request is forwarded to the memory controller. In other embodiments, the cache coherence controller does maintain such information and uses it to reduce the number of nodes probed within the home cluster.

For read block transactions on a shared memory line **1203**, there is no need to probe the remote clusters as the home cluster contains a valid copy of the memory line in either cache or the memory controller. Consequently the request is forwarded to the memory controller. For read block modify transactions on shared memory lines **1211**, the local node state is unknown and the request is sent to the memory controller.

For read block transactions on an owned memory line **1205**, there is no need to send a request to the target or probe local nodes as the owned state implies that the home cluster caches are invalid or shared. A probe is forwarded directly to the owning cluster to acquire the cached data. For read block write transactions on an owned memory line **1213**, the local state is unknown and consequently the request is forwarded to the memory controller. When the state of the memory line associated with the request is modified **1237**, there is no need to probe local nodes, as a modified state implies the home cluster state is invalid. A probe is forwarded to the cluster owning the memory line.

FIG. **13** shows one example of a sequence where a request does not need to be forwarded to the home cluster memory controller. According to various embodiments, processor **1301-1** in a local cluster **1300** sends a data access request such as a read request to a cache coherence controller **1303-1**. The cache coherence controller **1303-1** forwards the request to a cache coherence controller **1321-1** in a home cluster **1320**. The cache coherence controller **1321-1** at the home cluster **1320** receives the access request and determines whether the memory controller can be bypassed. Forwarding a probe to a remote or request cluster without forwarding the request to a memory controller is referred to herein as bypassing the memory controller. In one embodiment, the determination can be made by using memory controller filter information. If the probe characteristics fall within entries **1205**, **1207**, or **1215**, the memory controller is bypassed and a probe is sent to cache coherence controller **1341-1** in the remote cluster **1340**. In one example, the probe is forwarded with an indication that a completion bit should be used.

The cache coherence controller **1321-1** in the home cluster **1320** is acting as a serialization point in place of the memory controller to maintain cache coherency. Once it is determined that the memory controller can be bypassed, the cache coherence controller **1321-1** blocks all other incoming requests and outgoing probes until a final source done is received from the request cluster. The remote cluster cache coherence controller **1341-1** probes remote cluster nodes and sends a response with a completion indicator to the request cluster **1300**. The response is forwarded to CPU **1301-4** and a source done message is sent from cache coherence controller **1303-5** to home cluster cache coherence controller **1321-5**. The source done is not forwarded to the memory controller, because the memory controller never processed the transaction.

FIG. **14** is a flow process diagram showing request handling at a home cache coherence controller using memory controller filter information. At **1401**, a request associated with a particular memory line is received. At **1403**, characteristics associated with the request are identified. At **1411**, it is determined if the memory controller can be bypassed. According to various embodiments, memory controller filter information shown in FIG. **12** is used to determine whether a memory controller can be bypassed. If it is determined that a memory controller can be bypassed, requests associated with the same memory line are blocked at **1415** and a probe is sent to a remote or a request cluster. At **1417**, the memory line is unblocked after receiving a source done from the request cluster. If it is determined at **1411** that a memory controller can not be bypassed, the request is forwarded to a serialization point **1405**. The transaction sequence can then proceed with or without probe filtering and a completion indicator as shown in **1109** of FIG. **11**.

As described above and according to some embodiments, a cache coherence directory is a mechanism associated with each cache coherence controller which facilitates the tracking by that cache coherence controller of where particular memory lines within its cluster's memory are being cached in remote clusters. That is, a portion of the global memory space for the multi-cluster system is associated with each cluster. The cache coherence directory enables the cache coherence controller in each cluster to track which memory lines from the portion of the global memory space associated with its cluster have been cached with processors in remote clusters.

Each cache coherence controller in each cluster has such a cache coherence directory associated with it. Given the size of the memory space associated with each cluster, it is not practical to have an entry in the coherence directory for each memory line. Rather, the directory is sized in relation to the amount of cache memory associated with the processors in all remote clusters, a much smaller amount of memory. That is, the coherence directory is an associative memory which associates the memory line addresses with their remote cache locations. According to one embodiment, the cache coherence directory is fully associative. According to another embodiment, the directory is set-associative.

According to a specific embodiment, a typical entry in the cache coherence directory includes the memory address

corresponding to the cached memory line, the remote cache location, whether the line is "clean" or "dirty," and whether the associated processor has read-only access or read/write access. This information corresponds to the standard coherence protocol states which include "invalid" (not cached in any remote clusters), "shared" (cached as "clean" and read-only), "modified" (cached as "dirty" and read/write), and "owned" (cached as "dirty" but read-only). A coherence directory entry also includes one or more fields identifying which, if any, of the remote clusters have the line cached in the "dirty" state, and which other clusters have the line cached in the "shared" state.

When the cache coherence controller in a particular cluster, e.g., the home cluster, receives a request for a particular memory line in its memory, it transmits the request to a memory controller associated with one of the local nodes to which the requested address maps, e.g., the home controller. To determine whether the most recently modified copy of the memory line resides in any of the cache memories in the system, the home controller then generates probes to all of the nodes in the cluster (including the cache coherence controller) asking whether any of the nodes have the requested memory line stored in their corresponding caches in either a "dirty" (i.e., modified) or "clean" (unmodified) state. These probes can tell the nodes to invalidate their copies of the memory line, as well as to return the memory line in the case where the memory line has been modified.

Because the cache coherence controller in each cluster maps to the remainder of the global memory space outside of its cluster, it is responsible for ensuring that the appropriate processors in remote clusters receive corresponding probes. This is where the cache coherence directory comes into play. Without such a mechanism, the cache coherence controller would have to transmit probes to all of the nodes in all of the remote clusters having cache memories associated with them. By contrast, because the cache coherence directory provides information about where memory lines are cached as well as their states, probes only need be directed toward the clusters in which the requested memory line is cached. The state of a particular cached line will determine what type of probe is generated. As will be seen, such a cache coherence controller may also be configured to include information about locally cached memory lines and be operable to use such information to reduce the number of probes within its cluster.

The associative nature of the cache coherence directory of the present invention necessitates an eviction mechanism so that the most relevant information may be maintained in the limited number of directory entries. In addition, the distributed, multi-cluster architecture described herein also requires that the eviction mechanism be able to guarantee that the memory line corresponding to an evicted directory entry is purged from all remote caches. As mentioned above, the directory entry field indicating the location(s) of the memory line helps to reduce the number of transactions required to effect this purging. In addition, the appropriate type of request to effect the purging depends on the state of the remotely cached memory lines.

Thus, if a directory entry to be purged indicates that the line is only cached in the "clean" state, what is required is a mechanism which invalidates the memory line in each of the remote caches in which the line is cached. On the other hand, if the directory entry indicates that the line is in the "dirty" state in any of the remote caches, the modified memory line to memory must first be written back to memory before the line is invalidated.

In a conventional multiprocessor system, i.e., a system which does not have remote clusters of processors, there typically are not mechanisms by which external requests to a particular processor may be generated for the purpose of instructing the processor how to manage its cache. That is, in such a system, each processor is responsible for maintaining its own cache and evicting and/or writing lines back to memory to free up room for new entries. Thus, there is no provision for allowing one processor to instruct another processor to write a particular line back to memory. Similarly, there is no provision for allowing one processor to instruct another processor to invalidate a particular line in its cache without returning any data. That is, transactions between processor in a cache coherence protocol typically assume that one processor is trying to get a copy of the line from the other. Therefore, according to the present invention, mechanisms are provided for a system having a plurality of multiprocessor clusters by which such requests may be generated.

According to various specific embodiments of the invention, the semantics of transaction types developed for a single cluster system are altered to enable external devices to generate requests to specific processors to invalidate cache entries and to write cache entries back to memory. According to one embodiment which assumes the multi-cluster architecture described above, one such transaction type referred to herein as a "sized write" (i.e., a partial line write to memory) is employed to achieve the effect of instructing a processor having a "dirty" copy of a memory line stored in its cache to write the line back to memory.

The sized write transaction normally allows a processor to initiate a write to a any arbitrarily sized portion of a memory line (e.g., a particular byte or the entire line). That is, a request to write the byte to the memory line is sent to the memory controller which maps to the memory line. The memory controller then sends out a request to any other caches in the system having the corresponding line in the "dirty" state. If a positive response is received, i.e., if a modified copy of the line is returned in response to the request, the memory controller than merges the original byte with the retrieved memory line, and then writes the merged line back to memory.

Generally speaking, the eviction of a cache coherence directory entry corresponding to a "dirty" line in a remote cache requires that the remote cache write the line back to memory and invalidate its copy. Thus, a transaction is needed which results in the following actions:

1. A write back is generated for the cached memory line,
2. The copy of the line in the cache is invalidated, and
3. The eviction mechanism is notified when the memory line has been written back to memory.

According to a specific embodiment of the invention, the semantics of the sized write transaction are altered resulting in a transaction having these characteristics. The altered sized write is generated such that no data are provided for the partial write, i.e., the sized write request has zero size. When the cache coherence directory associated with the cache coherence controller in a particular cluster, i.e., the home cluster, determines that it needs to evict an entry which corresponds to remotely cached "dirty" memory line, it generates a sized write request specifying no data and directs the request to the local memory controller corresponding to the memory line, i.e., the home memory controller. The home memory controller then generates probes to all of the local nodes in the cluster (including the cache coherence controller) requesting the most recent copy of the memory

line. The local nodes respond as described above, returning any dirty copy of the line and invalidating the corresponding entries in their caches.

As described above, the cache coherence controller forwards the probe to the appropriate remote cluster(s) based on the information in its associated cache coherence directory which indicates the existence and location of any remotely cached copies of the memory line. The nodes in remote clusters which receive the probe behave similarly to the local nodes in that they respond by returning any dirty copy of the line and invalidating the corresponding entries in their caches.

The home memory controller receives the "dirty" copy of the memory line (if one exists), performs a NOP (because there are no data to merge with the modified line), writes the line back to memory, and notifies the cache coherence directory (i.e., the originator of the transaction) that the transaction is complete. In this way, the "altered" sized write transaction is employed to achieve the effect of instructing a remote processor to write back a specific "dirty" line in its cache to memory.

According to a specific embodiment of the invention, the notification by the home memory controller that the transaction is complete plays an important part in avoiding race conditions. That is, because the coherence directory is in flux during the period of time required to complete an eviction, it is possible that subsequent transactions corresponding to the same memory line might be generated somewhere in the system. Fortunately, as described above, the memory controllers of the multi-cluster architecture described herein act as serialization points for memory transactions. That is, once a memory controller accepts a transaction for one of its memory lines, it blocks all other transaction to that same memory line. Therefore, once the home memory controller accepts the sized write transaction, it does not allow any further transactions for the same memory line until the eviction process is completed.

Generally speaking, the eviction of a cache coherence directory entry corresponding to a "clean" line in a remote cache requires that the remote cache invalidate its copy. Thus, a transaction is needed which results in the following actions:

1. The copy of the line in the cache is invalidated, and

2. The eviction mechanism is notified when the invalidation is complete.

According to some embodiments, the zero sized write described above is employed as described with reference to dirty lines. According to another embodiment of the invention, the semantics for another type of transaction referred to herein as a "validate block" transaction are altered to achieve these results. That is, the semantics of the validate block transaction are altered such that it has the effect of instructing remote systems nodes having "clean" copies of a memory line to invalidate those lines in their caches without resulting in any returned copies of the line in response to the request.

The validate block transaction is normally intended for the case in which a processor or I/O device (via the I/O bridge) writes an entire memory line of data atomically. This might occur, for example, when an I/O device, such as a disk drive, is writing blocks of data to memory. Such a transaction does not require a data response from the memory controller responsible for the memory line. In such a case, however, there still is a need to invalidate all cached copies of the line. The transaction saves the bandwidth that would normally be consumed to send the line from the memory controller to the processor or I/O bridge, which would be completely overwritten.

Therefore, according to a specific embodiment of the invention, when the cache coherence directory associated with the cache coherence controller in a particular cluster, i.e., the home cluster, determines that it needs to evict an entry which corresponds to one or more remotely cached "clean" memory lines, it generates a validate block request and directs the request to the local memory controller corresponding to the memory line, i.e., the home memory controller. The home memory controller then generates invalidating probes to all of the local nodes in the cluster (including the cache coherence controller). The local nodes invalidate their copies of the memory line and send confirming responses to home memory controller indicating that the invalidation took place.

The cache coherence controller forwards the invalidating probe to the appropriate remote cluster(s) based on the information in its associated cache coherence directory which indicates the existence and location of any remotely cached copies of the memory line. The remote nodes behave similarly to the local nodes in that they also invalidate any copies of the memory line and send the corresponding responses back to the cache coherence controller in the home cluster. The cache coherence controller aggregates the responses and transmits the aggregated response to the home memory controller.

The home memory controller receives the responses from the local nodes and the cache coherence controller, and notifies the cache coherence directory (i.e., the originator of the transaction) that the transaction is complete. The cache coherence directory then transmits a "source done" to the memory controller in response to which the memory line is freed up for subsequent transactions. In this way, the validate block transaction is employed to achieve the effect of instructing a remote processor to invalidate its copy of a "clean" memory line. As with the altered sized write transaction, the home memory controller acts as a serialization point for the validate block transaction thereby avoiding race conditions caused by subsequent transactions corresponding to the same memory line.

As described above, the eviction mechanism employed to effect an eviction of an entry from the cache coherence directory may depend on the indicated state of the corresponding memory line, e.g., "clean" vs. "dirty." According to specific embodiments of the invention, the determination of which of the existing entries is to be evicted to make room for a new entry may be done in a wide variety of ways. For example, different approaches might select the oldest or least frequently used entries. According to one embodiment, "modified" lines are chosen ahead of "shared" lines, with a random mechanism being employed to select among like copies. It will be understood that any kind of policy for selecting the entry to be evicted may be employed without departing from the scope of the invention.

As described above, the serialization point of the home memory controller guarantees that transactions to the memory line corresponding to the directory entry being evicted will be locked out once the home memory controller receives the sized write or validate block request from the directory. However, it is possible that conflicting transactions may be generated during the time between when the cache coherence directory to evict a particular entry and the corresponding request is received by the memory controller. Until the sized write or validate block request corresponding to the entry being evicted is received by the memory

controller, it is desirable to guarantee that any other requests corresponding to the same memory line are properly serviced.

Therefore, according to a specific embodiment of the invention, an eviction buffer is provided in the cache coherence directory in which the directory places the entry it has determined should be evicted. The entry in the eviction buffer remains visible to the cache coherence controller as one of the entries in the directory, i.e., the cache coherence controller cannot distinguish between entries in the directory and entries in the eviction buffer. The entry in the eviction buffer remains there until the home memory controller receives the corresponding eviction request from the cache coherence directory and the cache coherence controller receives a probe corresponding to the eviction request, at which point the entry in the eviction buffer is invalidated. However, if an intervening request corresponding to the entry in the eviction buffer is received, it may be processed by the cache coherence controller with reference to the eviction buffer entry and, because of the ordering of transactions at the memory controller serialization point, it is guaranteed that this intervening transaction will complete before the eviction request is serviced by the memory controller. In this way, a cache coherence directory entry may be "earmarked" for eviction, but may still be used for servicing subsequent requests until the memory line is locked by the home memory controller for the eviction process. According to a specific embodiment, if the eviction buffer is full, a status bit instructs the cache coherence controller to stall, i.e., to queue up any new requests for which there are no corresponding entries already in the cache coherence directory.

FIG. **15** is a diagrammatic representation showing a transaction flow for a cache coherence directory eviction of a directory entry corresponding to a "dirty" memory line according to a specific embodiment of the invention. When the cache coherence directory **1501-1** determines that an eviction of one of its entries showing a "dirty" state must occur, e.g., in response to a new request for which no entry exists, it places the entry to be evicted into its eviction buffer and generates a sized write request (having zero size) to the local memory controller responsible for the memory line corresponding to the directory entry being evicted, i.e., the home memory controller **1502-1**.

Assuming a previous transaction corresponding to the same memory line is not currently being processed, the home memory controller **1502-1** accepts the sized write request and generates invalidating probes to all nodes in its cluster including local nodes **1503-1505** and cache coherence controller **1506-1**. Each of the local nodes **1503-1505** invalidates any copies of the memory line and responds accordingly to the home memory controller **1502-2**. When the cache coherence controller **1506-1** in the home cluster receives the invalidating probe, it forwards the invalidating probe to the remote cluster having the dirty copy of the memory line according to the directory information (i.e., the entry in the eviction buffer). The directory entry in the eviction buffer is then invalidated.

The cache coherence controller **1507-1** in the remote cluster receives the invalidating probe and forwards it to the local nodes in the remote cluster, i.e., local nodes **1508-1510**. The local node having the "dirty" copy of the memory line replies to cache coherence controller **1507-2** with a dirty data response (i.e., returning the modified copy of the memory line from its cache), and the other local nodes reply with clean responses. In addition, any copies of the memory line in the remote cluster's caches are invalidated. The cache coherence controller **1507-2** then forwards the dirty data response back to the cache coherence controller **1506-2** in the home cluster which forwards the response to the home memory controller **1502-3**.

The home memory controller **1502-3** receives the dirty data response and merges the modified data with the data from the sized write request (i.e., no data). Once all responses from the local nodes are received by the home memory controller **1502-3**, a target done (TD) message is sent by the home memory controller **1502-3** to the cache coherence directory **1501-2** which completes the transaction with a source done (SD) message back to the home memory controller **1502-4**, which then unlocks the memory line for subsequent transactions. As mentioned above, this mechanism may also be employed to evict directory entries corresponding to "clean" memory lines.

FIG. **16** is a diagrammatic representation showing a transaction flow for an eviction of a directory entry corresponding to a "clean" memory line according to another specific embodiment of the invention. When the cache coherence directory **1601-1** determines that an eviction of one of its entries showing a "clean" state must occur it places the entry to be evicted into its eviction buffer and generates a validate block request for the corresponding memory line and sends the request to the local memory controller responsible for the memory line, i.e., the home memory controller **1602-1**.

Assuming the memory line is not locked, the home memory controller **1602-1** accepts the validate block request and generates invalidating probes to all nodes in its cluster including local nodes **1603-1605** and cache coherence controller **1606-1**. Each of the local nodes **1603-1605** invalidates any copies of the memory line and responds accordingly to the home memory controller **1602-2**. When the cache coherence controller **1606-1** in the home cluster receives the invalidating probe, it forwards the invalidating probe to any remote clusters having a copy of the memory line according to the directory information (i.e., the entry in the eviction buffer). The directory entry in the eviction buffer is then invalidated.

The cache coherence controller **1607-1** in any such remote cluster receives the invalidating probe and forwards it to the local nodes in the remote cluster, i.e., local nodes **1608-1610**. Each of the local nodes **1608-1610** having a copy of the line invalidates its copy and responds accordingly to the cache coherence controller **1607-2**. The cache coherence controller **1607-2** aggregates and forwards these responses back to the cache coherence controller **1606-2** in the home cluster which sends a source done (SD) message to the home memory controller **1602-3**, which then unlocks the memory line for subsequent transactions.

In general, the entry in the eviction buffer may be invalidated by an earlier request, such as a write by a local processor. When the invalidating probe, associated with the eviction request, reaches the coherence controller, it will find the directory entry in the eviction buffer invalid. In this case, the coherence controller responds to the request without generating any remote probes.

The foregoing description assumes that the cache coherence directory includes processing functionality, e.g., an eviction manager, which may, according to different embodiments of the invention, be implemented in a variety of ways. For example, the directory may include its own memory controller configured to manage the directory and implement the various functionalities described above. Alternatively, these functionalities may reside in application specific hardware, e.g., an ASIC, as a separate eviction

manager. A further alternative might configure the cache coherence controller to incorporate at least some of the functionalities described.

According to a specific embodiment illustrated in FIG. **17**, the eviction manager **1702** is part of the cache coherence directory **1701** which is a functional block within the cache coherence controller **1700**. The protocol engine **1705** (which may actually be one or more protocol engines) is responsible for processing transactions and corresponds to the CCC blocks in FIGS. **15** and **16**. The cache coherence directory corresponds to the DIR blocks in FIGS. **15** and **16**. The remaining blocks within controller **1700** are similar to the corresponding blocks described above with reference to FIG. **3**. Eviction manager **1702** communicates with protocol engine **1705** via coherent interface **1707**. The protocol engine **1705** communicates with the coherence directory via a dedicated interface, which is used to communicate lookup and update commands and responses.

The basic architecture of FIG. **17** may also be used to implement a probe filtering unit which is operable to reduce probe traffic within a cluster of processing nodes. Various embodiments of such a probe filtering unit are described below with reference to FIGS. **18-22**.

As described above with reference to FIGS. **12-14**, embodiments of the invention are contemplated in which the memory controller in the home cluster may be bypassed with reference to characteristics of a received request in accordance with, for example, the memory controller information described with reference to FIG. **12**. According to such embodiments, if the request is forwarded to the memory controller in the home cluster, all of the local nodes in the home cluster are then probed as shown in and described above with reference to FIG. **4**. On the other hand, if the request is not forwarded to the memory controller in the home cluster, none of the local nodes are probed.

As will be understood, such embodiments are effective in reducing unnecessary probe traffic in the former case, but may still generate unnecessary probes in the latter. That is, for example, in cases where the memory controller filter information of FIG. **12** indicates that a valid copy of the requested memory line may exist in the home cluster, all of the local nodes in the home cluster end up being probed whether or not they have valid copies of the line in their caches. It is therefore desirable to provide techniques by which probe traffic may be more precisely "filtered" and system performance may be further enhanced. It will be understood that any reference herein to "filtering" includes any mechanism or technique by which the number of recipients of a probe is reduced.

According to specific embodiments of the invention, the techniques described above are adapted to reduce the number of probes within a cluster. Such techniques are referred to herein as local probe filtering. It should be noted that the following discussion applies both to systems having multiple multi-processor clusters such as those described above, as well as to systems having multiple processing nodes configured in a single cluster.

The behavior of a single cluster of processors implemented without local probe filtering will now be described again with reference to FIG. **4**. CPU **401-1** sends a read request to memory controller MC **403-1** which is responsible for controlling access to the memory range including the line identified in the request. If the memory controller MC **403-1** is available to respond to the request, it generates probes to the other processing nodes in the system (**405**, **407** and **409**), each of which sends a probe response back to the requesting CPU **401-2**. These probe responses may or may not include copies of the requested memory line depending on whether a valid copy of the line existed in the caches associated with these nodes. To account for the case in which the memory line does not exist in any of the caches, MC **403-1** also generates a read response back to the requesting CPU **401-3** which includes the memory line retrieved from main memory.

In implementations without local probe filtering, CPU **401** is programmed to expect responses from each of the nodes probed (including its own node) as well as a response from memory controller MC **403**. It will only send the "source done" message to the memory controller (which then unlocks the memory line) when all of the expected messages have been received. Thus, where the requested memory line is not held by a particular node's cache, there is unnecessary probe related traffic consuming the precious bandwidth of the system's point-to-point infrastructure.

It will be understood that the foregoing is an exemplary read transaction in which the probe responses and read responses are directed to the requesting processor. It should also be understood that the present invention is applicable to operations in which the responses are directed to the memory controller, e.g., write operations. The former probe responses are precipitated by what is called a probe source; the latter by a probe target. Probes also include "next state" information which indicates to each node what the state of its copy of the line should be at the end of the transaction. According to a specific embodiment, the next state information indicates one of three possibilities, i.e., that there should be no change to the line status, that it should be moved to "shared," or moved to "invalid." In general, for the typical memory transaction depicted in FIG. **4**, each of the nodes in the cluster is "consulted" and makes its own independent assessment of how to proceed based on its current condition.

As mentioned above, the filtering of probes within a cluster, i.e., local probe filtering, may be implemented in systems having multiple clusters as well as systems having a single cluster of processors. In the former and as described above, the probe filtering functionalities described herein may be implemented in a cache coherence controller which facilitates communication between clusters. In the latter, these functionalities may be implemented in a device which will be referred to herein as a probe filtering unit (PFU) which may occupy a similar location in the cluster as the cache coherence controller, and may include some subset of the other functionalities of the cache coherence controller. In either case, it should be noted that the functionalities described may be implemented in a single device, e.g., a cache coherence controller or probe filtering unit, or be distributed among multiple devices including, for example, the processing nodes themselves. It should be understood that the use of the term "probe filtering unit" or "PFU" in the following discussion is not intended to be limiting or exclusive. Rather, any device or object operable to perform the described functionalities, e.g., a cache coherency controller as described herein, is within the scope of the invention.

FIG. **18** is a diagrammatic representation of a multiple processor system **1800** in which embodiments of the invention relating to the filtering of probes within a single cluster of processors may be practiced. System **1800** may comprise one cluster in a multiprocessor system (as described above with reference to FIG. **2**) or the entirety of a single cluster system. System **1800** includes processing nodes **1802***a*-**1802***d*, one or more Basic I/O systems (BIOS) **1804**, a memory subsystem comprising memory banks **1806***a*-**1806***d*, and point-to-point communication links **1808***a*-

1808*e*. The point-to-point communication links are configured to allow interconnections between processing nodes **1802***a*-**1802***d*, I/O switch **1810**, and probe filtering unit **1830** according to a point-to-point communication protocol.

According to embodiments having multiple clusters of processors, PFU **1830** may comprise a cache coherence controller which facilitates communication with remote clusters as described above. According to one embodiment, PFU **1830** is an Application Specific Integrated Circuit (ASIC) supporting the local point-to-point coherence protocol. PFU **1830** can also be configured to handle a non-coherent protocol to allow communication with I/O devices. In one embodiment, PFU **1830** is a specially configured programmable chip such as a programmable logic device or a field programmable gate array. An exemplary architecture for PFU **1830** may be implemented as described above with reference to FIG. **17**. I/O switch **1810** connects the rest of the system to I/O adapters **1816** and **1820**. As mentioned above with reference to FIG. **2**, it should be understood that a node (e.g., processing nodes **1802***a*-**1802***d*) may comprise multiple sub-units, e.g., CPUs, memory controllers, I/O bridges, etc.

According to various embodiments of the invention, processing nodes **1802***a*-**1802***d* are substantially identical. FIG. **19** is a simplified block diagram of such a processing node **1802** which includes an interface **1902** having a plurality of ports **1904***a*-**1904***c* and routing tables **1906***a*-**1906***c* associated therewith. Each port **1904** allows communication with other resources, e.g., processors or I/O devices, in the computer system via associated links, e.g., links **1808***a*-**1808***e* of FIG. **18**.

The infrastructure shown in FIG. **19** can be generalized as a point-to-point, distributed routing mechanism which comprises a plurality of segments interconnecting the systems processors according to any of a variety of topologies, e.g., ring, mesh, etc. Each of the endpoints of each of the segments is associated with a connected processing node which has a unique node ID and a plurality of associated resources which it "owns," e.g., the memory and I/O to which it's connected.

The routing tables associated with each of the nodes in the distributed routing mechanism collectively represent the current state of interconnection among the computer system resources. According to a specific embodiment, each node has different routing tables for requests, broadcasts (e.g., probes), and responses. Each of the resources (e.g., a specific memory range or I/O device) owned by any given node (e.g., processor) is represented in the routing table(s) associated with the node as an address. When a request arrives at a node, the requested address is compared to a two level entry in the node's routing table identifying the appropriate node and link, i.e., given a particular address within a range of addresses, go to node x; and for node x use link y.

As shown in FIG. **19**, processing node **1802** can conduct point-to-point communication with three other processing nodes according to the information in the associated routing tables. According to a specific embodiment, routing tables **1906***a*-**1906***c* comprise two-level tables, a first level associating the unique addresses of system resources (e.g., a memory bank) with a corresponding node (e.g., one of the processors), and a second level associating each node with the link (e.g., **1808***a*-**1808***e*) to be used to reach the node from the current node.

Processing node **1802** also has a set of JTAG handshake registers **1908** which, among other things, may be used to facilitate modification of the routing tables (which are initially set by the BIOS). That is, routing table entries can be written to handshake registers **1908** for eventual storage in routing tables **1906***a*-**1906***c*. It should be understood that the processor architecture depicted in FIG. **19** is merely exemplary for the purpose of describing a specific embodiment of the present invention. For example, a fewer or greater number of ports and/or routing tables may be used to implement other embodiments of the invention.

According to a specific embodiment, the processing nodes in a single cluster are programmed according to their normal setup rules with a few exceptions. First, the broadcast routing tables in each of the nodes are programmed such that the broadcasts initiated from each node go directly to the PFU rather than on all of the node interfaces. Second, the broadcast routing table in each node is programmed such that broadcasts originating from the PFU enter the node and are not forwarded to any other node. Third, each node is programmed to expect only one or two probe responses instead of one from each node in the system. More specifically, each node is programmed to expect one probe response if the PFU contains temporary storage to hold dirty data, and two if it does not. Some of the exemplary embodiments described below will assume the latter. However, this should not be construed as limiting the scope of the invention.

Referring now to FIG. **20**, when a memory controller generates a probe (**2002**), the node's routing table is consulted (**2004**) and the probe is sent only to the PFU (**2006**), and not to any of the nodes (including the node associated with the memory controller). The PFU accepts the probe and looks up the address in its directory of shared cache states (**2008**). According to a specific embodiment, the directory of shared states may be implemented as described above with reference to FIGS. **7** and **8**, and indicates where particular memory lines are cached within the cluster. According to various embodiments, the directory may be full or sparse. And in embodiments where the directory is sparse, eviction mechanisms such as those described above with reference to the cache coherence controller may be employed.

If the directory lookup determines that the cache line is not cached anywhere in the system, i.e., ignoring the requesting node (**2010**), then the PFU responds to the probe with no traffic generated to any of the other nodes. That is, the response is sent back to the correct unit (either the CPU or the memory controller depending on the type of the probe) with an indication that there are no copies of the line in the system (**2012**). If the node counts are programmed to expect two probe responses, then the PFU sends two copies of the response.

If, on the other hand, the directory lookup determines the cache line may be cached in the system (**2010**), the PFU sends out a probe only on links corresponding to the nodes that may contain the cache line (**2014**). The outgoing probe is the same as the incoming probe, except that it is modified to identify the PFU as the target, i.e., the source of the probe, and the command is changed such that it is always a "Probe—respond to target" regardless of the original command (either respond to source or respond to target).

The nodes that receive the modified probe automatically look up the cache line (**2016**) and return their response to the PFU (**2018**). The PFU uses these responses to update the directory (which may remove the responding node from the list of nodes that is caching the data) (**2020**). Once all the nodes to which the probe was sent have responded (**2022**), the PFU accumulates the responses as described above with reference to remote probe filtering (**2024**), and responds to the node from which the original request originated (**2026**).

As mentioned above, embodiments are contemplated in which the requesting node is programmed to expect two responses from the PFU. This relates to the fact that it may be desirable to immediately forward "dirty" data to the requesting node even where the PFU has not yet received all of the expected responses from the probed nodes. That is, if a probed node has the requested line "dirty" in its cache, i.e., it is the exclusive owner of the most recent copy of the line, that node sends back a read response with the requested data. If the PFU receives a read response from one of the probed nodes, but waits for all probe responses before sending a final response to the requesting node, deadlock may occur (i.e., if the PFU's buffers are full of dirty data, it won't be able to receive the incoming read responses).

Therefore, according to this embodiment, when the PFU receives a read response from one of the probed nodes, it immediately forwards the response to the requesting node. A final response which is cumulative of all received probe responses is then sent to the requesting node to complete the transaction. In cases where the PFU does not receive a read response, i.e., none of the probed nodes own the line, two copies of the final probe response must be sent to the requesting node to complete the transaction, i.e., it is expecting two responses. Alternatively, embodiments are contemplated in which the PFU includes sufficient temporary storage for dirty data, and the requesting nodes are programmed to expect only one response. In either case, because the PFU centrally manages the probe traffic, cache coherency can be maintained without having all nodes respond to a requester.

FIG. **21** illustrates an example of a memory request in a multi-processor system designed according to a specific embodiment of the invention. In this example, a four processing node system with a PFU or cache coherence controller (e.g., the system of FIG. **18**) is assumed. A CPU makes a memory request Req to the memory controller M to which the requested line corresponds. The memory controller retrieves the requested line from the memory banks (as indicated by read response RR), and generates a probe to the probe filtering unit PFU for any cached copies of the line. The PFU, in turn, probes nodes N**0** and N**2** after it applies its directory lookup and probe filtering algorithm. As discussed above, the determination as to which nodes get probed depends on the state of the PFU's directory and is independent of the source of the request, i.e., the requesting node may receive a probe. The PFU then accumulates the responses from nodes N**0** and N**2** and sends two responses (one of which may be a read response from N**0** or N**2**) back to the requesting CPU. As mentioned above, some embodiments may only require a single response. The CPU then sends a source done to the memory controller to complete the transaction and unlock the memory line.

FIG. **22** is an example of a memory request in which the PFU does not have to probe any nodes. This example illustrates the case in which the local filtering mechanism has its greatest effect. That is, because the PFU determines that none of the nodes in the system (i.e., N**0**-N**3**) has the requested line in its cache (i.e., a directory miss), no probes are needed, and the two probe responses PR are immediately sent back to the requesting CPU which then sends the source done (SD) to the memory controller to complete the transaction. Thus, the probe traffic between the requesting CPU and each of nodes N**0**-N**3** which would have otherwise consumed bandwidth and clock cycles is almost entirely eliminated, i.e., only one probe to the PFU and two probe responses back to the requester are required. As mentioned above, some embodiments may only require a single response to the requesting CPU.

In some embodiments, the modification of routing tables also affects transactions that must go to every node in the system (and thus should not be filtered). Such broadcast transactions include, for example, lock requests and system management messages. In such embodiments, the probe filtering unit or cache coherence controller is programmed to send out such broadcasts and to accumulate the responses.

It should be noted that some of the arrows shown in these diagrams may represent multiple "hops" in the point-to-point infrastructure interconnecting the processing nodes. That is, depending on the number of processors and the topology in which the processors are interconnected, a probe from the PFU to a particular processor may need to go through another node before it arrives at its intended destination. In any case, whether a transmission requires one or multiple hops, it is represented in the figures by a single arrow for clarity.

One of the benefits of local probe filtering is that it allows a multi-processor system to scale better because it reduces or eliminates unnecessary probes that go to nodes that are known not to be caching the desired data. In addition, latency may be significantly reduced for lines which are not highly shared across nodes by reducing the number of messages that have to be sent. Moreover, where the underlying multi-processor architecture comprises the HyperTransport™ architecture from AMD, embodiments of the invention may be implemented with little or no alteration to the underlying architecture. That is, redirecting probes to a probe filtering unit and probe responses back to the probe filtering unit can be accomplished with little or no change to the current HyperTransport architecture and the implementation of that architecture. In addition, embodiments of the invention may be implemented in which non-probe related traffic (requests and responses) go directly between the nodes without having to go through the probe filtering unit.

While the invention has been particularly shown and described with reference to specific embodiments thereof, it will be understood by those skilled in the art that changes in the form and details of the disclosed embodiments may be made without departing from the spirit or scope of the invention. For example, embodiments of the present invention may be employed with multiple processor clusters connected through a point-to-point, switch, or bus architecture. In another example, multiple clusters of processors may share a single cache coherence controller, or multiple cache coherence controllers can be used in a single cluster. In addition, the mechanisms for facilitating local and remote probe filtering may be included in the same device or in separate devices. For example, the remote probe filtering functionality of a cache coherence controller in a multi-cluster system can be extended to facilitate local probe filtering. Alternatively, local probe filtering could be provided in a separate device deployed on a cluster's point-to-point interconnect.

In addition, although various advantages, aspects, and objects of the present invention have been discussed herein with reference to various embodiments, it will be understood that the scope of the invention should not be limited by reference to such advantages, aspects, and objects. Rather, the scope of the invention should be determined with reference to the appended claims.

What is claimed is:

1. A computer system comprising a plurality of processing nodes interconnected by a first point-to-point architecture, each processing node having a cache memory associated

therewith, the computer system further comprising a probe filtering unit which is operable to receive probes corresponding to memory lines from the processing nodes and to transmit the probes only to selected ones of the processing nodes with reference to probe filtering information representative of states associated with selected ones of the cache memories.

2. The computer system of claim **1** wherein the probe filtering unit corresponds to an additional node interconnected with the plurality of processing nodes via the first point-to-point architecture.

3. The computer system of claim **2** wherein the additional node comprises a cache coherence controller, and the probe filtering information comprises a cache coherence directory which includes entries corresponding to memory lines stored in the selected cache memories.

4. The computer system of claim **1** wherein the plurality of processing nodes comprises a first cluster of processors, the computer system comprising a plurality of clusters of processors including the first cluster, the plurality of clusters being interconnected via a second point-to-point architecture.

5. The computer system of claim **4** further comprising a cache coherence controller on the first point-to-point architecture which is operable to facilitate interconnection of the first cluster with others of the plurality of clusters via the second point-to-point architecture.

6. The computer system of claim **5** wherein the cache coherence controller comprises the probe filtering unit, and the probe filtering information comprises a cache coherence directory.

7. The computer system of claim **1** wherein the first point-to-point architecture comprises a HyperTransport architecture.

8. The computer system of claim **1** wherein each of the processing nodes is operable to transmit the probes only to the probe filtering unit.

9. The computer system of claim **8** wherein each of the processing nodes has at least one routing table associated therewith which governs which portions of the first point-to-point architecture the associated processing node employs for communicating with others of the processing nodes, the at least one routing table in each of the processing nodes being configured to direct all of the probes to the probe filtering unit.

10. The computer system of claim **9** wherein the at least one routing table in each of the processing nodes is configured to direct all broadcasts to the probe filtering unit.

11. The computer system of claim **1** wherein each of the processing nodes is programmed to complete a memory transaction after receiving a first number of responses to a first probe, the first number being fewer than the number of processing nodes.

12. The computer system of claim **11** wherein the probe filtering unit has temporary storage associated therewith for holding read response data from one of the cache memories, and the first number is one.

13. The computer system of claim **11** wherein the probe filtering unit is operable to forward read response data to a requesting node before accumulating all probe responses associated with the memory transaction, and the first number is two.

14. The computer system of claim **1** wherein the probe filtering unit is further operable to modify the probes such that the selected processing nodes transmit responses to the probes to the probe filtering unit.

15. The computer system of claim **1** wherein the probe filtering unit is operable to accumulate responses to each probe, and respond to requesting nodes in accordance with the accumulated responses.

16. A probe filtering unit for use in a computer system comprising a plurality of processing nodes interconnected by a first point-to-point architecture, each processing node having a cache memory associated therewith, the probe filtering unit being operable to receive probes corresponding to memory lines from the processing nodes and to transmit the probes only to selected ones of the processing nodes with reference to probe filtering information representative of states associated with selected ones of the cache memories.

17. An integrated circuit comprising the probe filtering unit of claim **16**.

18. The integrated circuit of claim **17** wherein the integrated circuit comprises an application-specific integrated circuit.

19. At least one computer-readable medium having data structures stored therein representative of the probe filtering unit of claim **16**.

20. The at least one computer-readable medium of claim **19** wherein the data structures comprise a simulatable representation of the probe filtering unit.

21. The at least one computer-readable medium of claim **20** wherein the simulatable representation comprises a netlist.

22. The at least one computer-readable medium of claim **19** wherein the data structures comprise a code description of the probe filtering unit.

23. The at least one computer-readable medium of claim **22** wherein the code description corresponds to a hardware description language.

24. A set of semiconductor processing masks representative of at least a portion of the probe filtering unit of claim **16**.

25. A computer implemented method for reducing probe traffic in a computer system comprising a plurality of processing nodes interconnected by a first point-to-point architecture, each processing node having a cache memory associated therewith, the method comprising:

transmitting a probe from a first one of the processing nodes only to a probe filtering unit, the probe corresponding to a memory line;

evaluating the probe with the probe filtering unit to determine whether a valid copy of the memory line is in any of the cache memories, the evaluating being done with reference to probe filtering information associated with the probe filtering unit and representative of states associated with selected ones of the cache memories;

transmitting the probe from the probe filtering unit only to selected ones of the processing nodes identified by the evaluating;

accumulating probe responses from the selected processing nodes with the probe filtering unit; and

responding to the probe from the first processing node only with the probe filtering unit.

* * * * *