US 20110004733A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2011/0004733 A1

Krakirian et al. (43) Pub. Date: Jan. 6, 2011

# (54) NODE IDENTIFICATION FOR DISTRIBUTED SHARED MEMORY SYSTEM

(75) Inventors: **Shahe Hagop Krakirian**, Palo Alto, CA (US); **Isam Akkawi**, Aptos, CA (US)

Correspondence Address:
**Huawei Technologies Co., Ltd.**
**IPR Dept., Building B1-3-A,, Huawei Industrial Base, Bantian**
**Shenzhen Guangdong 518129 (CN)**

(73) Assignee: **3 Leaf Networks**, Santa Clara, CA (US)

(21) Appl. No.: **12/755,113**

(22) Filed: **Apr. 6, 2010**

**Related U.S. Application Data**

(63) Continuation of application No. 11/740,432, filed on Apr. 26, 2007, now Pat. No. 7,715,400.

**Publication Classification**

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

(52) **U.S. Cl.** ................................ **711/147**; 711/E12.001

(57) **ABSTRACT**

An example embodiment of the present invention provides processes relating to a connection/communication protocol and a memory-addressing scheme for a distributed shared memory system. In the example embodiment, a logical node identifier comprises bits in the physical memory addresses used by the distributed shared memory system. Processes in the embodiment include logical node identifiers in packets which conform to the protocol and which are stored in a connection control block in local memory. By matching the logical node identifiers in a packet against the logical node identifiers in the connection control block, the processes ensure reliable delivery of packet data. Further, in the example embodiment, the. logical node identifiers are used to create a virtual server consisting of multiple nodes in. the distributed shared memory system.

| Bit / DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1-4 | Etherent Header | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0<br>1 | RDP header | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2<br>.<br>.<br>n | RDP payload (0 to 188 DWORDs) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| (n+1)<br>(n+2) | RDP Trailer (0, 1 or 2 DWORDs) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | Ethernet FCS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

RDP Over Ethernet Packet Format

| Bit / DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | DA [1] | | | | | | | | DA [0] | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | DA [5] | | | | | | | | DA [4] | | | | | | | | DA [3] | | | | | | | | DA [2] | | | | | | | |
| 2 | SA[3] | | | | | | | | SA[2] | | | | | | | | SA[1] | | | | | | | | SA[0] | | | | | | | |
| 3 | 0x00 | | | | | | | | 0x81 | | | | | | | | SA[5] | | | | | | | | SA[4] | | | | | | | |
| 4 | Len[0] (LSB) | | | | | | | | Len[1] (MSB) | | | | | | | | VID[0] (LSB) | | | | | | | | Pr | | | 0 | VID[1] (MSB) | | | |

Ethernet Header for RDP Packet

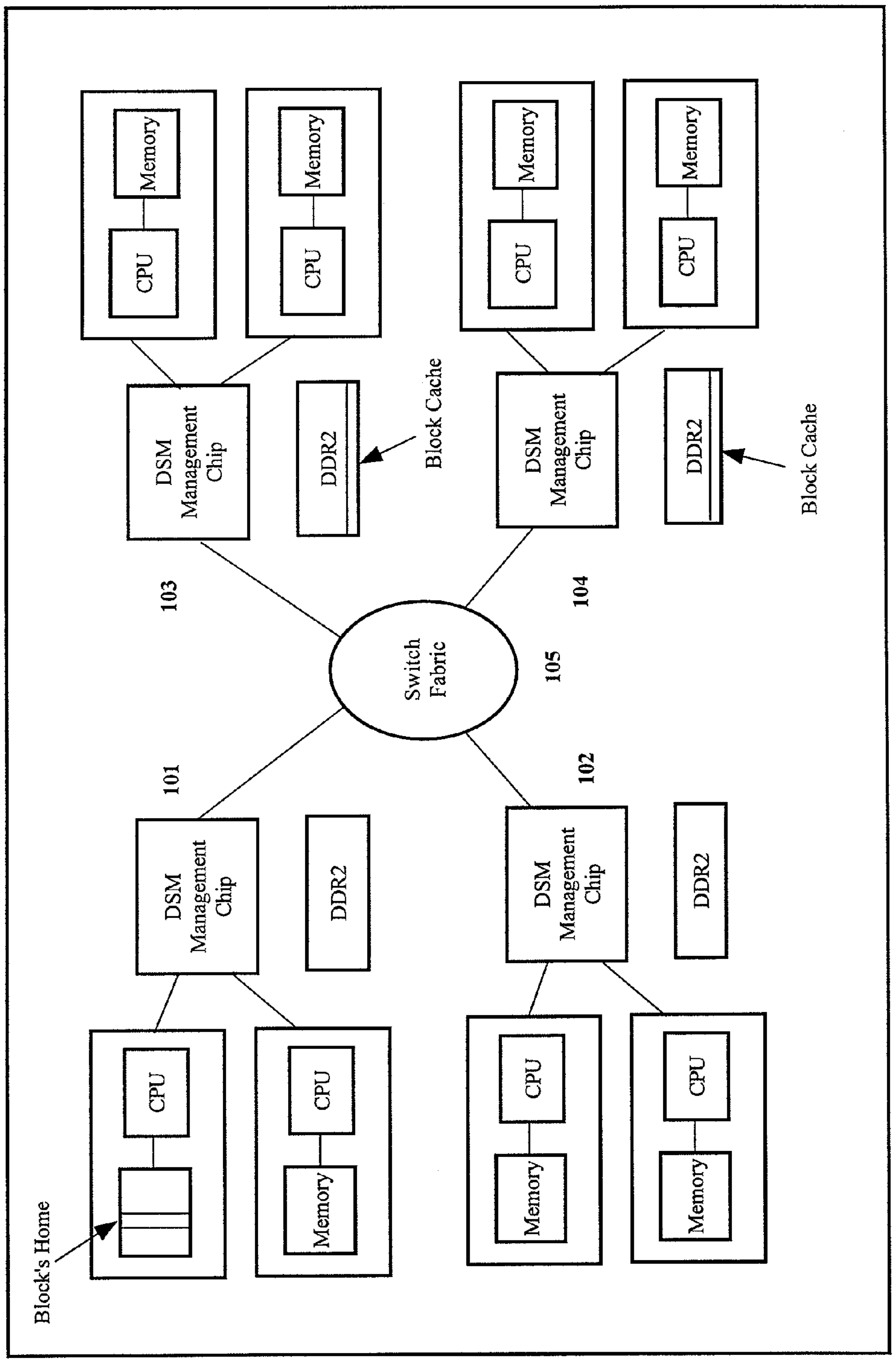
Block's Home
CPU
Memory
CPU
DSM Management Chip
DDR2
101
CPU
Memory
CPU
Memory
DSM Management Chip
DDR2
102
Switch Fabric
105
103
DSM Management Chip
DDR2
Block Cache
CPU
Memory
CPU
Memory
104
DSM Management Chip
DDR2
Block Cache
CPU
Memory
CPU
Memory

Fig. 1

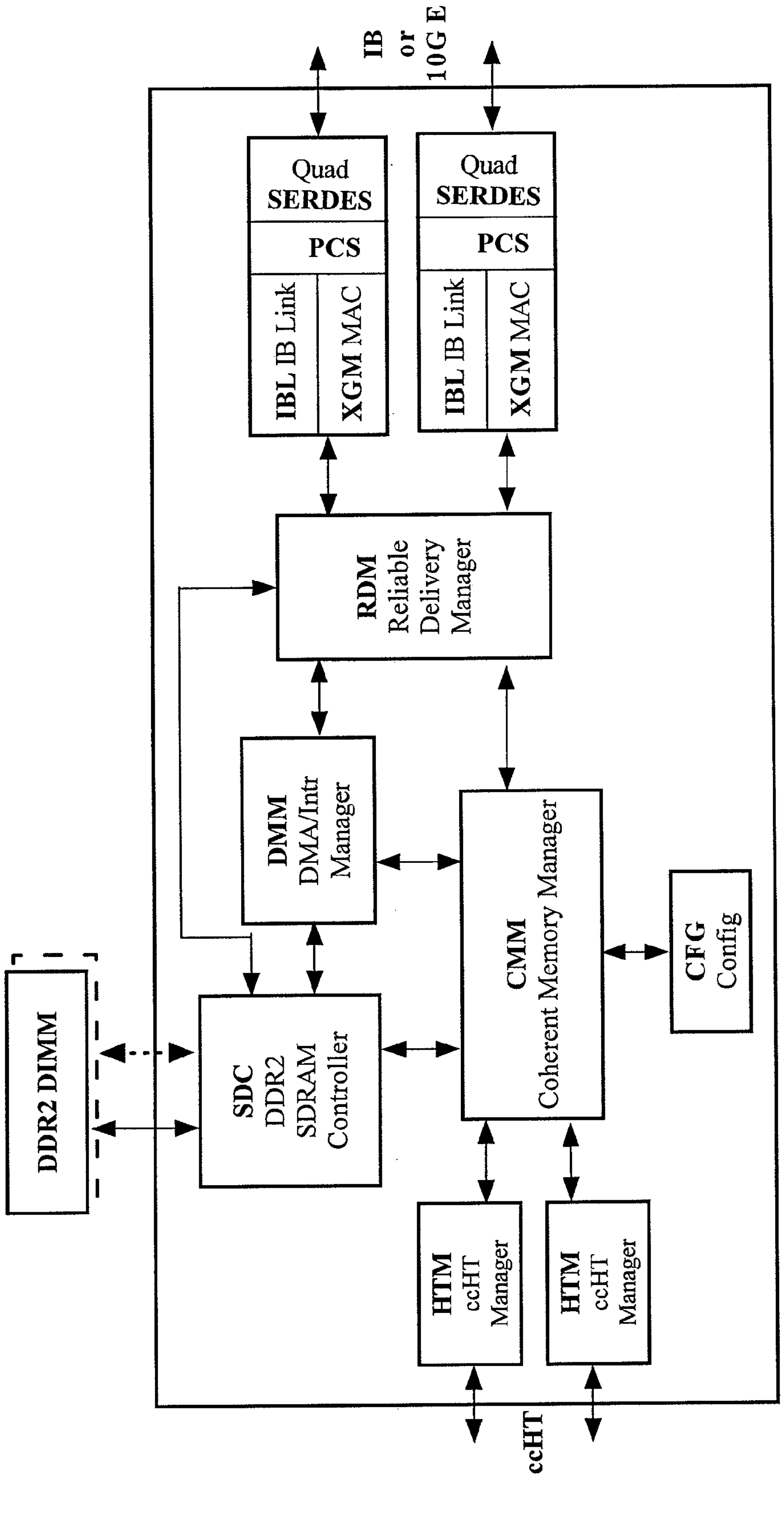
IB
or
10G E
Quad
SERDES
PCS
IBL IB Link
XGM MAC
Quad
SERDES
PCS
IBL IB Link
XGM MAC
RDM
Reliable
Delivery
Manager
DMM
DMA/Intr
Manager
CMM
Coherent Memory Manager
CFG
Config
DDR2 DIMM
SDC
DDR2
SDRAM
Controller
HTM
ccHT
Manager
HTM
ccHT
Manager
ccHT

Fig. 2

| Bit / DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1-4 | Etherent Header | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0<br>1 | RDP header | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2<br>.<br>.<br>n | RDP payload (0 to 188 DWORDs) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| (n+1)<br>(n+2) | RDP Trailer (0, 1 or 2 DWORDs) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | Ethernet FCS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

RDP Over Ethernet Packet Format

| Bit / DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | DA [1] | | | | | | | | DA [0] | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | DA [5] | | | | | | | | DA [4] | | | | | | | | DA [3] | | | | | | | | DA [2] | | | | | | | |
| 2 | SA[3] | | | | | | | | SA[2] | | | | | | | | SA[1] | | | | | | | | SA[0] | | | | | | | |
| 3 | 0x00 | | | | | | | | 0x81 | | | | | | | | SA[5] | | | | | | | | SA[4] | | | | | | | |
| 4 | Len[0] (LSB) | | | | | | | | Len[1] (MSB) | | | | | | | | VID[0] (LSB) | | | | | | | | Pr | | | 0 | VID[1] (MSB) | | | |

Ethernet Header for RDP Packet

Fig. 3

| Bit<br>DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0<br>1 | IB Local Route Header (LRH) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0<br>1 | RDP Header | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2<br>.<br>.<br>n | RDP payload (0 to 188 DWORDs) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| (n+1)<br>(n+2) | RDP Trailer (0, 1 or 2 DWORDs) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | IB Invariant CRC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | IB Variant CRC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

RDP Over InfiniBand Packet Format

| Bit<br>DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | VL | | | | LVer | | | | SL | | | | rsv | | LNH | | Destination Local ID | | | | | | | | | | | | | | | |
| 1 | rsv | | | | | Length | | | | | | | | | | | Source Local ID | | | | | | | | | | | | | | | |

InfiniBand Local Route Header for RDP Packet

Fig. 4

RDP Packet Format

| Bit / DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0<br>1 | RDP Header | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2<br>.<br>.<br>n | RDP Payload (0 to 188 DWORDs) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| (n+1)<br>(n+2) | RDP Trailer (0, 1 or 2 DWORDs) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Header for RDP Packet

| Bit / DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0x3 | | Ver | | AP | | VC | | CmdCode | | | | | | | | F | SendRN | | | SendSN_LSB | | | | | | | | | | | |
| 1 | DstLNID | | | | | | | | | | | | | | | | SrcLNID | | | | | | | | | | | | | | | |

Optional Trailer for RDP Packet

| Bit / DWORD | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (n+1) | N0 | AckRN0 | | | AckSN0 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| (n+2) | N1 | AckRN1 | | | AckSN1 | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Fig. 5

| DWORD \ Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | PS | Reserved | | | | | | | FAIL STAT | | | | DOWN STAT | | | | MY_LNID | | | | | | | | | | | | | | | |
| 1 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | Path 0 Middle Fabric Address | | | | | | | | | | | | | | | | Path 0 Low Fabric Address | | | | | | | | | | | | | | | |
| 3 | VD | PT | TMR SEL | | Reserved | | | | | | | | | | | | Path 0 High Fabric Address | | | | | | | | | | | | | | | |
| 4 | Path 1 Middle Fabric Address | | | | | | | | | | | | | | | | Path 1 Low Fabric Address | | | | | | | | | | | | | | | |
| 5 | VD | PT | TMR SEL | | Reserved | | | | | | | | | | | | Path 1 High Fabric Address | | | | | | | | | | | | | | | |
| 6 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | Transmit Replay Buffer Head and Tail Pointers for VC0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | Transmit Replay Buffer Head and Tail Pointers for VC0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | Transmit Replay Buffer Head and Tail Pointers for VC1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | Transmit Replay Buffer Head and Tail Pointers for VC1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | Transmit Replay Buffer Head and Tail Pointers for VC2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | Transmit Replay Buffer Head and Tail Pointers for VC2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | Transmit Replay Buffer Head and Tail Pointers for VC3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | Transmit Replay Buffer Head and Tail Pointers for VC3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | Send Sequence Number for VC1 | | | | | | | | | | | | | | | | Send Sequence Number for VC0 | | | | | | | | | | | | | | | |
| 17 | Send Sequence Number for VC3 | | | | | | | | | | | | | | | | Send Sequence Number for VC2 | | | | | | | | | | | | | | | |
| 18 | Expected Sequence Number for VC1 | | | | | | | | | | | | | | | | Expected Sequence Number for VC0 | | | | | | | | | | | | | | | |
| 19 | Expected Sequence Number for VC3 | | | | | | | | | | | | | | | | Expected Sequence Number for VC2 | | | | | | | | | | | | | | | |
| 20 | Transmitted Packet Count for VC0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | Transmitted Packet Count for VC1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | Transmitted Packet Count for VC2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | Transmitted Packet Count for VC3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | Received Packet Count for VC0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 25 | Received Packet Count for VC1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26 | Received Packet Count for VC2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 27 | Received Packet Count for VC3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 28 | Replayed Packet Count for VC1 | | | | | | | | | | | | | | | | Replayed Packet Count for VC0 | | | | | | | | | | | | | | | |
| 29 | Replayed Packet Count for VC3 | | | | | | | | | | | | | | | | Replayed Packet Count for VC2 | | | | | | | | | | | | | | | |
| 30 | Received Send SN Error Count | | | | | | | | | | | | | | | | Received ECRC Error Count | | | | | | | | | | | | | | | |
| 31 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

FIG. 6

Table 7.1

| Assigning Server | Node | Assigned LNID |
|---|---|---|
| Virtual Server A | VS Node A0 | 00 |
| | VS Node A1 | 01 |
| | VS Node A2 | 02 |
| | I/O Server D | 16 |
| Virtual Server B | VS Node B5 | 01 |
| | VS Node B1 | 05 |
| | I/O Server D | 18 |
| App Server C | I/O Server D | 03 |
| I/O Server D | VS Node A0 | 00 |
| | VS Node A1 | 02 |
| | VS Node A2 | 04 |
| | VS Node B5 | 06 |
| | VS Node B1 | 08 |
| | App Server C | 10 |

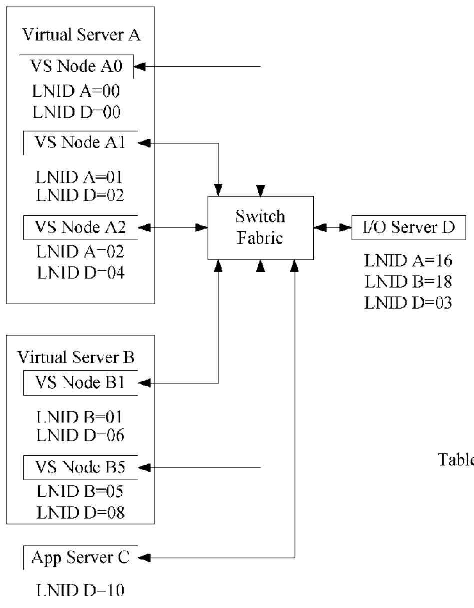

Fig 7

Table 7.2

| Source Node | Destination Node | srcLNID | DstLNID |
|---|---|---|---|
| VS Node A0 | VS Node A1 | 00 | 01 |
| VS Node A0 | VS Node A2 | 00 | 02 |
| VS Node A1 | VS Node A0 | 01 | 00 |
| VS Node A1 | VS Node A2 | 01 | 02 |
| VS Node A2 | VS Node A0 | 02 | 00 |
| VS Node A2 | VS Node A1 | 02 | 01 |
| VS Node B1 | VS Node B5 | 01 | 05 |
| VS Node B5 | VS Node B1 | 05 | 01 |
| VS Node A0 | I/O Server D | 00 | 16 |
| VS Node A1 | I/O Server D | 02 | 16 |
| VS Node A2 | I/O Server D | 04 | 16 |
| VS Node B1 | I/O Server D | 06 | 18 |
| VS Node B2 | I/O Server D | 08 | 18 |
| App Server C | I/O Server D | 10 | 03 |
| I/O Server D | VS Node A0 | 16 | 00 |
| I/O Server D | VS Node A1 | 16 | 02 |
| I/O Server D | VS Node A2 | 16 | 04 |
| I/O Server D | VS Node B1 | 18 | 06 |
| I/O Server D | VS Node B2 | 18 | 08 |
| I/O Server D | App Server C | 03 | 10 |

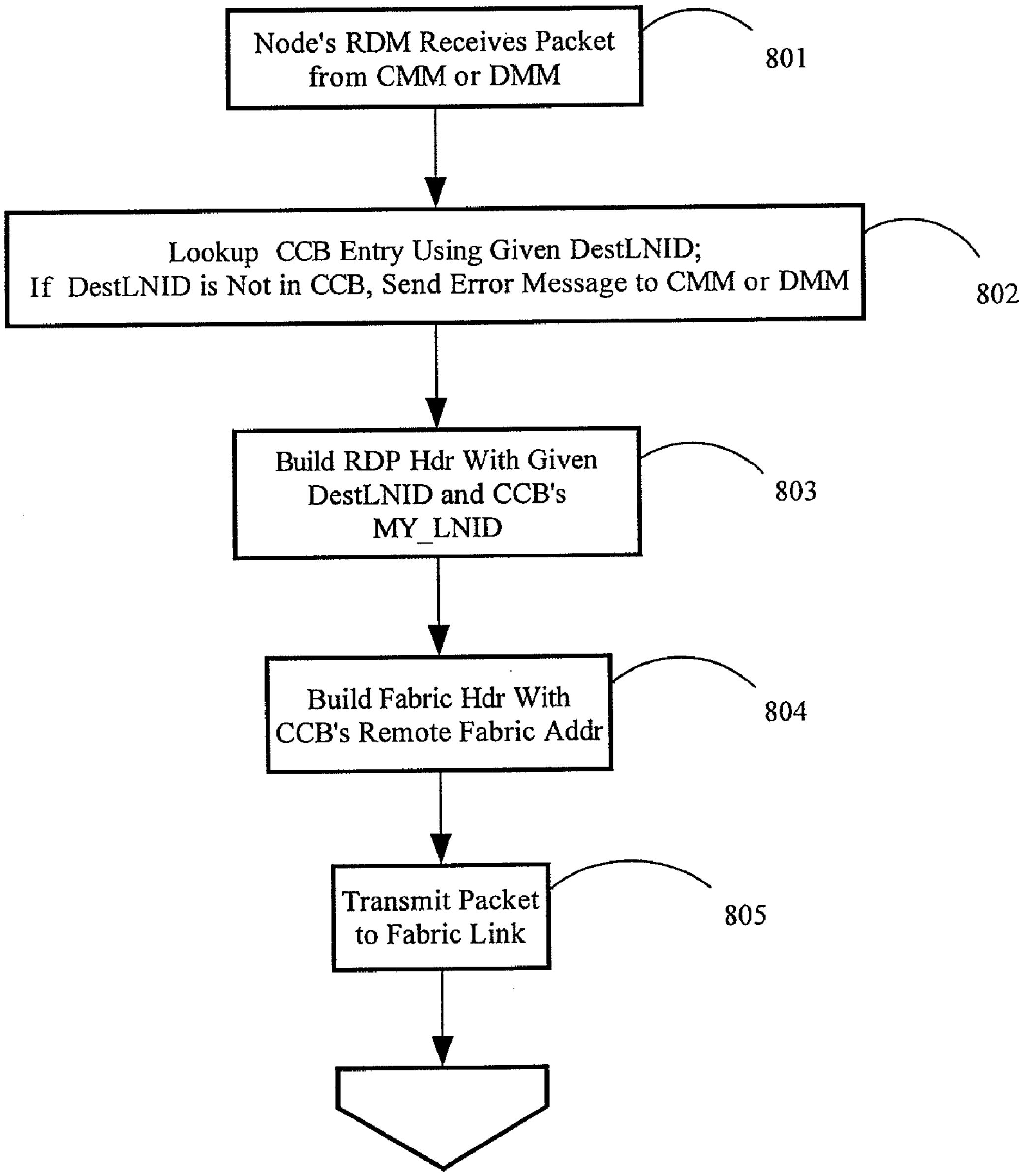
Node's RDM Receives Packet from CMM or DMM
801
Lookup CCB Entry Using Given DestLNID; If DestLNID is Not in CCB, Send Error Message to CMM or DMM
802
Build RDP Hdr With Given DestLNID and CCB's MY_LNID
803
Build Fabric Hdr With CCB's Remote Fabric Addr
804
Transmit Packet to Fabric Link
805

FIG. 8

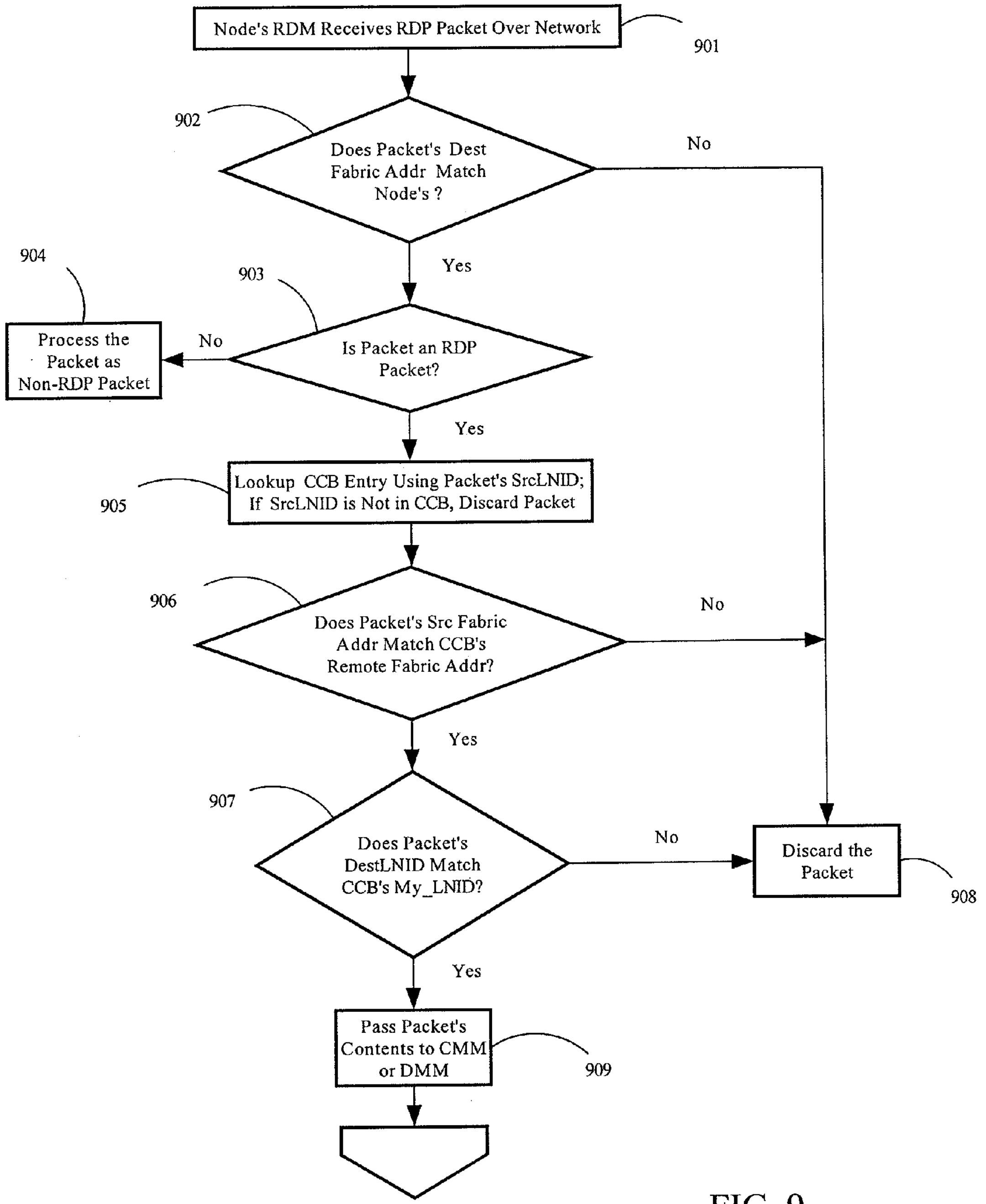
Node's RDM Receives RDP Packet Over Network
901
902
Does Packet's Dest Fabric Addr Match Node's ?
No
Yes
904
903
Process the Packet as Non-RDP Packet
No
Is Packet an RDP Packet?
Yes
Lookup CCB Entry Using Packet's SrcLNID; If SrcLNID is Not in CCB, Discard Packet
905
906
Does Packet's Src Fabric Addr Match CCB's Remote Fabric Addr?
No
Yes
907
Does Packet's DestLNID Match CCB's My_LNID?
No
Discard the Packet
908
Yes
Pass Packet's Contents to CMM or DMM
909

FIG. 9

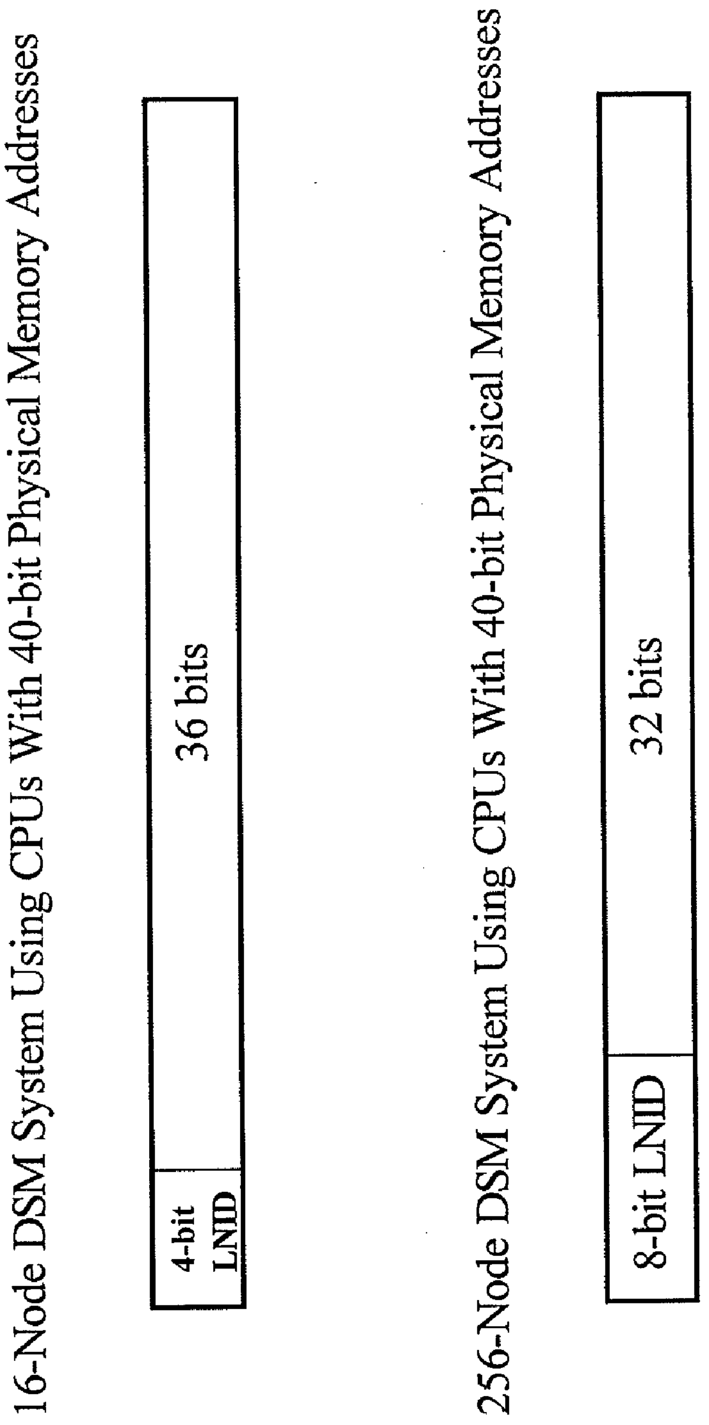
16-Node DSM System Using CPUs With 40-bit Physical Memory Addresses
4-bit LNID
36 bits
256-Node DSM System Using CPUs With 40-bit Physical Memory Addresses
8-bit LNID
32 bits

Fig. 10

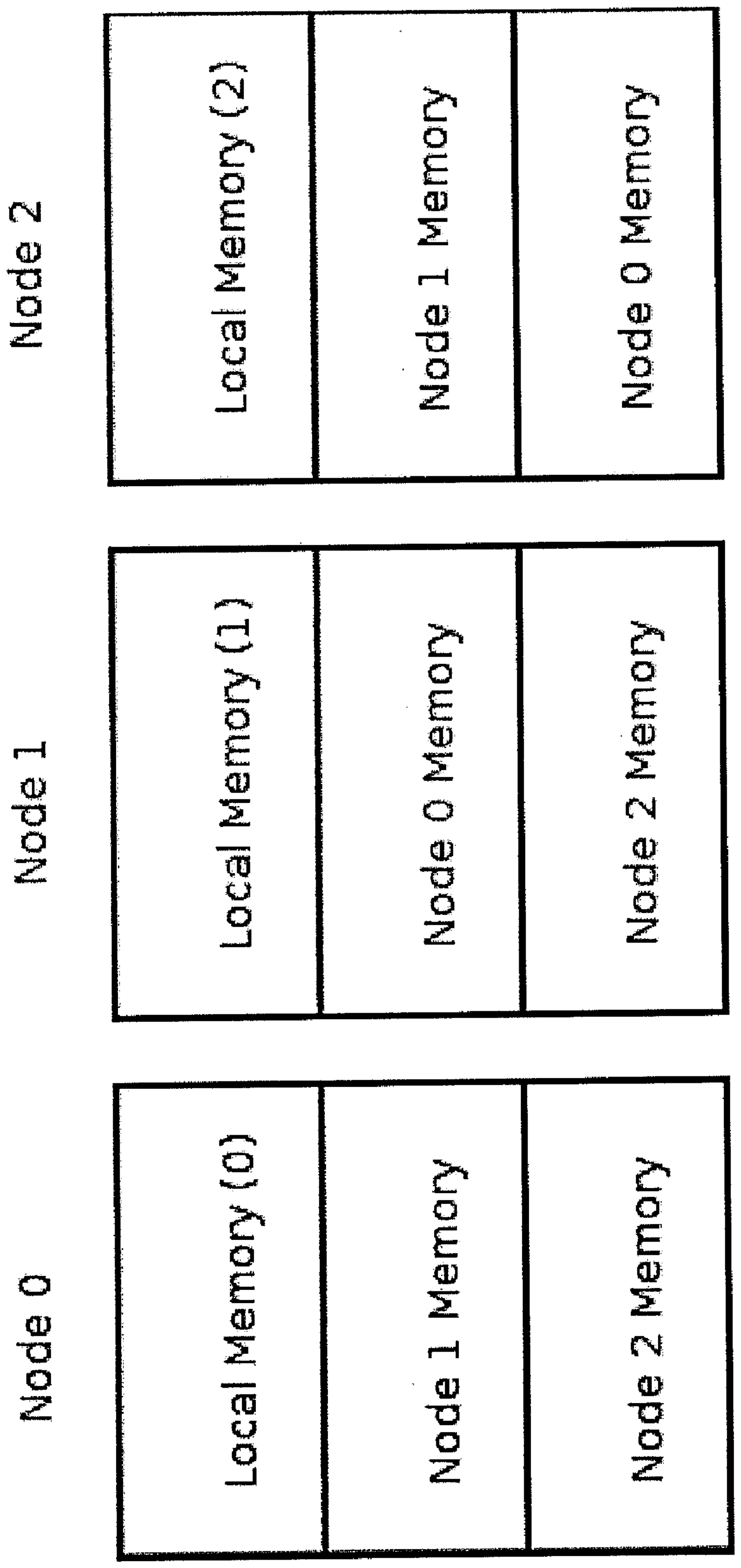
Node 0
Local Memory (0)
Node 1 Memory
Node 2 Memory
Node 1
Local Memory (1)
Node 0 Memory
Node 2 Memory
Node 2
Local Memory (2)
Node 1 Memory
Node 0 Memory

Fig. 11

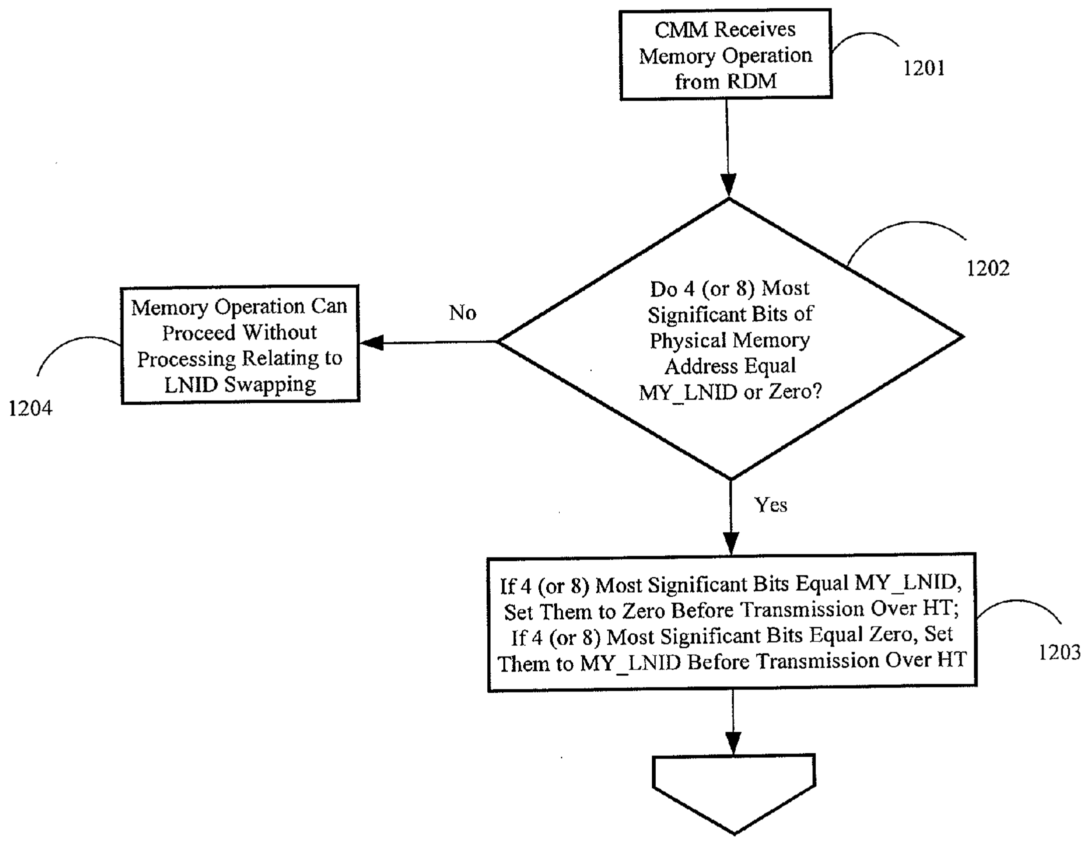
CMM Receives Memory Operation from RDM
1201
Do 4 (or 8) Most Significant Bits of Physical Memory Address Equal MY_LNID or Zero?
1202
No
Memory Operation Can Proceed Without Processing Relating to LNID Swapping
1204
Yes
If 4 (or 8) Most Significant Bits Equal MY_LNID, Set Them to Zero Before Transmission Over HT; If 4 (or 8) Most Significant Bits Equal Zero, Set Them to MY_LNID Before Transmission Over HT
1203

FIG. 12

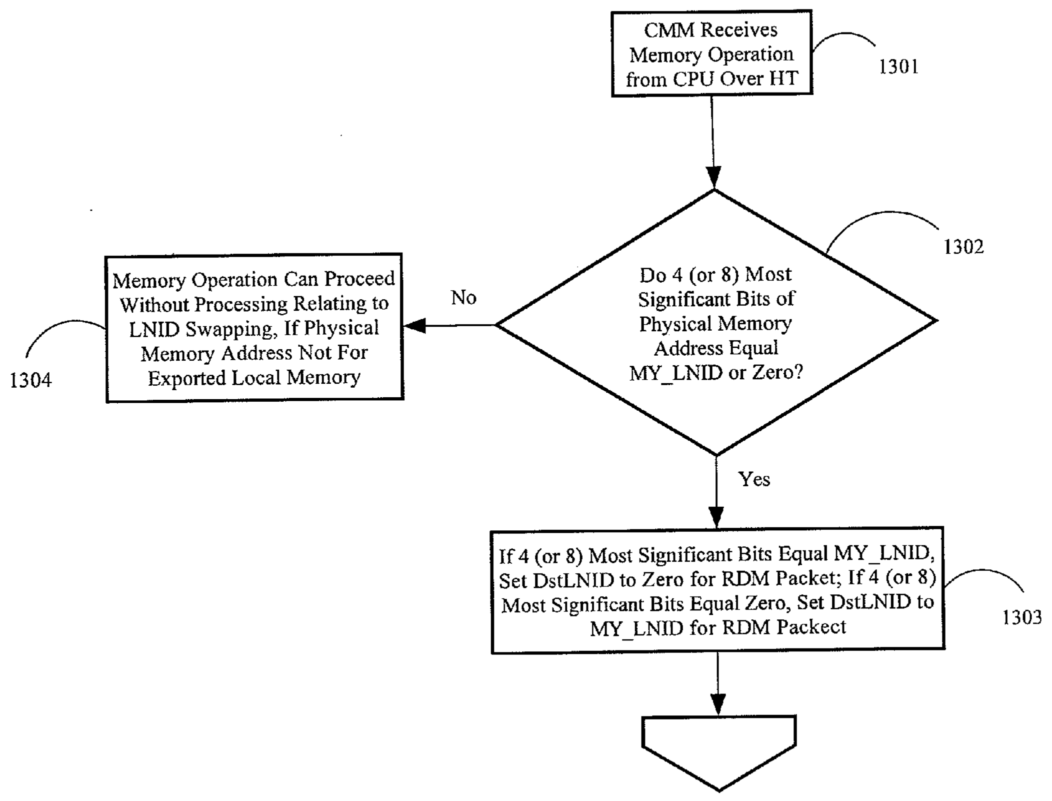
CMM Receives Memory Operation from CPU Over HT
1301
Do 4 (or 8) Most Significant Bits of Physical Memory Address Equal MY_LNID or Zero?
1302
No
Memory Operation Can Proceed Without Processing Relating to LNID Swapping, If Physical Memory Address Not For Exported Local Memory
1304
Yes
If 4 (or 8) Most Significant Bits Equal MY_LNID, Set DstLNID to Zero for RDM Packet; If 4 (or 8) Most Significant Bits Equal Zero, Set DstLNID to MY_LNID for RDM Packect
1303

FIG. 13

# NODE IDENTIFICATION FOR DISTRIBUTED SHARED MEMORY SYSTEM

## CROSS-REFERENCE TO RELATED APPLICATIONS

**[0001]** The present application is a continuation application of pending U.S. patent application Ser. No. 11/740,432 filed Apr. 26, 2007 and entitled "Node Identification for Distributed Shared Memory System," which is related to the following commonly-owned U.S. utility patent application, filed on Jan. 29, 2007, whose disclosure is incorporated herein by reference in its entirety for all purposes: U.S. patent application Ser. No. 11/668,275, entitled "Fast Invalidation for Cache Coherency in Distributed Shared Memory System".

## TECHNICAL FIELD

**[0002]** The present disclosure relates to an identification process for the nodes in a distributed shared memory system.

## BACKGROUND

**[0003]** A distributed shared memory (DSM) is a multiprocessor system in which the processors in the system are connected by a scalable interconnect, such as an InfiniBand switched fabric communications link, instead of a bus. DSM systems present a single memory image to the user, but the memory is physically distributed at the hardware level. Typically, each processor has access to a large shared global memory in addition to a limited local memory, which might be used as a component of the large shared global memory and also as a cache for the large shared global memory. Naturally, each processor will access the limited local memory associated with the processor much faster than the large shared global memory associated with other processors. This discrepancy in access time is called non-uniform memory access (NUMA).

**[0004]** A major technical challenge in DSM systems is ensuring that the each processor's memory cache is consistent with each other processor's memory cache. Such consistency is called cache coherence. To maintain cache coherence in larger distributed systems, additional hardware logic (e.g., a chipset) or software is used to implement a coherence protocol, typically directory-based, chosen in accordance with a data consistency model, such as strict consistency. DSM systems that maintain cache coherence are called cache-coherent NUMA (ccNUMA).

**[0005]** Typically, if additional hardware logic is used, a node in the system will comprise a chip that includes the hardware logic and one or more processors and will be connected to the other nodes by the scalable interconnect. For purposes of initial connection and later communication between nodes, the system might employ node identifiers, e.g., serial, random, or centrally assigned numbers, which in turn might be used as part of an address for physical memory residing on the node.

## SUMMARY

**[0006]** In particular embodiments, the present invention provides methods, apparatuses, and systems directed to node identification in a DSM system. In one particular embodiment, the present invention provides node-identification processes for use with a connection/communication protocol and a memory-addressing, scheme in a DSM system.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0007]** FIG. **1** is a block diagram showing a DSM system, which system might be used with some embodiments of the present invention.

**[0008]** FIG. **2** is a block diagram showing some of the physical and functional components of an example DSM-management chip or logic circuit, which chip might be used as part of a node with some embodiments of the present invention.

**[0009]** FIG. **3** is a diagram showing the format of an RDP over Ethernet packet and its header, which formats might be used in some embodiments of the present invention.

**[0010]** FIG. **4** is a diagram showing the format of an RDP over InfiniBand packet and its header, which formats might be used in some embodiments of the present invention.

**[0011]** FIG. **5** is a diagram showing the format of an RDP packet, its header, and its optional trailer, which formats might be used in some embodiments of the present invention.

**[0012]** FIG. **6** is a diagram showing the format of a connection control block, which format might be used in some embodiments of the present invention.

**[0013]** FIG. **7** is a diagram showing an example illustrating the use of LNIDs with respect to the RDP protocol, which protocol might be used with an embodiment of the present invention.

**[0014]** FIG. **8** is a diagram showing a flowchart of an example process for building an RDP packet for transmission over the switched fabric network, which process might be used with an embodiment of the present invention.

**[0015]** FIG. **9** is a diagram showing a flowchart of an example process for validating an RDP packet received over the switched fabric network, which process might be used with an embodiment of the present invention.

**[0016]** FIG. **10** is a diagram showing the format of a 40-bit physical memory address in a 16-node DSM system and the format of a 40-bit physical memory address in a 256-node DSM system, which formats might be used with embodiments of the present invention.

**[0017]** FIG. **11** is a diagram showing, for didactic purposes, the local views of a physical address space for a virtual server comprised of three nodes.

**[0018]** FIG. **12** is a diagram showing a flowchart of an example process for altering a physical memory address prior to transmission over a HyperTransport bus, which process might be used with an embodiment of the present invention.

**[0019]** FIG. **13** is a diagram showing a flowchart of an example process for altering a physical memory address prior to transmission over a switched fabric, which process might be used with an embodiment of the present invention.

## DETAILED DESCRIPTION OF THE INVENTION

**[0020]** The following example embodiments are described and illustrated in conjunction with apparatuses, methods, and systems which are meant to be examples and illustrative, not limiting in scope.

### A. ccNUMA DMA System with DSM-Management Chips

**[0021]** A DSM system has been developed that provides cache-coherent non-uniform memory access (ccNUMA) through the use of a DSM-management chip. In a particular embodiment, a DSM system may comprise a distributed computer network of up to 16 nodes, connected by a switched fabric, where each node includes two or more Opteron CPUs and one DSM management chip. In another embodiment, this DSM system comprises up to 256 nodes connected by the switched fabric.

**[0022]** The DSM system allows the creation of a multi-node virtual server which is a virtual machine consisting of multiple CPUs belonging to two or more nodes. In some embodiments, the nodes use a connection/communication protocol to communicate with each other and with virtual I/O servers in the DSM system. Enforcement of the connection/communication protocol is also handled by the DSM-management chip. Consequently, virtual I/O servers include a DSM-management chip, though they do not contribute any physical memory to the DSM system and consequently do not make use of the chip's functionality directly related to cache coherence, in particular embodiments. For a further description of a virtual I/O server, see U.S. patent application Ser. No. 11/624,542, entitled "Virtualized Access to I/O Subsystems", and U.S. patent application Ser. No. 11/624,573, entitled "Virtual Input/Output Server", both fled on Jan. 18, 2007 which are incorporated herein by reference for all purposes. As explained below, the connection/communication protocol uses an identifier called a logical node identifier (LNID) to identify source and destination nodes for packets that travel over the switched fabric.

**[0023]** FIG. **1** is a diagram showing a ccNUMA DSM system, which system might be used with a particular embodiment of the invention. In this DSM system, four nodes (labeled **101**, **102**, **103**, and **104**) are connected to each other over a switched fabric (labeled **105**) such as Ethernet or InfiniBand. In turn, each of the four nodes includes two Opteron CPUs, a DSM-management chip, and memory in the form of DDR2 S DRAM (double-data-rate two synchronous dynamic random access memory). In this embodiment, each Opteron CPU includes a local main memory connected to the CPU. This DSM system provides NUMA (non-uniform memory access) since each CPU can access its own local main memory faster than it can access the other memories shown in FIG. **1**.

**[0024]** Also as shown in FIG. **1**, a block of memory has its "home" in the local main memory of one of the Opteron CPUs in node **101**. That is to say, this local main memory is where the system's version of the memory block is stored, regardless of whether there are any cached copies of the block. Such cached copies are shown in the DDR2s for nodes **103** and **104**. The DSM-management chip includes hardware logic (e.g., the CMM) to enforce a coherence protocol and make the DSM system cache-coherent (e.g., ccNUMA) when multiple nodes are caching copies of the same block of memory.

## B. Example System Architecture of a DSM-Management Chip

**[0025]** FIG. **2** is diagram showing the physical and functional components of a DSM-management chip, which chip might be used as part of a node with particular embodiments of the invention. The DSM-management chip includes interconnect functionality facilitating communications with one or more processors, which might be Opteron processors offered by Advanced Micro Devices (AMD), Inc., of Sunnyvale, Calif., in some embodiments. As FIG. **2** illustrates, the DSM-management chip includes two HyperTransport Managers (HTM), each of which manages communications to and from a processor over a HT (HyperTransport) bus. More specifically, an HTM provides the PHY and link layer functionality for a cache coherent HT interface such as Opteron's ccHT. The HTM captures all received HT packets in a set of receive queues per interface (e.g., posted/non-posted command, request command, probe command and data) which are consumed by the Coherent Memory Manager (CMM). The HTM also captures packets from the CMM in' a similar set of transmit queues per interface and transmits those packets on the HT interface. As a result of the two HTMs, the DSM-management chip becomes a coherent agent with respect to any bus snoops broadcast over the cache-coherent HT bus by a processor's memory controller. Of course, other inter-chip or bus communications protocols might be used in other embodiments of the present invention.

**[0026]** Also as shown in FIG. **2**, the two HTMs are connected to a Coherent Memory Manager (CMM), which enforces a coherence protocol and thereby provides cache-coherent access to memory shared by the nodes that are part of the DSM fabric. In addition to interfacing with the Opteron processors through the HTM, the CMM interfaces with the fabric via the RDM (Reliable Delivery Manager). Additionally, the CMM provides interfaces to the HTM for DMA (Direct Memory Access) and configuration.

**[0027]** In some embodiments, the CMM behaves like both a processor cache on a cache-coherent (e.g., ccHT) bus and a memory controller on a cache-coherent (e.g., ccHT) bus, depending on the scenario. In particular, when a processor on a node performs an access to a home (or local) memory address, the home (or local) memory will generate a probe request that is used to snoop the caches of all the processors on the node. The CMM will use this probe to determine if it has exported the block of memory containing that address to another node and may generate DSM probes (over the fabric) to respond appropriately to the initial probe. In this scenario, the CMM behaves like a processor cache on the cache-coherent bus.

**[0028]** When a processor on a node performs an access to a remote memory, the processor will direct this access to the CMM. The CMM will examine the request and satisfy it from the local cache, if possible, and, in the process, generate any appropriate probes. If the request cannot be satisfied from the local cache, the CMM will send a DSM request to the remote memory's home node to (a) fetch the block of memory that contains the requested data or (b) request a state upgrade. In this case, the CMM will wait for the DSM response before it responds back to the processor. In this scenario, the CMM behaves like a memory controller on the ccHT bus.

**[0029]** The RDM manages the flow of packets across the DSM-management chip's two fabric interface ports. The RDM has two major clients, the CMM and the DMA Manager (DMM), which initiate packets to be transmitted and consume received packets. The RDM ensures reliable end-to-end delivery of packets using a connection/communication protocol called Reliable Delivery Protocol (RDP). On the fabric side, the RDM interfaces to the selected link/MAC (XGM for Ethernet, IBL for InfiniBand) for each of the two fabric ports. In particular embodiments, the fabric might connect nodes to other nodes. In other embodiments, the fabric might also connect nodes to virtual IO servers. In particular embodiments, the processes using LNIDs described below might be executed by the RDM.

**[0030]** The XGM provides a 10 G Ethernet MAC function, which includes framing, inter-frame gap handling, padding for minimum frame size, Ethernet FCS (CRC) generation and checking, and flow control using PAUSE frames. The XGM supports two link speeds: single data rate XAUI (10 Gbps) and double data rate XAUI (20 Gbps). In particular embodiments, the DSM-management chip has two instances of the XGM, one for each fabric port. Each XGM instance interfaces to the RDM, on one side, and to the associated PCS, on the other side.

**[0031]** The IBL provides a standard 4-lane IB link layer function, which includes link initialization, link state machine, CRC generation and checking, and flow control. The IBL block supports two link speeds, single data rate (8 Gbps) and double data rate (16 Gbps), with automatic speed negotiation. In particular embodiments, the DSM-management chip has two instances of the IBL, one for each fabric port. Each IBL instance interfaces to the RDM, on one side, and to the associated Physical Coding Sub-layer (PCS), on the other side.

**[0032]** The PCS, along with an associated quad-serdes, provides physical layer functionality for a 4-lane InfiniBand SDR/DDR interface, or a 10 G/20 G Ethernet XAUI/10GBase-CX4 interface. In particular embodiments, the DSM-management chip has two instances of the PCS, one for each fabric port. Each PCS instance interfaces to the associated IBL and XGM.

**[0033]** The DMM shown in FIG. **2** manages and executes direct memory access (DMA) operations over RDP, interfacing to the CMM block on the host side and the RDM block on the fabric side. For DMA, the DMM interfaces to software through the DmaCB table in memory and the on-chip DMA execution and completion queues. The DMM also handles the sending and receiving of RDP interrupt messages and non-RDP packets, and manages the associated inbound and outbound queues.

**[0034]** The DDR2 SDRAM Controller (SDC) attaches to a one or two external 240-pin DDR2 SDRAM DIMM, which is actually external to the DMS-management chip, as shown in both FIG. **1** and FIG. **2**. In particular embodiments, the SDC provides SDRAM access for the CMM and the DMM.

**[0035]** In some embodiments, the DSM-management chip might comprise an application specific integrated circuit (ASIC), whereas in other embodiments the chip might comprise a field-programmable gate array (FPGA). Indeed, the logic encoded in the chip could be implemented in software for DSM systems whose requirements might allow for longer latencies with respect to cache coherence, DMA, interrupts, etc.

## C. RDP Packets and Their Headers

**[0036]** FIG. **3** is a diagram showing the format of a packet for RDP over Ethernet and the packet's header, which formats might be used in some embodiments of the present invention. When RDP runs over the Ethernet MAC layer, an RDP packet is encapsulated in an Ethernet MAC frame. The Ethernet header of an encapsulated RDP packet is a VLAN-tagged header (where VLAN stands for virtual local area network). In FIG. **3**, SA identifies the 6-byte source MAC address and DA identifies the 6-byte destination MAC address.

**[0037]** The Reliable Delivery Protocol allows RDP and non-RDP packets to co-exist on the same fabric. When RDP runs over the Ethernet MAC layer, RDP and non-RDP packets are distinguished from each other by the presence of the VLAN header and the value of the Length/Type field following it. For an RDP packet: (a) the VLAN header is present, i.e., the first Length/Type field (following the last SA byte) has a value of 0x0081; and (b) the second Length/Type field (following the VLAN header) has a value less than 1536 (frame length). An Ethernet frame that does not satisfy both of the above conditions is a non-RDP packet.

**[0038]** FIG. **4** is a diagram showing the format of a packet for RDP over InfiniBand and the packet's header, which formats might be used in some embodiments of the present invention. It will be appreciated that the header includes fields for Source Local ID and Destination Local ID. When RDP runs over the IB link layer, an RDP packet is encapsulated into an IB packet. The format of an IB Local Transport Packet is used, although the 12-byte Base Transport Header (BTH) which is normally present after the Local Route Header (LRH) is replaced by the RDP header (8 bytes) and the first 4 bytes of the RDP payload. From the standpoint of the 113 standard, bits 31:24 of the first DWORD of the RDP Header is the OpCode field of Base Transport Header (BTH). The most significant two bits (31:30) of that field have a fixed value of 0x3 (binary 11) for RDP packets, which specifies a 'Manufacturer Specific OpCode'. The Rsv8 field of the BTH (bits 31:24 of the second DWORD) is not protected by the 32-bit IB Invariant CRC (ICRC). This corresponds to the most significant 8 bits of the DstLNID. Thus, these bits do not have end-to-end protection but do have point-to-point protection by the 16-bit Variant CRC (VCRC), which presents an insignificant risk of failure since the DstLNID is only used as a packet validation field at the destination node in conjunction with many other validation fields. A false match of a corrupted LNID MSB (most significant bit) with good VCRC has very low probability and would only occur if the connection parameters were set up inconsistently at the source and destination nodes.

**[0039]** When RDP runs over the InfiniBand link layer, RDP and non-RDP packets are distinguished by the values of the LNH field in the IB Local Route Header and the QpCode field in the IB Base Transport Header. For an RDP packet: (a) LNH=0x2 (IBA Local); and (b) OpCode bits [7:6]=0x3 (Manufacturer Specific OpCode). An InfiniBand packet that does not satisfy both of the above conditions is a non-RDP packet.

**[0040]** FIG. **5** is a diagram showing the format of an RDP packet and its header, which formats might be used in some embodiments of the present invention. An RDP packet consists of a header, payload, and optional trailer. As shown in FIG. **5**, another field in the RDP packet is the SrcLNID (Source Logical Node ID) which identifies the packet's source node. This is the connection identifier (i.e., remote LNID) at the destination node. This field is also 16 bits wide. Also as shown in FIG. **5**, one of the fields in an RDP packet is the DestLNID (Destination Logical Node ID) which identifies the packet's destination node. This is the connection identifier (i.e., remote LNID) at the source node. This field is 16 bits wide.

## D. Using LNIDs with RDP

**[0041]** In particular embodiments, the DSM system uses a software data structure called the connection control block (CCB), stored in local memory such as the local main memory shown in FIG. **1**, to facilitate implementation of the RDP protocol. The RDM uses a received packet's source LNID as an index into the CCB to find an entry for the connection corresponding to the packet. FIG. **6** is a diagram showing the format of a CCB entry for a single connection, which format might be used in some embodiments of the present invention. As shown in FIG. **6**, each entry records the fabric address for two paths, Path **0** and Path **1**, which may correspond to the two fabric interface ports shown connected to the RDM in FIG. **2**. In other embodiments, there might be more than two paths, corresponding to more than two fabric interface ports. It will be appreciated that the CCB entry has a field called MY LNID, which identifies the LNID for the RDM's node.

**[0042]** For an RDP connection between a pair of nodes, the node at each end uses an LNID to refer to the node at the other end. Within a multi-node virtual server (VS), every node is assigned a unique LNID, possibly by some management entity for the DSM system. For example, within a three-node VS, the LNID values might be 0, 1, and 2, or 1, 3, and 4, i.e., they not need to be sequentially incrementing from 0. In addition, every server (multi-node virtual server or standalone server) assigns a unique LNID to each node that communicates with it. For example, a standalone server node that communicates with the virtual server described above might be assigned an LNID value of 16 by the VS. If that same node communicates with another server, it may be assigned the same LNID or a different LNID by that server. Therefore, LNID assignments are unique from the standpoint of a given server, but they are not unique across servers.

**[0043]** An example of LNID assignments is shown in FIG. **7**. In the example, a virtual computing environment (VCE) consists of two virtual servers (A and B), an application server (C), and a virtual I/O server (D). In this example, virtual server A assigns LNID values 0, 1, and 2 to each of its own nodes (VS nodes A**0**, A**1**, and A**2**, respectively) and an LNID value of 16 to virtual I/O server D. Virtual server B assigns values of I and 5 to each of its own nodes (VS nodes B**1** and B**5**, respectively) and an LNID value of 18 to virtual I/O server D. Application server C assigns an LNID value of 3 to virtual I/O server D. Virtual I/O server D assigns LNID values 0, 2, and 4, to VS nodes A**0**, A**1** and A**2**, respectively, and LNID values of 6 and 8 to VS nodes B**1** and B**5**. Finally, virtual I/O server D assigns a value of 10 to application server C. These various assignments are collected and summarized in Table **7.1** in FIG. **7**.

**[0044]** Table **7.2** shows the SrcLNID and DstLNID values used in the headers of RDP packets exchanged between different node pairs. For example, VS nodes **A0** and **A1** both belong to virtual server A, so a packet from **A0** to **A1** will have a SrcLNID value of 0 (LNID assigned to A**0** by VS A), and a DstLNID value of 1 (LNID assigned to **A1** by VS A). As another example, a packet from **A1** to I/O server D will have a SrcLNID value of 2 (LNID assigned to **A1** by I/O server D) and a DstLNID value of 16 (LNID assigned by V S A to I/O server D).

**[0045]** FIG. **8** is a diagram showing a flowchart of an example process for building an RDP packet for transmission over the switched fabric network, which process might be used with an embodiment of the present invention. In the process's first step **801**, the node's Reliable Delivery Manager (RDM) receives a DestLNID and data for an RDP packet from the node's CMM or DMM. The RDM uses the packet's DestLNID to look up the entry corresponding to the DestLNID in the Connection Control Block (CCB), in step **802.1***f* there is no corresponding entry, the RDM sends an error message to the CMM or DMM, as the case may be. Then in step **803**, the RDM builds an RDP header for an RDP packet for the data, using the DestLNID and the CCB entry's MY LNID value. In step **804**, the RDM builds a fabric header for the RDP packet, using information in the CCB entry's remote fabric address. Once the RDP packet is complete, the RDM sends the packet to the fabric link for transmission to the remote node, in step **805**.

**[0046]** FIG. **9** is a diagram showing a flowchart of an example process for validating an RDP packet received over the switched fabric network, which process might be used with an embodiment of the present invention. In the process's first step **901**, a node's RDM receives an RDP packet over the switched fabric network. The RDM then checks to see whether the packet's destination fabric address (e.g., the 6-byte MAC DA in an Ethernet header or the Destination Local ID in an Infiniband LRH) matches the node's fabric address, in step **902**. If not, the RDM discards the packet. Otherwise, the RDM goes' to step **903** and determines whether the packet is an RDP packet. If not, the RDM will process the packet as a non-RDP packet, in step **904**. Otherwise, if the packet is an RDP packet, the RDM uses the packet's SrcLNID to look up the entry corresponding to the SrcLNID in the Connection Control Block (CCB), in step **905**. If there is no corresponding entry, the RDM discards the packet. Then the RDM goes to step **906** and checks to make sure that the packet's source fabric address (e.g., the 6-byte MAC SA in an Ethernet header or the Source Local ID in an Infiniband LRH) matches the CCB entry's remote fabric address (e.g., for Path **0** or Path **1**). If not, the RDM discards the packet. Otherwise, the RDM checks to determine whether the packet's DestLNID matches the CCB entry's MY_LNID, in step **907**. If not, the RDM discards the packet. But if there is a match, the RDM forwards the packet to the CMM or DMM for further processing.

## E. Using LNIDs With Memory-Addressing Scheme

**[0047]** As indicated earlier, the DSM system also uses LNIDs in its memory-addressing scheme. In particular embodiments, the physical memory address width is 40-bits (e.g., in DSM systems that use the present generation of Opteron CPUs), though it will be appreciated that there are numerous other suitable widths. FIG. **10** is a diagram showing the format of a 40-bit physical memory address in a 16-node DSM system and the format of a 40-bit physical memory address in a 256-node DSM system. As shown in FIG. **10**, the four most significant bits comprise an LNID in the 16-node DSM system and the eight most significant bits comprise an LNID in the 256-node DSM system.

**[0048]** In particular embodiments of the DSM system, the physical address space for a virtual server is arranged so that the local node's memory always starts at address **0** (zero). One reason for using this arrangement is compatibility with legacy system software, in particular embodiments. Specifically, with local memory starting at address **0**, system software (e.g., boot code) accesses local memory the same way that it does on a standard server. Another reason for using this arrangement is that it simplifies the address lookup in the CMM. For a memory read/write request from a local processor, an address in the lower 1116th or 11256th segment of the 40-bit address space is always local and all other addresses map to memory in other nodes.

**[0049]** To see how the arrangement works, consider the example of a virtual server consisting of three nodes: 0, 1, and 2. In a 16-node DSM system, the total addressable memory space for this virtual server would be 1 terabyte (2^40) and each node would be allocated a segment which is 1116 of that space (64 GB or 2^36). From a global view, the first 64 GB segment of the physical address space starting at address **0** would be allocated to node **0** (i.e., the node whose LNID equals 0), the next 64 GB segment to node **1**, and the following segment to node **2**. The remaining 13 segments would be unused since LNIDs 4-15 are not used.

**[0050]** FIG. **11** shows this physical address space from the local view of each of the three nodes in the virtual server. The local view of node **0** would be the same as the global view and is shown in FIG. **11** under the label "Node **0**", with Local Memory (**0**) first, Node **1** Memory second, and Node **2** Memory third. The local view of node **1** would be as shown under the label "Node **1**", with Local Memory (**1**) first, Node **0** Memory second, and Node **2** Memory third. And the local view of node **2** would be as shown under the label "Node **2**", with Local Memory (**2**) first, Node I Memory second, and Node **0** Memory third.

**[0051]** It will be appreciated that in order to accomplish this arrangement, the locations of the local segment and the node **0** segment are swapped in the address map. And since MY_LNID, as defined above, is the LNID assigned to the local node, this is equivalent to swapping MY_LNID with LNID **0** in the address map. However, such a swapping would create confusion in the DSM system if it were applied to memory traffic leaving the node ver the switched fabric. Therefore, the node's CMM reverses the swapping for traffic leaving the node.

**[0052]** FIG. **12** is a diagram showing a flowchart of an example process for altering a physical memory address, by the swapping a described above, prior to transmission over a HyperTransport bus. In the process's first step **1201**, a node's CMM receives a memory operation (e.g., a read, write, or probe) pertaining to a physical memory address from the RDM on the DSM-management chip. In step **1202**, the CMM determines whether the four (or eight) most significant bits in the physical address are equal to: (1) the MY LNID value for the node; or (2) zero. If so, the CMM goes to step **1203**, where: (1) if those bits are equal to the MY_LNID value, the CMM sets the bits to zero (e.g., by changing to zero the four (or eight) most significant bits in the physical memory address) before transmission of the operation over the HyperTransport bus; and (2) if those bits are equal to zero, the CMM sets those bits to MY_LNID (e.g., by changing to MY LNID the four (or eight) most significant bits in the physical memory address) before transmission of the operation over the HyperTransport bus. Otherwise, if those bits are not equal to MY_LNID or zero, the CMM goes to step **1204** and allows the memory operation to proceed without processing relating to LNID swapping.

**[0053]** FIG. **13** is a diagram showing a flowchart of an example process for altering a physical memory address, by reversing the swapping as described above, prior to transmission over a switched fabric. In the process's first step **1301**, a node's CMM receives a memory operation (e.g., a read, write, or probe) pertaining to a physical memory address from one of the node's CPUs over the HyperTransport (e.g., ccHT) bus that connects the node's CPUs to the node's DSM-management chip. In step **1302**, the CMM determines whether the four (or eight) most significant bits in the physical address are equal to: (1) the MY_LNID value for the node; or (2) zero. If so, the CMM goes to step **1303**, where: (1) if those bits are equal to the MY_LNID value, the CMM sets the DstLNID value to zero (e.g., by changing to zero the four (or eight) most significant bits in the physical memory address) before transmission of the operation to the RDM; and (2) if those bits are equal to zero, the CMM sets the DstLNID value to MY_LNID (e.g. by changing to MY_LNID the four (or eight) most significant bits in the physical memory address) before transmission of the operation to the RDM. Otherwise, if those bits are not equal to MY_LNID or zero, the CMM goes to step **1304** and allows the memory operation to proceed without processing relating to LNID swapping, if the physical memory address is not for exported local memory. (If the physical memory address is for exported local memory, a probe operation to another physical memory address might result, feeding back into the process at step **1301**.)

**[0054]** Particular embodiments of the above-described processes might be comprised of instructions that are stored on storage media. The instructions might be retrieved and executed by a processing system. The instructions are operational when executed by the processing system to direct the processing system to operate in accord with the present invention. Some examples of instructions are software, program code, firmware, and microcode. Some examples of storage media are memory devices, tape, disks, integrated circuits, and servers. The term "processing system" refers to a single processing device or a group of inter-operational processing devices. Some examples of processing devices are integrated circuits and logic circuitry. Those skilled in the art are familiar with instructions, storage media, and processing systems.

**[0055]** Those skilled in the art will appreciate variations of the above-described embodiments that fall within the scope of the invention. In this regard, it will be appreciated that there are many other possible orderings of the steps in the processes described above and many other possible modularizations of those orderings. Also, it will be appreciated that the above processes relating to memory-addressing will work with physical memory addresses that exceed 40-bits in width and DSM systems that have more than 256 nodes. Further, it will be appreciated that the DSM system will work with nodes whose CPUs are not Opterons having a ccHT bus. As a result, the invention is not limited to the specific examples and illustrations discussed above, but only by the following claims and their equivalents.

What is claimed is:

**1**. A method, comprising:

receiving, at a distributed shared memory circuit of a first node in a distributed shared memory system, a message from a second node in the distributed shared memory system comprising a plurality of nodes each having a unique logical unit identifier, wherein the message indicates a memory operation related to a local memory of the first node and identifies a memory address;

if a first plurality of contiguous bits of the memory address equal a logical node identifier of the first node, changing the first plurality of contiguous bits to a predetermined value;

if the first plurality of contiguous bits of the memory address equal the predetermined value, changing the first plurality of contiguous bits to the logical node identifier of the first node;

forwarding the message to a processor of the first node for processing.

**2**. The method of claim **1** wherein the predetermined value is zero.

**3**. The method of claim **1** wherein the first set of contiguous bits of the memory address are the most significant bits.

**4**. The method of claim **1** wherein the plurality of nodes internally access their respective local memories having the first plurality of contiguous bits set to the predetermined value.

**5**. The method of claim **1** wherein the plurality of nodes access the local memory of the node having a logical unit identifier equal to the predetermined value using its own respective logical node identifier.

**6**. The method of claim **1** wherein the memory operation is a read command.

**7**. The method of claim **1** wherein the memory operation is a write command.

**8**. The method of claim **1** wherein the memory operation is a probe.

**9**. A method comprising

receiving, at a distributed shared memory circuit of a first node in a distributed shared memory system, a message from a processor of the first node identifying a memory operation related to a local memory of a second node in the distributed shared memory system comprising a plurality of nodes each having a unique logical unit identifier, wherein the message identifies a memory address;

if a first plurality of contiguous bits of the memory address equal a logical node identifier of the first node, changing the first plurality of contiguous bits to a predetermined value;

if the first plurality of contiguous bits of the memory address equal the predetermined value, changing the first plurality of contiguous bits to the logical node identifier of the first node;

forwarding the message to the second node for processing.

**10**. The method of claim **9** wherein the predetermined value is zero.

**11**. The method of claim **9** wherein the first set of contiguous bits of the memory address are the most significant bits.

**12**. The method of claim **9** wherein the plurality of nodes internally access their respective local memories having the first plurality of contiguous bits set to the predetermined value.

**13**. The method of claim **9** wherein the plurality of nodes access the local memory of the node having a logical unit identifier equal to the predetermined value using its own respective logical node identifier.

**14**. The method of claim **9** wherein the memory operation is a read command.

**15**. The method of claim **9** wherein the memory operation is a write command.

**16**. The method of claim **9** wherein the memory operation is a probe.

**17**. A distributed shared memory system, comprising:

a plurality of interconnected nodes, wherein each node has a logical node identifier comprising a plurality of contiguous bits; wherein each of the nodes comprises one or more processors and a local memory; and wherein each of the nodes further comprises a distributed memory logic circuit operative to share the local memory of a respective node in a distributed shared memory system to create a shared memory in connection with other nodes of the plurality of nodes accessible using binary addresses comprising a plurality of bits, wherein a first set of contiguous bits of the binary addresses of the shared memory correspond to a logical node identifier of a node in the plurality of nodes, and

wherein the one or more processors of each of the nodes are operative to access the local memory of its own node having the first set of contiguous bits of the binary addresses set to a uniform predetermined value; and

wherein the distributed memory logic circuit is further operative to map the uniform predetermined value to the logical node identifier of the local node in memory management traffic transmitted between the nodes that include binary addresses of the shared memory.

**18**. The system of claim **17** wherein each of the one or more processors access the local memory of the node having a logical node identifier equal to the predetermined value using the logical node identifier of its own node.

**19**. The method of claim **17** wherein the predetermined value is zero.

**20**. The method of claim **17** wherein the first set of contiguous bits of the memory address are the most significant bits.

**21**. The method of claim **17** wherein the distributed memory logic circuit is operative to

if a first plurality of contiguous bits of the binary address equal a logical node identifier of the node, change the first plurality of contiguous bits to the predetermined value;

if the first plurality of contiguous bits of the memory address equal the predetermined value, change the first plurality of contiguous bits to the logical node identifier of the node.

* * * * *