Title: SYSTEM AND METHOD FOR COMMUNICATING ON A RICHLY-CONNECTED MULTI-PROCESSOR COMPUTER SYSTEM USING A POOL OF BUFFERS FOR DYNAMIC ASSOCIATION WITH A VIRTUAL CHANNEL.

Abstract: Systems and methods for communicating on a richly-connected multiprocessor computer system using a pool of buffers for dynamic association with a virtual channel. Packets are communicated in a multiprocessor computer system having a large plurality of processing nodes interconnected by a defined interconnection topology, in which a communication from a source processing node to a target processing node may pass through one or more intermediate nodes en route to the target processing node. A set of virtual channels is associated for each link in the interconnection topology. A first subset of buffers is dedicated for fixed correspondence to virtual channel identifiers, and a second subset of buffers is dedicated for dynamic allocation and assignment to virtual channels.
System and Method for Communicating on a Richly-Connected Multi-processor Computer System Using a Pool of Buffers for Dynamic Association with a Virtual Channel

Cross-reference to Related Applications

[0001] This application is related to the following U.S. patent applications, the contents of which are incorporated herein in their entirety by reference:

U.S. Pat. Appl. No. 11/335,421, filed January 19, 2006, entitled SYSTEM AND METHOD OF MULTI-CORE CACHE COHERENCY;

U.S. Pat. Appl. No. 11/594,426, filed on November 8, 2006, entitled SYSTEM AND METHOD FOR PREVENTING DEADLOCK IN RICHLY-CONNECTED MULTI-PROCESSOR COMPUTER SYSTEM USING DYNAMIC ASSIGNMENT OF VIRTUAL CHANNELS;

U.S. Pat. Appl. No. 11/594,421, filed on November 8, 2006, entitled LARGE SCALE MULTI-PROCESSOR SYSTEM WITH A LINK-LEVEL INTERCONNECT PROVIDING IN-ORDER PACKET DELIVERY;

U.S. Pat. Appl. No. 11/594,442, filed on November 8, 2006, entitled MESOCHRONOUS CLOCK SYSTEM AND METHOD TO MINIMIZE LATENCY AND BUFFER REQUIREMENTS FOR DATA TRANSFER IN A LARGE MULTI-PROCESSOR COMPUTING SYSTEM;

U.S. Pat. Appl. No. 11/594,427, filed on November 8, 2006, entitled REMOTEDMA SYSTEMS AND METHODS FOR SUPPORTING SYNCHRONIZATION OF DISTRIBUTED PROCESSES IN A MULTIPROCESSOR SYSTEM USING COLLECTIVE OPERATIONS;

U.S. Pat. Appl. No. 11/594,420, filed on November 8, 2006, entitled SYSTEM AND METHOD FOR ARBITRATION FOR VIRTUAL CHANNELS TO PREVENT LIVELOCK IN A RICHLY-CONNECTED MULTI-PROCESSOR COMPUTER SYSTEM;

U.S. Pat. Appl. No. 11/594,441, filed on November 8, 2006, entitled LARGE SCALE COMPUTING SYSTEM WITH MULTI-LANE MESOCHRONOUS DATA TRANSFERS AMONG COMPUTER NODES;

U.S. Pat. Appl. No. 11/594,443, filed on November 8, 2006, entitled RDMA SYSTEMS AND METHODS FOR SENDING COMMANDS FROM A


SOURCE NODE TO A TARGET NODE FOR LOCAL EXECUTION OF COMMANDS AT THE TARGET NODE;

U.S. Pat. Appl. No. 11/594,447, filed on November 8, 2006, entitled SYSTEMS AND METHODS FOR REMOTE DIRECT MEMORY ACCESS TO PROCESSOR CACHES FOR RDMA READS AND WRITES; and

U.S. Pat. Appl. No. 11/594,446, filed on November 8, 2006, entitled SYSTEM AND METHOD FOR REMOTE DIRECT MEMORY ACCESS WITHOUT PAGE LOCKING BY THE OPERATING SYSTEM.

U.S. Pat. Appl. No. 11/594,423, filed on November 8, 2006, entitled COMPUTER SYSTEM AND METHOD USING A KAUTZ-LIKE DIGRAPH TO INTERCONNECT COMPUTER NODES AND HAVING CONTROL BACK CHANNEL BETWEEN NODES;

U.S. Pat. Appl. No. 11/594,416, filed on November 8, 2006, entitled COMPUTER SYSTEM AND METHOD USING EFFICIENT MODULE AND BACKPLANE TILING TO INTERCONNECT COMPUTER NODES VIA A KAUTZ-LIKE DIGRAPH

Background

Field of the Invention

[0002] This invention relates to deadlock prevention in richly-connected multiprocessor computer systems and more specifically to virtual channel techniques to prevent deadlock in large multi-node computing systems interconnected by complicated topologies, including but not limited to Kautz and de Bruijn topologies.
Description of the Related Art

[0003] Massively parallel computing systems have been proposed for scientific computing and other compute-intensive applications. The computing system typically includes many nodes, and each node may contain several processors. Various forms of interconnect topologies have been proposed to connect the nodes, including Hypercube topologies, butterfly and omega networks, tori of various dimensions, fat trees, and random networks.

[0004] One of the problems encountered in building computer systems with complex, richly-connected communication networks is deadlock. Deadlock is the condition or situation in which actions are mutually blocked from progress because they are waiting for some form of resource. That is, some form of cycle of resource dependency exists that cannot be satisfied.

[0005] Figure 1 depicts a simple communications system to illustrate the deadlock problem. In Figure 1, the system has three nodes, 102, 104, and 106, each with a communication buffer, 108, 110, and 112 respectively. The buffers are resources the nodes need to send, or receive, data to, or from, a connected node (as suggested with links 114, 116, and 118). For node 102 to send its data forward from buffer 108 to buffer 110 in node 104, buffer 110 must be empty, free, or available. In this example, however, there is a cycle of resource dependency. That is, each node needs the adjacent node's buffer to be empty so that it can transmit data to an empty buffer (and not overwrite data in the buffer prematurely). Since a cycle exists from node 102, 104, and 106 and back to 102, the potential exists that no node will be able to send traffic forward, and each node will be waiting for the adjacent node's buffer to empty. Figure 1 thus shows a system susceptible to deadlock.

[0006] Many communication architectures and protocols, including TCP/IP and the global telephone network, address the deadlock issue by discarding traffic whenever a potential deadlock situation arises. This approach, however, imposes a substantial cost in any higher-level protocol which aims to be reliable, because it must recognize and recover from the loss of message traffic. It is therefore very desirable to provide guarantees against deadlock, as opposed to detection of such and then discarding of traffic.

traffic on each virtual channel, and structures the flow of data through those virtual channels such that there is no cycle of dependency among the channels. By preventing cycles, this strategy ensures that all traffic is able to leave the network, and therefore that deadlock does not occur. Virtual channel assignment is constant. A communication travels on its route using the same virtual channel assignment throughout. Typically buffer resources are mapped and fixed to virtual channel assignments.

[0008] Another potential problem in richly-connected networks is the issue of livelock or starvation. In this situation, an action is starved (as opposed to blocked) from getting access to necessary resources. Typically, timers are used to age the action. If the action gets old enough, then its priority may be elevated to increase the probability that it will win arbitration when contending for a required resource (such as access to a buffer or output link).

Summary

[0009] The invention provides systems and methods for communicating on a richly-connected multiprocessor computer system using a pool of buffers for dynamic association with a virtual channel.

[0010] Under one aspect of the invention, packets are communicated in a multiprocessor computer system having a large plurality of processing nodes interconnected by a defined interconnection topology, in which a communication from a source processing node to a target processing node may pass through one or more intermediate nodes en route to the target processing node. A set of virtual channels is associated for each link in the interconnection topology. A virtual channel is assigned to convey each communication between a source processing node and a target processing node. A first subset of buffers is dedicated for fixed correspondence to virtual channel identifiers, and a second subset of buffers is dedicated for dynamic allocation and assignment to virtual channels. In response to a communication request with an associated virtual channel, one of the fixed buffers or dynamically allocated buffers is provided to store packet data for the communication.

[0011] Under another aspect of the invention, if the processing node dynamically allocates buffer from the second subset, the processing node records the order of delivery for the communication utilizing the buffer from the second subset and records the virtual channel for the communication.
Under another aspect of the invention, the number of buffers allocated to a virtual channel is dynamically allocated and varies depending on traffic load.

Brief Description Of The Figures

In the Drawing.

Figure 1 is a diagram depicting a three-node communications system to illustrate the deadlock problem;
Figures 2A-2B are exemplary, simple Kautz topologies;
Figure 3 is a block diagram of node switching logic according to certain embodiments of the invention;
Figure 4 is a flow chart depicting dynamic virtual channel assignment according to certain embodiments of the invention;
Figure 5 depicts the arbitration scheme for an egress link according to certain embodiments of the invention; and
Figure 6 is an architectural diagram depicting arbitration logic to avoid livelock of packet transmission according to certain embodiments of the invention.

Detailed Description

Preferred embodiments of the invention use a virtual channel technique to prevent deadlock in richly-connected multiprocessor computer systems. By using virtual channels, there is no need to detect deadlock and discard traffic. Moreover, the inventive technique uses virtual channel assignments more efficiently than conventional virtual channel approaches. In addition, arbitration logic is provided which avoids livelock of packet transmission on virtual channels and does not require timer circuitry. Moreover, a pool of buffers is provided and buffers are dynamically allocated to virtual channels; thus a virtual channel may utilize more than one buffer at a node to improve data latency.

To discuss preferred embodiments of the invention, first an exemplary richly-connected computer topology is described. Then the virtual channel technique is discussed in the context of the exemplary topology. It will be appreciated that the virtual channel technique may be employed in topologies other than the exemplary one.

Exemplary embodiment of a Richly-connected computer topology
Certain embodiments of the invention are utilized on a large scale multiprocessor computer system in which computer processing nodes are interconnected in a Kautz interconnection topology. Kautz interconnection topologies are unidirectional, directed graphs (digraphs). Kautz digraphs are characterized by a degree $k$ and a diameter $n$. The degree of the digraph is the maximum number of arcs (or links or edges) input to or output from any node. The diameter is the maximum number of arcs that must be traversed from any node to any other node in the topology.

The order $O$ of a graph is the number of nodes it contains. The order of a Kautz digraph is $(k + 1)F^{-1}$. The diameter of a Kautz digraph increases logarithmically with the order of the graph.

Figure 2A depicts a very simple Kautz topology for descriptive convenience. The system is order 12 and diameter 2. By inspection, one can verify that any node can communicate with any other node in a maximum of 2 hops. Figure 2B shows a system that is degree three, diameter three, order 36. One quickly sees that the complexity of the system grows quickly. It would be counter-productive to depict and describe preferred systems such as those having hundreds of nodes or more.

The table below shows how the order $O$ of a system changes as the diameter $n$ grows for a system of fixed degree $k$. 

- 6 -
If the nodes are numbered from zero to 0-1, the digraph can be constructed by running a link from any node x to any other node y that satisfies the following equation:

\[ y = (-x^k \mod 2) \mod 0, \text{ where } 1 \leq j \leq k \]  

(1)

Thus, any (x,y) pair satisfying (1) specifies a direct egress link from node x. For example, with reference to figure 2B, node 1 has egress links to the set of nodes 30, 31 and 32.

Iterating through this procedure for all nodes in the system will yield the interconnections, links, arcs or edges needed to satisfy the Kautz topology. (As stated above, communication between two arbitrarily selected nodes may require multiple hops through the topology but the number of hops is bounded by the diameter of the topology.)

Each node on the system may communicate with any other node on the system by appropriately routing messages onto the communication fabric via an egress link. Under certain embodiments, any data message on the fabric includes routing information in the header of the message (among other information). The routing information specifies the entire route of the message. In certain degree three embodiments, the routing information is a bit string of 2-bit routing codes, each routing code specifying whether a message should be received locally (i.e., this is the target node of the message) or identifying one of three egress links. Naturally other topologies may be implemented with different routing codes and with different structures and methods under the principles of the invention.

Under certain embodiments, each node has tables programmed with the routing information. For a given node x to communicate with another node z, node x accesses the table and receives a bit string for the routing information. As will be explained below, this
bit string is used to control various switches along the message's route to node z, in effect specifying which link to utilize at each node during the route. Another node j may have a different bit string when it needs to communicate with node z, because it will employ a different route to node z and the message may utilize different links at the various nodes in its route to node z. Thus, under certain embodiments, the routing information is not literally an "address" (i.e., it doesn't uniquely identify node z) but instead is a set of codes to control switches for the message's route.

[0023] Under certain embodiments, the routes are determined apriori based on the interconnectivity of the Kautz topology as expressed in equation 1. That is, the Kautz topology is defined, and the various egress links for each node are assigned a code (i.e., each link being one of three egress links). Thus, the exact routes for a message from node x to node y are known in advance, and the egress link selections may be determined in advance as well. These link selections are programmed as the routing information. This routing is described in more detail in the related and incorporated patent applications.

[0024] In certain preferred embodiments, embodiments of the invention may be practiced in a degree three, diameter six Kautz topology of order 972. The 972 nodes are tiled on modules each having 27 nodes. Each node may be a multiprocessor node, e.g., six computer processors per node. Moreover, node to node transfers may be multi-lane mesochronous data transfers using 8B/10B codes.

Exemplary Node Switching Logic

[0025] Figure 3 is a block diagram of the switching logic 300 used in processor nodes of certain embodiments of the invention that use a degree three topology. The switching logic receives data from (and provides some control information to) parent nodes via links 302. The logic provides data to (and receives some control information from) children nodes via links 304.

[0026] Item 302 identifies a set of lanes to receive data from parent nodes in the interconnection topology. They connect the input data to corresponding appropriate input blocks IB0-IB2. Item 304 identifies a set of lanes 304 to provide data to children nodes in the topology. They connect to the appropriate corresponding output blocks OB0-OB2. The links include data lanes or circuits (shown in heavy line) and a reverse control channel (shown in lighter line shading).

[0027] The switching logic includes various crosspoint buffers, e.g., XB 00, for each crosspoint connection. Thus, if data from input link 0 (from the set 302) is addressed to
egress on a link 2 (from the set 304), then the data will be stored in cross point buffer XB 02, if the data needs to be stored in this node. (In some situations the data can cut through and avoid local buffering.)

[0028] The input blocks IBx distribute incoming data from the input links 302 to one of four cross point buffers XBxx based on the routing field in the packet's header. It also performs various other functions such as checking CRC on the packet, determining if a packet has been poisoned (to support cut-through functionality), supporting flow control with packet sequence numbers, and the like. It is also responsible for passing control packets to parent nodes via the control circuits of input links 302. In certain embodiments, the IB also modifies the routing information in the packet header by shifting the contents two bits and shifting in the two-bit code corresponding to the link on which the packet was received. (In this fashion, the LSBs - or MSBs depending on the embodiment - may be used to identify the egress link a packet should use as it is received at the next node.)

[0029] The output blocks OBxs , among other things, are responsible for performing arbitration among the four cross point buffers XBxx attached to a given OB. They also maintain a replay buffer used in flow control and compute CRC on data outbound on egress links 304.

[0030] The DMA blocks are responsible for performing remote DMA actions to the node DMAIx or output from the node DMAOx.

**Virtual Channel Assignment**

[0031] Under certain embodiments, as a packet is created it is also assigned a virtual channel number, and this number is encoded in the packet header along with other information such as the routing information discussed above. The initial value for a virtual channel number is determined by software at the time the routing tables are constructed and the virtual channel is encoded in the routing tables.

[0032] Unlike conventional approaches, under preferred embodiments the virtual channel number may be changed dynamically as the message moves along its route. It can be shown that this dynamic assignment of virtual channels still prevents cycles of resource dependency, and it can also be shown that this technique requires fewer buffer resources than the conventional static and constant assignment of virtual channel numbering.

[0033] For example, under certain embodiments, when a message arrives at a node, the virtual channel number is decremented if the current node number is greater than both the node number of the parent node (i.e., node from which the message came on immediately
prior hop) and the node number of the child node (i.e., node to which the message is going to on the next hop). The basic requirement in assigning the initial virtual channel is simply that its assignment value must be large enough so that it does not become negative as the packet makes its way to the final destination. That could be accomplished by starting every packet with a virtual channel equal to the maximum number of decrements it might have to encounter, but there are other ways to program as well, e.g., starting with the highest virtual channel. Alternatively, rules could be changed and still keep the principle of dynamic assignment, e.g., the channel number can be incremented, or the comparison can be "less than" rather than "greater than".

Under certain embodiments, the modification of virtual channel numbers while the message is en route may be implemented as follows. The nodes in the topology are each assigned a corresponding node number identification. For example, in a Kautz topology of order 972, the nodes may be assigned numbers from 0 to 971. The interconnectivity is known and satisfies equation (1) above. The nodes' connections are assigned to the various egress and ingress links of each node. Thus the node to link mapping may be determined \textit{a priori}. From this, it may be determined for each node whether a given input link/output link pair satisfies the rule for changing the virtual channel number.

Each input block IB, in these embodiments, is constructed to have a programmable register to indicate whether the input link/output link pairing should cause a change in virtual channel number. For example, the register may have a bit for each egress link (and by design the register is associated with a particular input link to which the IB is connected). Each bit may then indicate whether the IB should change the virtual channel number for packets destined for the corresponding egress link. Any packet received on that input link has its routing information in its header analyzed to see which egress link should be used. The bit register is then considered, and depending on the bit setting, the virtual channel number may be changed, e.g., decremented. The packet is then stored in the relevant corresponding cross point buffer XBxx, or it may immediately forwarded on an egress link if the egress link is available and arbitration rules allow such immediate bypass. Thus, if a packet arrives on a link and is assigned virtual channel 3, it will be stored in a local buffer (in the corresponding XB for the input link/output link pair) for virtual channel 3. However, depending on the node numbers for the current, parent and child nodes, the virtual channel may be changed to 2, e.g., the packet header may be modified to indicate virtual channel 2. That packet (though stored in a buffer slot for virtual channel 3) will arbitrate for downstream buffer resources corresponding to the new, dynamically-assigned virtual channel 2.
Under certain embodiments, each cross point buffer XBxx has 16 buffers. Some portion of these are dedicated for specific virtual channel assignment (e.g., as defined in a programmable register). The remainder may then be used in a common pool of buffers that can be dynamically allocated for any virtual channel. For a diameter six Kautz topology, only three virtual channels are needed. Thus three of the buffers are dedicated to the three virtual channels, and the remaining buffers are assigned to a pool of buffers that may be dynamically allocated to any virtual channel. Thus, in embodiments using 16 buffers per cross point XBxx, 13 buffers can be used for the pool. That is, a virtual channel need not be fixed to one and only one buffer and instead can be associated with a set of buffers, the actual size of which set will be dynamic, based on traffic flow. For example, if one virtual channel is particularly busy, it may use several buffers at a given cross point. The XBxx includes logic to associate a virtual channel number with each buffer (including those in the pool) and to record the order of arrival for data in the buffers. In this way, the data can be forwarded or consumed in the same order as it arrived (on a per channel basis). By efficiently assigning virtual channels (i.e., using less channels to avoid deadlock), more buffers may be associated with the pool, which in turn should improve data flow and reduce latency.

In certain embodiments, the amount of buffers dedicated to virtual channels (i.e., the fixed assignments) is programmable. Among other things, this facilitates the use of the module design in various system designs. For example, by reprogramming the amount of buffers to directly map to virtual channels, the module design may be used in systems of different diameter (larger diameter systems need more virtual channel assignments to avoid deadlock as discussed above).

Figure 4 is a flow diagram depicting steps for virtual channel assignment according to certain embodiments of the invention.

The logic starts in step 402 and progress to step 404 in which node numbers are assigned. As described above, in some embodiments, this is as simple as assigning nodes unique integers to identify the nodes, e.g. 0 to 971. However, the unique identifier can be a number (not necessarily decimal), a string of characters, or other code. Any type of unique identifier can be used as long as they are ordered by some known set of ordering rules.

Ordered means that they can be compared to one another, and it is possible to determine if any unique identifier is less than, greater than, or equal to another unique identifier.

In step 406 a particular node creates a packet and populates the packet header with appropriate routing information, including for example an initial assignment of virtual
channel. This initial assignment may be pulled from a table look-up in conjunction with retrieving other routing information.

[0041] The packet is eventually sent and eventually received by a child node in step 408. 

[0042] The current node compares its node number to the node number of the parent node (the node which sent the packet) and the next node, i.e., child node (assuming this is not the final node in the route). As explained above, in certain embodiments this comparison is done by use of programmable registers that have pre-recorded the comparison results based on input link/output link mappings. In other embodiments, the node numbers may be directly available and in such case the node numbers may be consulted and compared directly. 

[0043] In step 412, the results of the comparison are considered in conjunction with the dynamic assignment rules. For example in some embodiments, the current node is compared to the parent and child nodes of the packet route to see if the current node number is greater than both the parent and the child. Other combinations of rules and comparisons exist, which can be shown to avoid cycles of resource dependency. 

[0044] If the comparison indicates that the virtual channel identifiers should be changed while the packet is en route, the flow proceeds to step 414. In step 414, the virtual channel number is changed. For example, in the above embodiments, the virtual channel number is decremented, if the current node is greater than both the parent and child node numbers. Generally speaking, however, the virtual channels may be considered as ordered identifiers in a set of identifiers (e.g., integers). Each virtual channel has an adjacent channel identifier in the ordered set of channel identifiers, and step 414 changes the virtual channel identifier accordingly. So if the identifiers were simply integers, next would mean the next highest integer. 

[0045] In step 416, the packet is either stored or forwarded accordingly using the virtual channel numbers assigned to the packet (and possibly changed).

**Arbitration**

[0046] Multiple buffers may contend for an egress link. In the embodiment of figure 3, there are three cross point buffers associated with each egress link. For example, XB00, XB10, and XB20 are all associated with output block OBO and output link 0. (Another cross point buffer XB30 is depicted but this is associated with a DMA engine not an input link from another node.) Each XBxy may have data that is in contention for use of egress link y. In addition, within each given XB, there are multiple buffers (e.g., 16), and thus there may be
more than one buffer within an XB that may wish to contend for use of an egress link
(perhaps on the same virtual channel or perhaps on a different virtual channel).

[0047] In certain embodiments, arbitration is performed within an XBxy to select a buffer
from within that XB to bid for contention to egress link y. Arbitration is also performed at
the OBy to select from among the various XBxys having data bidding for contention on link
y.

[0048] Under certain embodiments, the cross point buffer XB does not identify a buffer
entry as available for transmission on the egress link (i.e., bid for contention), until the XB
can determine that there is sufficient space for the packet in the downstream node. In short, a
debit/credit mechanism is arranged in which the receiving node (i.e., the IB of the
downstream node) tells the sending node (i.e., the OB of the sending node) how much space
is available in the receiver's buffers for each virtual channel and port. (This buffer status in
certain embodiments is conveyed from the downstream node to the upstream node using the
control channel, e.g., one 16 bit buffer mask per XB connected to the relevant internode link.)
For example, if a packet route requires that the packet arrive at an input link 0 on a node K
and leave on egress link 2, and the route also requires that the packet arrive at downstream
node D and leave node D on egress link 1, the packet will be stored in XB02 of node K.
XB02 will keep that data in a buffer, until XB02 can determine that there is appropriate
buffer space available in downstream node D's cross point buffer XB? 1. (The '?' is used to
signify that the topology does not require downstream node D to connect egress link 2 of the
upstream node as an input link 2. So if the downstream node D connected its input link 1 to
the upstream node's egress link 2, then the upstream node would determine whether
downstream node D's XB1 had buffer space.)

[0049] To determine whether a buffer entry should bid for contention for an egress link,
the XBxxs query the buffer busy masks to determine if space is available in downstream
nodes. If space is available, the XB identifies the relevant packet as one that can bid for
contention for use of the egress link. For example, if the packet in the above example is
traveling on virtual channel 3, XB02 will query a relevant buffer busy mask to determine if
the corresponding slot for virtual channel 3 is empty on the downstream node D, or if there
are any free buffers in the pool. If the corresponding slot for virtual channel 3 is empty (or
there is space in the pool), the packet will be identified as available to contend for the egress
link. (Moreover, in certain embodiments, the packet may be modified to specify the
particular slot it should go into, when it is received in node D; i.e., buffer entry slot is
recorded in the packet header separately from the virtual channel identifier in the packet.
As will be explained in more detail below, the XB has a conservative (worst case) estimate of whether the relevant buffers are busy.

Each XB (by definition associated with an input link/output link pair) keeps track of the order in which a packet arrived in the XB, and the virtual channel to which that packet is assigned. If multiple buffers within an XBxx may contend for use of the egress link (e.g., because relevant downstream buffers are free) the XB selects the oldest such packet within the XB to contend for use of the link.

In certain embodiments, the output block OB tracks buffer busy status. It does this both for the local buffers and for the downstream buffers for the node to which it connects via its associated egress link. When a packet is sent, the OB marks the corresponding buffers downstream as being busy. Buffers are marked as free when a control packet from a downstream node indicates that the buffer has been forwarded or consumed (this control packet need not be a packet-by-packet acknowledgement). The control packet also contains "busy masks" for each of the downstream XBs. Thus, for example, a downstream IB may send four 16 bit masks to indicate whether buffers are free or busy in each XB the IB is connected to, including the XB for DMA processing, and the OB of the upstream node will maintain such status information so the upstream node's XBxxs know when they should bid for contention. Thus the OB maintains busy masks for the local XBss to which it is connected, and also busy masks for the downstream XBss connected to the egress link. Performing a logical OR on the busy masks provides a conservative (worst case) picture of whether buffers are busy.

Figure 5 is a conceptual diagram illustrating how an OB may select which buffer should gain access to an egress port, and figure 6 is a block diagram of exemplary logic for such. Both should be referred to conjointly in connection with the following description. In short, the logic seeks the XBxy that has been waiting the longest for an egress link.

However, before granting access to that XBxy, it also seeks to see if any XBxy has a buffer bidding for contention for that link and for the virtual channel that has been waiting longer. In this fashion, livelock of virtual channel traffic is avoided (i.e., absent such, one might be fair to XBxys but be unfair - and even starve - traffic for particular virtual channels on a link).

The output block may have as many as four XBxxs bid for contention of the output link, in the above-described embodiments. In figure 6, the XBxxs are depicted as ports 612-618, and packets 0-15 in each port refer to the buffers in an XB. In certain embodiments, the output block considers the XBxxs that are contending for use of the egress
link 620 and selects 622 the least recently chosen XB, which is then requesting use of the egress link. Aging vectors and other techniques may be utilized for such. This is shown at 502 and 504 in figure 5.

The downstream egress link (i.e., next port or "NP" in figure 5) and virtual channel number (i.e., VC) are then identified; see 506 and 624. Then all XBxxs that have buffers that were bidding for contention (i.e., the oldest for that XBxx that were ready to be transmitted to an available downstream buffer) that are also addressing the same downstream egress link (or next port) and the same virtual channel are identified. This is shown in 508 and 626-632.

The virtual channel and NP are used to consult (e.g., index) into a history structure 634, which records the last XBxx to have won use of the egress link for that particular downstream egress link (NP) and virtual channel (VC) pair. This is shown in 510.

The OB logic then starts with the next XBxx identification (i.e., from the one identified in the history structure) and cycles through in round robin fashion 636 to see if an XB is bidding for use of the egress link with the same NP and VC. If so, that XB will be granted access. This will continue in round robin fashion, to identify XBxs with the same NP and VC. This is shown in 512. Thus in the example of figure 5, even though XB12 was selected because it was the XB that had least recently used the OB, it is not the first to use the egress link in this arbitration event. Instead, XB32 precedes XB12 in this arbitration event. This was done to combat livelock of the link/VC pair. XB32 may have used the OB more recently than XB12 but it was not for the particular link (NP) and VC of this event. Indeed the history structure revealed in this instance that XB12 in fact was the last XB to send data to this NP/VC pair and the arbitration result in effect avoided XB12 from starving out XB32 for this NP/VC pair. Moreover, timers were avoided. As a result, traffic on all virtual channels is treated fairly, a result that is difficult to achieve even with complex timers.

Many of the teachings here may be extended to other topologies including de Bruijn topologies. Likewise, though the description was in relation to large-scale computing system, the principles may apply to communication fabrics.

While the invention has been described in connection with certain preferred embodiments, it will be understood that it is not intended to limit the invention to those particular embodiments. On the contrary, it is intended to cover all alternatives, modifications and equivalents as may be included in the appended claims. Some specific
figures and source code languages are mentioned, but it is to be understood that such figures and languages are, however, given as examples only and are not intended to limit the scope of this invention in any manner.

[0059] What is claimed is:
1. A method of communicating packets in a multiprocessor computer system having a large plurality of processing nodes interconnected by a defined interconnection topology, in which a communication from a source processing node to a target processing node may pass through one or more intermediate nodes en route to the target processing node, the method comprising:

   for each link in the interconnection topology associating a set of virtual channels, each virtual channel having an associated virtual channel identifier;
   for each communication between a source processing node and a target processing node, assigning an virtual channel to convey the communication;
   at a processing node providing a set of buffers to associate with virtual channels, a first subset of buffers being dedicated for fixed correspondence to virtual channel identifiers, and a second subset of buffers being dedicated for dynamic allocation and assignment to virtual channels;
   in response to a communication request with an associated virtual channel, providing one of the fixed buffers or dynamically allocated buffers to store packet data for the communication.

2. The method of claim 1 wherein the processing node records the order of delivery for the communication utilizing the buffer from the second subset and records the virtual channel for the communication.

3. The method of claim 1 wherein the size of the first subset and the size of the second subset are dynamically configurable.

4. The method of claim 1 wherein the number of buffers allocated to a virtual channel is dynamically allocated and varies depending on traffic load.
5. A large scale multiprocessor computer system, comprising:
   a large plurality of processing nodes;
   a defined interconnection topology having links interconnecting the processing nodes;
wherein each processing node includes
   link logic to transmit data on, and receive data from, links connected to the
   processing node,
   buffers to store communications, including buffers dedicated for fixed
   correspondence to virtual channel identifiers, and buffers for dynamic
   allocation and assignment to virtual channels; and
wherein, in response to a communication request with an associated virtual
channel, one of the fixed buffers or dynamically allocatable buffers
stores communication data.

6. The system of claim 5 wherein the processing node includes logic to record the order
   of receipt for all communications in the fixed or allocatable buffers associated with a
   particular virtual channel.

7. The system of claim 5 wherein the size of the first subset and the size of the second
   subset are dynamically configurable.
Stage 1 Contenders
Three out of four XBs request. The requests are shown, along with their VC and NextPort.

<table>
<thead>
<tr>
<th>XB02, VC=4, NP=7</th>
</tr>
</thead>
<tbody>
<tr>
<td>XB12, VC=0, NP=3</td>
</tr>
<tr>
<td>XB32 no request</td>
</tr>
<tr>
<td>XB32, VC=0, NP=3</td>
</tr>
</tbody>
</table>

Stage 1 Arbitration
Find Least Recently Chosen XB that is requesting. Use a 4x4 Age Vector Matrix.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XB02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XB12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>XB22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>XB32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This table shows the XB12 won least recently, followed by XB22, then XB02. XB32 won most recently. Eliminate the rows and columns for the one that is not requesting, and look for a row full of zeroes. XB12 is the winner.

Stage 1 Result
The stage 1 winner is:

XB12, VC=0, NP=3

Any requests with the same VC and NextPort will proceed to stage 2.

Stage 2 Contenders
There are two contenders which had the same VC and NextPort as the stage 1 winner.

<table>
<thead>
<tr>
<th>XB12, VC=0, NP=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>XB32, VC=0, NP=3</td>
</tr>
</tbody>
</table>

Stage 2 Arbitration
Do Round Robin by NextPort+VC using a 64-entry Table of which XB won last.

<table>
<thead>
<tr>
<th>VC 0, NextPort 0</th>
<th>XB02</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC 0, NextPort 1</td>
<td>XB32</td>
</tr>
<tr>
<td>VC 0, NextPort 2</td>
<td>XB02</td>
</tr>
<tr>
<td>VC 0, NextPort 3</td>
<td>XB12</td>
</tr>
<tr>
<td>VC 1, NextPort 0</td>
<td>XB12</td>
</tr>
<tr>
<td>VC 1, NextPort 1</td>
<td>XB12</td>
</tr>
<tr>
<td>VC 1, NextPort 2</td>
<td>XB02</td>
</tr>
<tr>
<td>VC 1, NextPort 3</td>
<td>XB32</td>
</tr>
<tr>
<td>VC 14, NextPort 3</td>
<td>XB22</td>
</tr>
<tr>
<td>VC 15, NextPort 0</td>
<td>XB22</td>
</tr>
<tr>
<td>VC 15, NextPort 1</td>
<td>XB12</td>
</tr>
<tr>
<td>VC 15, NextPort 2</td>
<td>XB02</td>
</tr>
<tr>
<td>VC 15, NextPort 3</td>
<td>XB32</td>
</tr>
</tbody>
</table>

Stage 2 Result
Last time there was a stage 2 arb for VC=0, NP=3, the table says that XB12 won. Give it lowest priority this time. XB32 is the winner.

XB32, VC=0, NP=3

Last step: Update the stage 1 age vector and stage 2 history table.

Figure 5
Each packet buffer requests service when it is occupied and an appropriate downstream buffer is available.

First level arbitration (for bandwidth equalization and fairness among ports):
Each port selects the oldest packet for which there is a buffer available downstream.

Port-W buffer req
Port-X buffer req
Port-Y buffer req
Port-Z buffer req

Second level arbitration:
Round robin among ports to select the downstream buffer/VC to be filled.

Target virtual channel

Identify the most recent port to use the selected virtual channel.

Third level arbitration:
Round robin among ports contending for selected virtual channel.