Performance Evaluation of Wormhole Routed Network Processor-Memory Interconnects

Taskin Kocak and Jacob Engel
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816
{tkocak, jengel}@cs.ucf.edu

Abstract

Network line cards are experiencing ever increasing line rates, random data bursts, and limited space. Hence, they are more vulnerable than other processor-memory environments, to create data transfer bottlenecks and hot-spots. Solutions to the memory bandwidth bottleneck are limited by the area available on the line card and network processor I/O pins. As a result, we propose to explore more suitable off-chip interconnect and communication mechanisms that will replace the existing systems and that will provide extraordinary high throughput. We utilize our custom-designed, event-driven, interconnect simulator to evaluate the performance of wormhole routed packet-based off-chip k-ary n-cube interconnect architectures for line cards. Our performance results show that wormhole routed k-ary n-cube based interconnect topologies significantly outperform the existing line card interconnects and they are able to sustain higher traffic loads.

1 Introduction

Routers and switches are the fundamental equipments of most network infrastructures. These devices provide the functionality to receive, decode, repack, and switch packets of data within the network. The basic hardware building block for a mid-end to high-end router or switch is the line card. Currently, two major trends are impacting the architecture and design of line cards. First, line cards are required to handle more functions to support new services such as quality of service (QoS) and policy management, which increases the traffic overhead to incoming line rates by 40%-100%. This, in turn, raises memory, chip interconnect and back plane bandwidth requirements. Second, ever increasing memory capacity requirements which are due to higher link rates and unstoppable expansion in lookup tables. The heavily stressed memory is used by network processors for two main tasks: packet storage (buffering) and lookup table searches. Packet buffer memory is accessed at least four times per packet. Therefore, in order to sustain wire-speed performance the buffer memory should be able to provide at least four times the bandwidth of the network link.

In this paper, our primary goal is to explore different types of interconnect architectures to increase the off-chip memory bandwidth on line cards. There are many candidates in the area of interconnects that can be used to provide a communication link between processors and memories. Networks such as k-ary n-cubes, which include hypercubes, mesh and torus networks. But the uniqueness of the interconnect architecture we seek is contained by the physical constraints characterizing the line card board. Area and I/O pins are limited on the line card. Hence, the number of optional designs that can physically and functionally fit, given those constraints, is limited. Each embedded chip has fixed and limited number of I/O pins. Therefore, a low-dimension, packet-switched network may be a good solution.

Shared-bus is commonly used as a communication link between network processors and multiple memories. A shared-bus cannot scale well as the number of modules (processing elements or memories) connected to it increases. In addition, it requires an arbitration mechanism that becomes distributed (rather than centralized) as the number of modules connected to the bus grows. Another commonly used communication alternative is the crossbar. A crossbar can support multiple simultaneous connections as long as no

1-4244-0054-6/06/$20.00

contentions occur. Once contention occurs, its performance degrades. Other disadvantages include higher cost (function of its switching and wiring complexity) and scalability (cannot scale once embedded into the line card).

There is a plethora of works in k-ary n-cube networks [1, 2, 4, 5, 6, 7]. In this paper, we are not proposing yet another one. We propose to explore them in the context of line card designs. To the best of our knowledge, this will be the first time, the k-ary n-cube networks will be used on a board.

There are two well-known architectures that conceptually and structurally resemble somewhat the work we propose here: Cosmic Cube [9] and Cray 3TE [8]. However, there are a couple of major differences. First, the scope of the interconnect architectures is different. Nodes in our case are at the chip-level (i.e., processors and memory devices), not boards or cases (i.e., computers). The physical dimensions of our interconnect must be smaller since area on the line card is limited. Second, the application workload shows very different behaviour. Unlike data traffic generated in supercomputers, the network traffic is proven to be self-similar and exhibits burstiness. Third, the cost of implementing the interconnect architecture is lower, since its performance superiority and reliability are primarily depend on its routing algorithm and message flow-control not its hardware. This allows us to use simple logic and bus lines between chips making it an inexpensive interconnect to fabricate.

2 Processor-Memory Interconnect Architectures

The interconnect resides in the middle of the line card and provides a multipath off-chip linkage between processing elements (PEs) and memory modules. The interconnect allows packets to be shared and transferred by different processor and memory modules on the network line card simultaneously. Fig. 1 portrays a generic line card architecture in which processing, communication and memory components can communicate via the interconnect. Besides the NPUs several processing components are shown in the figure, such as co-processor (Co-P), traffic manager (TM), hardware accelerator (HWA), and quality of service (QoS) co-processor. The memory banks, which differ in size, are distributed over the interconnect structure to allow data sharing among modules and direct processor memory storage. The fabric and physical interfaces handle incoming/outgoing traffic.

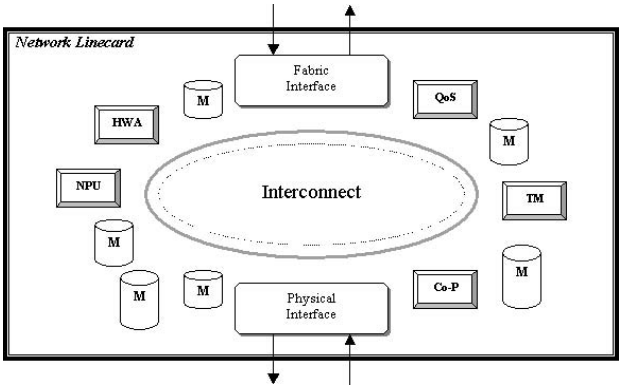

Figure 1. Generic network line card architecture

Each component, which requires memory access, sends its data encapsulated in packets in the shortest path possible. If there is a congested area (hot spot), packets in transit will take a different route, using predetermined and prioritized set of directions using the traffic controller (TC) located in each node. A node connects multiple channels together and it includes a TC. Messages propagate from one node to another adjacent node, where in each node routing decisions are made, until they reach destination. A faulty link will not discontinue the transmission of a message to its destination since packets will be rerouted through alternative paths using other available nodes. Routing ensures that a faulty link will be limited only locally and other links from the intermediate nodes should ensure that connectivity continues.

There are two additional features required to enhance message flow and avoid deadlocks: virtual channels and sub-channelling. Each node will contain two bidirectional virtual channels (VC). VCs are used to avoid transmission failure. Transmission failure can occur when a packet (fraction of message) cannot take any of the output ports. A propagating message occupies two ports (out of four ports available in one node) simultaneously, when moving from input to output port. Only two VCs are required per node, since only two messages can arrive on the other two ports which compete over the same output port. Channel width partitioning allows a unidirectional channel to become bidirectional so that more than one message can share each channel. There can be multiple ways to partition the channel into smaller bidirectional subchannels. If three sub-channelling configurations are set, then, the first configuration will support unidi-

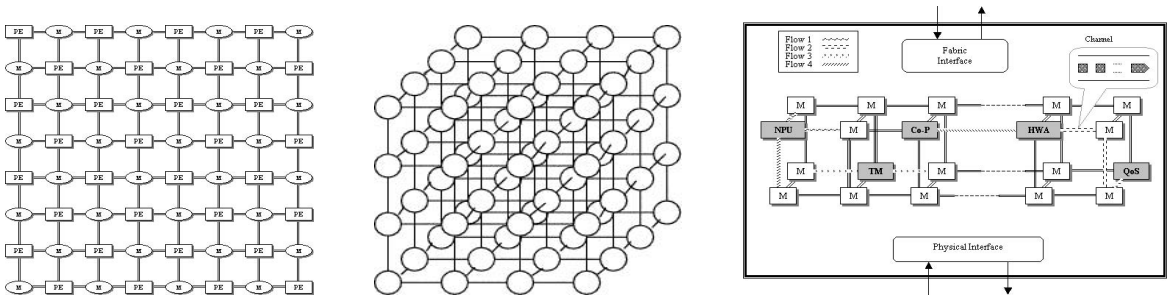

Figure 2. a) 8-ary 2-cube network b) 4-ary 3-cube network c) 3D-mesh

rectional channels (one message occupies the complete channel width), the second bidirectional channels (two messages can share the same channel and can move in either direction), and the third will support quad-directional channels, which partition channels into four (each channel can contain four simultaneous messages in either direction). As a result of channel partitioning, the packet size of each message will decrease. Channel partitions are predetermined before messages enter the interconnect and affect all the channels in the interconnect. Each message can only occupy one sub-channel at a time. The main advantage of channel partitioning is the additional freedom and flexibility in paths that can be selected by propagating messages.

2.1 Example k-ary n-cube based interconnect architectures

In this paper, we consider k-ary n-cube based direct network topologies. A k-ary n-cube network consists of $N = k^n$ nodes, where n represents the dimension of the network and k represents the number of nodes in each dimension of the structure. Fig. 2a and 2b present an 8-ary 2-cube and 4-ary 3-cube networks, in which processors and memories are evenly distributed. Each node in k-ary n-cube interconnect is uniquely labelled and elements of the same plane are connected to each other.

Besides the standard k-ary n-cube structures, we also consider variations of them. Fig. 2c shows a 3D-mesh architecture, which is a 2-ary 3-cube interconnect architecture, extended in the x-direction to provide both the dimensionality required to improve routing as well as space constraints on the line card. PEs and memories are distributed throughout the interconnect in different configurations and allow each PE to use multiple memories as storage as well as data sharing with other processing elements.

2.2 Traffic Controller Architecture

Each node employs a traffic controller (TC) to forward messages. The TC includes five components: the routing algorithm, multi-port switch, channel sampler, channel partitioning mechanism and virtual channels (Fig. 3).

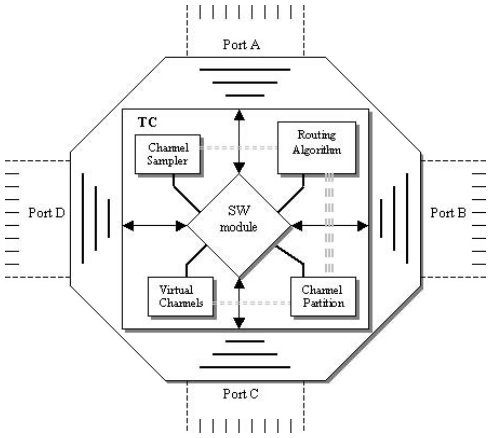

Figure 3. Traffic controller structure

Each cycle, the channel sampler samples each port to determine its status (total of 4 ports). If a port is currently busy transferring a message, the channel sampler will not allow any new messages to be routed to it. The channel partitioning module can divide a unidirectional channel into two or four bidirectional subchannels, as shown in Fig. 4. For example, a channel of 32 bits can be partitioned into 4 sub-channels of 8 bits each, and transfer 4 different messages simultaneously. It receives channel configuration information from the user interface and sets the TC's internal parameters accordingly.

Virtual channels (VC) are used when an incoming message cannot be routed to any output port since all output ports are busy transferring other messages. Fig.

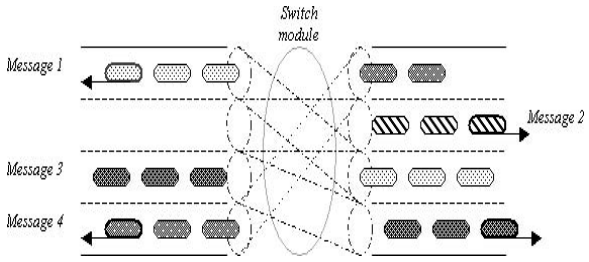

Figure 4. Channel partitioning to 4 sub-channels

5 depicts a situation where two messages compete over the same output port (West port). Since message 1 is granted permission to continue in its path, message 2 will have to be queued in one of the available VCs. There are only two VCs per node since each message deploys two ports simultaneously, if two ports are used by a massage then in the worst case only two messages can arrive to a node and require to use these ports. The VC module, within the TC, sets virtual channels to enabled/disabled status and if enabled, it also allocates its buffer size (in KB).

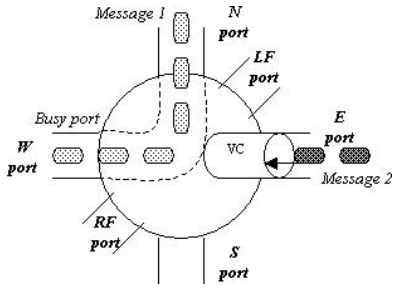

Figure 5. Virtual channels

2.3 Routing Mechanism

The routing algorithm dynamically collects both configurational and real-time changes of all channels to fully utilize routing capabilities. The routing algorithm routes a packet from a source device $s = \{s_1, s_2, ..., s_m\}$ to a destination module $d = \{d_1, d_2, ..., d_n\}$, by choosing a direction to travel in each of the three dimensions (x, y, or z) to balance the channel load. The default route is in the x-axis quadrant and each message will always attempt to be routed using the shortest path as long as the packets are admissible (accepted by idle nodes). If a node is oversubscribed (i.e., all ports are occupied), the routing algorithm can determine if packets will take an alternative route (by choosing the y-axis or z-axis quadrants), or store them in the local virtual channel until conditions improve. If a message cannot use any of the resources available to it in order to continue in its path to destination (i.e., all ports are busy and virtual channels are disabled or fully occupied) the message will be retransmitted from the same source.

In order to improve performance metrics, such as latency and throughput, wormhole message passing mechanism is selected to send messages between source and destination modules. Each message (also called worm) is segmented into smaller size packets, known as flow control digits (flits). The size of each packet is determined by the channel width. In wormhole message passing mechanism, the header flit is sent first, while all other packets follow it in a pipeline manner (resembles a worm movement). As the header propagates through the interconnect, it sets the node switches (if idle) in a certain position corresponding to its shortest path and traffic conditions on the node's channels. The key to achieve high performance using an interconnect architecture combined with wormhole message passing is hidden in the message transfer mechanism. Latency of each worm is composed of the sum of latencies of each propagating packet belongs to the same worm. The dominant part of latency is obtained by the transfer of the header, since it sets the switching element within each TC it passes. Switching latency is the leading delay in the system. The rest of the packets incur only propagation and routing delays, which are smaller in magnitude. Therefore, as the message size becomes larger, the ratio of consecutive latencies decreases while throughput increases.

The three latencies involve in transferring a message through the interconnects are: Ts, Tr and Tw. Ts represents the switching delay (accounts for the header packet and packet generating/receiving modules at the source/destination nodes), Tw denotes the propagation delay of one bit in a unit length (it is equal to 62.5 ps per 1 cm using the current manufacturing technology [11]) and Tr signifies the routing delay. The ideal packet transmission rate from source to destination can be achieved in $Tw + Tr$ units. This becomes a great advantage in attaining high throughput, compared to the shared bus, which can only send those packets like a store and forward type architecture.

2.3.1 Packet Forwarding

Routing decisions are based on the worm's header information and the TC status. The worm's header contains data fields such as worm ID, source node and

destination node. Each worm has a unique ID to differentiate itself from other entering worms. The worm tries to take the shortest path to its destination, if possible. The shortest path is calculated by taking the difference of each direction (x, y, and z) derived from its source-destination addresses. When a worm enters a node, the TC switches the worm to an output port, giving priority to the port pointing in its shortest path to destination. If the highest priority port is used by other worms, the TC will route the worm to an alternative port to bypass nodes which experience hot-spots. Even when an alternative route is taken, the worm continuously calculates its position to follow as close as possible to its original shortest path. If a worm cannot be switched to any of the node ports, it will occupy a virtual channel, assuming virtual channels are enabled.

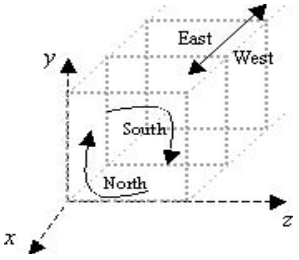

Figure 6. Routing directions

The routing algorithms is derived and based on [3, 6, 10]. The following rules must be satisfied:

- Ensure the shortest path first, by comparing source and destination vectors (in terms of x, y, and z coordinates) and move forward by evaluating the variance in each dimension.

- If one of the chosen output ports is occupied (busy transferring another message), it samples the status of other ports in the following order (Fig. 6): EW (East-West) a movement from one face to another, NS (North-South also up-down) resembles a clockwise vs. counter-clockwise movement on individual face.

- Avoid certain consecutive turns. This rule seeks to avoid deadlocks. A worm following an EN, ES will not take west movement as the third direction. Similarly, if WN, WS movements are taken then, it will not take an east movement as the third direction.

- Since worms are generated either from PEs to memories and vice versa then the worm's relative direction is always towards its destination and will never move backward (towards its source). This step attempts to avoid livelocks.

3 Performance Evaluation

We developed an event-driven simulator to explore different types of interconnect architectures to increase the off-chip memory bandwidth on line cards. The simulator block diagram, given in Fig. 7, contains both the off-chip interconnect architectures to be evaluated and the control modules to adjust, collect and modify interconnect settings, dataflow, and performance metrics. The interconnects configuration manager sets the interconnect type properties such as channel width, VC on/off, bidirectional channel.

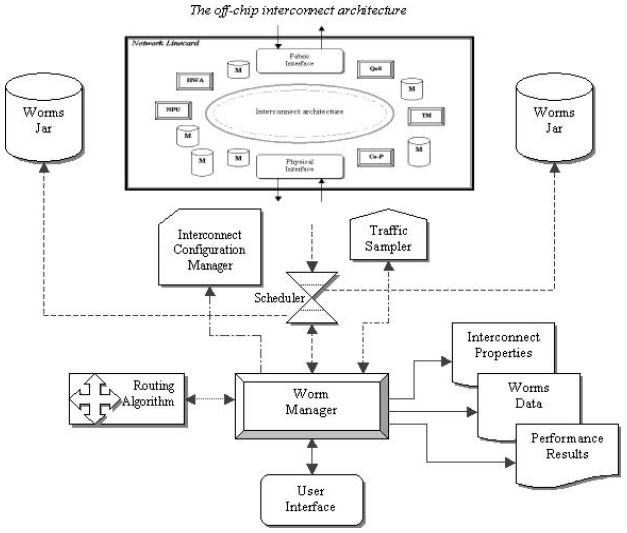

Figure 7. Simulation control modules

The interconnect properties are collected from the user interface and transferred to the configuration manager via the worm manager. The worm manager must get this configuration information to configure, accordingly, other modules in the system that participate in simulation. The traffic sampler collects performance data such as throughput, routing accuracy and interconnect link utilization parameters. Those parameters are sent back to the worm manager to adjust worm generation rate and load balance the traffic. The routing algorithm is attached to the worm manager and receives data of each individual worm and its current position. The routing algorithm assists the worms which cannot continue in the shortest path due to traffic hotspots. Worms are constantly generated and stored in a worm jar. The scheduler is responsible to inject

worms into the interconnect taking into account the total network capacity and traffic load. Since the worm manager knows the total number of worms that are modelled throughout the simulation, it must inform the scheduler the end of the simulation (no more worms to model). The user interface allows the user to change interconnect architecture, parameters, the number of worms to model, and the simulation length. The simulator takes into account all practical parameters such as switching delays (T_s), routing delays (T_r) and propagation delays (T_w) as well as the complete functionality of each system components (nodes, links, PE/memory, interfaces, virtual channels, and channel partitioning). Simulation time is based on a unit cycle which is equal to one clock cycle ($T_w + T_r$). All other delays are calculated as multiples of it. This provides the advantage of having single uniform simulation clock. Message size in bytes and message generation time are obtained using pseudo-random number generator, which is utilized to resemble the randomness of packet transmission by both processors and memories. Each worm is linked to performance-bookkeeping function, which records its latency, throughput, simulation cycles, failures, and route-taken from the moment the worm enters the interconnect until it completely reaches its destination. Performance results are provided at the end of each simulation where the software provides the average of each parameter separately.

3.1 Simulation Results

3.1.1 Latency

Latency represents the time it takes for a worm to reach its destination. Depending on the worm movement, latency sums wire transfer, switching and routing delays at each cycle. The resulting latency is an average of latencies collected from all worms generated, at the end of the simulation. We chose three representative k-ary n-cube interconnects for our simulations: 8-ary 2-cube, 4-ary 3-cube and 3D-mesh (all three interconnects have 64 nodes).

Fig. 8 shows a comparison among all three interconnects with VCs and channel partitioning enabled. The results shown are an average of 10 different simulations with both short (128B-1KB) and long (1KB-8KB) worms and identical interconnect settings. The lowest latency was recorded for the 3D-mesh, while the 4-ary 3-cube network has slightly higher latency than the 3D-mesh. The 8-ary 2-cube interconnect has the highest average latency.

Fig. 9 portrays the latency of each interconnect with respect to the offered load. Offered load determines the probability that each node, comprising the interconnect, will generate a message within each simulation cycle. For example, if the offered load is set to 0.1 there is a probability that 10% of the total nodes in the interconnect will generate a message at each simulation cycle. Fig. 9 shows that as offered load increases the latency increases exponentially for all interconnects. Further, 3D-mesh interconnect can sustain the highest offered load out of the three interconnects.

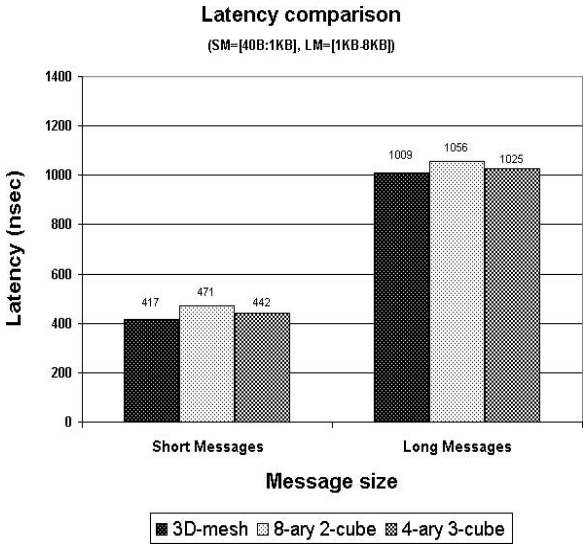

Figure 8. Latency comparison

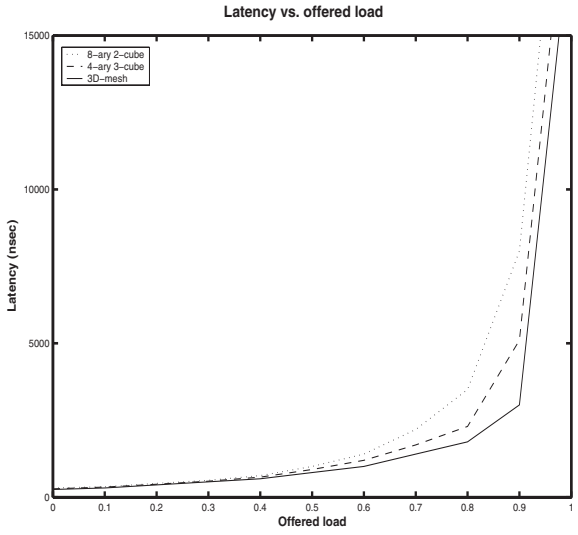

Figure 9. Latency vs. offered load

3.1.2 Throughput

Throughput is measured by taking samples of the total bits processed within the interconnect at each cycle. Throughput significantly increases when VCs are enabled since VCs allow more worms to occupy the interconnect without transmission failures. Fig. 10 shows that the highest throughput was reached by the 3D-mesh interconnect for both short and long messages.

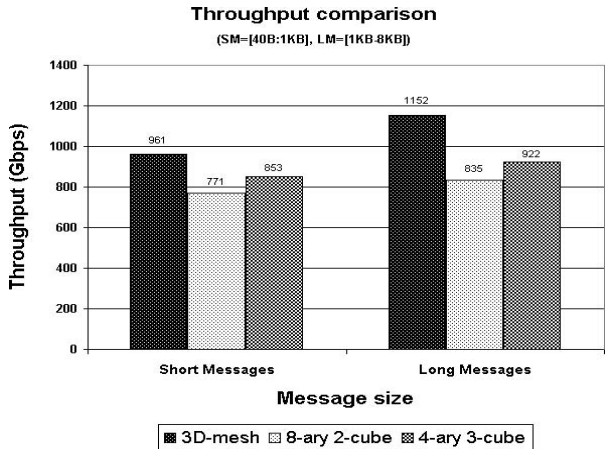

Figure 10. Throughput comparison

3.1.3 Interconnect resources utilization

Interconnect resource utilization counts the number of busy ports within each traffic controller per simulation cycle. At the end of simulation it provides the average of ports that were set to busy status out of the total number of ports available in the interconnect throughout simulation. The results of interconnect utilization is shown in Fig. 11 reveal that 4-ary 3-cube ports are set to busy status more often than the 3D-mesh or 8-ary 2-cube.

3.2 Performance comparison with commercial interconnects

Finally, we evaluate our 3D-mesh, 8-ary 2-cube, and 4-ary 3-cube interconnects with other currently used high-performance interconnect technologies such as Hypertransport [12], Infiniband [13] and PCI-Express [14].

We used reported results provided by each individual vendor to compare with our results. In addition, the performance properties of these technologies takes into account a constant channel size of 32-bits and a

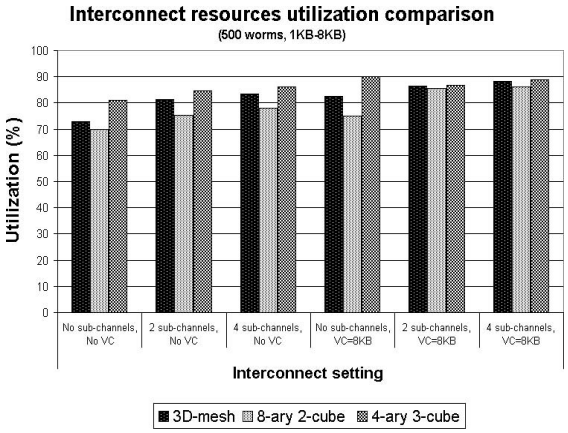

Figure 11. Interconnect utilization rate

single communication link. For the 3D-mesh interconnect the settings are: channel width is 32 bits, interconnect size is 16 cubes, number of worms generated is 10, each worm is 1KB in size. Virtual channels as well as channel partitions were enabled. The throughput comparison results are shown in Fig. 12. The throughput values of the 3D-mesh, 8-ary 2-cube and 4-ary 3-cube interconnects represent the average throughput of each interconnect for worm sizes 40B-32KB. 3D-mesh shows superior results compared to all of its competitors reaching a peak throughput of 452 Gbps. This is more than twice the throughput of the best interconnect available not including the other types of k-ary n-cubes tested.

4 Conclusions and Future Work

In this paper, we proposed to use k-ary n-cube based off-chip packet-switched networks as memory I/O interconnect for high throughput network processors. We developed an event-driven simulator to evaluate the performance of the proposed interconnect arhcitectures. Results show that k-ary n-cube based topologies significantly outperforms the current solutions on network line cards. Future directions for this work include physical implementation of the interconnect on a printed-circuit board and testing it. It will be also interesting to see if this interconnect can be used instead of bus mechanisms used in PC architectures and on-chip communications mechanisms used within the network processors.

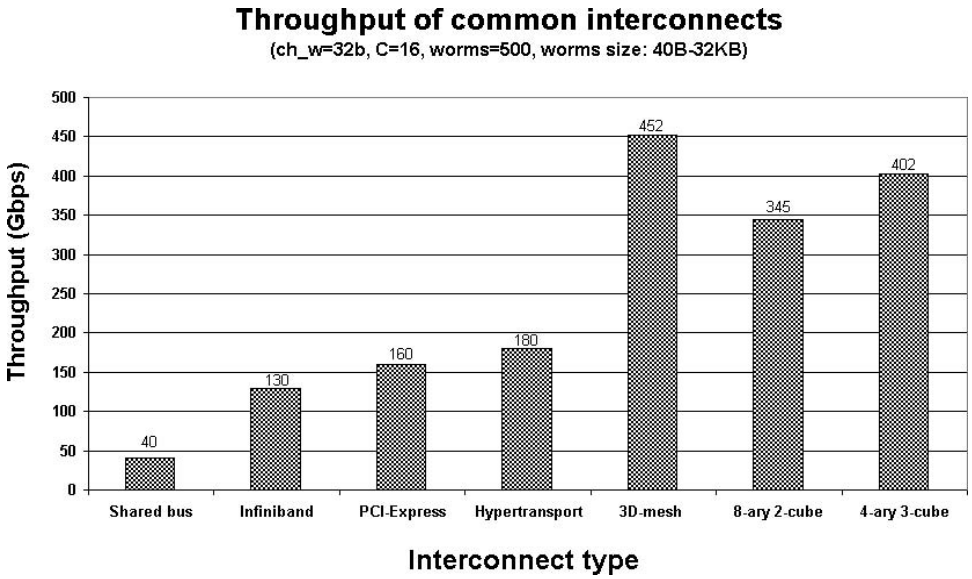

Figure 12. Throughput comparison

References

[1] A. Agarwal, "Limits on interconnection network performance", *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398-412, 1991.

[2] H.S. Azad and M. Ould-Khaoua, "Analytical modeling of wormhole routed k-ary n-cubes in the presence of hot-spot traffic", *IEEE Trans. on Computers*, vol. 50, no. 7, pp. 623-634, 2001.

[3] G. M. Chiu, "The odd-even turn model for adaptive routing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729-738, 2000.

[4] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks", *IEEE Tran. on Computers*, vol. 39, no. 6, pp. 775-785, 1990.

[5] W. J. Dally, A. Singh, B. Towels, and A. K. Gupta, "Globally adaptive load-balanced routing on tori", *Computer Architecture Letters*, vol.3, Mar. 2004.

[6] O. Lysne, "Deadlock avoidance for switches based on wormhole networks", *Proceedings of the Annual International Conference of Parallel Processing*, pp. 68-74, 1999.

[7] G. Min and M. Ould-Khaoua, "Performance analysis of wormhole switching in k-ary n-cubes under multimedia traffic", *Proc. of IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2001.

[8] S. Scott and G. Thorson, "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus," *Proc. of Hot Interconnects IV*, Stanford University, August 1996.

[9] C. L. Seitz, "The cosmic cube", *ACM Trans. on Communications*, pp. 23-33, vol. 1, 1985.

[10] A. Singh, W. J. Dally, A. K. Gupta, B. Towles, "GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks", *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pp. 194 - 206, 2003.

[11] Y. Zhang, "Microstrip-multilayer delay line on printed-circuit board", Technical Report, University of Nebraska, Lincoln, April, 2003.

[12] HyperTransport Consortium, "HyperTransport Technology Specifications", 2005. (web: http://www.hypertransport.org/tech/index.cfm)

[13] Infiniband Trade Association, "Infiniband architecture specification, rev. 1.0", Oct. 2000 (web: http://www.infinibandta.org)

[14] PCI Special Interest Group, "PCI express base specification rev. 1.0a", Apr. 2003.