Engel, J., & Kocak, T. (2006). K-ary n-cube based off-chip communications architecture for high-speed packet processors. 1903 - 1906. 10.1109/MWSCAS.2005.1594497

# *K*-ary *n*-cube Based Off-Chip Communications Architecture for High-Speed Packet Processors

Jacob Engel and Taskin Kocak
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816
e-mail: {jengel, tkocak}@cs.ucf.edu

***Abstract*— A *k*-ary *n*-cube interconnect architecture is proposed, as an off-chip communications architecture for line cards, to increase the throughput of the currently used memory system. The *k*-ary *n*-cube architecture allows multiple packet processing elements on a line card to access multiple memory modules. The main advantage of the proposed architecture is that it can sustain current line rates and higher while distributing the load among multiple memories. Moreover, the proposed interconnect can scale to adopt more memories and/or processors and as a result increasing the line card processing power. Our results portray that *k*-ary *n*-cube sustained higher incoming traffic load while keeping latency lower than its shared-bus competitor.**

## I. INTRODUCTION

In this paper, we present an off-chip *k*-ary *n*-cube interconnect that provides an effective solution, under certain constraints, to the increasing demand for memory bandwidth on line cards. Memory bandwidth is affected by two major factors: higher line rates and increase in deep packet inspection operations. Network line rates are constantly increasing, currently reaching 40 Gb/sec. Line cards are required to perform multiple functions to support new services which increases the traffic overhead to incoming line rates by 40%-100%. Moreover, there exist an unstoppable expansion in lookup tables which requires more memory space and more parameters to examine within each packet.

In most line card architectures there exist a direct interconnect, such as busses or switches, that connects different processing elements to memory modules. The heart of the line card is the network processor which performs different operations in order to analyze the flow of incoming packets. The nature of packet processing requires frequent read/write operations to memories which are distributed around the NPU.

As line-rate data and NPU processing power increase, memory access time becomes the main system bottleneck (it requires a minimum of 8 ns processing time at 40 Gbps line-rate) during data store/retrieve operations. The growing demand for memory bandwidth contrasts the notion of direct interconnect methodologies and replaced it with indirect, packet-based networks such as mesh, torus or k-ary n-cubes.

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.

Solutions to the memory bandwidth bottleneck are limited by area on the line card and NPU I/O pins. Pin constraints bound the bus size that can be interfaced with the NPU. Hence, only a packet-based network-on-board can provide the required performance improvement between the NPU and off-chip memory modules.

## II. *K*-ARY *n*-CUBE INTERCONNECT ARCHITECTURE

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. Each node is connected to all of its nearest neighbors via bi-directional channels. The address/location of a node can be represented as a vector consists of two bit-vector fields. One bit vector represents the location of the node within its plane. This vector can repeat its value within other planes. The second vector represents the node dimension location. For example, node *m* can have an address vector with $m_i$ being the node's location in its dimension.

The *k*-ary *n*-cube architecture shown in Fig. 1 is a packet-based multiple path interconnect that allows network packets to be shared by different processor and memory modules on the network line card. Memories are distributed around processing elements, such as traffic manager, QoS co-processor, classification processor, to allow data sharing among modules and direct processor memory storage. If a link goes down, not only should the fault be limited to the link, the additional links from the intermediate nodes should ensure the connectivity continues.

### *A. Routing mechanism*

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 to balance the channel load. A memory packet sent by a processing element (PE) will always attempt to take the shortest path as long as packets are admissible (accepted by ideal nodes). If a node is oversubscribed (i.e., all ports are occupied), packets in transit will take a different route using the traffic controller (TC) in each corner (node). The architecture path diversity offers alternative paths between source and destination modules.

0-7803-9197-7/05/$20.00 © 2005 IEEE.

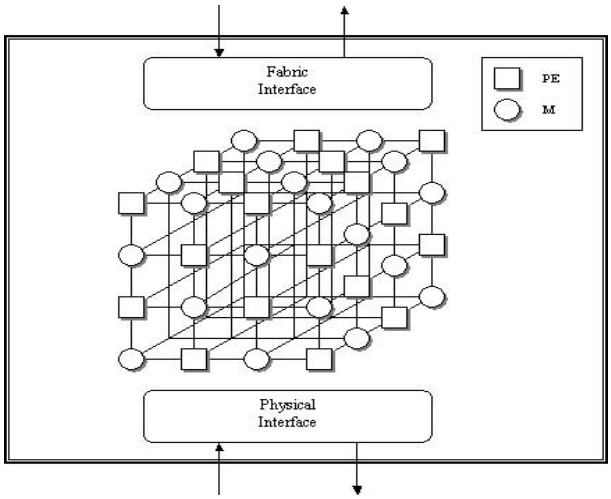


Fig. 1. *k*-ary *n*-cube architecture on the line card

In wormhole routing, the header flit is sent first. While the header propagates through the interconnect, it sets the node switches in a certain position corresponding to the traffic load on the node's channels. The rest of the packets comprising the message are transmitted in a pipeline manner following the message header. The main advantage in using wormhole routing in this *k*-ary *n*-cube structure is that it diminishes the latency as the size of the message increases while increasing its throughput. The major part of the latency is hidden in the transfer of the first packet. The rest of the packets follow it and introduce only wire transfer delay. As the message size increases, the ratio of consecutive latencies decreases.

Before a node forwards a packet to one of its adjacent nodes, it polls the status of each node. The traffic controller at each node has a "sensor status" flag which determines if the node is currently processing a packet (i.e., busy), or if the node is idle and ready to accept a new packet (i.e., not busy). Depending on its direction preferences (some nodes may get higher preference than others if they are located closer to the packet's destination), TC will chose an admissible node to forward the packet. If at least one adjacent node is available to forward a packet, it will require only one clock cycle to do so.

## III. Performance evaluation

We use standard performance metrics such as latency to evaluate the *k*-ary *n*-cube interconnect performance. Latency is defined as the time it takes for a complete message to reach its destination. The load or offered-load is the number of packets injected into the interconnect network per second and depends on the processing elements which are generating them.

The *k*-ary *n*-cube architecture and performance measures are based on the following assumptions:

- Messages are uniformly distributed among modules.
- Nodes are generating traffic independently of each other.
- It takes only one propagation delay cycle for a flit to move from one node to another.
- Each physical channel consists of five virtual channels.

### A. Latency of packet-switched multiprocessor shared-bus

Bus Performance, in terms of latency and throughput, of a shared bus is affected by the following factors:

- Number of processors or memories connected to it (as the number of modules connected to the shared-bus increases, performance degrades)
- The shared-bus length (average is between 5-12 inches).
- Shared-bus width (increasing width results in higher cost).

Most shared bus systems do not have more than 30 processors / memories. Communication protocol is trivial and is based on simple connection oriented mechanism. In addition, shared bus systems require arbitration mechanism to send/receive data from multiple modules. Bus arbitration adds delay to the overall system latency. Some systems use multiple buses to reduce the effect of the factors mentioned above. Although the bandwidth of the multiple bus architecture is higher than that using a single shared bus, the system is more costly and requires complex bus arbitration logic. As a result of the shared bus disadvantages many multi processor-memory systems include multiple shared buses and/or other type of interconnects.

The system illustrated in Fig. 2 describes a multiple-processor (multiple-memory) packet-based shared bus. A processor-memory communication is only allowed when the bus is not in use by other devices.

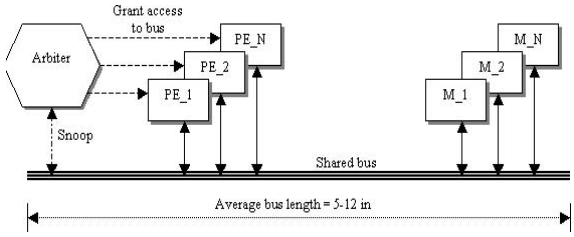


Fig. 2. Shared-bus multiple processor with arbitration

A processor wishes to communicate with a memory receives permission to send data from the arbiter. The data is transferred using flits. The bus width determines the flit size. When multiple processors wish to use the bus it can cause contention. Therefore, the bus is being monitored by the arbiter and only the arbiter grants processors access to the bus. Bus of this type fabricated on a PCB line card has an average length between 5 inches to 12 inches. The bus length is an important factor in determining the packet propagation delay through the bus (not including queuing, arbitration or transfer time).

$$L_{shared-bus} = \frac{M}{w} * T_w * l + W_q \quad (1)$$

Equ. 1 represents the latency of a shared bus. The first component embodies the delay associated with message propagation from source to destination through the bus. $\frac{M}{w}$ represents the number of flits to send, $T_w$ (=254 psec) is the time it

takes for a packet to propagate in 1 inch [7]. $l$ represents the bus length. The second latency component, $W_q$, is due to queuing within each module and signifies the average waiting time between message transmissions of the same module. Each module can transmit a message in its turn. Messages are queued in each source until released.

Both arbitration latency and average wait in queue components provide the queuing latency. For example, in FCFS scheme the queuing latency is equal to (N-1)*r [6]. N symbolizes the number of modules (processors and/or memories) connected to the bus. The value of *r* varies in each system depending on the queuing model which represents the bus type and communicating modules connected to it. The (N-1) term represents the number of turns (bus cycles) a device must wait before it can receive bus access again. A packet-based multiple processors shared bus can be modelled as M/D/1 queue. M/D/1 queue model has a Poisson input (random bus access requests by processors) to a single-server queue with constant service times (this is the bus throughput), unlimited capacity and FIFO queue discipline. Arrival rate, $\lambda$, is the rate in which processor requires memory access and depends on the incoming line rate. The arrival rate we use represent an aggregate arrival rate, $m_g$, which includes the amount of traffic (in flits) generated by a node per cycle per second. Note that,

$$\sum_{i=1}^{PE} \lambda_i < \mu \tag{2}$$

The sum of all arrival rates cannot exceed the service rate of the system. Moreover, $\lambda$ is used in our model as an aggregate value of all issued communications by PEs to the shared-bus. Service rate, $\mu$, is the rate in which the bus can service a packet once a processor granted access to the bus. This rate is determined by the bus throughput (bits per second). $\mu$ includes the arbitration cycles which each module is required to wait before receiving access to the bus. That is,

$$\mu = \frac{1}{(N-1) * r} \tag{3}$$

where,

$$r = \frac{M}{w} * T_w * l \tag{4}$$

The average waiting time of a message in its generating node queue is denoted by the following equation

$$W_q = \frac{\rho}{2 * \mu * (1-\rho)} \tag{5}$$

### *B. K-ary n-cube latency equation*

The latency model for *k*-ary *n*-cubes was initially modeled by M. O. Khaoua in [1] and was developed also in [3][4][5]. Under uniform traffic pattern, a message passes on average $\bar{k}$ hops across the network

$$\bar{k} = \frac{(k-1)}{2} \tag{6}$$

The average distance of this message is

$$\bar{d} = n * \bar{k} \tag{7}$$

The mean message latency consists of two parts, the delay due to message transmission and the time a message spends if blocking occurs. For an average of $\bar{d}$ hops from source to destination, latency can be expressed as

$$Latency_{k-ary,n-cube} = M + \bar{d} + \sum_{i=1}^{\bar{d}} B_i + W_{ej} \tag{8}$$

The first term, M, is the message length in flits. $B_i$ is the average blocking time seen by a message at any $i^{th}$ hop, where $(1 \leq i \leq \bar{d})$. $W_{ej}$ represents the mean waiting time between message transmission at the ejection node. In this equation, it is assumed that channel to channel transfer time is 1 unit cycle. V virtual channels are used per physical channel in the model introduced by [2][9]. Although virtual channels are divided into two types adaptive and deterministic, there is no distinction between them when computing virtual channels occupancy probabilities [8]. A message is blocked at any $i^{th}$ hop channel when all virtual channels are busy. If $W_b$ denote the average waiting time due to blocking and $P_b$ represents the probability of blocking then the mean blocking time expression is

$$B_i = P_{b_i} * W_b \tag{9}$$

The blocking probability, $P_b$, is determined by calculating the probability that all virtual channels are busy. Virtual channels analysis and analytical model is discussed in [1]. Since multiple virtual channels share the bandwidth of the same physical channel, a multiplexer is required to select which virtual channel will use the physical channel. This multiplexer functions in a TDM manner and adds the following component to the latency equation

$$\bar{V} = \frac{\sum_{i=0}^{V} i^2 * P_i}{\sum_{i=0}^{V} i * P_i} \tag{10}$$

The mean waiting time at any node within the message's path is given as

$$W_b = \frac{m_c * S^2 * (1 + \frac{(S-M)^2}{S^2})}{2 * (1 - m_c * S)} \tag{11}$$

where, the latency of the *k*-ary *n*-cube network, S, is measured in cycles and $m_c$ is the traffic rate on a given channel and is expressed as

$$m_c = \frac{m_g * \bar{d}}{n} \tag{12}$$

The mean waiting time of a message at any ejection channel is given as

$$W_{ej} = \frac{m_g * M^2}{2 * (1 - m_g * M)} \tag{13}$$

The mean waiting time in a source node is determined by modeling the injection channel at the source node as an M/G/1 queue gives

$$W_s = \frac{\frac{m_g}{V} * S^2 * (1 + \frac{(S-M)^2}{S^2})}{2 * (1 - \frac{m_g}{V} * S)} \tag{14}$$

The term $\frac{m_g}{V}$ denotes the mean arrival rate.

The overall message latency is composed of the sum of the mean network latency, S, and the mean waiting time at the source node, $W_s$, multiplied by the multiplexing factor $\bar{V}$ to account for the virtual channels multiplexing that takes place over the physical channel. Thus,

$$L_{msg} = (S + W_s) * \bar{V} \tag{15}$$

### *C. Performance results k-ary n-cube interconnect vs. shared-bus*

Performance of *k*-ary *n*-cube is compared against a shared-bus. Since line cards contain multiple network processing elements and can reach 64 co-processors (including memory modules), we chose two combinations of *k*-ary *n*-cubes which comprised of 64 nodes and evaluate their performance. Both shared-bus and the *k*-ary *n*-cube have 32 bits channel width. Fig. 3 compares the latency of 8-ary 2-cube network with 4-ary 3-cube network while traffic load increases. Moreover, performance for both types of *k*-ary *n*-cubes were analytically computed using 32-flits and 64-flits messages.

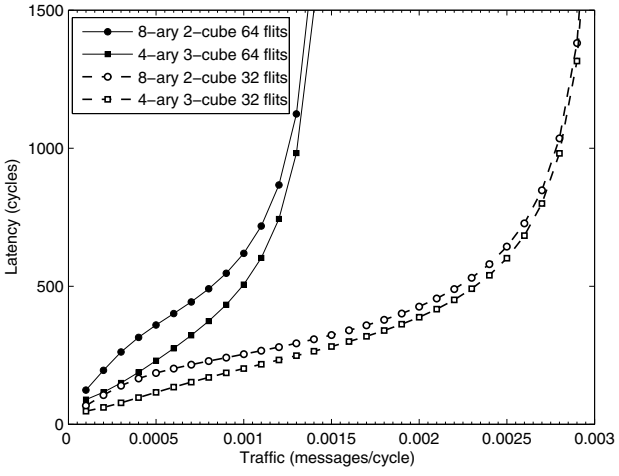


Fig. 3. Latency comparison between 4-ary 3-cube and 8-ary 2-cube

Fig. 3 depicts that 4-ary 3-cube network was superior to 8-ary 2-cube network with respect to load vs. latency for both 32 and 64 flits message. Once the better *k*-ary *n*-cube was chosen we compared its performance against a shared-bus (Fig. 4).

Fig. 4 portrays latency comparison results for shared bus vs. 4-ary 3-cube network. In both interconnects the channel width is 32-bits. 4-ary 3-cube network was able to sustain much higher traffic rate while keeping lower latency than its competitor the shared-bus. Moreover, the 4-ary 3-cube network maintained its exceptional latency for both low as well as high traffic loads.

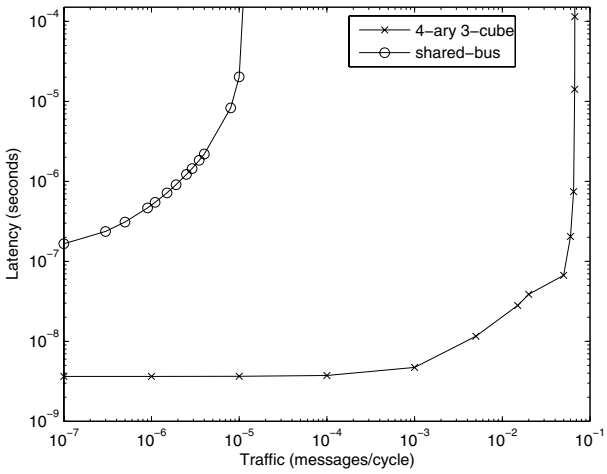


Fig. 4. Latency comparison between 4-ary 3-cube and shared-bus

## IV. CONCLUSION

We presented a *k*-ary *n*-cube interconnect as an off-chip solution for line cards. Results show that the *k*-ary *n*-cube interconnect had superior performance over commonly used share-bus and it can adapt to higher line rates while maintaining low latency. The *k*-ary *n*-cube interconnect can be easily scalable to accommodate more processors/memories in order to increase its processing power or if line rates exceed the current rates and require a change in network size.

## REFERENCES


[1] M. O. Khaoua, "A Performance Model for Duato's Fully Adaptive Routing Algorithm in *k*-Ary *n*-Cubes", *IEEE Transactions on Computers*, vol. 48, no. 12, pp. 1297-1304, 1999.

[2] W. J. Dally, "Virtual-Channel Flow Control", *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-199, 1992.

[3] J. Kim and C. R. Das, "Hypercube Communication Delay with Wormhole Routing", *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 806-813, 1994.

[4] A. Agarwal, "Limits on Interconnection Network Performance", *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398-412, 1991.

[5] W. A. Najjar, A. Lagman, S. Sur and P. K. Srimani, "Analytical Models of Adaptive Routing Strategies", Department of Computer Science, Colorado State University, August 10, 1994.

[6] W. L. Bain and S. R. Ahuja, "Performance Analysis of High-Speed Digital Buses for Multiprocessing Systems", *Proceedings of the 8th annual symposium on Computer Architecture*, pp. 107-133, 1981.

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

[8] H. S. Azad and M. O. Khaoua, "A Simple Mathematical Model of Adaptive Routing in Wormhole *k*-ary *n*-cubes", *Proceedings of the 2002 ACM symposium on Applied computing*, pp. 835-839, 2002.

[9] Y. M. Boura and C. R. Das, "Modeling Virtual Channel Flow in Hypercubes", *Proceedings of the First IEEE Symposium on High Performance Computer Architecture* , pp. 166-175, 1995.