The BlackWidow High-Radix Clos Network

Steve Scott* sscott@cray.com
Dennis Abts* dabts@cray.com
John Kim† jjk12@stanford.edu
William J. Dally† dally@stanford.edu

*Cray Inc.
Chippewa Falls, Wisconsin 54729

† Stanford University
Computer Systems Laboratory
Stanford, California 94305

Abstract

This paper describes the radix-64 folded-Clos network of the Cray BlackWidow scalable vector multiprocessor. We describe the BlackWidow network which scales to 32K processors with a worst-case diameter of seven hops, and the underlying high-radix router microarchitecture and its implementation. By using a high-radix router with many narrow channels we are able to take advantage of the higher pin density and faster signaling rates available in modern ASIC technology. The BlackWidow router is an 800 MHz ASIC with 64 18.75Gb/s bidirectional ports for an aggregate off-chip bandwidth of 2.4Tb/s. Each port consists of three 6.25Gb/s differential signals in each direction. The router supports deterministic and adaptive packet routing with separate buffering for request and reply virtual channels. The router is organized hierarchically [13] as an 8×8 array of tiles which simplifies arbitration by avoiding long wires in the arbiters. Each tile of the array contains a router port, its associated buffering, and an 8×8 router subswitch. The router ASIC is implemented in a 90nm CMOS standard cell ASIC technology and went from concept to tapeout in 17 months.

1 Introduction

The interconnection network plays a critical role in the cost and performance of a scalable multiprocessor. It determines the point-to-point and global bandwidth of the system, as well as the latency for remote communication. Latency is particularly important for shared-memory multiprocessors, in which memory access and synchronization latencies can significantly impact application scalability, and is becoming a greater concern as system sizes grow and clock cycles shrink.

The Cray BlackWidow system is designed to run demanding applications with high communication requirements. It provides a globally shared memory with direct load/store access, and, unlike conventional microprocessors, each processor in the BlackWidow system can support thousands of outstanding global memory references. The network must therefore provide very high global bandwidth, while also providing low latency for efficient synchronization and scalability.

Over the past 15 years the vast majority of interconnection networks have used low-radix topologies. Many mutiprocessors have used a low-radix k-ary n-cube or torus topology [6] including the SGI Origin2000 hypercube [14], the dual-bristled, sliced 2-D torus of the Cray X1 [3], the 3-D torus of the Cray T3E [20] and Cray XT3 [5], and the torus of the Alpha 21364 [18]. The Quadrics switch [1] uses a radix-8 router, the Mellanox router [17] is radix-24, and the highest radix available from Myrinet is radix-32 [19]. The IBM SP2 switch [22] is radix-8.

The BlackWidow network uses a high-radix folded Clos [2] or fat-tree [15] topology with sidelinks. A low-radix fat-tree topology was used in the the CM-5 [16], and this topology is also used in many clusters, including the Cray XD1 [4]. The BlackWidow topology extends this previous work by using a high-radix and adding sidelinks to the topology.

During the past 15 years, the total bandwidth per router has increased by nearly three orders of magnitude, due to a combination of higher pin density and faster signaling rates, while typical packet sizes have remained roughly constant. This increase in router bandwidth relative to packet size motivates networks built from many *thin* links rather than fewer *fat* links as in the recent past[13]. Building a network using *high-radix* routers with many narrow ports reduces the latency and cost of the resulting network.

The design of the YARC[1] router and BlackWidow network make several contributions to the field of interconnection network design:

- The BlackWidow topology extends the folded-Clos topology to include *sidelinks*, which allow the global network bandwidth to be statically partitioned among the peer subtrees, reducing the cost and the latency of the network.
- The YARC microarchitecture is adapted to the constraints imposed by modern ASIC technology — abundant wiring but limited buffers. The abundant wiring available in the ASIC process enabled an $8\times$ speedup in the column organization of the YARC switch, greatly simplifying global arbitration. At the same time, wormhole flow control was used internal to YARC because insufficient buffers were available to support virtual cut-through flow control.
- YARC provides fault tolerance by using a unique routing table structure to configure the network to avoid bad links and nodes. YARC also provides link-level retry and automati-

[1] YARC stands for 'Yet Another Router Chip', and is also Cray spelled backwards.

cally reconfigures to reduce channel width to avoid a faulty bit or bits.

- YARC employs both an adaptive routing method and a deterministic routing method based on address hashing to balance load across network channels and routers.

This paper describes the BlackWidow (BW) multiprocessor network and the microarchitecture of YARC, the high-radix router chip used in the BW network. The rest of the paper is organized as follows. An overview of the BlackWidow network and the high-radix Clos topology is described in Section 2. We provide an overview of the microarchitecture of the YARC router used in the BlackWidow network in Section 3. In Section 4, the communication stack of the router is discussed and the routing within the BlackWidow network is described in Section 5. The implementation of the YARC router is described in Section 6. We provide some discussions in Section 7 on key design points of the BlackWidow network and the YARC router, and present conclusion in Section 8.

2 The BlackWidow Network

2.1 System Overview

The Cray BlackWidow multiprocessor is the follow-on to the Cray X1. It is a distributed shared memory multiprocessor built with high performance, high bandwidth custom processors. The processors support latency hiding, addressing and synchronization features that facilitate scaling to large system sizes. Each BlackWidow processor is implemented on a single chip and includes a 4-way-dispatch scalar core, 8 vector pipes, two levels of cache and a set of ports to the local memory system.

The system provides a shared memory with global load/store access. It is globally cache coherent, but each processor only caches data from memory within its four-processor node. This provides natural support for SMP applications on a single node, and hierarchical (e.g.: shmem or MPI on top of OpenMP) applications across the entire machine. Pure distributed memory applications (MPI, shmem, CAF, UPC) are of course also supported, and expected to represent the bulk of the workload.

2.2 Topology and Packaging

The BlackWidow network uses YARC high-radix routers, each of which has 64 ports that are three bits wide in each direction. Each BW processor has four injection ports into the network (Figure 1), with each port connecting to a different network *slice*. Each slice is a completely separate network with its own set of YARC router chips. The discussion of the topology in this section focuses on a single slice of the network.

The BlackWidow network scales up to 32K processors using a variation on a folded-Clos or fat-tree network topology that can be incrementally scaled. The BW system is packaged in modules, chassis, and cabinets. Each compute module contains eight processors with four network ports each. A *chassis* holds eight compute modules organized as two 32-processor rank 1 (R1) subtrees, and up to four R1 router modules (each of which provides two network slices for one of the subtrees). Each R1 router module

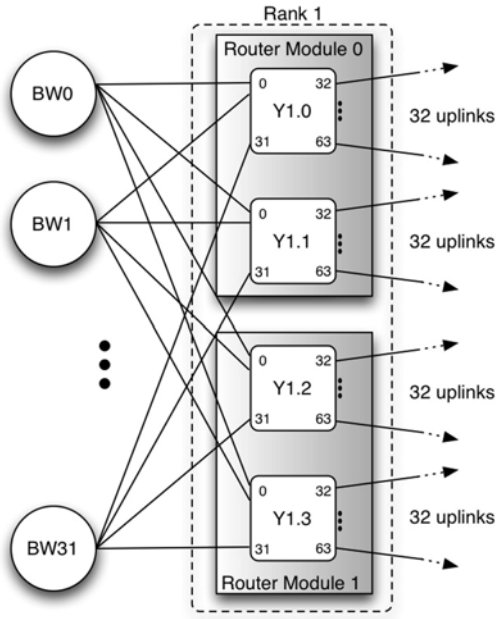


Figure 1. The BlackWidow network building blocks are 32-processor local groups connected via two rank 1 router modules each with two YARC (Y) router chips.

contains two 64-port YARC router chips (Figure 1) providing 64 *downlinks* that are routed to the processor ports via a mid-plane, and 64 *uplinks* (or *sidelinks*) that are routed to eight 96-pin cable connectors that carry eight links each.[2] Each *cabinet* holds two chassis (128 processors) organized as four 32-processors R1 subtrees. Machines with up to 288 processors, nine R1 subtrees, can be connected by directly cabling the R1 subtrees to one another using *sidelinks* as shown in 2(a) and 2(b) to create a rank 1.5 (R1.5) network.

To scale beyond 288 processors, the uplink cables from each R1 subtree are connected to rank 2 (R2) routers. A rank 2/3 router module (Figure 2c) packages four YARC router chips on an R2/R3 module. The four radix-64 YARC chips on the R2/R3 module are each split into two radix-32 *virtual* routers (see Section 7.4). Logically, each R2/R3 module has eight radix-32 routers providing 256 network links on 32 cable connectors. Up to 16 R2/R3 router modules are packaged into a stand-alone router cabinet.

Machines of up to 1024 processors can be constructed by connecting up to 32 32-processor R1 subtrees to R2 routers. Machines of up to 4.5K processors can be constructed by connecting up to 9 512-processor R2 subtrees via side links. Up to 16K processors may be connected by a rank 3 (R3) network where up to 32 512-processor R2 subtrees are connected by R3 routers. In theory networks up to 72K processors could be constructed by connecting nine R3 subtrees via side links; however, the maximum-size BW system is 32K processors.

The BW topology and packaging scheme enables very flexible provisioning of network bandwidth. For instance, by only using

[2] Each network cable carries eight links to save cost and mitigate cable bulk.

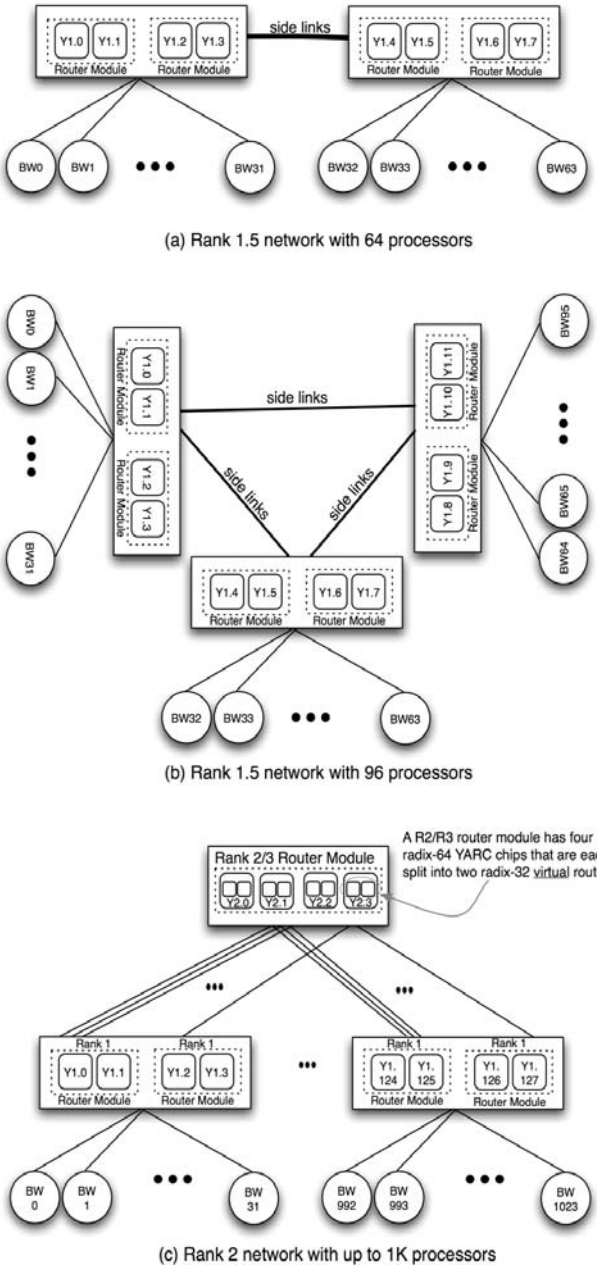


Figure 2. The BlackWidow network scales up to 32K processors. Each rank 1 (R1) router module connects 32 BW processors and the rank 2/3 (R2/R3) modules connect multiple R1 subtrees.

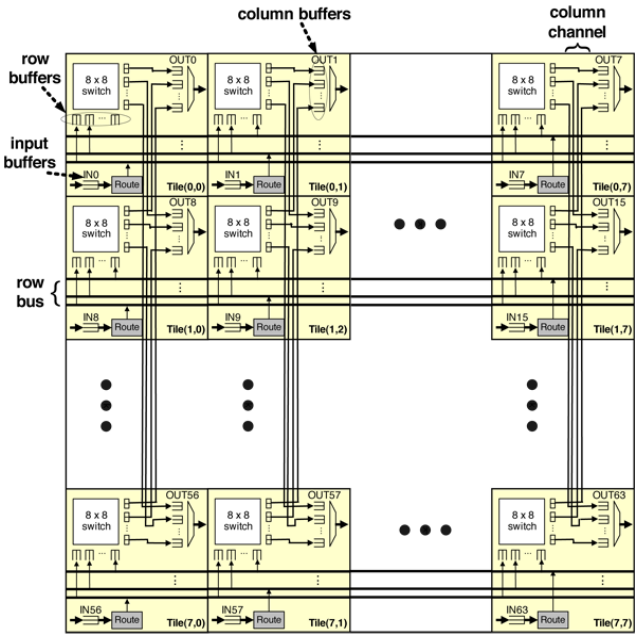


Figure 3. YARC router microarchitectural block diagram. YARC is divided into a 8×8 array of tiles where each tile contains an input queue, row buffers, column buffers, and an 8×8 subswitch.

a single rank 1 router module (instead of two as shown in Figure 1), the port bandwidth of each processor is reduced in half — halving both the cost of the network and its global bandwidth. An additional bandwidth *taper* can be achieved by connecting only a subset of the rank 1 to rank 2 network cables, reducing cabling cost and R2 router cost at the expense of the bandwidth taper.

3 YARC Microarchitecture

The input-queued crossbar organization often used in low-radix routers does not scale efficiently to high radices because the arbitration logic and wiring complexity both grow quadratically with the number of inputs. To overcome this complexity, we use a hierarchical organization similar to that proposed by [13]. As shown in Figure 3, YARC is organized as an 8×8 array of *tiles*. Each tile contains all of the logic and buffering associated with one input port and one output port. Each tile also contains an 8×8 switch and associated buffers. Each tile's switch accepts inputs from eight row buses that are driven by the input ports in its row, and drives separate output channels to the eight output ports in its column. Using a tile-based microarchitecture facilitates implementation, since each tile is identical and produces a very regular structure for replication and physical implementation in silicon.

The YARC microarchitecture is best understood by following a packet through the router. A packet arrives in the input buffer of a tile. When the packet reaches the head of the buffer a routing decision is made to select the output *column* for the packet. The packet is then driven onto the *row bus* associated with the input port and buffered in a row buffer at the input of the 8×8 switch at the junc-

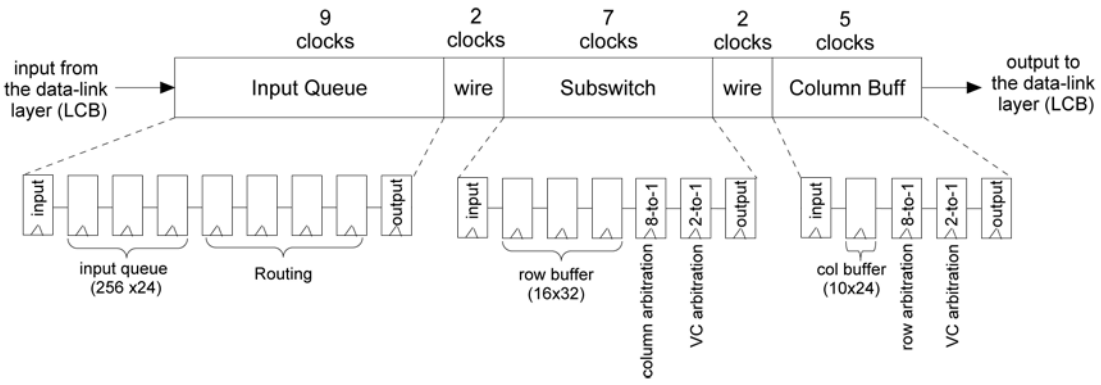


Figure 4. YARC pipeline diagram shows the tile divided into three blocks: input queue, subswitch, and column buffers.

tion of the packet's input row and output column. At this point the routing decision must be refined to select a particular output port within the output column. The switch then routes the packet to the column channel associated with the selected output port. The column channel delivers the packet to an output buffer (associated with the input row) at the output port multiplexer. Packets in the per-input-row output buffers arbitrate for access to the output port and, when granted access, are switched onto the output port via the multiplexer.

There are three sets of buffers in YARC: input buffers, row buffers, and column buffers. Each buffer is partitioned into two virtual channels. One input buffer and 8 row buffers are associated with each input port. Thus, no arbitration is needed to allocate these buffers — only flow control. Eight column buffers are associated with each subswitch. Allocation of these column buffers takes place at the same time the packet is switched.

Like the design of [13], output arbitration is performed in two stages. The first stage of arbitration is done to gain access to the output of the subswitch. A packet then competes with packets from other tiles in the same column in the second stage of arbitration for access to the output port. Unlike the hierarchical crossbar in [13], the YARC router takes advantage of the abundant on-chip wiring resources to run separate channels from each output of each subswitch to the corresponding output port. This organization places the column buffers in the output tiles rather than at the output of the subswitches. Co-locating the eight column buffers associated with a given output in a single tile simplifies global output arbitration. With column buffers at the outputs of the subswitch, the requests/grants to/from the global arbiters would need to be pipelined to account for wire delay which would complicate the arbitration logic.

As shown in Figure 4, a packet traversing the YARC router passes through 25 pipeline stages which results in a zero-load latency of 31.25ns. To simplify implementation of YARC, each major block: input queue, subswitch, and column buffers, was designed with both input and output registers. This approach simplified system timing at the expense of latency. During the design, additional pipeline stages were inserted to pipeline the wire delay associated with the row busses and the column channels.

4 Communication Stack

This section describes the three layers of the communication stack: network layer, data-link layer, and physical layer. We discuss the packet format, flow control across the network links, the link control block (LCB) which implements the data-link layer, and the serializer/deserializer (SerDes) at the physical layer.

4.1 Packet Format

The format of a packet within the BlackWidow network is shown in Figure 5. Packets are divided into 24-bit phits for transmission over internal YARC datapaths. These phits are further serialized for transmission over 3-bit wide network channels. A minimum packet contains 4 phits carrying 32 payload bits. Longer packets are constructed by inserting additional payload phits (like the third phit in the figure) before the tail phit. Two-bits of each phit, as well as all of the tail phit are used by the data-link layer.

The head phit of the packet controls routing which will be described in detail in Section 5. In addition to specifying the destination, this phit contains a *v* bit that specifies which virtual channel to use, and three bits, *h*, *a*, and *r*, that control routing. If the *r* bit is set, the packet will employ source routing. In this case, the packet header will be accompanied by a routing vector that indicates the path through the network as a list of ports to select the output port at each hop. Source routed packets are used only for maintenance operations such as reading and writing configuration registers on the YARC. If the *a* bit is set, the packet will route adaptively, otherwise it will route deterministically. If the *h* bit is set, the deterministic routing algorithm employs the hash bits in the second phit to select the output port.

4.2 Network Layer Flow Control

The allocation unit for flow control is a 24-bit phit — thus, the phit is really the flit (flow control unit). The BlackWidow network uses two virtual channels (VCs) [7], designated request (v=0) and response (v=1) to avoid request-response deadlocks in the network. Therefore, all buffer resources are allocated according to the virtual channel bit in the head phit. Each input buffer is 256 phits and is sized to cover the round-trip latency across the network channel. *Virtual cut-through* flow control [12] is used across the network links.

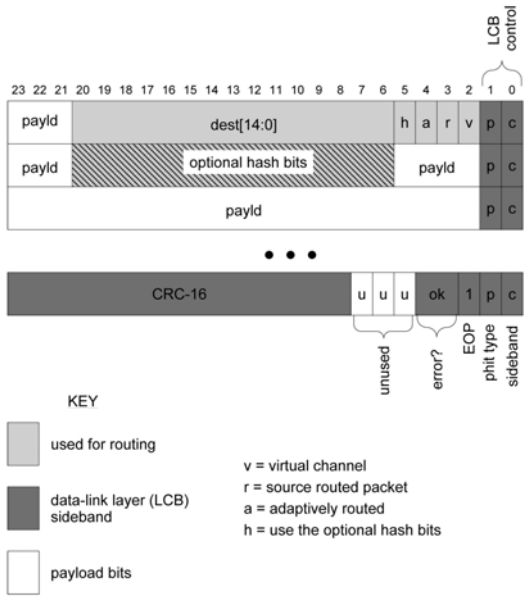


Figure 5. Packet format of the BlackWidow network.

4.3 Data-link Layer Protocol

The YARC data-link layer protocol is implemented by the link control block (LCB). The LCB receives phits from the router core and injects them into the serializer logic where they are transmitted over the physical medium. The primary function of the LCB is to reliably transmit packets over the network links using a sliding window go-back-N protocol. The send buffer storage and retry is on a packet granularity.

The 24-bit phit uses 2-bits of sideband dedicated as a control channel for the LCB to carry sequence numbers and status information. The virtual channel acknowledgment status bits travel in the LCB sideband. These VC acks are used to increment the per-vc credit counters in the output port logic. The *ok* field in the EOP phit indicates if the packet is healthy, encountered a transmission error on the current link (*transmit_error*), or was corrupted prior to transmission (*soft_error*). The YARC internal datapath uses the CRC to detect soft errors in the pipeline data paths and static memories used for storage. Before transmitting a tail phit onto the network link, the LCB will check the current CRC against the packet contents to determine if a soft error has corrupted the packet. If the packet is corrupted, it is marked as *soft_error*, and a good CRC is generated so that it is not detected by the receiver as a transmission error. The packet will continue to flow through the network marked as a bad packet with a soft error and eventually be discarded by the network interface at the destination processor.

The narrow links of a high-radix router cause a higher serialization latency to squeeze the packet over a link. For example, a 32B cache-line write results in a packet with 19 phits (6 header, 12 data, and 1 EOP). Consequently, the LCB passes phits up to the higher-level logic *speculatively*, prior to verifying the packet CRC, which avoids store-and-forward serialization latency at each hop. However, this early forwarding complicates various error conditions in order to correctly handle a packet with a transmission error and reclaim the space in the input queue at the receiver.

Because a packet with a transmission error is speculatively passed up to the router core and may have already flowed to the next router by the time the tail phit is processed, the LCB and input queue must prevent corrupting the router state. The LCB detects packet CRC errors and marks the packet as *transmit_error* with a corrected CRC before handing the end-of-packet (EOP) phit up to the router core. The LCB also monitors the packet length of the received data stream and *clips* any packets that exceed the *maximum packet length*, which is programmed into an LCB configuration register. When a packet is clipped, an EOP phit is appended to the truncated packet and it is marked as *transmit_error*. On either error, the LCB will enter error recovery mode and await the retransmission.

The input queue in the router must protect from overflow. If it receives more phits than can be stored, the input queue logic will adjust the tail pointer to excise the bad packet and discard further phits from the LCB until the EOP phit is received. If a packet marked *transmit_error* is received at the input buffer, we want to drop the packet and avoid sending any virtual channel acknowledgments. The sender will eventually timeout and retransmit the packet. If the bad packet has not yet flowed out of the input buffer, it can simply be removed by setting the tail pointer of the queue to the tail of the previous packet. Otherwise, if the packet has flowed out of the input buffer, we let the packet go and decrement the number of virtual channel acknowledgments to send by the size of the bad packet. The transmit-side router core does not need to know anything about recovering from bad packets. All effects of the error are contained within the LCB and YARC input queueing logic.

4.4 Serializer/Deserializer

The serializer/deserializer (SerDes) implements the *physical* layer of the communication stack. YARC instantiates a high-speed SerDes in which each *lane* consists of two complimentary signals making a balanced differential pair.

The SerDes is organized as a *macro* which replicates multiple lanes. For full duplex operation, we must instantiate the 8-lane receiver as well as an 8-lane transmitter macro. YARC instantiates 48 8-lane SerDes macros, 24 8-lane transmit and 24 8-lane receive macros, consuming $\approx$91.32 mm^2 of the 289 mm^2 die area, which is almost 1/3 of the available silicon (Figure 6).

The SerDes supports two full-speed data rates: 5 Gbps or 6.25 Gbps. Each SerDes macro is capable of supporting full, half, and quarter data rates using clock dividers in the PLL module. This allows the following supported data rates: 6.25, 5.0, 3.125, 2.5, 1.5625, and 1.25 Gbps. We expect to be able to drive a 6 meter, 26 gauge cable at the full data rate of 6.25 Gbps, allowing for adequate PCB foil at both ends.

Each port on YARC is three bits wide, for a total of 384 low voltage differential signals coming off each router, 192 transmit and 192 receive. Since the SerDes macro is 8 lanes wide and each YARC port is only 3 lanes wide, a naive assignment of tiles to SerDes would have 2 and 2/3 ports (8 lanes) for each SerDes macro. Consequently, we must aggregate three SerDes macros (24 lanes) to share across eight YARC tiles (also 24 lanes). This grouping of eight tiles is called an *octant* (tiles belonging to the same octant are shown in Figure 6a) and imposes the constraint that each *octant* must operate at the same data rate.

The SerDes has a 16/20 bit parallel interface which is managed by the link control block (LCB). The *positive* and *negative* components of each differential signal pair can be arbitrarily swapped between the transmit/receive pair. In addition, each of the 3 lanes which comprise the LCB port can be permuted or "swizzled." The LCB determines which are the positive and negative differential pairs during channel initialization, as well as which lanes are "swizzled". This degree of freedom simplifies the board-level river routing of the channels and reduces the number of metal layers on a PCB for the router module.

5 Routing

Routing in the BlackWidow network is performed on variable length packets. The first phit of a packet is the *header*, which contains all the mandatory routing fields, and the last phit of a packet is an *end of packet* (EOP) phit which contains the packet checksum.

In a folded-Clos topology, packet routing is performed in two stages: routing up to a *common ancestor* of the source and destination processors, and then routing down to the destination processor. Up routing can use either adaptive or deterministic routing. Downrouting, however, is always deterministic, as there is only a single path down the tree from any router to a destination processor. The BlackWidow memory consistency model requires that requests to the same address maintain ordering in the network. Therefore, request packets always use deterministic routing. Response packets do not require ordering, and so are routed adaptively.

Packet routing is algorithmic and distributed. At each hop in the network, routing logic at the head of the input queue calculates the output port for the local router. This is performed using routing registers and an eight-entry routing table. The routing logic is replicated in each tile, allowing multiple *virtual routers* (see Section 7.4) per physical router and providing the needed bandwidth for parallel routing in all 64 tiles.

There are three types of links (routes):

uplinks from the injection port to a rank 1 router or a rank n router to a rank $n+1$ router,

sidelinks from a rank n router to a peer rank n router (only for R1.5, R2.5 and R3.5 networks), and

downlinks from a rank n router to a rank $n-1$ router or from a rank 1 router to the destination processor.

En route from the source to the common ancestor, the packet will take either an uplink or a sidelink depending on the class of the network (e.g.: rank 2 or rank 2.5, respectively). Upon arrival at the common ancestor, the router will begin routing the packet down the fat tree toward its final destination using the downlinks. The down route is accomplished by extracting a logical port number directly from the destination processor number. Each YARC router chip in the BlackWidow network has 64 ports which have both a physical number, and an arbitrary *logical* number. System software will perform network discovery when the system is initialized and assign a logical port number to each physical port number.

5.1 Up Routing

Each tile has a *root detect* configuration register that identifies the *subtree* rooted at this router, using a 15-bit router location and a 15-bit mask. As an example, the root detect register of a rank-1 router connected to destinations 96-127 would have a router location of 0x0060 (96), and a mask of 0x001F (covering 32 destinations). If the unmasked bits of the packet destination and the router location match, then the destination processor is contained within the router's subtree, and the packet can begin traversing downward. Otherwise the packet must continue to route up (or over if sidelinks are used).

Routing *up* or *over* is accomplished using an eight-entry table, where each entry contains a location and mask bits (like the root detect register) identifying a subtree of the network. The packet destination is associatively checked against the routing table entries. The packet matches an entry if its destination is contained within the subtree identified by that entry. The matching entry then provides the set of uplinks/sidelinks that the packet may use to reach its destination.

In a healthy network, only a single entry is required for up routing, matching the *entire network*, and identifying the full set of available uplinks. In a system with faults, additional routing table entries are used to provide alternative uplinks for affected regions of the machine. If multiple entries match, then the entry with the highest index is chosen. Thus, entry 0 could be set to match the entire network, with a full uplink mask, and entry 1 could be set to match the subtree rooted at the fault, using a constrained uplink mask that avoids sending packets to a router that would encounter the fault en route to any destination processors in that subtree.

A given network fault casts a shadow over some subtree of endpoints that can be reached going down from the fault. We only need fault entries in the routing table for faults that do *not* cast a shadow over the local router. A router can also ignore a fault if it cannot be reached from this router (such as faults in another network slice).

In a router with configured sidelinks, each peer subtree is given its own routing table entry, which defines the set of sidelinks usable to route to that subtree. No additional routing entries are required for faults.

Packets in the BlackWidow network may adaptively route on a per-packet basis. Each packet header (Figure 5) has an adapt (*a*) bit that chooses the routing policy. If *a*=1 then the packet will choose the output port adaptively during up or siderouting. The routing table of the input port produces a 64-bit mask of allowable ports. The *column mask* is formed by OR-ing together the eligible ports within each column — the resultant 8-bit mask will have bit i set if any of the eight output ports of column i are set in the output port mask produced by the routing table. After constructing the set of allowable columns, we choose the winner (the eventual output column) based on the amount of space available in the row buffer for each column. Ties are broken fairly using a matrix arbiter [9]. When the packet is sent across the row bus to the chosen column it is accompanied by an 8-bit mask corresponding to the allowable output rows within that column. This row mask is used by the 8x8 subswitch to select an exit row. The row selection at the subswitch is guided by the space available in the column buffers at the outputs, the row with the most space available in the column buffers is chosen.

Packets that are not marked as adaptive (*a*=0) are routed deterministically based on the output of a hash function. To uniformly spread the packets across the available uplinks, the hash function does an XOR of the *input port*, *destination processor*, and *optional hash bits* if the hash bit (*h*) is set in the packet header. The hash value is then mapped onto the set of output links identified by the routing table. The input port and destination processor are hashed on to avoid non-uniformities in many-to-one traffic patterns. For request packets, the hash bit is set, and a portion of the packet's address is included in the hash function to further spread the traffic across the uplinks. In this way, we can load balance and still guarantee in-order delivery of packets from source to destination targeting a given address.

5.2 Down Routing

Once the packet reaches the common ancestor it will begin routing down the subtree. The first step in routing down is to select a logical downlink number. The *down route* configuration register contains shift (*s*) and mask (*m*) values that are used by first right-shifting the destination processor number by *s* bits and then masking the bottom *m* bits to produce the *logical* output port number for the downlink. A rank 1 router, for example, would have *s*=0 and *m*=00011111. The logical port number is converted to a physical port number by a 64-entry port mapping table. The packet proceeds down the tree, shifting and masking the bits of destination processor to determine the downlink at each level, until it reaches the final egress port where it is sent to the processor's network interface.

6 ASIC Implementation

This section discusses the various implementation trade-offs and challenges that were encountered during the development of YARC. The design was implemented using standard-cell ASIC technology from Texas Instruments. The ASIC development spanned 17 months and involved a team of 11 developers: architecture (2), RTL logic design (4), verification (2), RTL netlist processing (1), electrical and mechanical packaging (2). The 17 month development pace was subdivided as:

- architecture and documentation ≈8%
- RTL logic design and verification ≈58%
- timing closure and netlist preparation ≈12%
- physical design and back-end netlist processing ≈22%

YARC is implemented in a 90nm CMOS standard-cell ASIC process operating at 800 MHz. The silicon area is 289 mm^2 (17 mm on an edge) using the Texas Instruments SR50LX (low power) standard-cell ASIC library and a small number of cells from the SR50 [23] cell library for relay latches on long global wires. The SR50 ASIC technology from Texas Instruments has a maximum gate density of 255k gates per mm^2 (at 100% utilization) with a 330nm minimum metal pitch and 7 layers of copper (with low-K dielectric) interconnect.

A summary of the logic gates and RAM cells used is shown in Table 1. YARC has approximately 28.2M cells, which does not include the SerDes macros, or test circuitry. Register cells make up about half of the overall area, at 12.98M cells, and SRAM memory only 3.92M cells. Each SRAM macro requires a BIST (built-in self test) collar and test circuitry. Thus, if a memory was relatively small, less than 1Kbit, it was implemented as a group of latches, rather than instantiated as a small SRAM macro with the relatively large overhead for memory test circuits.

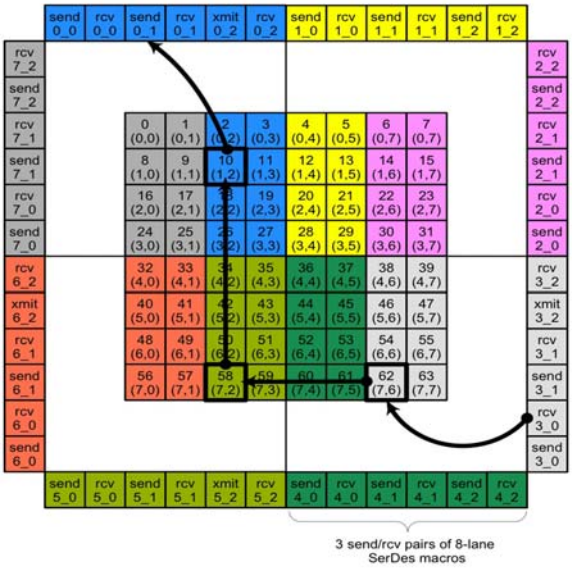
Table 1. YARC logic gate and SRAM usage.

Block Name	Number of Instances	Logic cells	Register cells	SRAM cells
Tile	64	95,777	175,560	61,185
Config Registers	1	49,088	5,170	0
Logic Monitor	1	103,390	11,426	0
Startup	1	414	64	0
Clock Generator	1	1,112	24	0
Row control	8	6,145	988	0
Column control	8	4,860	810	0
Config control	1	5,600	896	0
TOTAL	-	11.32M	12.98M	3.92M

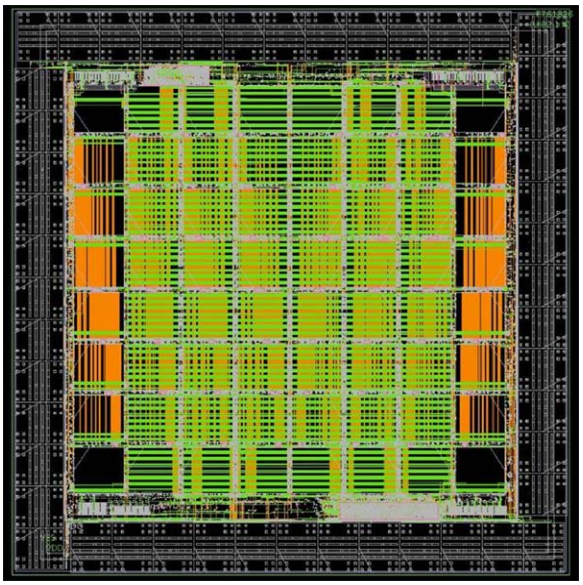
The tile is broken into four blocks: the link control block (LCB), input buffers, 8×8 subswitch, and column buffers. The input buffer block contains 122k cells (46% registers, 35% logic, and 19% SRAM) which includes the routing table and routing logic. A considerable amount of this logic is dedicated to handling speculative data forwarding — the LCB passing data up from the data-link layer prior to verifying the CRC — to handle error cases due to transmission errors and soft errors. The 8×8 subswitch accounts for 141k cells (54% registers, 25% logic, and 21% SRAM), or approximately 1/3 of the logic in the tile. The subswitch contains the row buffers and logic that performs the 8-to-1 arbitration among the row buffers, and a 2-to-1 arbitration amongst the virtual channels. The column buffer block which also performs the same two-stage arbitration as the subswitch only accounts for 62k cells (71% registers, and 29% logic). The column buffers are implemented in latches, not SRAMs, so the bulk of the area in the column buffers is dedicated to latches. The remaining 111k cells, or 25% of the tile area, is consumed by the LCB.

The estimated power consumption when the chip is idle is 80W, with an expected 87W peak dynamic power dissipation. This relatively modest power dissipation allows conventional air cooling to be used for the router cabinets. Figure 6 shows the ASIC chip floorplan with tile (0,0) corresponding to port 0 in the upper-left corner of the diagram. The SerDes macros occupy the perimeter of the chip where they are physically close to the I/O pads. This allows 12 SerDes macros (six 8-lane transmit and six 8-lane receive) to be abutted around each edge of the die (Figure 6) for a total of 48 8-lane macros, or 384 serial lanes.

The network channels have a physical layer operating at 6.25 Gbaud providing a plesiochronous interface between the physical layers on each router chip. Within the chip, there are several clock domains. The core logic in the YARC tiles operates at the 800 MHz local clock (Lclk) frequency. The SerDes has a 311 MHz external input reference clock (REFCLK) and can operate in 16b20b mode or 16-bit mode. The data clock (Dclk) rate depends on the encoding – if 16b20b is used then the Dclk rate is 1/20th of the 6.25 GHz rate, or 311MHz. Otherwise, when configured for 16-bit mode with frame-synchronous scrambling (FSS) used to balance the transition density on the SerDes serial lanes, the Dclk rate is 1/16th the 6.25 GHz rate, or 390MHz.



(a) YARC logical layout.



(b) YARC floorplan.

Figure 6. (a) The logical layout of YARC showing an example of packet traversal through the router, and (b) the top-level ASIC floorplan of its implementation.

7 Discussion

This section provides additional discussion around the key design points. We explore what the *optimal* radix would be given the ASIC and packaging constraints. Then, we discuss why a folded-Clos topology was chosen. Then, we turn to more detailed design decisions of subswitch degree, fault tolerance, and network load balancing.

7.1 Optimal Radix

The radix at which a network has minimum latency is largely determined by the *aspect ratio* [13] of the network router. Aspect ratio is given by: $A = \frac{Bt_r \log N}{L}$ where B is the total bandwidth of a router, t_r is the per router delay, N is the size of the network, and L is the length of a packet. Using the parameters of the Black Widow Network,[3] the aspect ratio is 1600, which gives an optimal radix of 82.

While the optimal radix is 82, this is not a practical value. To simplify implementation and routing, the radix must be a power of 2. A radix that is not a power of 2 would require an integer division and modulo operation to determine the output port from a destination address. In the design of the BW router we considered radices of 64, and 128. Both of these values give network latency within 2% of the optimal value. Although the higher radix of 128 theoretically leads to lower cost [13], this theory assumes that port widths can be varied continuously. We selected a radix of 64 because it gives better performance with our pinout and integral port-width constraints. Area constraints limited us to no more than 200 SerDes on the router chip. A radix-64 router using 3-bit wide ports requires 192 SerDes, fitting nicely within this constraint. A radix-128 router, on the other hand, is limited to 1-bit wide ports requiring 128 SerDes. Such a router has only 2/3 the bandwidth of the radix 64 router, resulting in significantly lower performance.

[3]The parameters for the BlackWidow Network are B=2.4Tb/s, t_r=20ns, N=32K nodes, and L=312. The L value is averaged over the different type of packets in the network, including read and write request and response packets, and the t_r and B are estimated values prior to the actual implementation, based on the technology.

7.2 Long Links

BlackWidow systems with over 4K processors have cabinet-to-cabinet spacing that requires network links longer than six meters, the maximum length that can be driven reliably at full signaling rate (6.25Gb/s). Such long links can be realized using optical signaling or using electrical cables with in-line repeaters. However, both of these alternatives carry a significant cost premium.

YARC's support for variable signaling rate (see Section 4) and flexible routing enable these long links to be realized using electrical signaling over passive cables by using a reverse taper. By reducing the signaling rate on the link, significantly longer electrical cables can be driven. The reduced signaling rate can be offset for by doubling the number of links provisioned at that level of the network (a reverse taper) to preserve network bandwidth.

7.3 High-Radix Clos vs. Torus

We chose a high-radix folded-Clos topology for the BlackWidow network because it offered both lower latency and lower cost than alternatives such as a torus network while still providing 8.33 GB/s of global memory bandwidth. Table 2 compares the

zero-load latency of the two topologies, a folded-Clos and a 3-D torus, as we vary the size of the network. In the comparison, we assume both topologies were implemented based on the technology available for the BlackWidow network. We assume uniform random traffic to calculate the average hop count[4]. The port bandwidth of the routers are held constant in the comparison. Thus, with a torus network, the channels will be fatter, reducing the serialization latency but the header latency is increased because of the larger hop count.

NetworkSize	576	2304	9216	30720
Torus	0.144	0.210	0.326	0.484
Clos	0.078	0.091	0.104	0.116

Table 2. Zero-load latency (μs) comparison of two different topologies. For the high-radix Clos network, radix-64 routers were used. For the 3-D torus, the configurations used were identical to those of the Cray XT3. Uniform random traffic was assumed in calculating the average hop count of the network.

As shown in Table 2, for a small size network, there is a 2x reduction in latency but as the size of the network increases, there is an over 4x reduction in latency. With the lower hop count, the high-radix Clos not only reduces latency but also reduces cost. The network cost is approximately proportional to the total router bandwidth and with the network bisection held constant, it is proportional to the hop count. Thus, high-radix Clos network leads to a lower latency and a lower cost network.

There are also several *qualitative* attributes of the high-radix folded-Clos network which made it an attractive choice. Routing in torus is more complex as turn rules [10] or virtual channels [8] are needed to prevent deadlocks and complex routing algorithms are needed to properly load balance across adversarial traffic pattern [21]. Compared to a torus, the folded-Clos has very a straightforward routing algorithm. Because of the path diversity in the topology, load balancing is achieved by selecting any one of the common ancestors. The folded Clos is also cycle-free by design so no additional virtual channels are needed to break deadlock. VC allocation is often the critical path in the router implementation [9] and with fewer VCs, the VC allocation is also simplified.

7.4 Virtual routers

The radix-64 YARC router can be divided into multiple *virtual* routers with lower degree. For instance, a single YARC can serve as two radix-32, four radix-16, or ten radix-6 virtual routers. Since each tile has its own set of routing tables and keeps track of the set of allowable exit ports, system software can partition the router into multiple virtual routers by programming the routing tables associated with each virtual router with a set of masks that restricts output traffic to the ports of that virtual router. This flexibility enables a YARC router to be used in systems where packaging constraints require multiple lower radix routers. Virtual routers can also be used to support multiple network slices in a single YARC chip. For example, a single YARC chip can be configured as two radix-32 routers to provide a radix-32 first stage switch for two of the four BW network slices as shown in Figure 2(c).

[4]If worst-case traffic pattern is used, the latency for folded-Clos will not change significantly but latency for a 3-D torus will increase significantly.

7.5 Flow Control Within YARC

YARC employs virtual cut-through flow control externally but uses wormhole flow-control internally due to buffer size constraints. The 64 input buffers are each sized deep enough (256 phits) to account for a round-trip credit latency plus the length of a maximum-length packet (19 phits). This enables us to perform virtual cut-through flow control (with packet granularity) on external links as discussed in Section 4.2. It was infeasible, however, to size the 512 row buffers or 512 column buffers large enough to account for credit latency plus maximum packet size. Thus using wormhole flow control[6] (at flit=phit granularity) is performed over both the row buses and the column channels to manage these buffers. The row buffers are 16 phits deep and the column buffers are 10 phits deep — large enough to cover the credit latency over the global column lines. Here a maximum-length packet can block traffic from the same input row to other outputs in the same column (by leaving its tail in the row buffer).

7.6 Degree of the Subswitch

In a hierarchical high-radix router, a radix-k router is composed of $(k/p)^2$ $p \times p$ subswitches. The cost and performance of the router depend on p. As p is reduced, the design approaches that of a fully buffered crossbar and becomes prohibitively expensive but provides higher performance [13]. As p is increased, the design approaches an input-buffered crossbar and is inexpensive but has poor performance.

To stress the hierarchical organization, we apply worst-case traffic to the router in which *all* of the offered traffic "turns the corner" at a single subswitch. With this approach, with an offered load of λ, one subswitch in each row sees λp packets per cycle while the other subswitches in the row are idle. In contrast, uniform random (UR) traffic does not stress the hierarchical organization because it evenly distributes traffic across the $\frac{k}{p}$ subswitches in a row with each subswitch seeing only $\frac{\lambda p}{k}$ packets per cycle.

We wrote a simulator to evaluate the performance on worst-case traffic for subswitches with degree p of 2, 4, 8, 16, and 32. Table 3 summarizes the sustained throughput on worst-case traffic as a function of subswitch degree p. The 8, 16, and 32-input subswitches perform almost identically with a throughput of about 60% [11]. Since a $p \times p$ subswitch provides an internal speedup of $\frac{k}{p}$, (8, 4 and 2 respectively for p=8, 16 and 32), a sustained throughput of 60% provides more than sufficient performance for uniform traffic. With the 8×8 subswitch used in YARC, we can sustain approximately five times the average traffic demand through our subswitch on uniform traffic, providing plenty of headroom for non-uniform traffic patterns.

Although 8, 16, or 32 input subswitches provide nearly identical performance, higher degree subswitches give lower cost because the buffering required is $O(\frac{k^2}{p})$. However, we chose the more expensive $p = 8$ configuration for YARC for two reasons. First, the higher-degree subswitches required too much time to perform the p-to-1 switch arbitration which is a timing critical

Subswitch Configuration	No. of Subswitches Needed for Radix-64	Sustained Throughput	Buffer Area
2x2	1024	75.0%	4096
4x4	256	65.6%	2048
8x8	64	61.8%	1024
16x16	16	60.1%	512
32x32	4	59.3%	256

Table 3. Evaluation of subswitch configurations.

path in the implementation. Early results showed that an 8-to-1 arbitration can be done within a single 800 MHz clock cycle. A 16- or 32-to-1 arbitration would require a longer clock cycle or a pipelined arbiter. Second, a subswitch of size $p = 8$ resulted in a modular design in which the number of ports was equal to the number of subswitches. This enabled us to build a tile that contained a single subswitch, a single input, and a single output. A higher subswitch size would require each tile to have multiple inputs/outputs, while a smaller subswitch size would require several subswitches to share an input/output complicating the design effort of the tiles.

7.7 Fault Tolerance

The high path diversity of a high-radix folded-Clos network can be exploited to provide a degree of fault tolerance. The YARC chip is designed to construct a network that provides graceful degradation in the presence of the following faults:

- a failed network cable or connector,
- a faulty router (a YARC that stops responding), and
- a noisy high-speed serial lane that is causing excessive retries.

In a fault-free network, only a single entry in the routing table is necessary to specify the uplinks for the entire system. However, higher-priority table entries can be used to override this master entry to restrict routing to a set of destinations. If a fault occurs at a particular node of the network, the routing tables can be set so that traffic with destinations in the subtree beneath the fault do not route to the fault or any ancestors of the fault. This is done by creating an entry that matches this set of destinations that has an uplink *mask* with the bits corresponding to the faulty node and/or its ancestors cleared.

The sender-side of each YARC port maintains a forward progress countdown timer for each virtual channel. If the forward progress timer expires, it indicates that a packet has not flowed in a long time and the router must prevent the error from propagating throughout the network. A forward progress timeout may happen if the attached BlackWidow processor stops accepting requests causing the network to back pressure into the routers. Upon detection of a forward progress timeout, an interrupt is raised to the maintenance controller to inform the system software that a node has stopped responding. The router will begin discarding packets that are destined to port which incurred the timeout.

The link control block (LCB) handles the data-link layer of the communication stack. It provides reliable packet delivery across each network link using a sliding window go-back-N protocol. It manages the interface between the higher-level core logic and the lower-level SerDes interface (physical layer). The LCB counts the number of retries on a per-lane basis as a figure of merit for that serial channel. System software defines a threshold for the number of tolerable retries for any of the serial lanes within the 3-lane port. If the LCB detects that the retry count exceeded the threshold, it will automatically decommission the noisy lane and operate in a degraded (2-bit wide or 1-bit wide) mode until the cable can be checked and possibly replaced. This allows the application to make forward progress in the presence of persistent retries on a given network link.

If all the lanes in the link are inoperable and must be disabled, the LCB will deassert the *link_active* signal to the higher-level logic which will cause a system-level interrupt to the maintenance controller and cause the sending port to discard any packets destined to the dead port. This prevents the single link failure from *cascading* into the rest of the network.

A folded-Clos topology is cycle free and under normal operating conditions is deadlock-free. The router ensures the following invariant: *once a packet begins traversing downward, it remains going downward until it reaches the destination*. That is, packets that arrived from an *uplink* must route to a *downlink*. This prevents packets from being caught in a cycle containing uplinks and downlinks. If the router is configured properly, this should never happen. However, software and the programmers who create it are fallible. This dynamic invariant should help reduce the debugging time when investigating early-production routing software.

7.8 Network Load Balancing

Non-uniform traffic can cause local hot spots that significantly increase contention in interconnection networks. To reduce this network load imbalance, the BlackWidow network performs two types of load balancing: hashing of deterministic routes to split bulk transfers up over multiple paths; and adaptive routing. Section 7.8.1 analyzes the ability of hashing to reduce network hotspots. Section 7.8.2 discusses the technique used to ensure diversity in the hash function. Section 7.8.3 discusses how adaptive routing is implemented over 64 ports.

7.8.1 Hot Spot Contention

To assess the potential contention from non-uniform traffic, we analyzed random permutations among processors, in which each processor is sending a block of data to one other random processor and receiving data from one other processor. We looked at both torus and radix-64 fat-tree networks, calculating the average worst-case link utilization for 10,000 random permutations. For the torus, we assumed minimal, dimension order routing. For the fat tree, we analyzed the effect of *path diversity*, varying the number of unique network paths taken by a single transfer from 1 up to 32.

Table 4 shows the results, for networks up to 32K endpoints. Link traffic is shown relative to injection port bandwidth. For the torus, both the average link traffic and the average worst case link traffic are shown. Average link traffic for global traffic in a bidirectional, radix-k torus is simply $k/8$. Thus, contention starts to degrade global bandwidth per endpoint for radices larger than 8. We see that the average worst-case link traffic is an *additional* 4-5 times higher (the "degrade factor" shown in the table).

Table 4. Simulation results of worst-case link utilization for 10,000 random permutations.

	Torus				Radix 64 Fat Tree						
Num Endpoints	Radix	Avg Cont.	Avg WC Cont.	Degrade Factor	Rank	Average Worst Case Contention					
						np=1	np=2	np=4	np=8	np=16	np=32
64	4	0.5	2.46	4.92	1	1.00	1.00	1.00	1.00	1.00	1.00
216	6	0.75	3.92	5.23	2	4.50	2.97	2.07	1.53	1.18	1.00
512	8	1	5.02	5.02	2	5.18	3.45	2.40	1.76	1.34	1.00
1728	12	1.5	7.02	4.68	2	5.88	3.89	2.70	1.95	1.46	1.00
4096	16	2	8.75	4.38	3	6.42	4.22	2.90	2.08	1.54	1.19
8000	20	2.5	10.33	4.13	3	6.84	4.50	3.09	2.23	1.70	1.39
13824	24	3	11.80	3.93	3	7.21	4.73	3.26	2.36	1.82	1.50
21952	28	3.5	13.19	3.77	3	7.44	4.89	3.38	2.46	1.91	1.56
32768	32	4	14.51	3.63	3	7.64	5.04	3.48	2.54	1.97	1.61

A conventional fat tree is also susceptible to hot spots. The average link traffic for global traffic in a fully configured fat tree is equal to the injection bandwidth independent of size (that is, global bandwidth per endpoint stays constant as system size is increased). However, the average worst-case link traffic under random permutations is up to 7.6 times higher than the injection bandwidth. As the number of paths used by a transfer is increased, the averge worst-case link traffic is reduced. Using a path diversity of 32 results in near uniform usage of all the links in the system, with an average worst case of only 1.6 times the injection bandwidth at 32K endpoints. The BlackWidow network is capable of using all uplinks for deterministic traffic at each level, permitting up to thousands of diverse paths for a single block transfer.

7.8.2 Diversity in the Hash Function

For deterministic routing, a packet's uplink or sidelink is chosen at each hop by computing a hash value and then performing a modulo over the number of configured ports. The BW router calculates exact modulos of 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20, 24, 28 or 32 ports, by factoring the divisor into a power-of-two component, and an optional factor of 3, 5 or 7. Division by 3, 5 or 7 is easy to compute via the binary equivalent of the "casting out nines" technique for verifying long multiplication.

The hash function is an XOR of the input port, destination number, and *optional hash bits* (included if the *h* bit is set in the packet header). The BlackWidow request packets map the address bits *address[20:6]* into the hash region of the packet header, providing high diversity across packets, yet preserving in-order delivery of request packets on a per-cacheline basis.

Only an 8-bit subset of the 15-bit optional hash bits are actually used by the hash function. Each tile has a configuration register which indicates the bits to use. By only using a subset of the optional hash bits, we can hash on different bits at different ranks within the network. For example, the rank 1 routers might hash using hash bits 0..7, rank 2 routers hash using bits 5..12, and rank 3 routers hash using bits 10..14. In this way, we prevent successive routers from hashing on the same bits as the packet moves up the tree. A router with n uplinks will tend to "use up" the least significant $log_2(n)$ hash bits it employs. A parent of that router will see an incoming stream of packets with little or no diversity in these bits, as they were used to select the parent. Therefore, the parent should use a different set of the optional hash bits in order to maximize the diversity in the hash function.

7.8.3 Adaptive Routing

Implementing an adaptive routing scheme in a high-radix router is particularly challenging because of the large number of ports involved in the adaptive decision. Ideally, we would look at the congestion at all possible output ports (at most 32) and choose the queue with the most free space. Unfortunately, this is unrealistic in a 1.25 ns clock cycle. Instead, in keeping with the hierarchical organization of the router, we break the adaptive decision into two stages: choosing the output column, and choosing the output row within that column.

We first choose the column, c, by comparing the congestion of the row buffers in each of the c row buffers identified by bits in the column mask. A full eight-way, four-bit comparison of row buffer depths was too expensive. Instead we look only at the most-significant bit of the row buffer depth, giving priority to buffers that are less than half full. We then select the column based on a round-robin arbitration, and route to the row buffers of the cross-point tile. This algorithm ignores the number of eligible output ports in each of the target columns, giving no preference to a column with more eligible outputs. However, columns with more eligible outputs will tend to drain faster, leading to more space in their subswitch row buffers.

In the second stage of the adaptive route, we choose the output row based on the bits of the row mask which are set. The row mask identifies the set of valid output ports within the chosen column. We again must rely on imperfect information to choose the output tile based on the depth of the column buffers in the r rows, where r is the number of bits set in the row mask. We choose among the rows by comparing two bits of the 4-bit column buffer depth (which is at most 10). The most-significant bit indicates if the column buffer is "almost full" (i.e. 8 or more phits in the buffer), and the upper two-bits together indicate if the column buffer has more than 4 phits but less than 8 phits — corresponding to "half full." Finally, if the upper two bits of the buffer size are zero, then the column buffer is "almost empty." The adaptive decision will choose the column buffer based on its current state, giving preference to those ports which are "almost empty" then those that are "half full" and finally those buffers that are "almost full."

8 Conclusion

YARC is a high-radix router used in the network of the Cray BlackWidow multiprocessor. Using YARC routers, each with 64 3-bit wide ports, the BlackWidow scales up to 32K processors us-

ing a folded-Clos topology with a worst-case diameter of seven hops. Each YARC router has an aggregate bandwidth of 2.4Tb/s and a 32K-processor BlackWidow system has a bisection bandwidth of 2.5Pb/s.

YARC uses a hierarchical organization[13] to overcome the quadratic scaling of conventional input-buffered routers. A two-level hierarchy is organized as an 8×8 array of tiles. This organization simplifies arbitration with a minimal loss in performance. The tiled organization also resulted in a modular design that could be implemented in a short period of time.

The architecture of YARC is strongly influenced by the constraints of modern ASIC technology. YARC takes advantage of abundant on-chip wiring to provide separate column buses from each subswitch to each output port, greatly simplifying output arbitration. To operate using limited on-chip buffering, YARC uses wormhole flow control internally while using virtual-cut-through flow control over external channels.

To reduce the cost and the latency of the network, BlackWidow uses a folded-Clos network that is modified by adding *sidelinks* that connect peer subtrees and statically partition the global network bandwidth. We showed the benefits of high-radix Clos, compared to the previous torus networks, in terms of fault tolerance, bandwidth spreading, and simpler routing algorithm. Both adaptive and deterministic routing algorithms are implemented in the network to provide load-balancing across the network and still maintain ordering on memory requests. Deterministic routing is performed using a robust hash function to obliviously balance load while maintaining ordering on a cache line basis.

As more networks take advantage of high-radix routers, many opportunities are open for further research on high-radix routers and networks. High-radix Clos network can exploit high-radix routers. However they are far from the lower bounds of achievable cost and latency. Research on alternate topologies may discover more efficient organizations. The large number of possible output ports on a high-radix router motivates the development of new routing algorithms. Of particular interest are algorithms that reduce the complexity of the routing decision - to avoid quadratic scaling of the routing function. Research on router microarchitecture and switch organization is also of interest. As wire delays become more critical, organizations that exploit locality, for example those based on a network-on-chip may provide reduced latency.

Acknowledgements

Many people contributed to the development of the YARC router and BlackWidow network. Foremost, we would like to thank Greg Hubbard, Roger Bethard and Kelly Marquardt for their countless whiteboard discussions. We would also like to thank the rest of the BW network team: Joe Kopnick, Jack Webber, Boone Severson, Greg Faanes, Brad Smith, Doug Carlson, and Paul Wildes. We also thank Amit Gupta for his helpful insights in the preliminary stages of this paper. Finally, we would like to thank three of the five referees for their detailed suggestions and comments.

References

[1] J. Beecroft, D. Addison, F. Petrini, and M. McLaren. Quadrics QsNet II: A Network for Supercomputing Applications. In *Hot Chips 15*, Stanford, CA, August 2003.

[2] C. Clos. A Study of Non-Blocking Switching Networks. *The Bell System technical Journal*, 32(2):406–424, March 1953.

[3] Cray X1. http://www.cray.com/products/x1/.

[4] Cray XD1. http://www.cray.com/products/xd1/.

[5] Cray XT3. http://www.cray.com/products/xt3/.

[6] W. J. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[7] W. J. Dally. Virtual-channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, 1992.

[8] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.

[9] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Francisco, CA, 2004.

[10] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *ISCA '92: Proceedings of the 19th annual international symposium on Computer architecture*, pages 278–287, 1992.

[11] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus Output Queueing on a Space-division Packet Switch. *IEEE Transactions on Communications*, COM-35(12):1347–1356, 1987.

[12] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[13] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta. Microarchitecture of a high-radix router. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 420–431, Madison, WI, USA, 2005. IEEE Computer Society.

[14] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proc. of the 24th Annual Int'l Symp. on Computer Architecture*, pages 241–251, 1997.

[15] C. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE Transactions on Computer*, C-34(10):892–901, October 1985.

[16] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong-Chan, S.-W. Yang, and R. Zak. The Network Architecture of the Connection Machine CM-5. *J. Parallel Distrib. Comput.*, 33(2):145–158, 1996.

[17] Mellanox. http://www.mellanox.com.

[18] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 network architecture. In *Hot Chips 9*, pages 113–117, Stanford, CA, August 2001.

[19] Myrinet. http://www.myricom.com/myrinet/overview/.

[20] S. Scott and G. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Hot Interconnects 4*, Stanford, CA, Aug. 1996.

[21] A. Singh. *Load-Balanced Routing in Interconnection Networks*. PhD thesis, Stanford University, 2005.

[22] C. B. Stunkel, D. G. Shea, B. Aball, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker. The SP2 High-performance Switch. *IBM Syst. J.*, 34(2):185–204, 1995.

[23] Texas Instruments - SR50. http://www.ti.com/.