Unifying on-chip and inter-node switching within the Anton 2 network

Brian Towles,[†] J.P. Grossman, Brian Greskamp, and David E. Shaw[*,†]
D. E. Shaw Research, New York, NY 10036, USA

Abstract

The design of network architectures has become increasingly complex as the chips connected by inter-node networks have emerged as distributed systems in their own right, complete with their own on-chip networks. In Anton 2, a massively parallel special-purpose supercomputer for molecular dynamics simulations, we managed this complexity by reusing the on-chip network as a switch for inter-node traffic. This unified network approach introduces several design challenges. Maintaining fairness within the inter-node network is difficult, as each hop becomes a sequence of many on-chip routing decisions. We addressed this problem with an inverse-weighted arbiter that ensures fairness with low implementation costs. Balancing the load of inter-node traffic across the on-chip network is also critical, and we adopted an optimization approach to design an appropriate routing algorithm. Finally, the on-chip routers carry inter-node traffic, so they must implement inter-node virtual channels to avoid deadlock. In order to keep the routers small and fast, we developed a deadlock-free routing algorithm that reduces the number of virtual channels by one-third relative to previous approaches. The resulting Anton 2 network implementation efficiently utilizes its inter-node channels and provides low messaging latency, while occupying a modest amount of silicon area.

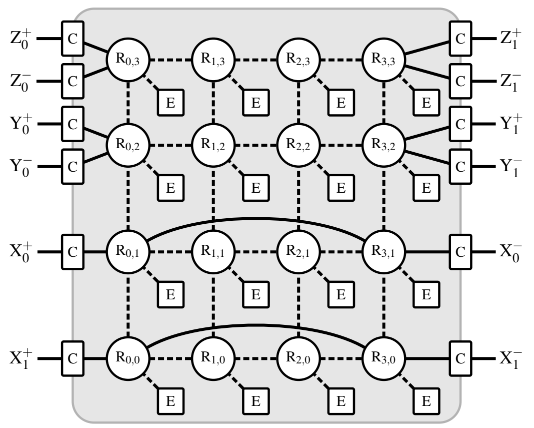

Z_0^+
Z_0^-
Y_0^+
Y_0^-
X_0^+
X_1^+
Z_1^+
Z_1^-
Y_1^+
Y_1^-
X_0^-
X_1^-
C
E
$R_{0,3}$ $R_{1,3}$ $R_{2,3}$ $R_{3,3}$
$R_{0,2}$ $R_{1,2}$ $R_{2,2}$ $R_{3,2}$
$R_{0,1}$ $R_{1,1}$ $R_{2,1}$ $R_{3,1}$
$R_{0,0}$ $R_{1,0}$ $R_{2,0}$ $R_{3,0}$

Figure 1: The on-chip topology of the Anton 2 network. Routers are annotated with their coordinates (U,V) in the two-dimensional mesh ($R_{U,V}$). Network adapters at endpoints (E) provide network connectivity to compute resources, and adapters at the edges of the network provide connections to external network channels (C). The inter-node network is channel-sliced, so that there are two physical channels connected to each of the node's six neighbors in the three-dimensional torus. The distinction between solid and dashed lines relates to deadlock avoidance, as discussed in Section 2.5.

1. Introduction

Continued scaling of CMOS technology, combined with an increasing focus on managing chip complexity, has resulted in the widespread adoption of high-performance, tightly coupled on-chip networks in a variety of distributed architectures [5, 8]. Performance-oriented networks were once incorporated primarily in supercomputers consisting of hundreds or thousands of individual processing nodes, but now that a single chip can contain as many transistors as a supercomputer from a few decades ago, the emergence of such networks within a chip is not unexpected [6]. While inter-node and on-chip networks are both designed to maximize performance, they are subject to significantly different requirements: on-chip networks tend to optimize area, power efficiency, and simplicity of the routers [22], whereas inter-node networks typically focus on making the most efficient use of costly inter-node bandwidth [10].

Interconnecting many chips that themselves contain on-chip networks adds a new layer of complexity to the network design problem. In Anton 2, a massively parallel special-purpose supercomputer designed to accelerate molecular dynamics (MD) simulations, we sought to manage this complexity by reusing the on-chip network to implement the inter-node network's switching functionality. This network architecture, which we refer to here as a *unified network*, offers significant advantages in simplicity and efficiency, including a single router design, a shared set of on-chip wires, and lower implementation complexity and cost. In a typical Anton 2 configuration, the inter-node network is a three-dimensional torus formed by hundreds of directly connected ASICs. The on-chip network is a two-dimensional mesh (Figure 1), which provides connectivity between the compute endpoints and the external network channels. Previous architectures have reused an on-chip crossbar [12] or, in the case of Anton 2's predecessor, a ring network [13] as an inter-node switch. The more complicated topology and large bandwidth requirements of Anton 2's on-chip network, however, gave rise to a number of unique challenges.

One of these challenges is ensuring fairness and maintain-

[*]David E. Shaw is also with the Department of Biochemistry and Molecular Biophysics, Columbia University, New York, NY 10032.
[†]To whom correspondence should be addressed: Brian.Towles@DEShawResearch.com and David.Shaw@DEShawResearch.com.

978-1-4799-4394-4/14/$31.00 © 2014 IEEE

ing the network's throughput beyond saturation. In a unified network, packets traveling through a node may pass through several intermediate on-chip routers, each with several arbiters. Small errors in the allocation of bandwidth at each network arbiter can accumulate, leading to exponentially increasing unfairness. Existing solutions designed for on-chip networks offer low overheads, but do not scale well to inter-node networks [16, 18, 19], and solutions intended for inter-node networks significantly complicate the routers [1]. Here, we introduce an arbiter design that sets the ratio of grants issued to requestors at each network arbitration point based on a small number of pre-computed traffic patterns. This arbiter design, which we refer to as an *inverse-weighted arbiter*, greatly improves the throughput of the network with only a modest increase in arbitration latency and network area. Inverse-weighted arbiters are most effective when the application's traffic patterns are known in advance. This is the case for our MD simulations, and this also arises in the context of many other on-chip network design applications [7].

Another challenge is preventing the on-chip network from becoming a bottleneck for inter-node traffic, which could lead to a reduction in the utilization of the inter-node channels. We addressed this concern by evaluating a set of candidate routing algorithms over all expected inter-node switching demands. This evaluation can be expressed as a linear program [27], the solution of which yields an on-chip routing algorithm that is both simple and robust.

In addition to routing the inter-node traffic, the on-chip network must ensure that this traffic remains deadlock-free. To meet this requirement, the on-chip routers implement the inter-node virtual channels (VCs), which increases the costs of the routers. In Anton 2, these costs are mitigated by a VC promotion algorithm that improves on previous approaches, reducing the total number of VCs by one-third. We describe the algorithm, which can be applied to any n-dimensional torus network, and show that it is deadlock-free under the assumption of minimal (shortest-path) routing.

The combination of these techniques resulted in an Anton 2 network implementation that met our performance goals while occupying less than 10% of the total ASIC area. Measurements of a working Anton 2 machine demonstrate nearest-neighbor, software-to-software messaging latencies that are substantially lower than on a typical commodity network. Anton 2's inter-node network channels provide 2.15 Tb/s of effective I/O bandwidth per ASIC, which is comparable to recent dedicated network switch chips [3], and we observe approximately 90% utilization of this bandwidth for many traffic patterns.

2. Architecture

To meet the demands of massively parallel MD simulations, the Anton 2 network was designed to provide low-latency delivery of fine-grained packets while maximizing bandwidth efficiency, especially for communication with three-dimensional spatial locality. Particular emphasis was placed on the performance of the network beyond saturation, as MD simulations tend to have bursty communication demands that are not self-throttling. These goals, combined with the engineering desire to keep the network as simple as possible, drove basic design decisions for the programming model, topology, and packaging of the network. This section describes these aspects of the network architecture, along with the routing algorithms we developed to maximize network throughput while ensuring that the network is deadlock-free.

2.1. Programming model

The Anton 2 network presents a global, distributed-memory model to its endpoints. The network supports remote reads, but most network transactions are remote writes, with synchronization provided by a counted-write mechanism at the endpoints [15]. Separate request and reply traffic classes are provided to avoid protocol deadlocks.

Anton 2's predecessor supported packets up to 288 bytes in size. While fine-grained by most standards, the complexity and latency of assembling these packets in software still proved nontrivial. The Anton 2 network is optimized to handle even finer-grained packets: a typical packet carries 16 bytes of payload with 8 bytes of header information, and the largest packet is limited to twice this amount (32 bytes of payload and 16 bytes of header).

2.2. Topology and packaging

The Anton 2 ASICs are interconnected using a three-dimensional torus topology—a natural choice for capturing the three-dimensional spatial locality inherent in MD simulations [24]. Because the topology is matched to the application, most packets travel just a few hops between nodes of the network, reducing both latency and inter-node bandwidth demands. Throughout this paper, we refer to the torus dimensions as X, Y, and Z.

To reduce physical design complexity within the ASIC, the three-dimensional torus is channel-sliced [9]: two physical channels are connected to each of the node's six immediate neighbors. Each of these 12 total physical channels comprises eight bidirectional Avago Technologies SerDes operating at 14 Gb/s, which provide a total of 112 Gb/s/direction of raw bandwidth. Physical and link layers provide framing, error checking, and go-back-N retransmission, leaving 89.6 Gb/s/direction of effective bandwidth.

Internally, the ASIC contains a 4×4 mesh network that serves two roles (Figure 1): it provides network connectivity between the endpoints within the ASIC, and it acts as a switch between the 12 torus channels. Each mesh channel contains 192 bits per direction and is clocked at 1.5 GHz. These channels are wide enough to transmit the common-case packet (24 bytes) in a single cycle. To avoid confusion with the inter-node torus dimensions, we refer to the two mesh dimensions as U and V.

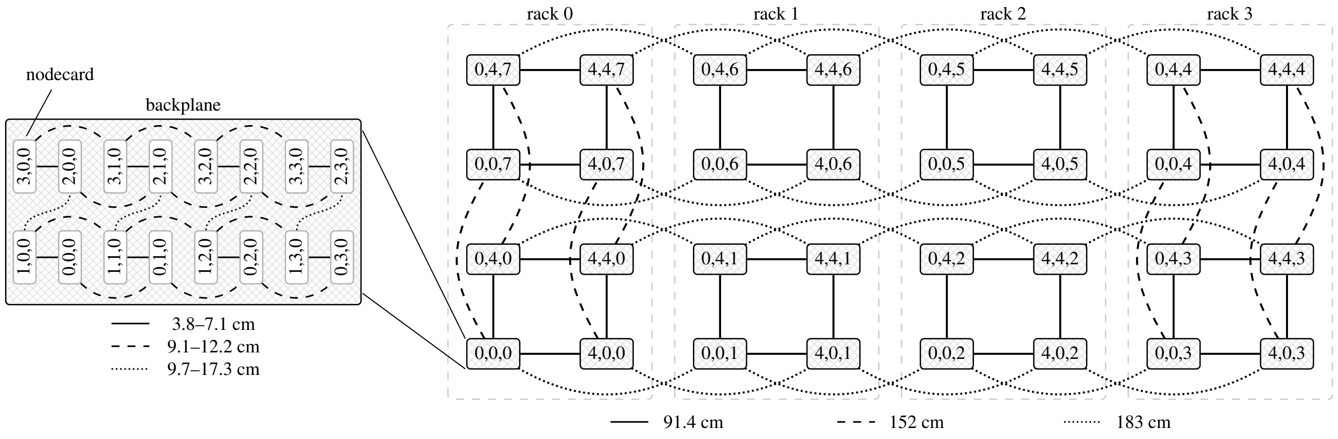


Figure 2: Network packaging of a 512-node Anton 2 machine. Each nodecard contains a single ASIC and is labeled with its ASIC's coordinates in the three-dimensional torus network. Sixteen nodecards connect to a backplane PCB using right-angle edge connectors, and short backplane traces connect the nodecards in a $4 \times 4 \times 1$ configuration. Eight backplanes are mounted into a single rack and four racks are grouped to form the entire machine. Each backplane is labeled with the lexicographically smallest torus coordinates over all of the ASICs that are connected to it. All inter-backplane connections (both within and between racks) are cabled. Trace and cable lengths are indicated by the line pattern used to connect units, with a key below the backplane and rack indicating the length associated with each connection. Although not shown, the ASIC-to-edge-connector trace lengths on the nodecard vary from 7.1 to 11.7 cm.

Due to package- and board-level design considerations, the high-speed I/O pins are distributed along two opposite edges of the ASIC. This arrangement enables isolation of the analog circuits within each I/O's SerDes from the noisy digital power that supplies the core of the ASIC. It also allows digital power to be fed from wide metal layers along the two remaining edges of the ASIC, lowering inductance, and ultimately improving digital power delivery during rapid changes in current draw.

The Y and Z channels are placed so that packets traveling along a Y or Z dimension traverse only a single router as they pass through an ASIC (Figure 1). In addition, the Y and Z channels associated with the same torus slice are placed on the same side of the ASIC. Individual packets use the same slice for their entire route, so this placement reduces latency for packets turning from the Y to Z dimension and vice versa.

The X channels are placed differently to reduce printed circuit board (PCB) complexity. Each ASIC is mounted to a *nodecard* PCB, and these nodecards mate with the front of a backplane using high-performance, right-angle edge connectors. Each backplane holds 16 nodecards arranged in a $4 \times 4 \times 1$ array. Torus channels between the nodecards are routed completely within the backplane; the remaining torus channels are connected to cables on the rear of the backplane. The flexibility provided by the cabling allows this single backplane design to be used for machine configurations from $4 \times 4 \times 1 = 16$ ASICs all the way up to the maximum supported size of $16 \times 16 \times 16 = 4,096$ ASICs. Figure 2 illustrates the packaging of a 512-node Anton 2 machine with 32 backplanes mounted into 4 racks.

Splitting the two directions ($+$ and $-$) of the X dimension across the two edges of the ASIC simplifies routing in the backplane, but at the cost of additional complexity within the ASIC. Packets traveling through the ASIC along the X dimension must travel from one side of the ASIC to the other. To mitigate the additional latency of this path, channels were added to the on-chip mesh topology that allow these packets to skip two intermediate routers. These channels, which we refer to as *skip channels*, also play a critical role in deadlock avoidance, as described in a later section.

2.3. Inter-node routing

Unicast routing in the Anton 2 network is oblivious—routes are randomized and independent of network load [28]. Packets follow a minimal, dimension-order route through the inter-node network, and each packet can follow any of the six possible dimension orders: XYZ, XZY, YXZ, YZX, ZXY, or ZYX. In addition, a packet is assigned to one of the two torus slices, and then only uses torus links associated with that slice during its route. Typically, a unicast packet's dimension order and torus slice are randomized. Adaptive routing was not employed in Anton 2 because our traffic demands tend to be node-symmetric, which simplifies the task of load balancing, and because maintaining throughput beyond saturation can be made more difficult by adaptive routing [21].

The network also supports table-based multicast to an arbitrary set of destinations. In MD simulations, multicast is particularly effective at reducing the bandwidth required to broadcast a particle to a set of endpoints on its neighboring nodes, which is an extremely common communication pattern. As an example, Figure 3 shows a representative multicast destination set for a single particle's position in one plane of the

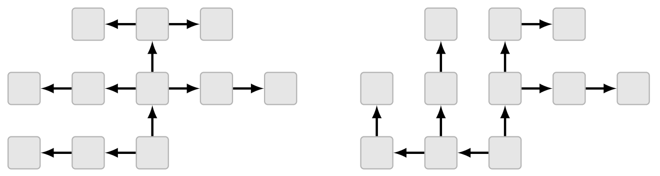


Figure 3: An example of two possible inter-node multicast routes for a single set of destination nodes in a two-dimensional plane of the torus network. By alternating between these two routes for different packets, load balance on the torus channels is improved.

torus network [24]. In this particular example, multicast saves 12 torus hops of bandwidth when compared to unicasts of the packet to each destination node. The example also illustrates the utility of supporting arbitrary dimension orders to improve load balance across the torus channels. By alternating between the two routing patterns shown in Figure 3, the load on the most heavily utilized torus channels can be balanced.

An MD simulation may contain several hundred distinct destination sets per node, but these sets can be quickly computed and loaded into the multicast tables at initialization; they remain constant throughout the entire simulation. In addition, each destination set may contain multiple endpoints within each node, with separate copies written to each endpoint in order to minimize the latency of retrieving the packet data. In this case, the inter-node bandwidth savings offered by multicast quickly multiply.

2.4. On-chip routing

There are two types of on-chip routes: 1) *through routes* of packets that are passing through the ASIC as they travel along a single dimension of the torus, and 2) *local routes* of packets that are turning between torus dimensions or routing to or from an on-chip endpoint.

Through routes in the Y and Z dimensions simply hop through the single on-chip router that connects their input and output torus channels. Referring to Figure 1, consider a packet traveling through an ASIC along the Y^- direction of the torus on slice 0. This packet would follow the path:

$$Y_0^+ \longrightarrow R_{0,2} \longrightarrow Y_0^-.$$

By convention, we refer to the bidirectional torus links by the direction of packets departing the ASIC, so arriving packets enter the ASIC on the torus channel labeled with the opposite direction.

Through routes in the X dimension use the skip channel. A packet traveling along the X^+ direction of the torus on slice 1 would follow the path:

$$X_1^- \longrightarrow R_{3,0} \xrightarrow{\text{skip channel}} R_{0,0} \longrightarrow X_1^+.$$

From the point of view of the external torus channels, the ASIC would ideally appear as a perfect switch whose throughput is limited only by the bandwidths of these channels. In reality, however, the switch is implemented by the on-chip mesh, so there may be some traffic demands placed on the ASIC that, in combination with the local routing algorithm, result in an on-chip mesh channel becoming a bottleneck. The local routing algorithm was designed to avoid this situation.

The design process began by constraining the search for possible on-chip local routing algorithms to direction-order algorithms [23]. Direction-order algorithms specify the order in which packets must traverse the network directions and include dimension-order routing algorithms as a special case. For the on-chip mesh, the directions are U^+, U^-, V^+, and V^-. Direction-order algorithms are deterministic and deadlock-free without the need for multiple VCs, which keeps their implementation in the routers as simple as possible.

In order to emulate the behavior of a perfect switch as closely as possible, the objective of the search was to find the algorithm that minimized the maximum traffic load placed on a mesh channel for all possible switching demands. From [27], maximizing a load given a routing algorithm can be posed as a linear programming problem over the set of possible switching demands. An optimal solution can always be obtained at an extreme point of the possible switching demands, and these extreme points correspond to permutation traffic patterns.

Assuming the two slices of the torus are load-balanced, the search yields a common worst-case permutation for all direction-order routing algorithms:

$$\begin{pmatrix} X^+ & X^- & Y^+ & Y^- & Z^+ & Z^- \\ Z^- & X^+ & Y^- & Z^+ & X^- & Y^+ \end{pmatrix}, \qquad (1)$$

where each column of this matrix contains one source-destination pair from the permutation.

The search also revealed that routing V^-, U^+, U^-, and then V^+ outperforms all other direction-order routing algorithms. Figure 4 shows the routes for the worst-case permutation using this algorithm: the most heavily loaded mesh channels carry two torus channels' worth of traffic. Each mesh channel has a bandwidth of 288 Gb/s, however, which is enough to carry twice the effective torus channel bandwidth of 89.6 Gb/s, with substantial bandwidth left over for packets routing to and from on-chip endpoints.

2.5. Deadlock avoidance

The Anton 2 network employs virtual cut-through flow control, and avoids deadlock by ensuring that the set of dependencies that can be formed between the VCs within each traffic class is acyclic [11]. Both traffic classes have an identical set of dependencies, so we simply focus on the dependencies within a single class. Furthermore, multicast does not introduce any new dependencies between the VCs, because each path from source to destination in a multicast tree is required to be a valid unicast route, so it suffices to restrict our attention to unicast routing.

For the analysis of VC dependencies, the network channels are divided into two groups: the M-group consists of all of the

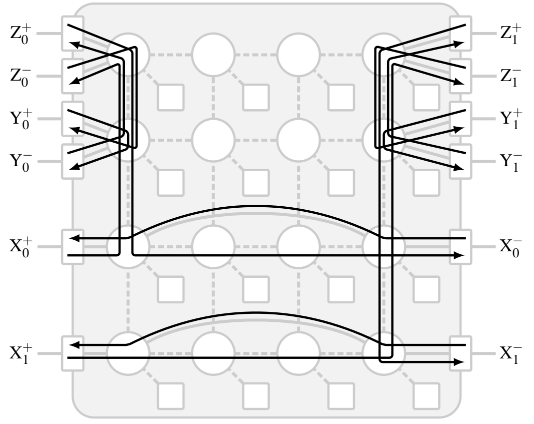


Figure 4: A superposition of the local routes induced by the worst-case permutation (1). The most heavily loaded mesh channels carry two traffic flows from the external channels along a single direction.

channels in the on-chip mesh *except* the skip channels and the channels between the routers and external torus channels; the T-group consists of the remaining channels, including all torus channels. Both groups are shown in Figure 1, with channels in the M-group shown as dashed lines and channels in the T-group shown as solid lines.

All routes begin in the M-group, switch to the T-group to route along a single dimension of the torus, and then switch back to the M-group to reach to the destination endpoint or to turn to a new torus dimension. In the latter case, routing continues to alternate between the two groups until a packet reaches its destination, with the M-group visited at most four times and the T-group at most three times (once per dimension of the torus).

The direction-order, on-chip routing algorithm used within the M-group is acyclic using a single VC [23]. The T-group does contain cycles, however, because each torus dimension is a ring. The cycles within a ring can be broken by using two VCs with a dateline [11].

Previous implementations [20] enable arbitrary dimension-order routing in an n-dimensional mesh by incrementing a packet's VC as it turns from one dimension to the next. This approach is then extended to tori by using two VCs with a dateline for each dimension. The resulting implementation uses a total of $2n$ VCs and yields an acyclic set of inter-group dependencies:

$$\begin{tikzcd}
 & \mathrm{T}_0 \arrow[d] \arrow[dr] & & \mathrm{T}_2 \arrow[d] \arrow[dr] & & \mathrm{T}_4 \arrow[d] \arrow[dr] & \\
\mathrm{M}_0 \arrow[ur] & \mathrm{T}_1 \arrow[r] & \mathrm{M}_1 \arrow[ur] & \mathrm{T}_3 \arrow[r] & \mathrm{M}_2 \arrow[ur] & \mathrm{T}_5 \arrow[r] & \mathrm{M}_3
\end{tikzcd}$$

M_v and T_v refer to a VC v within the M- and T-groups, respectively, and the dependencies $\mathrm{T}_v \to \mathrm{T}_{v+1}$ are introduced by the datelines. Since the dependencies for a single VC within each T- and M-group are also acyclic, it follows that the entire VC dependency graph is acyclic. With this simple approach, the network for a three-dimensional torus is deadlock-free using 6 VCs for the T-group channels and 4 VCs for the M-group.

The Anton 2 network improves on this, requiring only 4 VCs for each of the T- and M-groups. Instead of using a distinct set of VCs for each dimension of torus routing, a packet's VC is only incremented when it 1) crosses a dateline, or 2) finishes routing along a torus dimension in which it did not cross a dateline. This ensures that a packet's VC will be incremented at most once per dimension.

The resulting inter-group dependencies are:

$$\begin{tikzcd}
 & \mathrm{T}_0 \arrow[d] \arrow[dr] & & \mathrm{T}_1 \arrow[d] \arrow[dr] & & \mathrm{T}_2 \arrow[d] \arrow[dr] & \\
 & & \mathrm{M}_1 \arrow[ur] & & \mathrm{M}_2 \arrow[ur] & & \mathrm{M}_3 \\
\mathrm{M}_0 \arrow[uur] & \mathrm{T}_1 \arrow[ur] \arrow[uurr, dash, dashed] & & \mathrm{T}_2 \arrow[ur] \arrow[uurr, dash, dashed] & & \mathrm{T}_3 \arrow[ur] &
\end{tikzcd}$$

where the dotted lines emphasize the fact that the T_1 and T_2 groups are repeated in the dependency diagram. There are two requirements that must be met in order to avoid cyclic dependencies involving the T_1 or T_2 groups, so that the resulting routing algorithm is deadlock-free. First, all torus routes must follow shortest paths (minimal routes). Second, the $+$ and $-$ direction datelines must be placed between the same two nodes along a dimension. For a torus with k_D nodes along dimension D, we place the dateline between nodes $k_D - 1$ and 0, so that a packet's VC is incremented when its coordinate changes from $k_D - 1$ to 0 (or vice versa) in that dimension.

To see that the routing algorithm is deadlock-free under these assumptions, suppose instead that a cycle exists within a single VC, and assume without loss of generality that this cycle includes a hop in the X^+ direction from x to $x+1$. The cycle cannot cross the X dateline between nodes 0 and $k_X - 1$ (since it is contained within a single VC), so it must also include a hop in the X^- direction from $x+1$ to x, with turns between these hops.

The turn away from X^+ following $x+1$ corresponds to packets that have already crossed the dateline from $k_X - 1$ to 0 and have thus traveled at least $x+2$ hops in X^+ (if these packets had not crossed the dateline, their VC would be incremented upon turning, exiting the single VC under consideration). It follows from the minimal-route assumption that $x+2 \leq k_X/2$. Similarly, the turn away from X^- following x corresponds to packets that have crossed the dateline from 0 to $k_X - 1$, hence $k_X - x \leq k_X/2$. Adding these inequalities gives $k_X + 2 \leq k_X$, a contradiction. A cycle is thus impossible, so the routing algorithm is deadlock-free.

3. Inverse-weighted arbiters

Achieving fairness and maintaining throughput stability during network saturation is a challenge for any network [1]. In the Anton 2 network, the choice of a unified network only adds to this challenge as each inter-node hop is implemented as sequence of on-chip routing decisions, providing multiple opportunities for the introduction of unfairness. If the routers had been built using simple, locally fair arbiters [9], then the network would exhibit significant global unfairness and loss of throughput beyond saturation, even for benign, symmetric traffic patterns.

The goal of the Anton 2 network arbiters is to provide equality of service (EoS) [19], such that each endpoint gets an equal share of any bottleneck resource. This ensures stability and fairness as the network is pushed beyond saturation. The desire for a small, low-latency on-chip router, however, meant that heavy-weight techniques for providing EoS, such as age-based arbitration [1], would have been prohibitively expensive. Several alternatives to age-based arbitration have been proposed in the context of on-chip networks [16, 18, 19], but they are either specialized for specific topologies or do not scale to the thousands of endpoints typical in an Anton 2 deployment.

Instead, the Anton 2 network employs an inverse-weighted arbiter design. The design takes advantage of the fact that the application-level communication of MD can be captured with a small set of traffic patterns, allowing it to avoid the per-packet overhead and complex arbitration required in a dynamic scheme. Knowledge of these patterns is used to program static, per-pattern weights that are stored within each arbiter. Unlike existing approaches that incorporate a single set of weights [9], the inverse-weighted arbiter can accommodate multiple traffic patterns, and continues to ensure fairness as the patterns are blended. When combined with an optimized arbiter design, the resulting network has greatly improved throughput and fairness with a minimal increase in arbitration latency and a modest increase in the area of the network.

3.1. Equality of service

EoS is achieved when access to each network resource is granted in proportion to the requestor's contribution to the load on the resource [19]. The load on a resource is found by summing individual contributions from each source, where the contribution from a single source is the expected number of packets sent from that source that would use the resource per unit time. Figure 5 shows an example of a simple topology and traffic pattern along with the resulting loads.

3.2. Implementation concept

To achieve service proportional to load, a set of accumulators is used to track the service history of each arbiter input. For arbiter input i, the value of the accumulator A_i at time t is

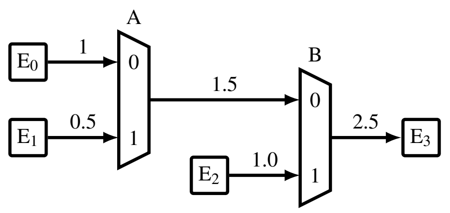
$$A_i(t) = S_i(t)/\bar{\gamma}_i, \tag{2}$$

where $S_i(t)$ is the total number of packets serviced by input i prior to time t and $\bar{\gamma}_i$ is the average load on input i. All accumulators are initialized to zero.

The arbitration policy is simply that the requesting input with the lowest accumulator value has the highest priority. This policy implies that all of the accumulator values across all of an arbiter's inputs will be roughly equalized. Then, for any

$$\begin{array}{c|c} & E_3 \\ E_0 & 1 \\ E_1 & 0.5 \\ E_2 & 1 \end{array}$$

(a)



(b)

Figure 5: (a) One column of a traffic pattern expressed as a matrix. Rows of the matrix correspond to sources. The column corresponds to a destination. Each entry contains the expected number of packets sent from a source to the destination per unit time. (b) A topology and the loads induced on its channels for the traffic pattern given in (a). The channel leaving arbiter A, for example, carries the combined loads of sources E_0 and E_1: 0.5 + 1 = 1.5. At arbiter A, the load on input zero is 1 and the load on input one is 0.5, so granting input zero 1/0.5 = 2 times as often as input one achieves EoS. Similarly, at arbiter B, input zero should be granted 1.5 times as often as input one to achieve EoS.

two inputs i and j, it follows from (2) that $S_i(t)/S_j(t) \approx \bar{\gamma}_i/\bar{\gamma}_j$, so the ratio of service is indeed proportional to the ratio of the loads. For a known traffic pattern, the values can be pre-computed offline and stored at each arbiter input.

This idea can also be extended to multiple traffic patterns. Define $\gamma_{i,n}$ as the load on input i due to traffic pattern n. Also, define $s_{i,n}(t)$ as the total number of packets from input i associated with traffic pattern n serviced prior to time t. Then, by marking each packet as belonging to one of N traffic patterns, the accumulator can track service across all patterns as

$$A_i(t) = \sum_{n=0}^{N-1} s_{i,n}(t)/\gamma_{i,n}, \tag{3}$$

If all packets belong to a single traffic pattern, then (3) is trivially equivalent to (2). In practice, however, packets in the network may be divided among several different traffic patterns simultaneously, so that the effective traffic pattern is a blend of the N pre-computed patterns. Load is linear, so the load when blending multiple patterns is simply

$$\bar{\gamma}_i = \sum_{n=0}^{N-1} \alpha_n \gamma_{i,n}, \tag{4}$$

where the mixing coefficients α_n are non-negative and sum to one.

The sequence of packets arriving at a particular input is divided among the traffic patterns in proportion to an individual pattern's contribution to the blended load. It follows that

$$s_{i,n}(t) = \alpha_n \gamma_{i,n} S_i(t)/\bar{\gamma}_i. \tag{5}$$

Substituting (4) and (5) into (3) and using the definitions $S_i(t) = \sum_{n=0}^{N-1} s_{i,n}(t)$ and $\sum_{n=0}^{N-1} \alpha_n = 1$ yields the original value of the accumulator (2).

Thus, by identifying N traffic patterns offline, precomputing their loads, and then tracking the accumulator value using (3), this arbitration policy achieves EoS across any blend of these N traffic patterns without explicit knowledge of how the patterns are combined (the mixing coefficients are not needed to track the accumulator's value).

3.3. Approximating the accumulator

The implementation of the arbiter begins with approximating (3). All of the accumulators at a given arbiter are scaled by a positive factor β, and $\gamma_{i,n}^{-1}$ is approximated as $m_{i,n} = \mathrm{nint}(\beta\gamma_{i,n}^{-1})$, where nint is the nearest integer function—this ensures that all accumulators will be integer-valued. Increasing β increases the accuracy of the approximation, but requires more bits to represent the inverse-weight values $m_{i,n}$. The number of inverse-weight bits M is chosen so that $m_{i,n} < 2^M$ for each $m_{i,n}$.

To limit the range of the accumulators, their values are stored relative to a sliding window that contains 2^{M+1} values. Then, a single bit conveys the priority of each input: if an accumulator's value is in the lower half of the window, the corresponding input is high priority, otherwise, it is low priority. Ties are broken using a round-robin rule. This approximates the arbitration policy of the previous section: smaller accumulators have higher priority.

The Anton 2 implementation supports two traffic patterns ($N = 2$), and each packet header contains a field that identifies the traffic pattern to which the packet belongs. When a packet using pattern n is granted at input i, the accumulator A_i is incremented by the inverse-weight $m_{i,n}$. If the granted input had low priority, then there are no high-priority requesting inputs, so the accumulators of all requesting inputs must be greater than or equal to 2^M. In this case, the sliding window is shifted by subtracting 2^M from all of the accumulator values. This update rule ensures that the accumulator values will always be less than 2^{M+1}. When this subtraction is applied, it is possible for a non-requesting input's accumulator to fall below zero. In this underflow case, the accumulator is simply set to zero. As long as the inputs remain largely non-empty, as is typically the case when the network is saturated, this underflow case should be rare. These accumulator update rules can be implemented using a single adder per accumulator and simple control logic, as shown in Figure 6.

```
module accumulator_update
  #( parameter K = 2, parameter M = 5 )
   ( input logic [K-1:0][M:0] accum,
     input logic [K-1:0][M-1:0] inv_weight,
     input logic [K-1:0] grant,
     output logic [K-1:0] pri,
     output logic [K-1:0][M:0] accum_nxt );

  logic low_grant;
  assign low_grant = |( grant & ~pri );

  always_comb for ( int i = 0; i < K; i++ ) begin
    logic [M:0] accum_msb0;

    accum_msb0 = { 1’b0, accum[i][M-1:0] };
    pri[i] = !accum[i][M];
    accum_nxt[i] =
      grant[i] ? ( accum_msb0 + inv_weight[i] ) :
      low_grant ? ( pri[i] ? ’0 : accum_msb0 ) :
      accum[i];
  end
endmodule
```

Figure 6: A SystemVerilog implementation of the accumulator update logic for a *k*-input arbiter. The most significant bit of each accumulator is output as a priority value, which is passed to the arbiter. The grant vector returned from the arbiter is used to determine the updated values of the accumulators. Subtracting 2^M to adjust the window is implemented by either clearing the most significant bit of the accumulator or zeroing the entire accumulator (the underflow case).

3.4. Optimizing the arbiter

A typical approach for constructing low-latency, round-robin arbiters with a small number of priority levels is to create a separate un-prioritized round-robin arbiter for each priority level and then combine the results, where each round-robin arbiter is itself implemented using two fixed-priority arbiters [17]. The Anton 2 arbiter improves on this approach by reducing the number of fixed-priority arbiters. Figure 7 illustrates the core idea: applying round-robin priority means back-to-back fixed-priority arbiters are selecting from among a mutually exclusive set of requests, so these requests can be safely combined and handled with a single fixed-priority arbiter. This design uses fewer fixed-priority arbiters and simplifies the final step of combining arbiter results. For the inverse-weighted arbiter with two priority levels, this optimization reduces the number of fixed-priority arbiters from 4 to 3. In general, for P priority levels, the number of fixed-priority arbiters is reduced by almost half (from $2P$ to $P+1$). An arbiter implementation based on these ideas is shown in Figure 8.

4. Measurements

The Anton 2 ASIC was fabricated in the TSMC 40G process [26]. We collected a number of network performance measurements from an Anton 2 machine with 512 ASICs configured as an $8 \times 8 \times 8$ torus. The on-chip network was clocked at 1.5 GHz, and the network SerDes operated at 14 Gb/s. All tests were driven by processor cores connected to the on-chip network, with one core per router participating in each test. Time measurements were taken using free-running cycle counters local to each core.

4.1. Fairness and throughput

To measure the impact of fairness on throughput and the benefits of introducing inverse-weighted arbitration, traffic patterns with varying levels of locality were considered. In n-hop neighbor traffic [2], each packet travels to a random destination node that is at most n hops away along each dimension of the torus.

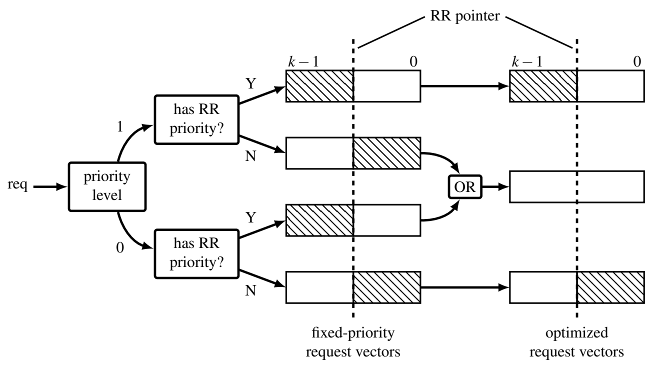


Figure 7: An example of arbiter optimization for a *k*-input arbiter with two priority levels. On the left, a *k*-bit request vector (one bit per arbiter input) is split into the requests at priority level 1 (top) and those at level 0 (bottom). A round-robin (RR) pointer divides the *k* request vector bits into upper bits with higher priority and lower bits with lower priority. The slashed areas in each of the four split request vectors have been disabled based on the RR pointer. A typical approach [17] is to arbitrate among these vectors using a fixed-priority rule: topmost request vectors get higher priority and, within each request vector, the most significant bit has highest priority. The optimized approach combines the middle pair of request vectors, which are mutually exclusive because of the RR split. A similar fixed-priority rule is then applied.

In uniform random traffic, each packet is sent to a random destination without any locality constraints.

For the measurements, all cores sent a given number of packets according to a particular traffic pattern. Throughput was then calculated as the time required to receive the last packet in the batch divided by the number of packets in the batch, and then normalized so that a throughput of 1 indicates full utilization of torus channels. Instead of tuning the arbiter weights for each traffic pattern, a single set of arbiter weights, based on the channel loads induced by the uniform pattern, was used for all traffic patterns.

Even for these benign traffic patterns, the throughputs of both 2-hop neighbor and uniform traffic fall significantly beyond saturation, with uniform traffic falling below 60% of the ideal throughput when round-robin arbiters are used throughout the network (Figure 9). Repeating this experiment with inverse-weighted arbiters yields significant improvements: both traffic patterns saturate near 90% of the ideal throughput and maintain this throughput as batch size increases. Although not shown, 1-hop neighbor traffic is stable with both round-robin and inverse-weighted arbitration.

These traffic patterns were similar enough that a single set of arbiter weights was sufficient to describe them, which suggests that, in general, a small set of weights may be sufficient for a large set of traffic patterns. Characterizing this property and optimizing a small set of arbiter weights for a larger number of traffic patterns simultaneously is an interesting open problem.

```
module priority_arb
  #( parameter K = 2, parameter P = 2 )
   ( input  logic [K-1:0] req,
     input  logic [K-1:0][$clog2(P)-1:0] pri,
     input  logic [K-1:0] rr_therm,
     output logic [K-1:0] grant );

   logic [P:0][K-1:0] req_unroll;
   logic [P:0][K-1:0] higher_pri_req;
   logic [P:0][K-1:0] grant_unroll;

   always_comb begin
       req_unroll[0] = req;
       for ( int p = 1; p <= P; p++ )
         for ( int i = 0; i < K; i++ )
           req_unroll[p][i] = req[i] &
             ( {pri[i],rr_therm[i]} >= 2*p-1 );

       // Kogge-Stone parallel prefix
       higher_pri_req = req_unroll >> 1;
       for ( int i = 0; i < $clog2(K-1); i++ )
         higher_pri_req |= higher_pri_req >> (1 << i);

       grant_unroll = req_unroll & ~higher_pri_req;
       for ( int i = 0; i < $clog2(P+1); i++ )
         grant_unroll |= grant_unroll >> (K << i);

       grant = grant_unroll[0];
   end
endmodule
```

Figure 8: A SystemVerilog implementation of a *k*-input arbiter with *P* priority levels and ties broken using a round-robin rule. Two priority levels ($P = 2$) are needed to implement the inverse-weighted arbiters. The round-robin state is thermometer-encoded [17], such that for all inputs `i > 0`, if `rr_therm[i] == 1`, then `rr_therm[i-1] == 1`. To minimize arbitration latency, the fixed-priority rule is implemented using a parallel-prefix network, the output of which is used to cancel lower-priority requests. The depth of this network is limited to $\lceil \log_2(k\text{-}1) \rceil$ stages by also thermometer encoding the fixed-priority request, such that for all priorities `p > 0` and all inputs `i`, if `req_unroll[i][p] == 1`, then `req_unroll[i][p-1] == 1`. Logic for computing an updated round-robin state is omitted in the interest of clarity.

4.2. Multiple traffic patterns

One of the important features of the inverse-weighted arbiter is that given multiple traffic patterns and their associated arbiter weights, any blend of these patterns is supported by simply labeling packets with the pattern to which they belong. To demonstrate this capability, two diametrically opposed traffic patterns were considered. In the "tornado" pattern [25], cores on node (x, y, z) send all of their packets to node

$$(x + k_X/2 - 1, y + k_Y/2 - 1, z + k_Z/2 - 1),$$

where k_D is the number of nodes along dimension D of the torus. The *reverse tornado* pattern is the opposite: cores on node (x, y, z) send to cores on node

$$(x - k_X/2 + 1, y - k_Y/2 + 1, z - k_Z/2 + 1).$$

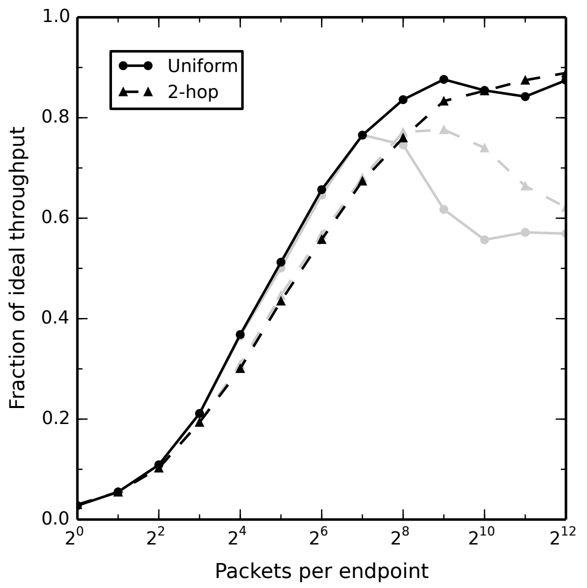


Figure 9: Throughput of 2-hop neighbor traffic and uniform random traffic when using weighted arbitration (black). Results of the same tests run using round-robin arbitration are shown in gray for comparison.

Figure 10 shows the throughput obtained when tornado and reverse tornado traffic are blended. When using a single set of arbiter weights, throughput is high when the traffic blend contains a substantial fraction of traffic that corresponds to those weights. As that fraction is reduced, however, throughput falls to that of round-robin arbitration. Unlike the experiments in the previous section, these two traffic patterns are different enough that the weights for one pattern do not work well for the other.

Using two sets of arbiter weights, one for each traffic pattern, results in a substantial improvement. Throughput is maintained at approximately 85% over the entire range of blending ratios, illustrating the advantage of incorporating multiple arbiter weights.

4.3. One-way message latency

The software-to-software messaging latency of the network was measured using a standard ping-pong test. A single ping-pong begins with software running on core A sending a remote write to memory associated with core B. The write contains 16 bytes of data. When the write is completed, a software handler is dispatched using a hardware synchronization mechanism [15]. Then, software running on core B sends a remote write back to core A. The same process of receiving the remote write and handler dispatch is repeated on core A, completing the ping-pong. The one-way message latency is computed as half the average time it takes to complete a single ping-pong, and includes software and synchronization latency in addition to latency incurred in the network.

Figure 11 shows the measured one-way latency versus the

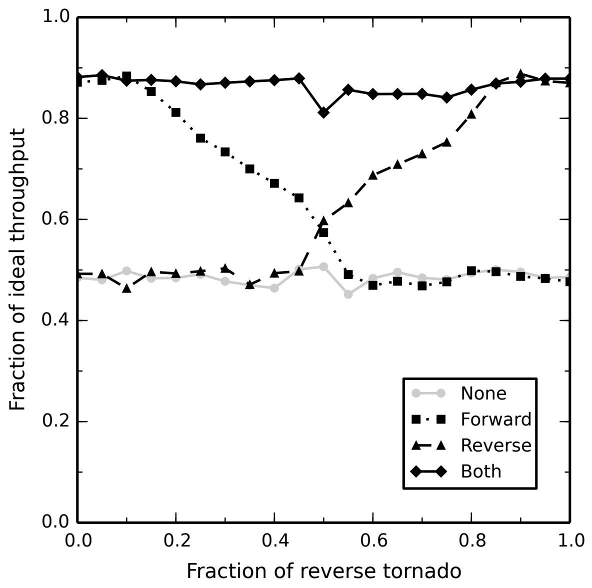


Figure 10: Throughput obtained by blending tornado and reverse tornado traffic for several different sets of arbiter weights. For each measurement, 1,024 packets are sent per processor core. Packets within each measurement are divided between the tornado and reverse tornado patterns, with the fraction assigned to each pattern varying along the horizontal axis. Results for round-robin arbitration (None), for a single set of arbiter weights based on tornado (Forward) or reverse tornado (Reverse), and for two sets of arbiter weights (Both) are shown.

number of inter-node hops taken by the message. This latency is an average over all endpoint pairs that are a given number of hops apart. A linear fit of 80.7 ns of fixed overhead plus 39.1 ns of per-hop latency gives a good approximation of this data. These measured latencies are approximately 1.6 times lower than the same measurements reported for the previous Anton machine [13], and are significantly lower than the 1–2 µs message latencies typical on commodity networks. The minimum measured inter-node latency was approximately 99 ns; Figure 12 shows how endpoint overhead and the various network components contribute to this latency.

4.4. Component area

As shown in Figure 1, the network comprises three distinct component types: routers (R), endpoint adapters (E), and torus-channel adapters (C). All three components use virtual cut-through flow control with credits for tracking VC occupancy. A total of eight VCs are employed in the router and channel adapters to provide separate request and reply traffic classes and to avoid deadlocks within those classes, while the endpoint adapters only need one VC per traffic class. Routers have six ports, each connected to bidirectional channels capable of carrying one 24-byte flit per cycle. Endpoint and channel adapters have two such ports: one connected to a router, and

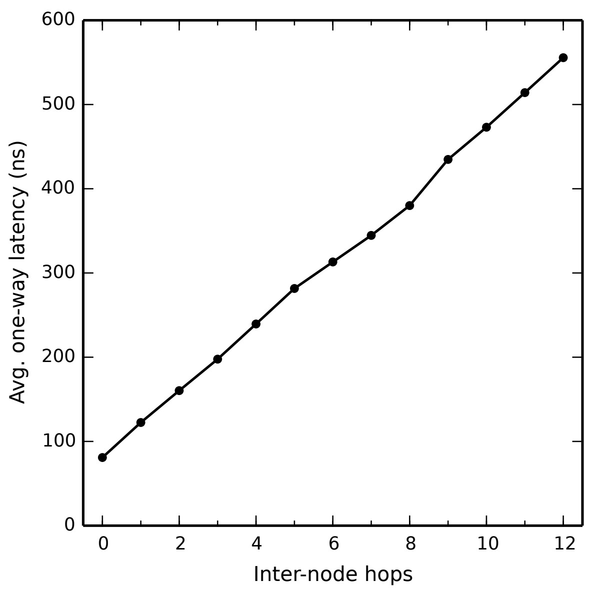


Figure 11: Average one-way message latency with 16 bytes of payload, measured as a function of the number of inter-node hops taken by the message.

Component	Component count	% Die area
Router	16	3.4
Endpoint adapter	23	1.1
Channel adapter	12	4.7

Table 1: Contributions of the network component types to the total die area.

Category	% Network area			
	Router	Endpoint	Channel	Total
Queues	21.2	2.7	22.7	46.6
Reduction	-	-	9.6	9.6
Link	-	-	8.9	8.9
Configuration	3.3	2.5	2.8	8.6
Debug	3.0	2.5	2.3	7.8
Miscellaneous	4.3	1.0	2.0	7.3
Multicast	-	3.2	2.5	5.7
Arbiters	5.2	< 0.1	0.2	5.4

Table 2: Network area by component and totaled across components (rightmost column). The area of multicast tables dominates the "Multicast" category. Registers store various network "Configuration" information and performance counters within each block. "Debug" logic provides a path for monitoring the in-silicon operation of the network components. Channel adapters also contain logic for accelerating in-network "Reductions," which will be described in a forthcoming paper. Logic dedicated to external torus channel framing, scrambling, error detection, and link-level retry is captured in the "Link" category. The "Miscellaneous" category includes credit counters, crossbars, parity generation and checking, and other minor logic.

the other connected to a computational endpoint or torus channel, respectively. Less than 10% of the ASIC's total die area is dedicated to these network components (Table 1).

The network's area can be broken down into several broad categories (Table 2). "Queues" is by far the largest category, and its area is roughly proportional to the number of VCs, highlighting the importance of the optimized deadlock avoidance algorithm introduced in Section 2.5. The inverse-weighted arbiters are the smallest category and account for 5.4% of the network area. Approximately three-quarters of the arbiter area is dedicated to storing the inverse-weight values, the accumulator values, and the accumulator update logic (Section 3.3). The remaining quarter is occupied by the prioritized arbiter itself (Section 3.4).

4.5. Router energy

The core power of each Anton 2 ASIC is supplied by a voltage-regulator module that can be queried by software to report the instantaneous current draw of the ASIC. Similarly, an analog-to-digital converter provides software access to the core voltage. We took advantage of both of these facilities to take power measurements of the network's routers.

The power of a single router hop was measured by first initializing the network so that a single processor core was able to send a continuous stream of single-flit packets on a circuitous route around the on-chip mesh. All routes were confined to a single ASIC, and packets flowed through the routers without any contention or stalling. Power was measured for two different routes: a 3-hop route and a 35-hop route. We then calculated the power of a single router hop by subtracting these two power measurements and dividing by the difference in their lengths (32 hops). The per-flit energy was recovered by dividing this power by the injection rate.[1]

Figure 13 shows the resulting energy measurements as a function of injection rate for three different payload patterns: all bits zero, all bits one, and each bit randomly chosen to be zero or one with equal probability. While the average energy is comparable to other studies [29], the flit energy clearly varies with injection rate. This variation is a feature not present in typical network power models [4, 14].

We propose a power model that captures this energy variation by incorporating the injection rate r as well as the *activation rate* a. For a stream of flits arriving at a router port, the activation rate is defined as the rate of transitions from a cycle that does not contain a valid flit to a cycle that does. The following sequences of valid flits (1) and empty cycles (0) are examples of different injection and activation rates:

[1]This methodology does not include power dissipated when the routers are completely idle. Ungated-clock and leakage power, for example, are excluded. Based on simulated results, this idle power accounts for about 30–50% (depending on the packet payloads) of the total router power when each port is sending at full rate.

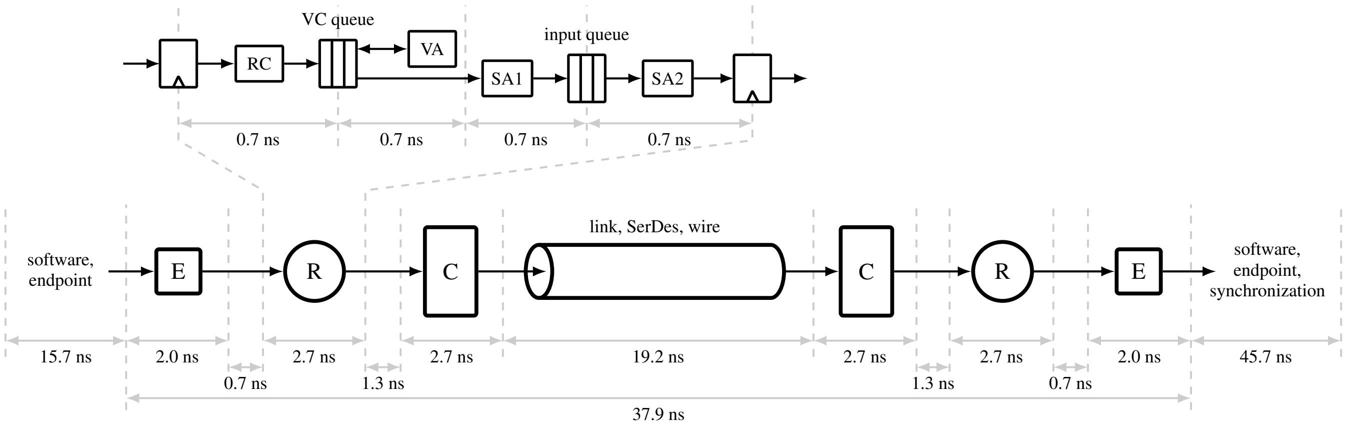


Figure 12: Detailed decomposition of the minimum inter-node messaging latency of 99 ns, including all software and synchronization overheads. Contributions of the endpoint adapters (E), routers (R), and torus-channel adapters (C) are included, along with the network pipeline delays. In addition, the four router pipeline stages are shown in greater detail: route computation (RC), virtual channel allocation (VA), input switch arbitration (SA1), and output switch arbitration (SA2). The actual network only accounts for about 40% of the latency; the majority of the latency is internal to the endpoints and the software.

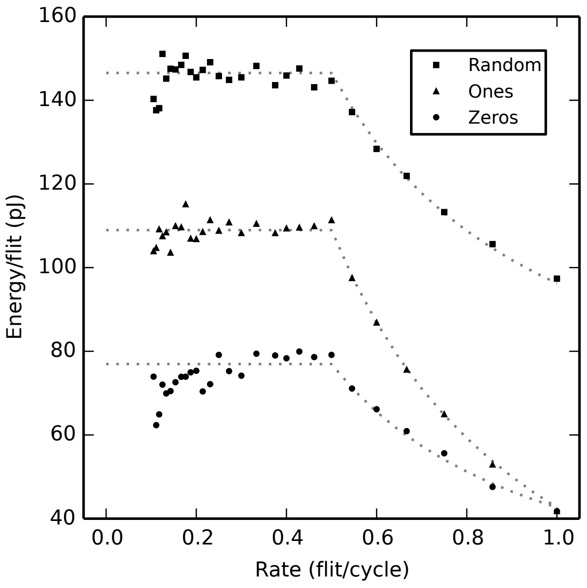


Figure 13: Measured router energy per flit as a function of the injection rate for three different payload patterns: all zeros, all ones, and random. A simple model fit to these measurements is shown as a dotted curve for each payload pattern.

…01010101…	$r = 0.5$	$a = 0.5$
…00110011…	$r = 0.5$	$a = 0.25$
…01110111…	$r = 0.75$	$a = 0.25$

In general, $0 \le a \le \min(r, 1-r)$. For our experiments, the activation rate was maximized, $a = \min(r, 1-r)$, to emphasize its impact on the per-flit energy.

Then, a simple model was fit to the energy measurements (Figure 13):

$$E = 42.7 + 0.837h + (34.4 + 0.250n)(a/r) \ \text{pJ},$$

where h is the average number of bit flips (Hamming distance) between successive valid flits in the sequence and n is the average number of set bits in each flit's payload.

The first two terms of the model capture the energy required to send a flit: a fixed component for data-independent energies, such as arbitration and other control functions, and a component proportional to the number of transitions required in the router's datapath. The remaining terms of the model capture the activation energy of the router, which is proportional to the number of activations per flit a/r. This energy should be non-zero because, at the very least, activation requires valid signals and clock-gate enables throughout the router to switch. The fact that the activation energy contributes a significant fraction of the overall energy at low packet rates, however, is a potential indication of energy inefficiencies in the Anton 2 router design. The extent to which activation energy could be reduced and its importance in developing more accurate router energy models for general use are left as open questions for future work.

5. Conclusion

The emergence of on-chip networks, combined with the existing desire to build tightly coupled multi-node systems, has complicated the network design problem. To manage this complexity, we reused the on-chip network as a switch for inter-node traffic in our design of the Anton 2 network. This unified network enables efficient use of the inter-node channels while occupying a modest of amount of on-chip area. Implementing a unified network is not without its challenges, however, and the specific solutions developed during the Anton 2 network design process may be applicable to other unified networks, as well as more general network design problems.

Reducing the number of VCs required for inter-node routing helps to limit the complexity of the routers. We introduced a deadlock-free routing algorithm that requires only $n+1$ VCs per traffic class for arbitrary dimension-order routing and that can be applied to any n-dimensional torus network. This improves over the $2n$ VCs required by existing approaches [20].

For applications whose traffic demands are well understood, inverse-weighted arbiters demonstrate significant improvements in network throughput and EoS, with low implementation costs. Measurements further reveal that the traffic model need not be exact to realize this improvement: an approximate model is often sufficient to gain the throughput advantages of inverse-weighted arbitration. Understanding the relationship between the quality of approximation and the resulting performance is an open question for future work.

The unified network approach appears to be a viable and attractive option when integrating dense computation, shared local interconnect, and inter-node network switching onto a single chip. We anticipate that continued refinement of this approach, as well as exploration of alternatives, will yield further improvements that can be applied to future generations of tightly coupled supercomputer systems.

Acknowledgements

We thank Larry Nociolo for providing details of the network packaging, Horia Toma, Jack Chao, and Shark Mok for their work on the verification of the Anton 2 network components, and Mollie Kirk and Rebecca Kastleman for editorial assistance.

References

[1] D. Abts and D. Weisser, "Age-based packet arbitration in large-radix k-ary n-cubes," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing (SC)*, Nov. 2007, pp. 1–11.

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

[3] B. Alverson, "Cray high speed networking," in *Proceedings of the 20th Annual Symposium on High-Performance Interconnects (HOTI)*, Aug. 2012.

[4] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proceedings of the 20th annual International Conference on Supercomputing (ICS)*, Jun. 2006, pp. 187–198.

[5] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64 processor: A 64-core SoC with mesh interconnect," in *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, Feb. 2008, pp. 88–89, 598.

[6] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[7] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006.

[8] G. Chrysos, "Intel® Xeon Phi™ coprocessor (codename Knights Corner)," in *Proceedings of the 24th Annual IEEE Hot Chips Symposium*, 2012.

[9] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco: Morgan Kaufmann Publishers Inc., 2003.

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

[11] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547–553, 1987.

[12] E. DeLano, "Tukwila — a quad-core Intel® Itanium® processor," in *Proceedings of the 20th Annual IEEE Hot Chips Symposium*, 2012.

[13] R. O. Dror, J. P. Grossman, K. M. Mackenzie, B. Towles, E. Chow, J. K. Salmon, C. Young, J. A. Bank, B. Batson, M. M. Deneroff, J. S. Kuskin, R. H. Larson, M. A. Moraes, and D. E. Shaw, "Exploiting 162-nanosecond end-to-end communication latency on Anton," in *Proceedings of the Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2010, pp. 1–12.

[14] N. Eisley and L.-S. Peh, "High-level power analysis for on-chip networks," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Sep. 2004, pp. 104–115.

[15] J. P. Grossman, J. S. Kuskin, J. A. Bank, M. Theobald, R. O. Dror, D. J. Ierardi, R. H. Larson, U. B. Schafer, B. Towles, C. Young, and D. E. Shaw, "Hardware support for fine-grained event-driven computation in Anton 2," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr. 2013, pp. 549–560.

[16] B. Grot, S. Keckler, and O. Mutlu, "Preemptive virtual clock: A flexible, efficient, and cost-effective QOS scheme for networks-on-chip," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2009, pp. 268–279.

[17] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *IEEE Micro*, vol. 19, no. 1, pp. 20–28, 1999.

[18] J. Lee, M. C. Ng, and K. Asanović, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in *Proceedings of the 35th annual International Symposium on Computer Architecture (ISCA)*, Jun. 2008, pp. 89–100.

[19] M. M. Lee, J. Kim, D. Abts, M. Marty, and J. W. Lee, "Probabilistic distance-based arbitration: providing equality of service for many-core CMPs," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2010, pp. 509–519.

[20] T. Nesson and S. L. Johnsson, "ROMM routing on mesh and torus networks," in *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Jul. 1995, pp. 275–287.

[21] T. D. Nguyen and L. Snyder, "Performance analysis of a minimal adaptive router," in *Parallel Computer Routing and Communication: 1st International Workshop (PCRCW)*, May 1994, pp. 31–44.

[22] J. Owens, W. Dally, R. Ho, D. N. Jayasimha, S. Keckler, and L.-S. Peh, "Research challenges for on-chip interconnection networks," *IEEE Micro*, vol. 27, no. 5, pp. 96–108, 2007.

[23] S. Scott and G. Thorson, "The Cray T3E network: Adaptive routing in a high performance 3D torus," in *Proceedings of the 4th Annual Symposium on High-Performance Interconnects (HOTI)*, Aug. 1996.

[24] D. E. Shaw, R. O. Dror, J. K. Salmon, J. P. Grossman, K. M. Mackenzie, J. A. Bank, C. Young, M. M. Deneroff, B. Batson, K. J. Bowers, E. Chow, M. P. Eastwood, D. J. Ierardi, J. L. Klepeis, J. S. Kuskin, R. H. Larson, K. Lindorff-Larsen, P. Maragakis, M. A. Moraes, S. Piana, Y. Shan, and B. Towles, "Millisecond-scale molecular dynamics simulations on Anton," in *Proceedings of the Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2009, pp. 1–11.

[25] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta, "Locality-preserving randomized oblivious routing on torus networks," in *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Aug. 2002, pp. 9–13.

[26] Taiwan Semiconductor Manufacturing Company (TSMC), "TSMC first to deliver 40nm process technology," Mar. 2008. Available: http://www.tsmc.com/tsmcdotcom/PRListingNewsAction.do?action=detail&newsid=2561

[27] B. Towles and W. J. Dally, "Worst-case traffic for oblivious routing functions," in *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Aug. 2002, pp. 1–8.

[28] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing (STOC)*, May 1981, pp. 263–277.

[29] H. Wang, L.-S. Peh, and S. Malik, "A technology-aware and energy-oriented topology exploration for on-chip networks," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, Mar. 2005, pp. 1238–1243.