Reproducing GOAL in 3D Torus

AI+X - Lab4

Ziren Wang, Chuan Liu

Yao Class 23 & 22

I. Preliminaries

3D Torus

Definition

A Torus, or k-ary n-cube, is an n dimensional torus network with k nodes per dimension giving a total of $N = k^n$ nodes. It has following properties and therefore becomes popular in network routing algorithm research:

- $\mathcal{O}(N)$ cost
- Exploit locality for near-neighbor traffic
- High path diversity
- Edge symmetric

Routing algorithm

Definition

A routing algorithm R maps a source-destination pair to a path through the network from the source to the destination. We can divide routing algorithms by the following criteria:

- **Oblivious** algorithms select the path using only the identity of the source and destination nodes.
- **Adaptive** algorithms may also base routing decisions on the state of the network.
- **Minimal** algorithms only route packets along a shortest path from source to destination.
- **Non-minimal** ones may route packets along longer paths.

II. GOAL

Algorithm details

The GOAL algorithm contains two parts:

Algorithm 1 Selecting quadrant

Input: Source node s, destination node d
Output: direction vector r
$x_s, y_s, z_s \leftarrow$ coordinates of s
$x_d, y_d, z_d \leftarrow$ coordinates of d
$\delta_x = \min(x_d - x_s, x_s - x_d)$
$p_x^+ = 1 - \delta_x / k$
$r_x \sim$ Bernoulli(p_x^+)
r_y and r_z are updated similarly

Algorithm 2 Adaptive routing

Input: s, d, r
Output: path from s to d
$x_d, y_d, z_d \leftarrow$ coordinates of d
$x, y, z \leftarrow x_s, y_s, z_s$
$path \leftarrow \emptyset$
while $(x, y, z) \neq (x_d, y_d, z_d)$ **do**
  calculate productive dimensions
  choose shorter queue length
  $path \leftarrow path +$ next hop
end while
return $path$

Flow Control

Virtual Channels

Our implementation of GOAL employs 4 VCs per physical channel to achieve deadlock freedom in the network. We group the 4 VCs into 3 pairs, $*_0$, $*_1$, and non-$*$.

- non-$*$ channels are fully adaptive and can be used at any time.
- $*$ can be used only when the packets are traversing the *most significant* productive dimension.
- Packets will (will not) move through $*_1$ ($*_0$) when crossing a wrap-around edge.

III. Implementation

Routing

Adaptive Routing

For Goal routing algorithm, we implement `OutputUnit::count_free_vc`. That's why GOAL routing can find the most vc_free outport and chose it.

```cpp
namespace garnet
OutputUnit::has_free_vc(int vnet, bool is_ring, bool is_ring_checkpoint, bool is_torus, std::vector<bool> is_torus_dims_checkpoint, bool is_minimal_torus, Routing
    int found = get_dim_of_direction();
    if (is_ring) {
        if (is_ring_checkpoint) {
            for (int vc = vc_base; vc < vc_base + m_vc_per_vnet/2; vc++) {
                if (is_vc_idle(vc, curTick()))
                    return true;
            }
        }
        else {
            for (int vc = vc_base + m_vc_per_vnet/2; vc < vc_base + m_vc_per_vnet; vc++) {
                if (is_vc_idle(vc, curTick()))
                    return true;
            }
        }
    }
    else if (is_torus) {
        if (found < 0 ){
            for (int vc = vc_base; vc < vc_base + m_vc_per_vnet; vc++) {
                if (is_vc_idle(vc, curTick()))
                    return true;
            }
        }
        if (routing_algorithm == Goal_ || routing_algorithm == Min_AD_) {
            for (int vc = vc_base; vc < vc_base + m_vc_per_vnet/3; vc++) {
                    if (is_vc_idle(vc, curTick()))
                        return true;
            }

            if (is_torus_dims_checkpoint[found]) {
                for (int vc = vc_base + m_vc_per_vnet/3; vc < vc_base + m_vc_per_vnet*2/3; vc++) {
                    if (is_vc_idle(vc, curTick()))
                        return true;
                }
            }
            else {
                for (int vc = vc_base + m_vc_per_vnet*2/3; vc < vc_base + m_vc_per_vnet; vc++) {
                    if (is_vc_idle(vc, curTick()))
                        return true;
                }
            }
        }

```

Flow Control

Virtual Channels

We maintain an `std::vector<bool>` in `RouteInfo`, each element assigned with a dimension and set to be true when the flit goes through a wrap-around edge.

Before send the flit, GOAL will call `route_compute` again and calculate new outport in time. This opreation avoids deadlock.

```cpp
namespace garnet
RoutingUnit::outportComputeTorus(RouteInfo route,
        // Create a random number generator
        std::random_device rd;  // Obtain a random number from hardware
        std::mt19937 gen(rd()); // Seed the generator
        std::uniform_real_distribution<> dis(0.0, 1.0);

        route.directions.resize(num_dims); // Resize to hold num_dims elements
        for (int i = 0; i < num_dims; ++i) {
            if (pos_probability[i] == 1) {route.directions[i]=0; continue;}
            double random_number = dis(gen);
            route.directions[i] = (random_number < pos_probability[i]) ? 1 : -1;

            // if (pos_probability[i] >0.5) route.directions[i] = 1;
            // else route.directions[i] = -1;// for testing

            if (pos_probability[i] >0.5 && route.directions[i] == 1) {route.is_minimal_torus = true;}
            else if (pos_probability[i] <0.5 && route.directions[i] == -1) {route.is_minimal_torus = true;}

        }
    }

    std::vector<unsigned> check_outports_idxs;
    for (int i = 0; i < num_dims; ++i) {
        if (dest_coords[i] == my_coords[i]) {
            route.directions[i] = 0;
            continue;
        }
        PortDirection temp_outport;
        if (route.directions[i] == 1) {
            temp_outport = "dim" + std::to_string(i) + "_pos";
        } else if (route.directions[i] == -1) {
            temp_outport = "dim" + std::to_string(i) + "_neg";
        }
        check_outports_idxs.push_back(m_outports_dirn2idx[temp_outport]);
    }

    // check these outports, count their vcs and pick the one with minimum occupation in vc.
    int max_vc_free = -1;
    int outport = check_outports_idxs[0];
    RoutingAlgorithm routing_algorithm =
        (RoutingAlgorithm) m_router->get_net_ptr()->getRoutingAlgorithm();
    for (int i = 0; i < check_outports_idxs.size(); ++i) {
        auto output_unit = m_router->getOutputUnit(check_outports_idxs[i]);
        int vc_free = output_unit->count_free_vc(route.vnet,false,false,true,route.is_torus_dims_checkpoint, route.is_minimal_torus, routing_algorithm);
        if (vc_free > max_vc_free) {
            max_vc_free = vc_free;
            outport = check_outports_idxs[i];
        }
    }

    return outport;
}
```

IV. Experiment Result

Experiment Setting

- network architecture: 4-ary 3-cube
- flow control: 4 VCs per physical channel
- traffic patterns: *uniform random*, *torus tornado*, *torus transpose*, *shuffle*, *bit reverse* and *bit rotation*.
- compared topologies: Ring, Mesh_XY
- sim-cycles: 20000

Baselines

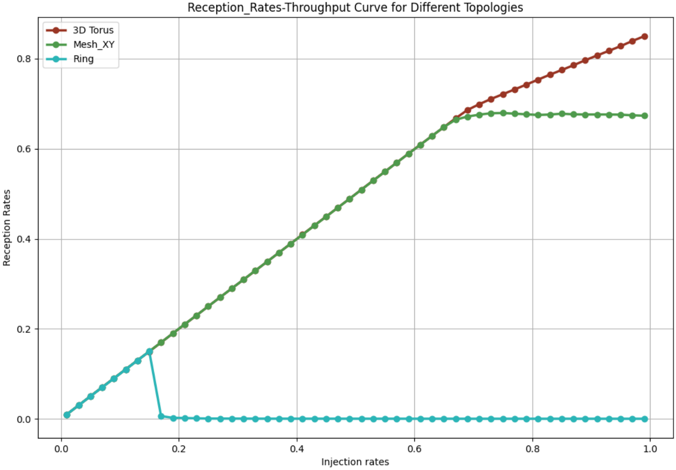
We use the following routing algorithms as baselines:

baseline

- *default TABLE* is a minimal routing algorithm that is implemented in `gem5`. It does not include any flow control.
- *Dor* is a minimal routing algorithm that routes packets along the shortest path from dimension X to Z. We expand 2 VCs for each physical channel.
- *Minimal Adaptive* first decides the closest quadrant to the destination and then adaptively routes the packet within the quadrant. We expand 3 VCs for each physical channel.

Topology Comparison under Uniform Random Traffic

In this page we show comparisons on reception rates of three topologies.

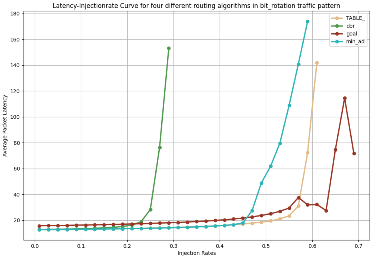


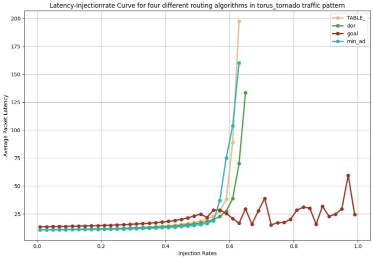
Max Reception Rate on all traffic patterns

In this page we show the saturation throughput of the four algorithms on a 4-ary 3-cube torus network on each traffic pattern.

Average Latency on Injection Rate

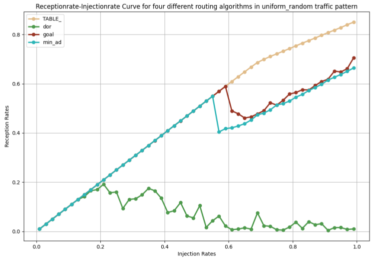
In this page we show the latency changing with injection rate of the four algorithms under *bit rotation* and *torus tornado* respectively.

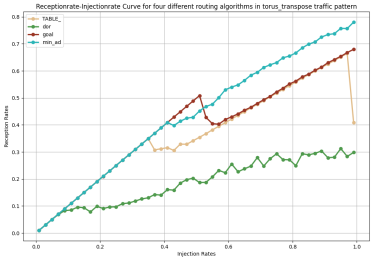




Reception Rate on Injection Rate

In this page we show reception rate changing with injection rate under *uniform random* and *torus transpose* respectively.
It can be seen that GOAL is relatively robust under various traffic patterns and has a satisfying lower bound empirically.





Conclusion

- The torus topology has a better performance than the ring and mesh topology because of its high path diversity, specifically, it has lower hops, higher and more stable throughput.
- The GOAL algorithm can balance the load among the quadrants selection and reduce the congestion in the network, which makes it perform more stable than other algorithms when the injection rates are increasing. But correspondingly, this algorithm sacrifices the low load performance.

Division of Labor

- **Ziren Wang:** Implementation of the Torus topology, GOAL routing algorithm, FlowControl and the evaluation of the performance of the Torus topology and GOAL. Chapter 4- of report. PPT presentation.
- **Chuan Liu:** Implementation of Dimension Order Routing (Dor) baseline. the evaluation of the performance of the Torus topology and GOAL. Chapter 1-3 of report. PPT producing and PPT presentation.

References

- Pablo E Berman, Luis Gravano, Gustavo D Pifarre, and Jorge LC Sanz. 1992. Adaptive deadlock-and livelock-free routing with all minimal paths in torus networks. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*. 3-12.
- Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh. 2022. *On-chip networks*. Springer Nature.
- Arjun Singh, William J Dally, Amit K Gupta, and Brian Towles. 2003. GOAL: a load balanced adaptive routing algorithm for torus networks. *ACM SIGARCH Computer Architecture News* 31, 2 (2003), 194-205.
- Leslie G. Valiant. 1982. A scheme for fast parallel communication. *SIAM journal on computing* 11, 2 (1982), 350-361.