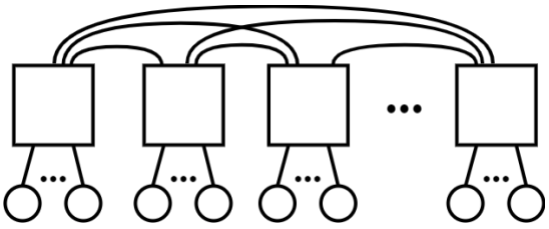
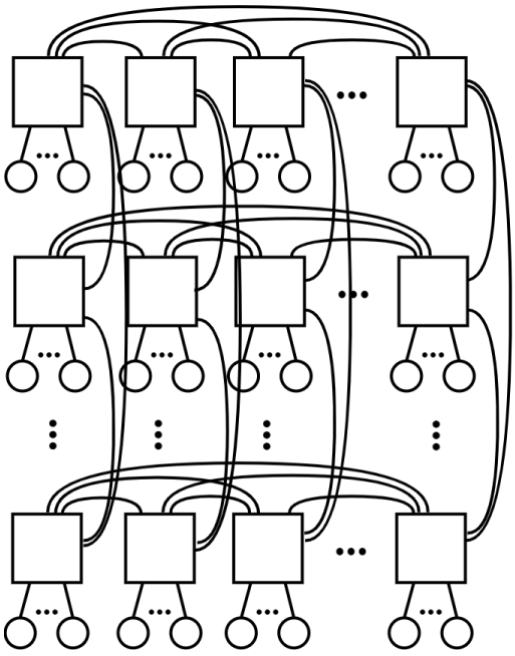
Task 0: Preliminaries

For this task, we will play around with Flattened Butterfly network from *"Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks"* paper.



1D Flattened Butterfly



2D Flattened Butterfly

List of important BookSim parameters:

Parameter	Definition
`k`	Number of routers per dimension
`n`	Number of dimensions
`c`	Concentration: number of nodes per router
`num_vcs`	Number of virtual channels
`vc_buf_size`	Buffer size per virtual channel
`alloc_iters`	Number of maximum allocation iterations at each cycle
`internal_speedup`	Ratio of router cycle over network cycle.
`routing_delay`	If set to 0, lookahead routing is used. If > 1, no lookahead routing.
`credit_delay`	The duration of time that the credit is held before being sent to the upstream router.
`use_noc_latency`	If set to 1, the channel latency will vary following the distance between each router. If set to 0, the latency of all channels is uniform.

Before proceeding, please remove 2 assertions in `networks/flatfly_onchip.cpp`, as shown below:

```
void FlatFlyOnChip::_ComputeSize( const Configuration &config )
{
  _k = config.GetInt( "k" );     // # of routers per dimension
  _n = config.GetInt( "n" );     // dimension
  _c = config.GetInt( "c" );     //concentration, may be different from k
  _r = _c + (_k-1)*_n ;          // total radix of the switch  ( # of inputs/outputs)

  //how many routers in the x or y direction
  _xcount = config.GetInt("x");
  _ycount = config.GetInt("y");
  //assert(_xcount == _ycount);
  //configuration of hohw many clients in X and Y per router
  _xrouter = config.GetInt("xr");
  _yrouter = config.GetInt("yr");
  //assert(_xrouter == _yrouter);
  gK = _k;
  gN = _n;
  gC = _c;
```

Don’t forget to run `make` after making any changes to the code (but not the configuration file).

Task 1: Measuring Performance

In network, performance is often shown using latency-throughput graph, which essentially shows the increase in average packet latency as more loads are given to the network. From the latency-throughput graph, we can also observe the zero-load latency and saturation throughput. Zero-load latency is the average latency when the injection rate is low (near zero), which gives us the lower bound of the average packet latency. Saturation throughput tells us the offered load where the network starts to *saturate*. At saturation, the average packet latency will continue to increase indefinitely.

Step 1: Configuration file

We will slightly modify the configuration file: `examples/flatflyconfig`. The variables that need to be changed / added are shown below.

Change buffer size and the number of virtual channels.

```
num_vcs     = 2;
vc_buf_size = 16;
```

Provide speedup for the routers by letting the routers operate more than once during each network cycle.

```
alloc_iters  = 3;
internal_speedup  = 1.6;
```

For simplicity, we won't use lookahead routing and no credit delay.

```
credit_delay   = 0;
routing_delay  = 1;
```

Define a 256-node 1D Flattened Butterfly.

```
c  = 16;
k  = 16;
n  = 1;

x  = 16;
y  = 1;
xr = 16;
yr = 1;
```

Assume that all wires has uniform latency

```
use_noc_latency = 0;
```

Step 2: Run simulation using script

To obtain a latency throughput graph, we have to run BookSim with different loads, which can be done by configuring the `injection_rate` parameter. However, instead of changing the parameter manually, we will utilize *shell scripting* to automate the simulations.

Create a script file (example name: `run_sim.sh`) in your BookSim directory. This script runs simulation on the 1D Flattened Butterfly network with uniform random traffic. The loads are varied from 0.025 to 1.00 flits / cycle with 0.025 increment.

```
for i in $(seq 0.025 0.025 1.000)
do
        ./booksim examples/flatflyconfig sim_type=latency
traffic=uniform injection_rate=${i} > latency_uni_ff_${i} &
done
```

Run `chmod +x run_sim.sh` to make the script file executable.

To run the simulations, simply run `./run_sim.sh` in your shell terminal.

Step 3: Obtain data using script

The script above will dump the report to text files. Since we don't want to open each report file manually, we can also create a script to help us obtain the data from the reports.

Make another script file (example name: `show_sim.sh`) in your BookSim directory.

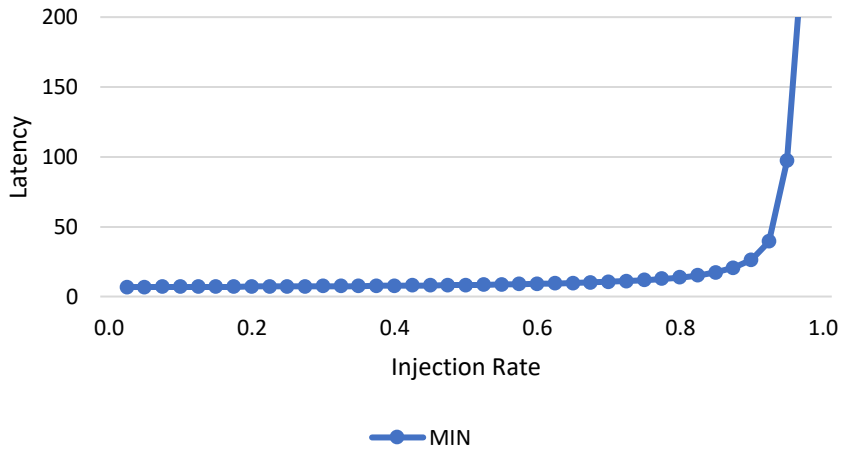
```
for i in $(seq 0.025 0.025 1.00); do grep 'Packet latency average'
latency_uni_ff_${i} | grep sample | awk '{print $5}'; done
```

Run `chmod +x show_sim.sh` to make the script file executable.

To show all the data, simply run `./show_sim.sh`. You should be able to get the average packet latency for each load, as shown below. If the network is saturated, the script does not return any average packet latency value.

```
[hanskasan@intel03 src]$ ./show_uni_ff
4.9616
4.98634
5.01523
5.04537
5.0769
5.11091
5.14367
5.18021
5.22181
5.26341
5.30845
5.35638
5.40956
5.46829
5.53166
5.60075
5.67625
5.76085
5.85601
5.95945
6.07214
6.2041
6.35173
6.52292
6.72186
6.94473
7.2073
7.52814
7.92025
8.39137
8.98918
9.80524
10.8642
12.4711
14.8741
19.7244
30.7625
91.1348
248.154
408.789
```

Plot these numbers in Excel (or any graph drawing tools) to draw the latency vs. throughput graph!



Task 2: Topology Configuration

For task 1, we simulated a 256-node 1D Flattened Butterfly with **1-cycle** channel latency. For task 2, we will obtain the latency-throughput graph for:

- 256-nodes 1D Flattened Butterfly, 50-cycle channel latency
- 216-nodes 2D Flattened Butterfly, 1-cycle channel latency
- 216-nodes 2D Flattened Butterfly, 50-cycle channel latency

To increase the channel latency (for both 1D and 2D Flattened Butterfly), we have to modify the `networks/flatfly_onchip.cpp` code, as shown below. For comparison purpose, we will change the latency value to 50.

```
#ifdef DEBUG_FLATFLY
        cout << "Adding channel : " << _output << " to node " << node <<" and "<<other<<" with length "<<length<<endl;
#endif

        if(use_noc_latency){
          _chan[_output]->SetLatency(length);
          _chan_cred[_output]->SetLatency(length);
        } else {
          _chan[_output]->SetLatency(1);
          _chan_cred[_output]->SetLatency(1);
        }
        _routers[node] ->AddOutputChannel( _chan[_output], _chan_cred[_output]);

        _routers[other]->AddInputChannel( _chan[_output], _chan_cred[_output]);

        if(gTrace){
          cout<<"Link "<<_output<<" "<<node<<" "<<other<<" "<<length<<endl;
        }
```

Change the latency value here!

To simulate a 2D Flattened Butterfly, we just need to make slight modifications to the configuration file.

```
c  = 6;
k  = 6;
n  = 2;

x  = 6;
y  = 6;
xr = 6;
yr = 1;
```

To simulate with long channel latency, we need to use deeper buffers to cover the credit round-trip latency (Dally's textbook, Section 13.3). Since the channel latency is 50 cycles, we have to make sure that the buffer depth is at least 100, which is twice the channel latency.

```
num_vcs     = 2;
vc_buf_size = 128;
```

Questions

1. As larger channel latency is used, the zero-load latency also increases. Why is this the case? (*Hint: the average packet latency at very low load, e.g., ≤0.025, is a good estimate of the zero-load latency)*
2. For both 1D and 2D Flattened Butterfly, we increased the channel latency from 1 to 50 cycles. However, you will notice that the increase in zero-load latency is higher in 2D than 1D Flattened Butterfly, even though the increase in channel length is exactly the same. Why is this the case?