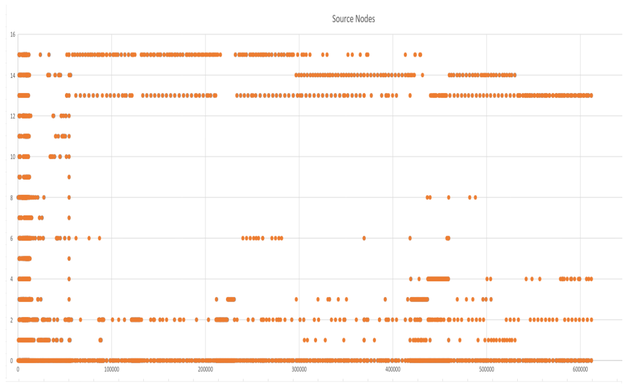
Reconfigurable Optical Networks-on-Chip

Nicholas Conrad (Elon University), Lei Zhang (UMES)

This work is supported by NSF.

ABSTRACT

We designed a simulator to test the re-configurability of an optical N-o-C for an Ouroboros network. The simulator receives a data file of requests to process and then simulates how an actual architecture of this design would function giving us (cost of transmissions and wait times) to answer two questions: “How should a network be reconfigured based on a file?” and “Is it worth it?”.



Graphical Visualization of a data file. On the left are the cores which core 0 is communicating with, with respect to time (clock cycles, on the x-axis)

OUROBOROS NETWORK

An Optical Network has a Head Core (HC) and multiple Body Cores. They are all connected to a ring that changes how the cores connect to each other based on wave guides. Each core can be connected or it can be bypassed. With this setup data can be transferred between any two cores as long as all of the cores in between those two are bypassed. The HC coordinates and controls the placement of these nodes and all of the data flow. With this type of network multiple sets of data can be run in parallel.

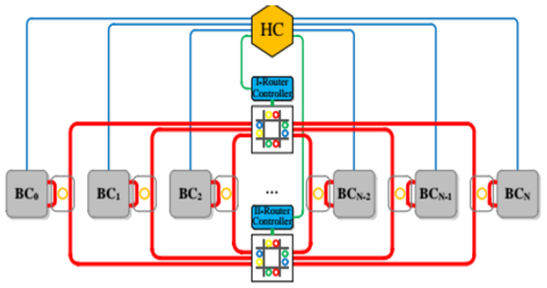
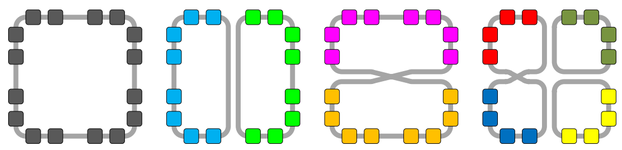


Fig. 1. Reconfigurable Ouroboros Network Overview

Reconfiguration

The network has two basic reconfiguration modes, ‘Order’ and ‘Territory’. The ‘Order’ mode allows for any or all nodes to change position to a specified location, thereby allowing two nodes that communicate a lot to be right next to each other. In the ‘Territory’ mode, the network can be split into multiple subnetworks then interconnected to allow the same result. The simulator I designed uses ‘Order’.



Examples of a 16-node ON Reconfiguration

Isolation, Tetris, Wormhole Rules

Three rules must be followed.
Isolation: If requests share either destination or source nodes then the first one has to be scheduled and finished before the second.
Tetris: A request can start as early as possible if the isolation rule is followed.
Wormhole: A request can cross existing channels and fit into a hole as long as the isolation rule is followed.

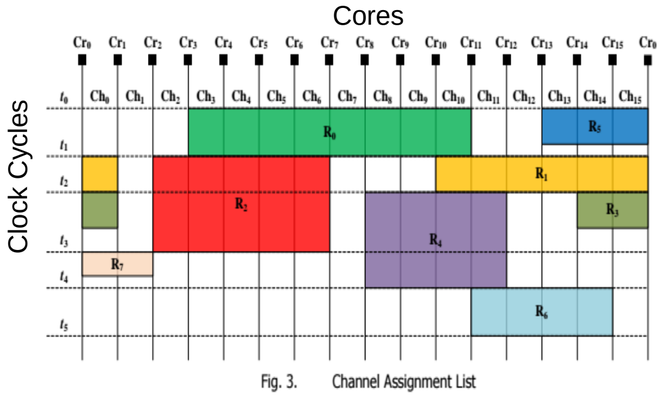
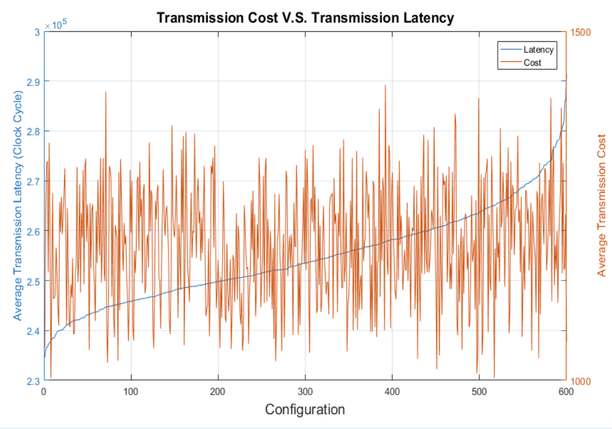


Fig. 3. Channel Assignment List

Simulator

Separated into three files written in python 2.7: Config.py, Request.py, Simulation2.py.
The config.py file handles all constants related to the machine being used and options to change how the data is looked at to determine the change in performance. One of these constants is one that controls wave-division-multiplexing (allowing more than one frequency to travel in the same space).
The request file handles the adding and removing of requests on and off the network(s).
The simulation file handles the reconfiguration of the nodes. It keeps the timing of all of the data. Controls rule coherence.



Results

Through the testing of several files with different configurations we have found that we can reduce the wait time by a huge amount. We have also determined that wave-division-multiplexing can reduce the wait time drastically depending on the configuration and file. Our third conclusion, to our surprise, is that cost and waiting time are not related at all! Instead, to reduce waiting time and figure out an optimal configuration we must base it off the timing and overlapping of requests.

SUMMARY

Creating this simulator allows us to see that a reconfigurable optical network-on-chip can drastically improve the performance of different programs. It allows us to see that changing the placement of the nodes (depending on an incoming file), and adding frequencies to allow for wave-division-multiplexing (depending on the file and configuration) reduce wait time immensely and are definitely worth it. Now we will need to determine how to find the optimal attributes and configurations based on the incoming file.