GR- Router : A Distributed GNU Radio Framework

Tommy Tracy II

Computer Engineering Graduate Student

Professor Mircea Stan

HPLP Advisor

Alliant Techsystems Inc. (ATK)

Funding Organization

University of Virginia

School of Engineering and Applied Science

hplp

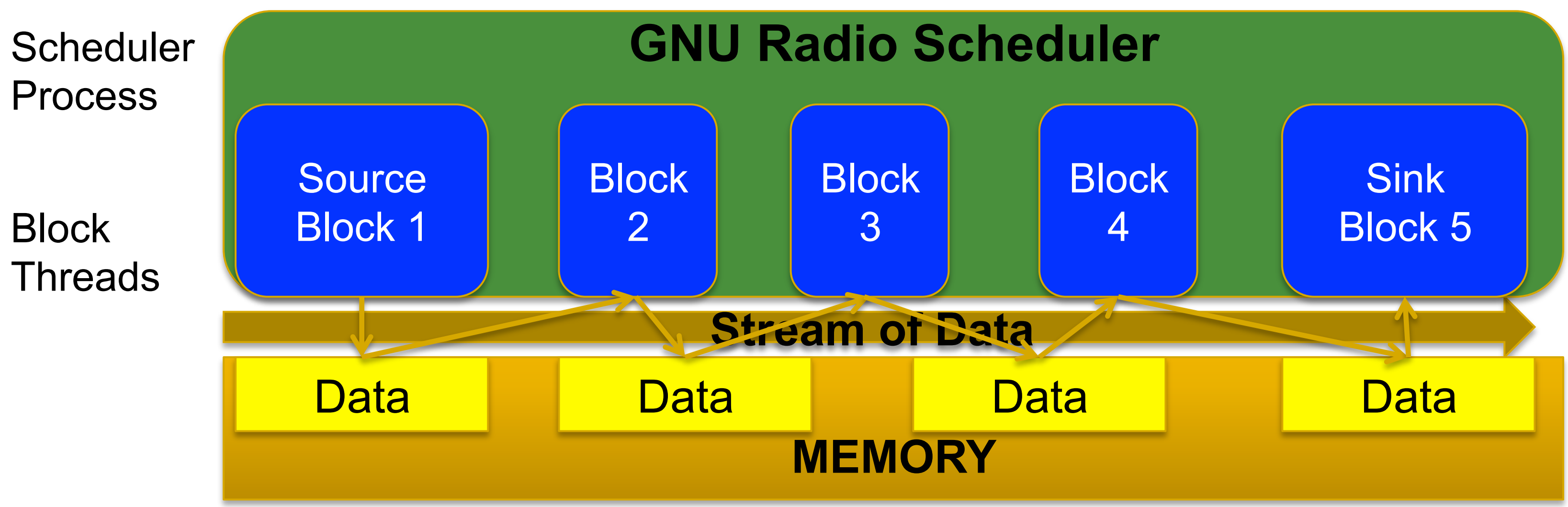
High-Performance Low-Power

Outline

- **Background**
- **Problem Statement** : Limited scalability in high-throughput shared-memory SDR system.
- **Solution** : Distribute the SDR system with GR-Router.
- **Example Implementation** : Test with GR-LDPC.
- **Future Work**

Background

- **GNU Radio**
  - Is an open-source software development toolkit.
  - It provides DSP blocks for SDR applications.
  - Each GNU Radio block is spawned as an independent processing thread which uses inter-thread communication mechanisms (via the scheduler) to share data between blocks.



The Problem

- Funding Source: Confidential high-throughput SDR communications system (multi-MIMO) that runs on a high-end, multi-core x86 processor (dozens) server with high-end memory configuration.

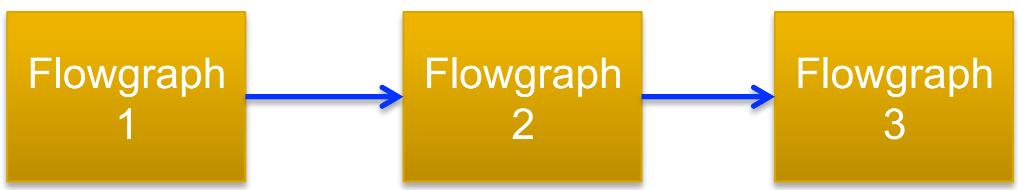
- They found that their throughput scales poorly as CPU count and memory are increased; they could not support additional real-time channels by introducing more hardware (within reason).

- After debugging and analysis, the problem was found to be <u>memory bandwidth contention</u>.

- Problem: Shared-memory system scales poorly with thread count.

Potential Solutions

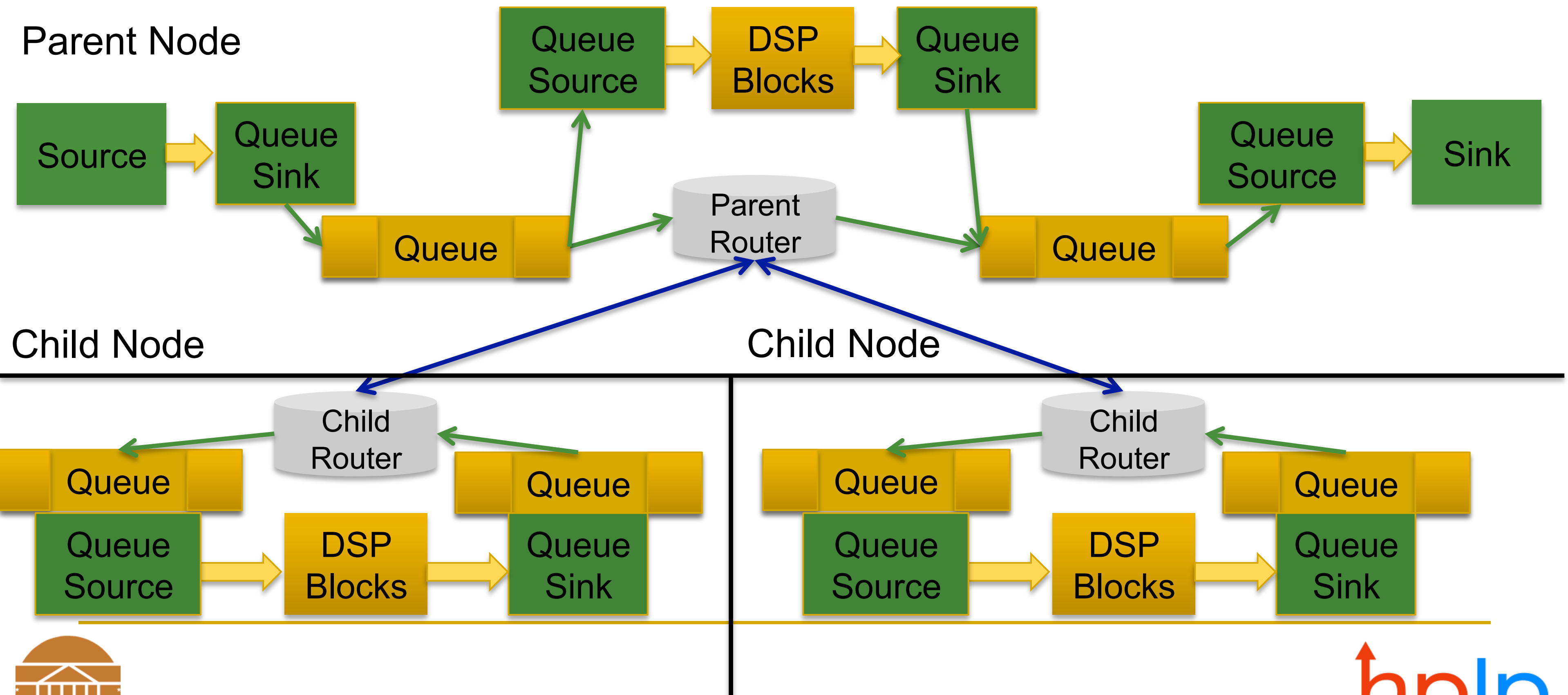
- Distribute block threads across disjoint memories and then use message passing between partitions.
  - Use existing TCP/UDP blocks and break flowgraph into networked chain
    - No dynamic load balancing; bottleneck slows down the entire system.



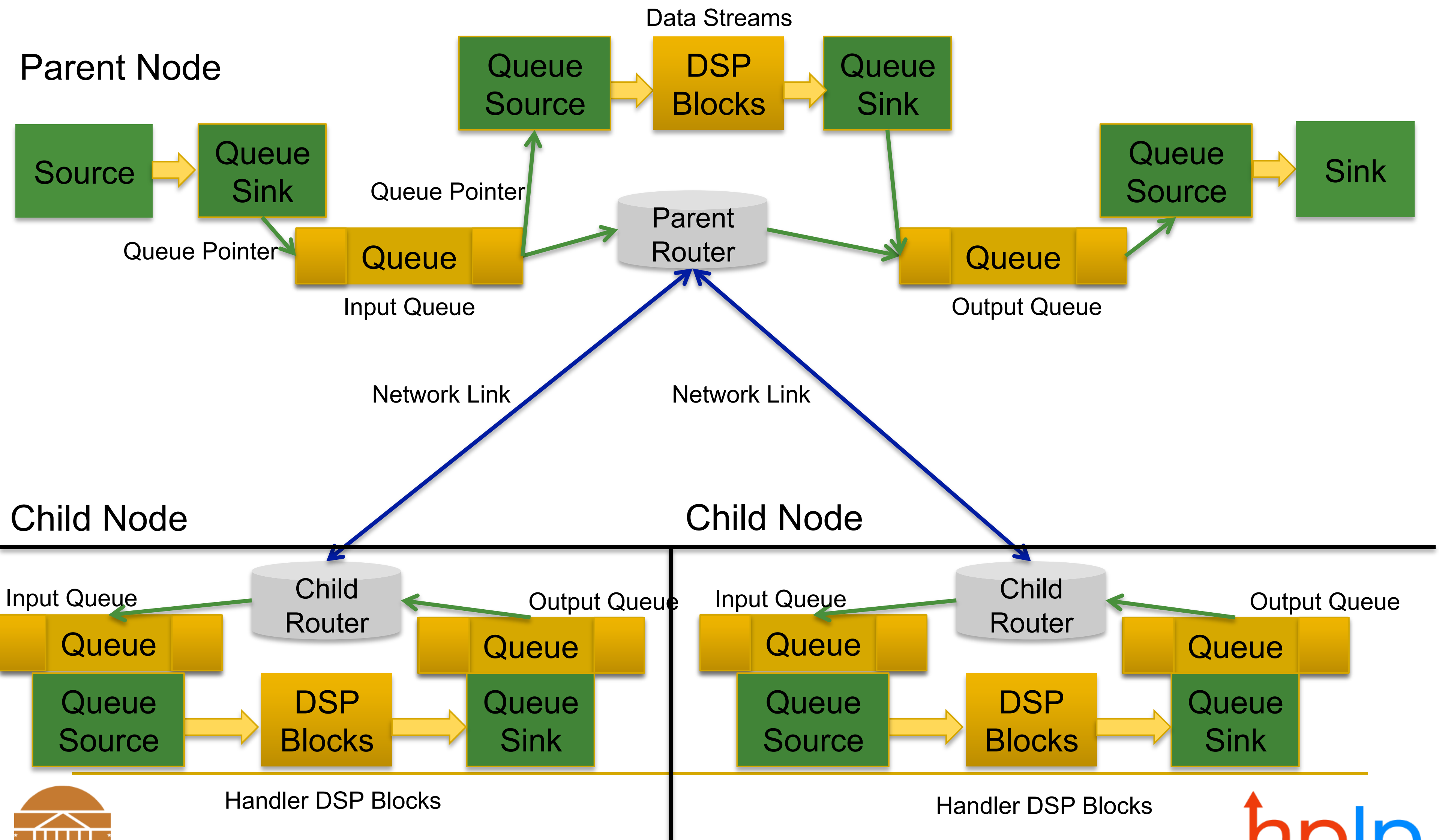
  - Instead use dynamic load-balancing to distribute the workload across multiple nodes; with the nodes running redundant blocks.

Solution: GR-Router

- GR-Router: Distributed GNU Radio framework
- Distributes the DSP workload across multiple machines tied to a common communication network



Terminology



Parent Code

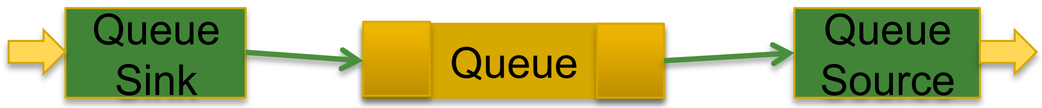
```
// Populate input queue
tb->connect(src, 0, throttle_0, 0);
tb->connect(throttle_0, 0, input_queue_sink, 0);

// Handler Code
tb->connect(input_queue_source, 0, decoder, 0);
tb->connect(decoder, 0, unpacked2packed, 0);
tb->connect(unpacked2packed, 0, output_queue_sink, 0);

//Order and sink results
tb->connect(output_queue_source, 0, throughput, 0);
tb->connect(throughput, 0, sink, 0);
tb->run();
```

Child Code

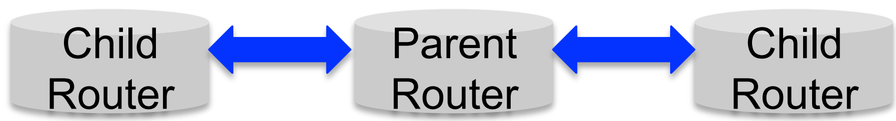
```
// Handler code
tb->connect(input_queue_source, 0, decoder, 0);
tb->connect(decoder, 0, unpacked2packed, 0);
tb->connect(unpacked2packed, 0, output_queue_sink, 0);
tb->run();
```

Blocks in GR-Router

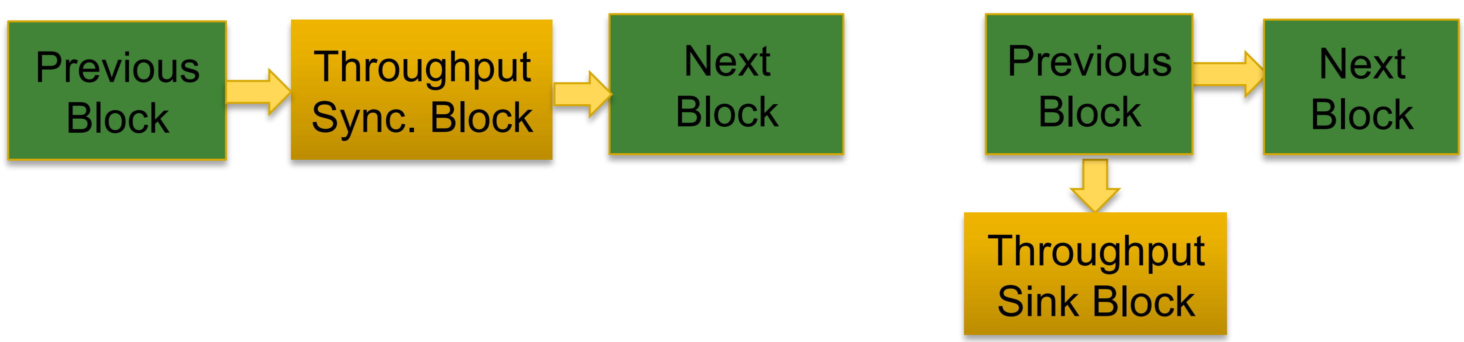
- Queue Source and Sink Blocks



- Parent & Child Router Blocks

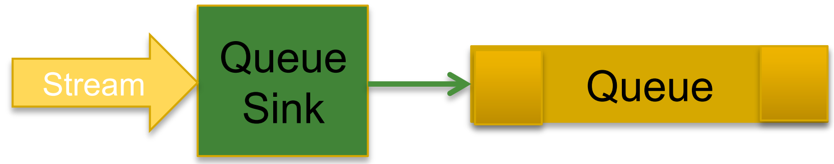


- Sync and Sink Throughput blocks



Queue_Sink Blocks

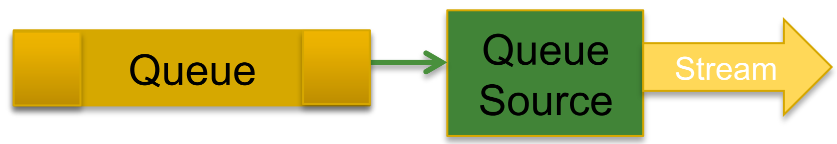
- **Queue Sink**: Converts stream to segments; pushes them onto queue
  - **Input**: Stream of data
  - **Output**: Pointer to queue
  - Supports reconstructing index
- **Segments**: Defined within the queue sink and source



Type 1 Msg.	type	index	size	Data[] : 768 floats	
Type 2 Msg.	type	index	size	Data[] : 50 bytes	
Type 3 Msg.	type	index	size	Data[] : 50 bytes	weight

Queue_Source Blocks

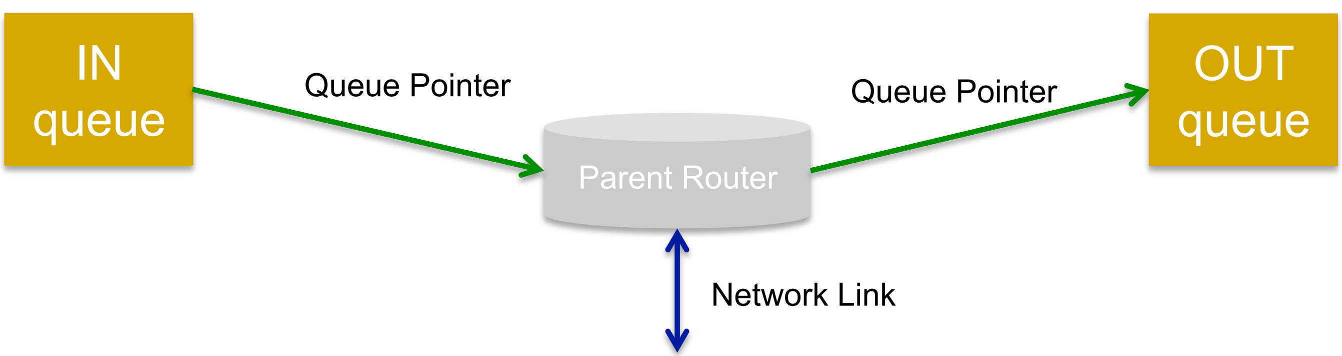
- **Queue Source**: Converts segments from a queue to streaming data
  - **Input**: Pointer to queue
  - **Output**: Stream of data
  - Supports ordering by index, and preserving index across stream with stream tags.



Type 1 Msg.	type	index	size	Data[] : 768 floats	
Type 2 Msg.	type	index	size	Data[] : 50 bytes	
Type 3 Msg.	type	index	size	Data[] : 50 bytes	weight

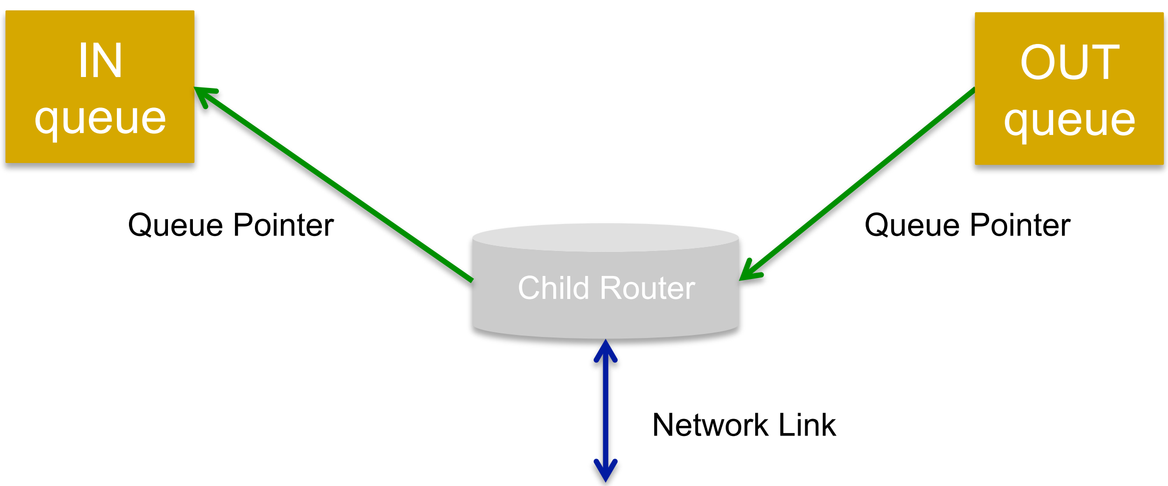
Parent_Router Block

- **Parent Router**: evenly distributes data segments among children
  - Keeps track of childrens' weights and distributes segments from the IN queue to balance the network.
  - Receives result segments from children (containing their 'weight')
  - **Input**: pointer from IN queue
  - **Output**: pointer to OUT queue
  - **Network Link**: Connection to Children.



Child_Router Block

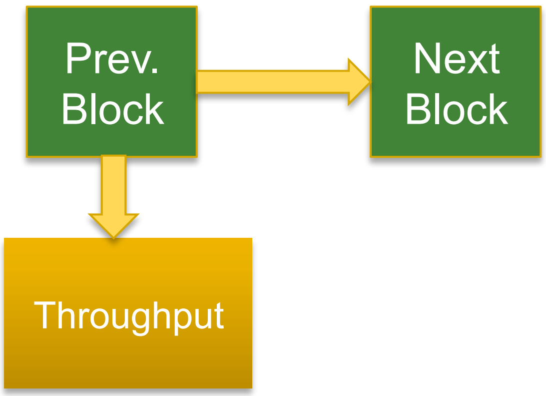
- **Child Router**: computes segments and returns results to Parent with weight.
  - Keeps track of current ‘busy-ness’, and sends weight to Parent
  - **Input**: Pointer from OUT queue
  - **Output**: Pointer to IN queue
  - **Network Link:** Connection to parent.



{Inline, Sink}_Throughput Blocks

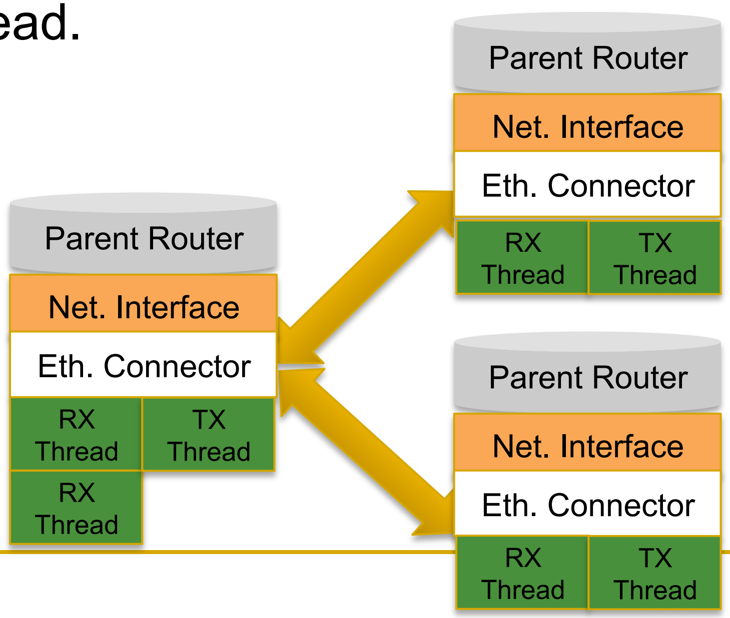
- **Inline Block**: In-line transparent block to calculate throughput
  - **Input:** Previous block
  - **Output:** Next block

- **Sink Block**: Out-of-line sink block used to calculate throughput
  - **Input:** Previous block
  - **Output:** None



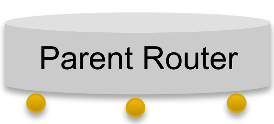
Communications

- We used Ethernet for communication between machines.
  - GR-Router was designed to work with any available communication technologies.
  - For each node there is a receive(RX) thread per child router, and a single transmit(TX) thread.

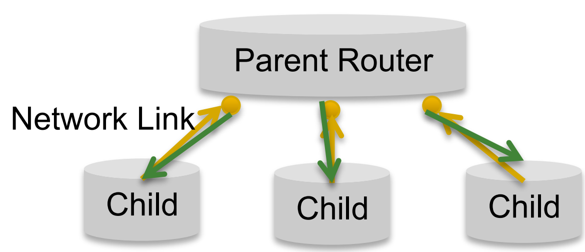


Network Interface

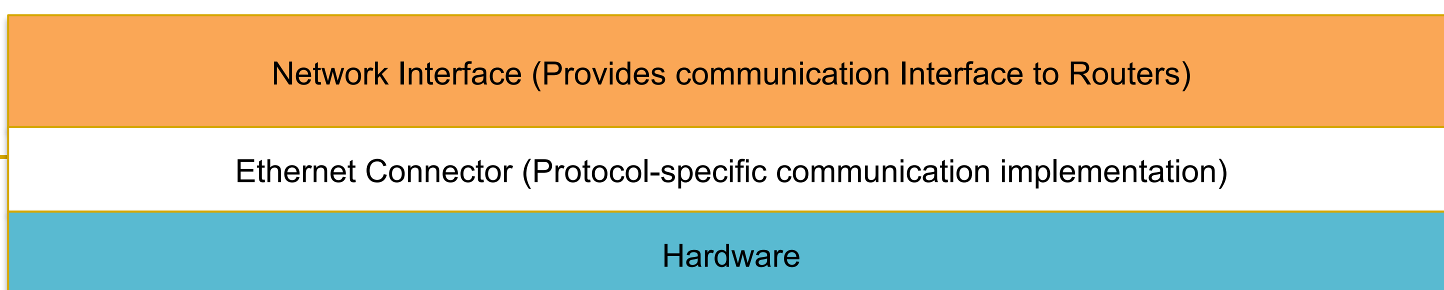
- Network Interface: Provides communication functions to Parent and Child Routers
  - **Connect**(char* hostname): connects the graph of routers
    - Each child specifies it's parent, and each parent knows the number of children it connects to.
  - **Receive**(int index, char* outbuf, int num_items): receives segments from node at index
  - **Send**(int index, char* msg, int num_items): sends message to given node
  - Uses Connector for communications; is only meant as a uniform interface for the routers to abstract away the technology-specific implementation



Wait for <number_children> connections …



After child has connected to parent, it waits for it's children to connect



Ethernet Connector

- Ethernet Connector: Ethernet-specific networking class
- Provides the following functions:
  - connect_to_parent(char* hostname, int port)
  - write_parent(char* msg, int size), read_parent(char* outbuf, int size)
  - connect_to_child(int index, int port)
  - write_child(int index, char* inbuf, int size), read_child(int index, char* outbuf, int size)
  - Stop()
- In order to use a different technology (PCIE), this connector would need to be written for that technology

Ethernet Connector (Protocol-specific communication implementation)

Hardware (Ethernet, PCIe, etc)

Example Implementation

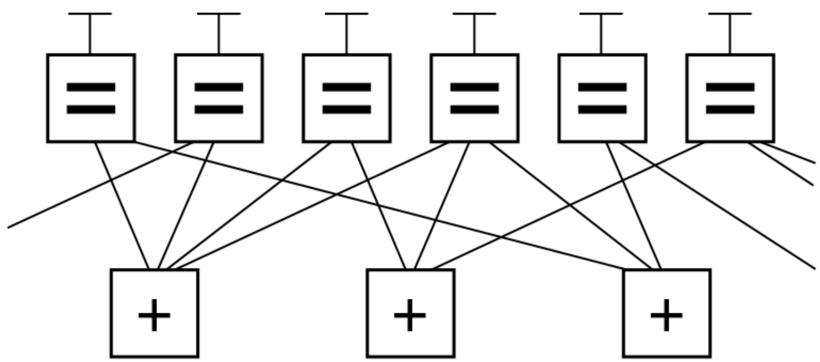
- **GR-LDPC** : Manu T S wrote an implementation of a Low-Density Parity-Check Code encoder and decoder for GSoC 2013.

- Goal: Parallelize his decoder to increase net throughput.

http://gnuradio.squarespace.com/storage/grcon13_presentations/grcon13_manuts_gsoc_ldpc.pdf

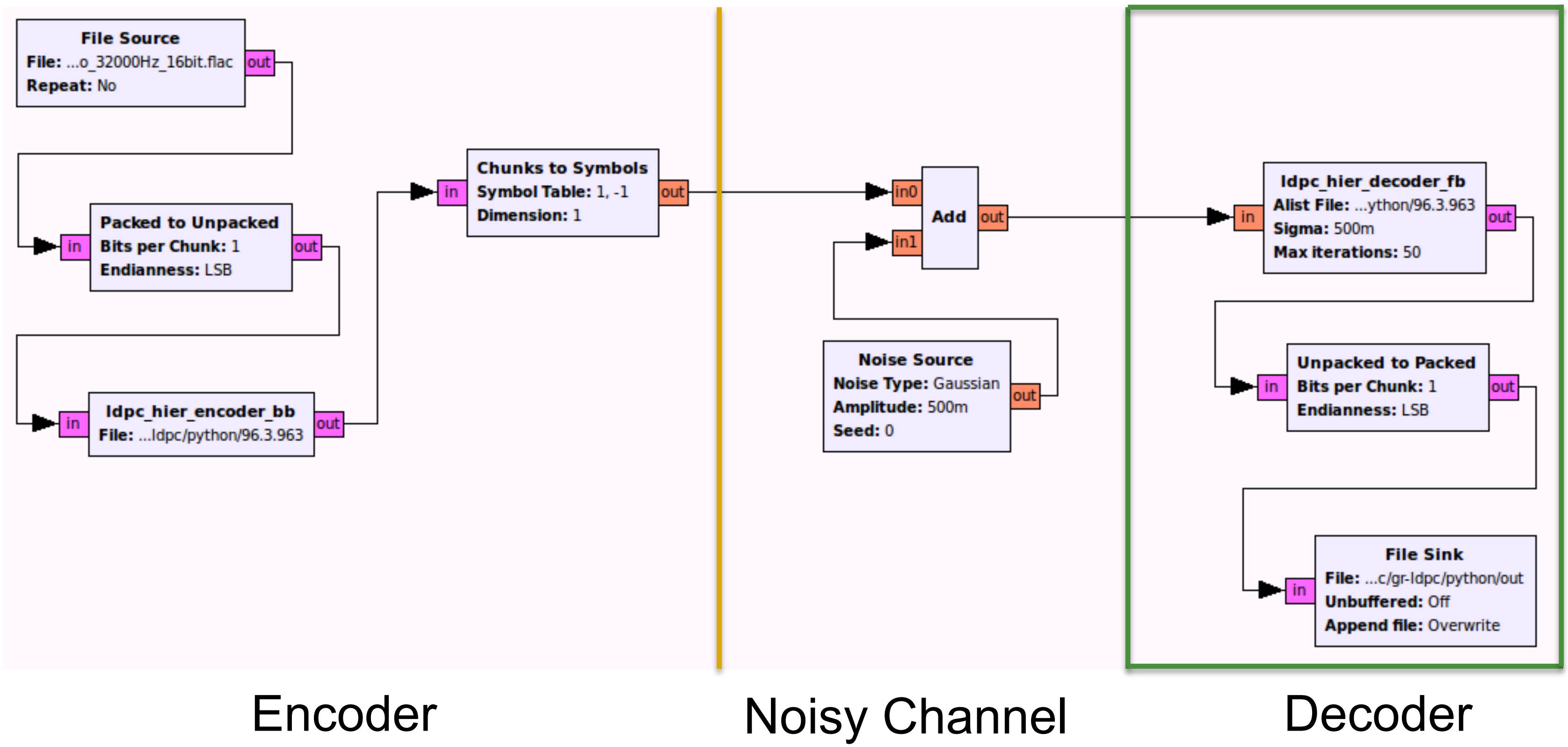
LDPC Decoding

- **LDPC**: forward error correcting code using a soft decision decoder to 'guess and check' until a valid code word is found.
  - once found, the data can be calculated or a LUT can do the translation.
- It makes these decisions independently on the n-bit code word. Therefore LDPC is data-parallelizable. Multiple code-words can be decoded in parallel.



http://en.wikipedia.org/wiki/Ldpc

Manu's LDPC Example

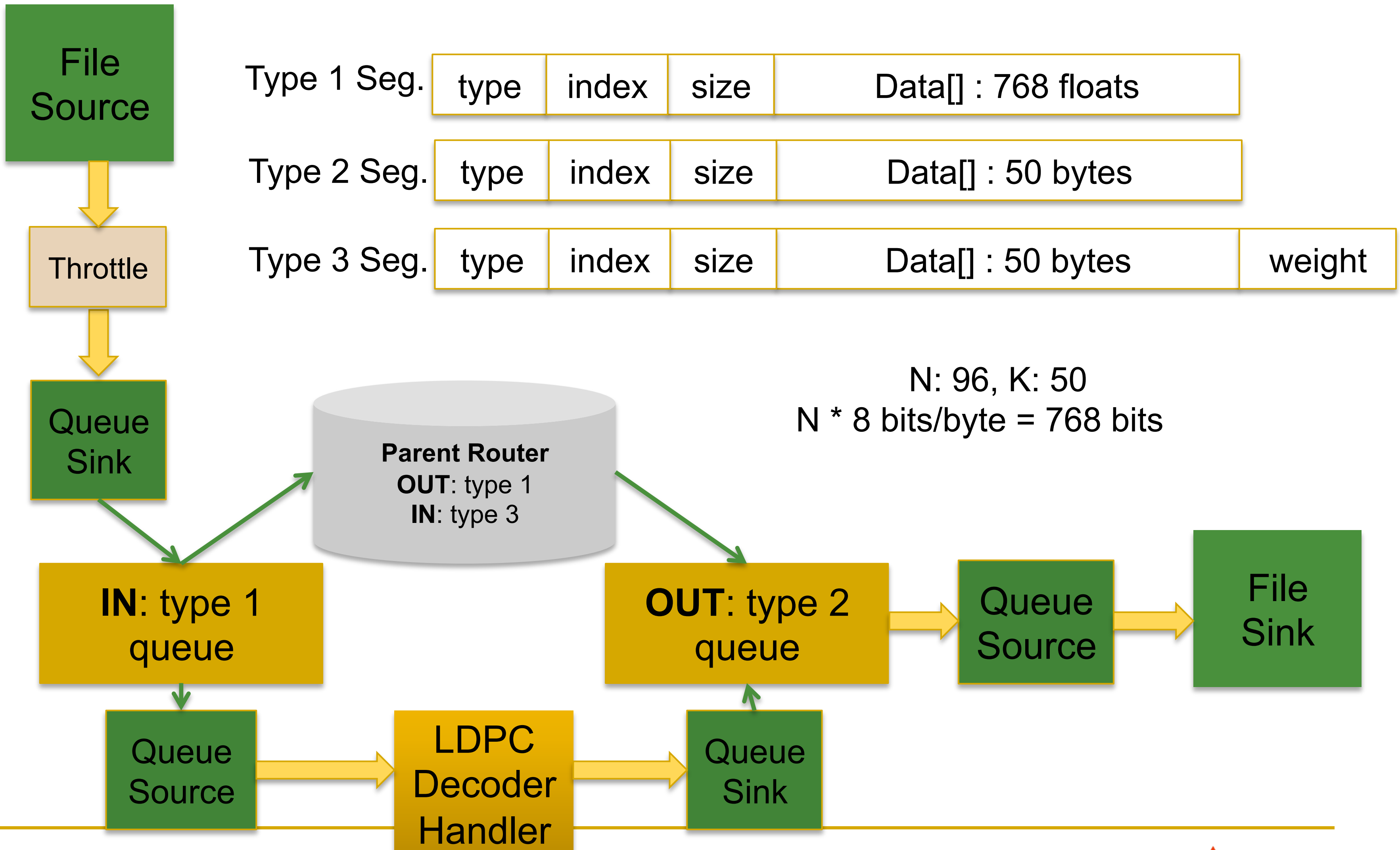


https://github.com/manuts/gr-ldpc

Parallelization Strategy

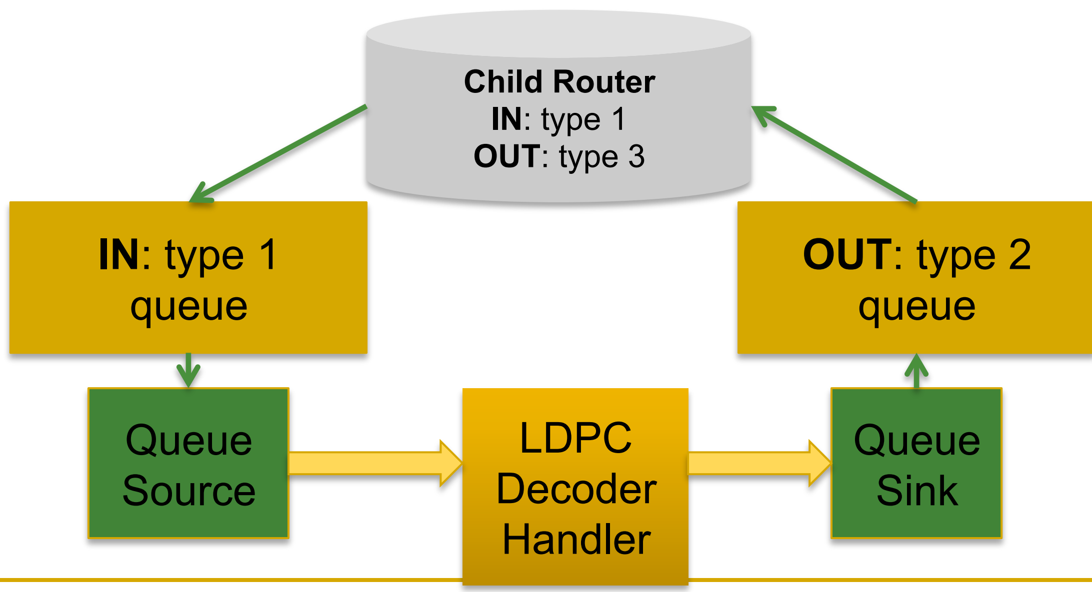
- 1. Port Manu's code to C++
- 2. Create no-noise and 0.5 pk-pk LDPC-encoded files.
- 3. Decode each of the inputs on each node and determine maximum throughput that the node can maintain without GR-Router. (base case)
- 4. Create a parent application that serves as the root of the graph.
- 5. Create a child application for all nodes that connect to the root.
- 6. Add nodes and compare throughput results.

LDPC Decoder Parent



LDPC Decoder Child

	type	index	size	Data[] : 768 floats	
Type 1 Msg.	type	index	size	Data[] : 768 floats	
Type 2 Msg.	type	index	size	Data[] : 50 bytes	
Type 3 Msg.	type	index	size	Data[] : 50 bytes	weight



N: 96, K: 50

N * 8 bits/byte = 768 bits

Throughput Results

- Results:
  - Overhead: 20% reduced throughput on single node
  - Without noise: 20% reduced throughput when running on two nodes.
  - With 0.5 pk-pk noise: 1.8x throughput on two nodes!

- Currently running tests with 2-6 ethernet-connected machines for additional testing.

Future Work

- Continue adding features to GR-Router
  - XML integration for easy reconfiguration
    - Define segments in an xml file, and have the queue_{source, sink} and routers read them.
- Implement redundancy and fault tolerance.
- Support additional network technologies.
- Support multi-layer GNU Radio systems.
- Support inter-child communication.
- Experiment with networked low-power platforms.

http://www.iconarchive.com/show/adobe-cs4-icons-by-hopstarter/File-Adobe-Dreamweaver-XML-01-icon.html

Questions?