The Modular Future of GNU Radio

Bastian Bloessl
Josh Morman

FOSDEM 2021

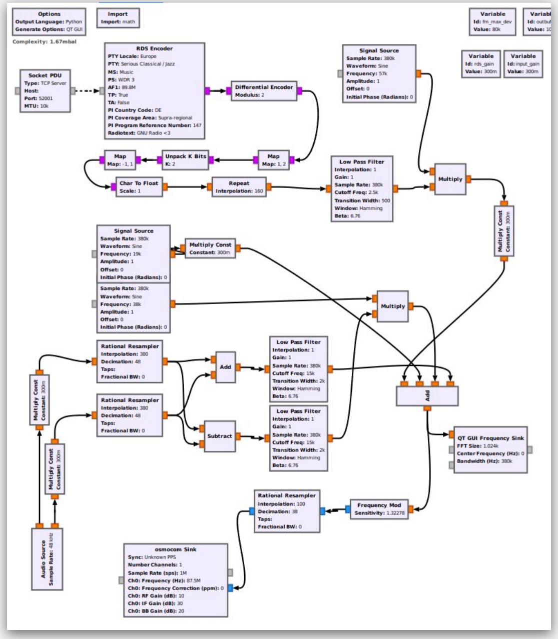
Why Modularity?

GNU Radio already supports a modular library of blocks.

But, we are restricted to **one** "scheduling" implementation and **one** type of buffer.

Current implementation has inherent limitations that cannot be overcome:

- We don’t have much control how threads/blocks are scheduled.
- CPU focused scheduling and memory model prevents us from adequately dealing with hardware accelerators and heterogeneous platforms.



What is "newsched"

https://github.com/gnuradio/newsched

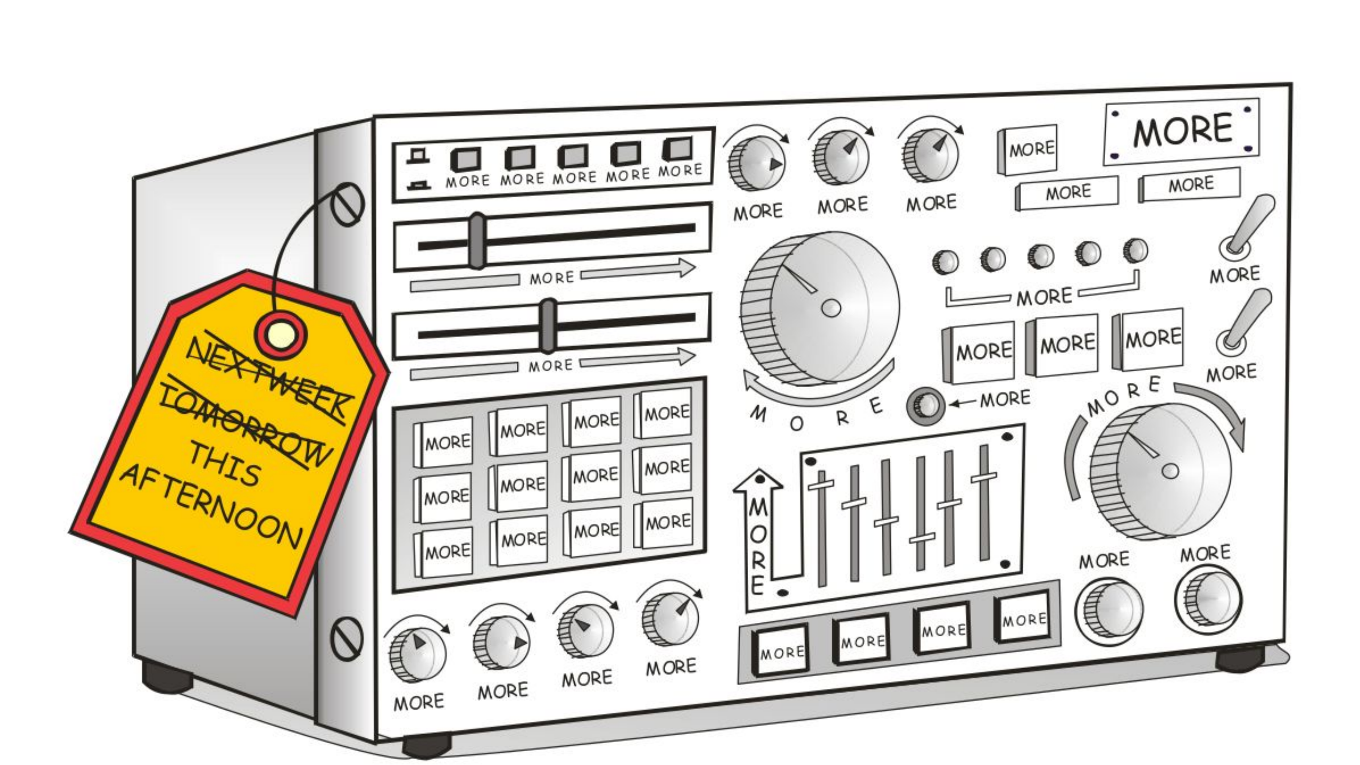
Project started 1 year ago at the pre-FOSDEM 2020 hackfest at the ESA, Noordwijk, Netherlands

Original Goal: A clean-slate approach to write a GNU Radio runtime that works for humans.

Since then

- our vision and goals have broadened
- the implementation has started to take shape
- newsched aims to be the basis for a GR 4.0 runtime

Working With the Community



Picture: fred harris, “Modems”, Dec 2010, SDR’10.

Scheduler Working Group

Following the breakout session from GRCON20, we have been meeting periodically - ~1x/month - times and meeting links shared via:

Chat room:
https://chat.gnuradio.org/#/room/#scheduler:gnuradio.org

Mailing List:
https://groups.io/g/gnuradio-scheduler

Topics Covered Thus Far:

- newsched status
- Custom Buffers
- Domains
- Hierarchical Scheduling
- Scheduling Paradigms
- Blocking I/O
- OpenCPI alignment
- Benchmarking
- SDR 4.0 Design Review
- PMTs

Future Topics

- Message Port interfaces
- GPU domain scheduling
- ...

Developer Tutorial

https://mormj.github.io/newsched-tutorial/dev_tutorial/01_Intro

Steps through the components in sequence of implementation

Newsched

In order to get more developers involved in the newsched project, which is aimed to be the GR 4.0 runtime and a sandbox to work out new ideas related revamping GNU Radio for enhanced modularity and native support of heterogenous and distributed platforms, we present a tutorial of the components of newsched.

What is newsched

Newsched is the attempt to create an SDR framework that is simple, yet flexible, and pleasant to develop applications with. Much is leveraged / copied from GNU Radio, but we are not attempting to keep GNU Radio entirely intact either.

Tutorial

- The Basics
- Ports
- Nodes and Blocks
- Graphs and Buffers
- Schedulers
- Flowgraph Monitor
- Single Threaded Scheduler
- The Flowgraph
- Some Blocks
- Build System
- Our First Test

Itemsize

The size of items to be processed in the work function - for instance a `100` length vector input of `uint16_t` would have a `datasize` of `200`

Typed Ports

Typed ports wrap the base `port` class with a standard c++ type through the template parameter and take care of all the sizing, etc.

Creating a typed port within the block factory would look something like:

```
auto p = port<float>::make("input",
            port_direction_t::INPUT,
            port_type_t::STREAM,
            std::vector<size_t>{ vlen });
```

This creates an input stream port, named as such that expects vectors of `vlen` floats.

Untyped Ports

Some blocks don't care about the underlying datatype and just process the raw data that passes through. Untyped ports preserve this behavior that was controlled previously by the io_signature.

Untyped ports are instantiated as

```
auto p = untyped_port::make(
            "out", port_direction_t::OUTPUT,
            itemsize,
            port_type_t::STREAM);
```

Core Concepts

* review for anyone who did not participate in Scheduler Working group or GRCON breakout session

Vision for Runtime 4.0

Modular GPP Scheduler

- Scheduler as plugin
- Application-specific schedulers

Heterogeneous Architectures

- Seamless integration of accelerators (e.g., FPGAs, GPUs, DSPs, SoCs)

Distributed DSP

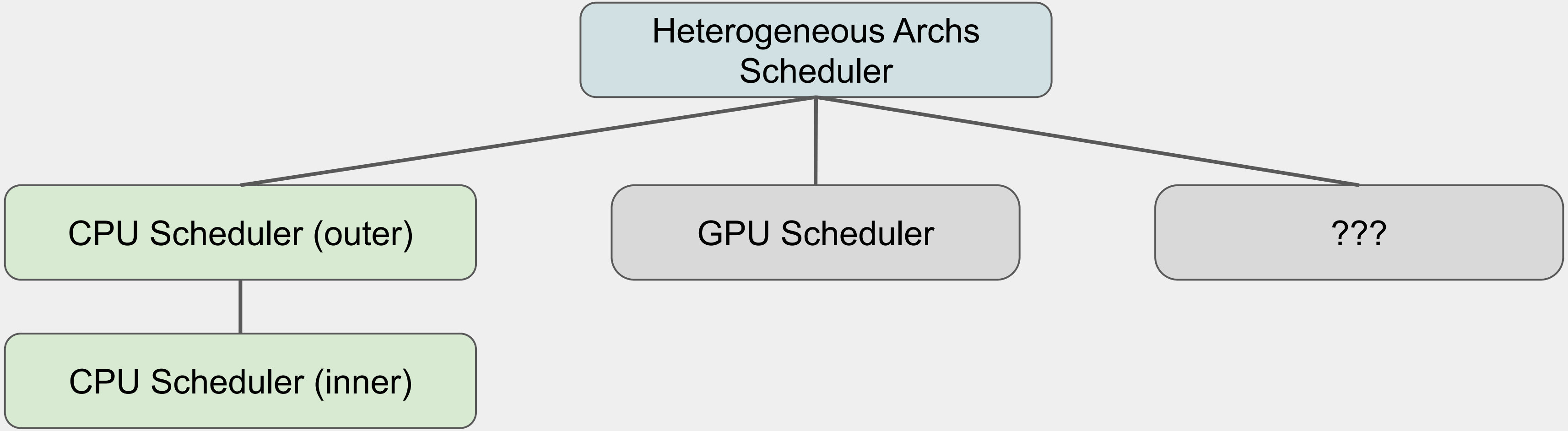
- Setup and manage flowgraphs that span multiple nodes

Straightforward implementation of (distributed) SDR systems that make efficient use of the platform and its accelerators

Scheduler Hierarchies

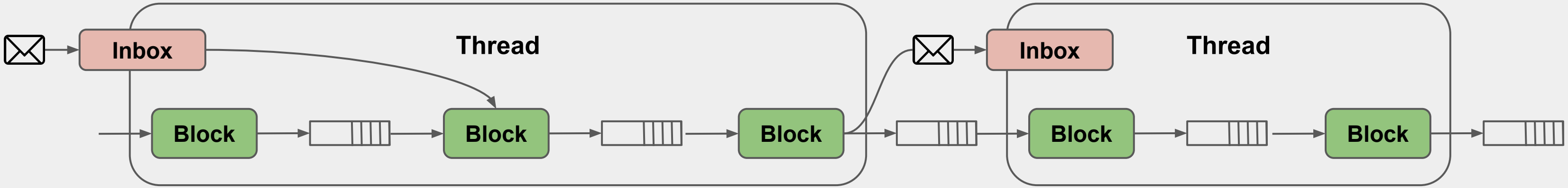
Idea: Separation of concerns, find good abstractions (similar to network stack)

- Might not results in the theoretical/global optimum but simplifies design and implementation. Main question: what's the right abstraction?



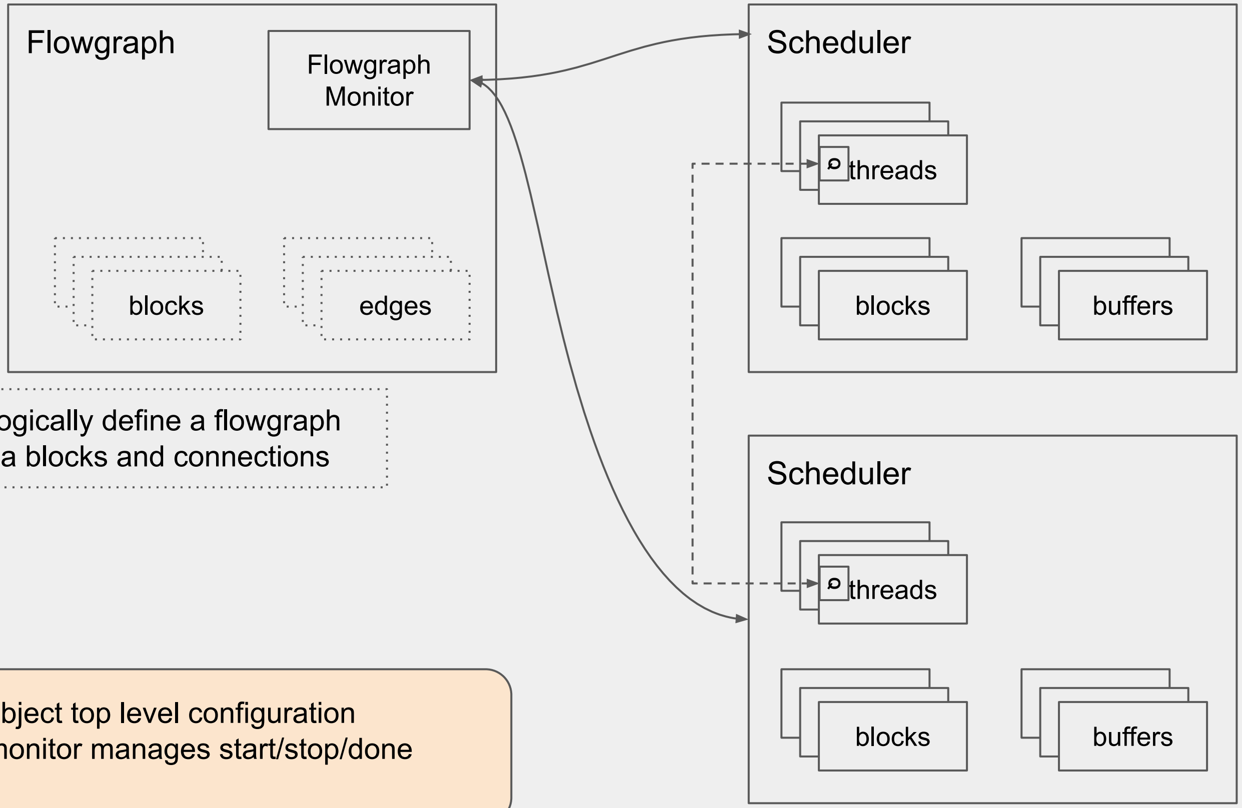
Recap: CPU Scheduler

- Init: **Outer scheduler** partitions blocks of flowgraph. Start one worker thread per partition that serves the corresponding blocks.
- Loop in worker thread
  - Read inbox non-blocking (update buffer pointers, execute async message handlers)
  - Activate blocks, if they (1) made progress in last round, (2) received updates
  - If there are active blocks
    - Use **inner scheduler** to execute active blocks
  - Else
    - Blocking-wait until inbox receives messages



Current Working State

Structure and Terminology



Logically define a flowgraph
via blocks and connections

Flowgraph object top level configuration
Flowgraph monitor manages start/stop/done

Single Threaded Scheduler

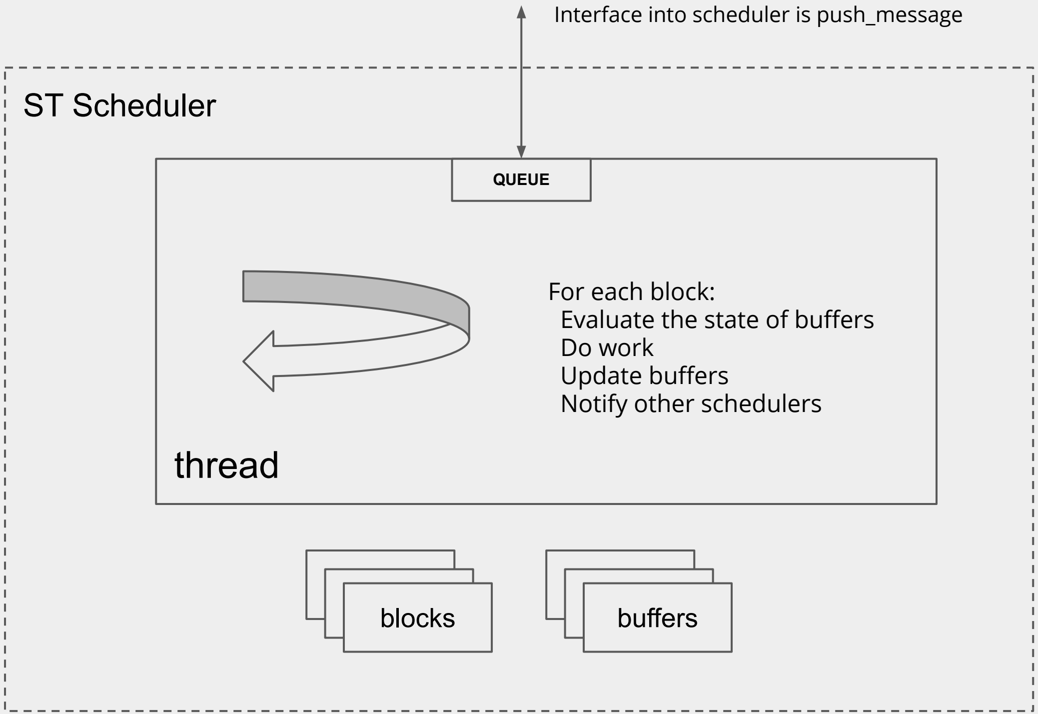
Single actor model - waits on queue messages

scheduler_st.cpp
- top level API

thread_wrapper.cpp
- runtime thread

graph_executor.cpp
- run_one_iteration()

buffer_management.cpp
- initializes buffers

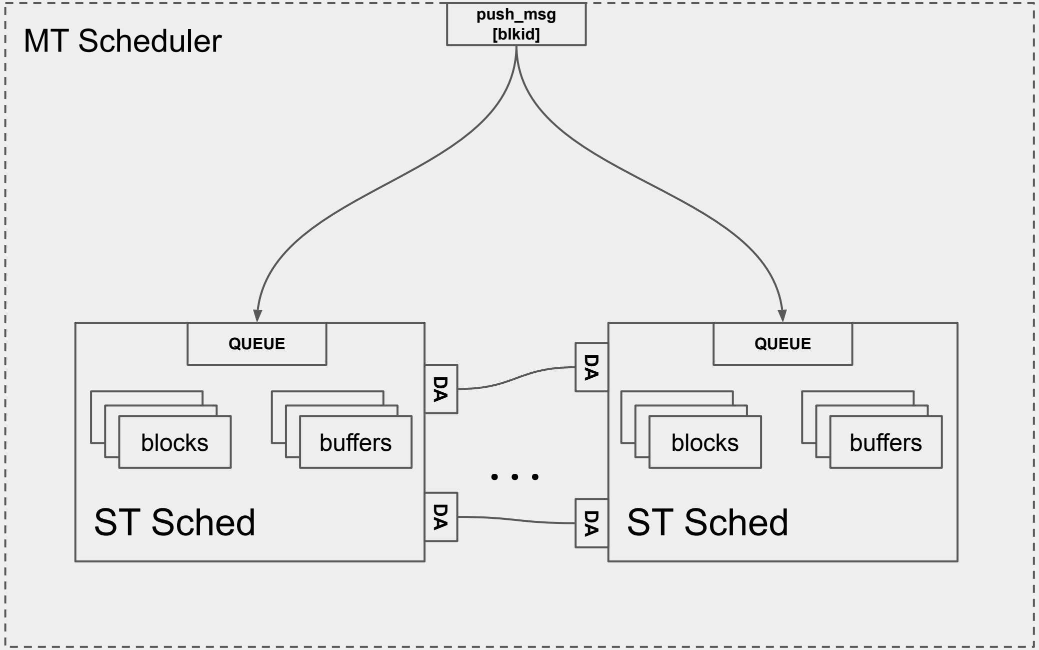


Multi-Threaded Scheduler

Just a bunch of ST Schedulers connected via "domain adapters"*

Defaults to TPB

Unless add_block_group (vector<block_sptr>) is called



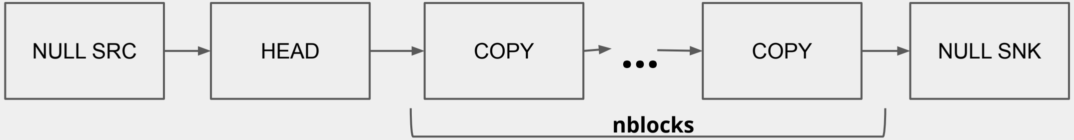
* The mechanism for sharing buffer pointers (or copies of data in the case of distributed connections)

Top Level API

```
auto src = blocks::vector_source_f::make(input_data, true);
auto mult = blocks::multiply_const_ff::make(k);
auto head = blocks::head::make(sizeof(float), samples);
auto snk = blocks::vector_sink_f::make();

auto fg = flowgraph::make();
fg->connect(src, 0, mult, 0);
fg->connect(mult, 0, head, 0);
fg->connect(head, 0, snk, 0);

auto sched = schedulers::scheduler_mt::make();
sched->add_block_group({mult,head}) // if I don't specify block group, defaults to TPB
fg->set_scheduler(sched);

fg->validate();
fg->start();
fg->wait();
```

Scheduler Benchmarks

Following methodology from gr-sched and associated paper



Flowgraph with **nblocks** copy blocks

Using 4 cores cpu shielded

No real-time scheduling

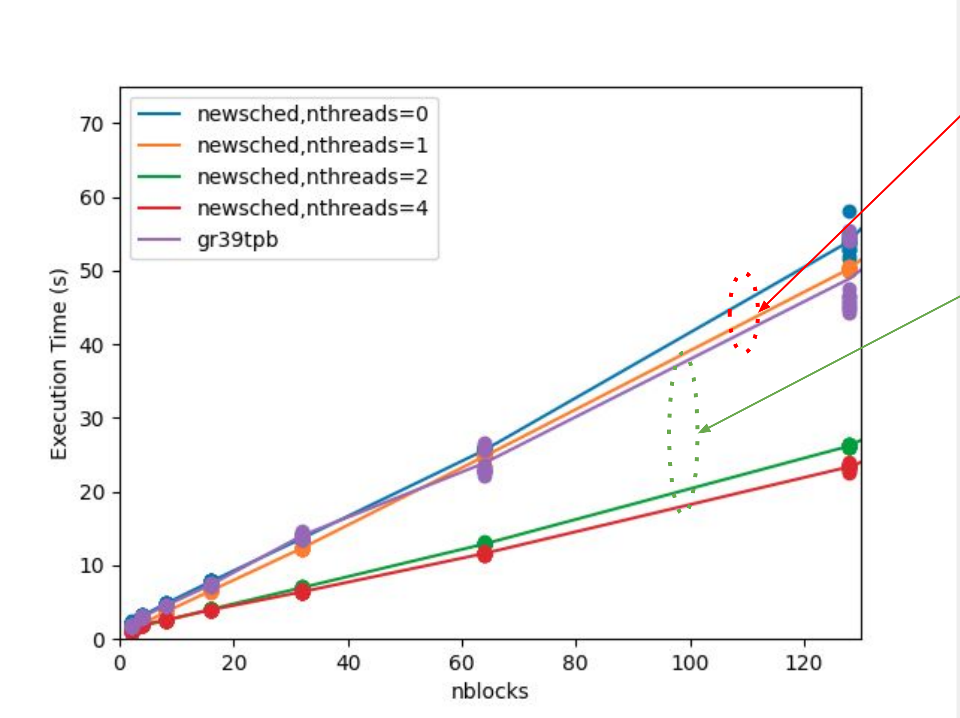
Pushing 1e9 samples through the flowgraph

Also verified optimizations are set the same (-O2) and buffer sizes are equivalent (half filled 32768 bytes)

benchmarked with scripts in https://github.com/mormj/gr-bench and https://github.com/bastibl/gr-sched

Scheduler Benchmarks

With implementation on Master (`186a3f2b8`)



Still not matching the performance of GR 3.9 scheduler

NEEDS INVESTIGATION

But the thread grouping has the intended effect

For newsched:

- nthreads=0 => TPB
- nthreads=4, blocks grouped sequentially in nblocks/nthreads with the src, snk, head joining the adjacent block groups

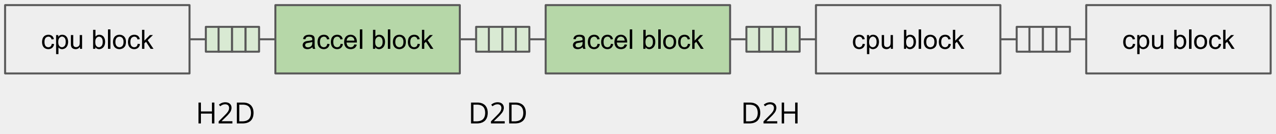
Custom Buffers

Interface

Buffer is associated with edge in graph

Assumption: in work(), in and out buffers are already in appropriate device memory - e.g. should not have H2D or D2H memcpy in work()

Depending on placement of accelerated block, custom buffers need to be on both upstream and downstream edge



Interface

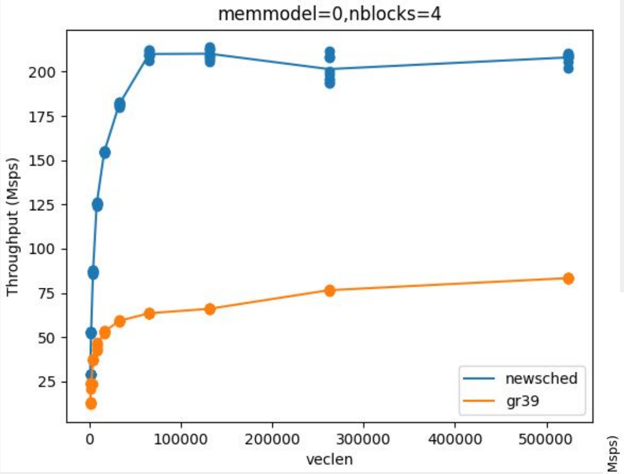
```
flowgraph->connect(src,blk1)->set_buffer(CUDA_BUFFER_ARGS_H2D)
flowgraph->connect(blk1,blk2)->set_buffer(CUDA_BUFFER_ARGS_D2D)
flowgraph->connect(blk2,blk3)->set_buffer(CUDA_BUFFER_ARGS_D2H)
flowgraph->connect(blk3,snk) // uses default buffer

flowgraph->run()
```

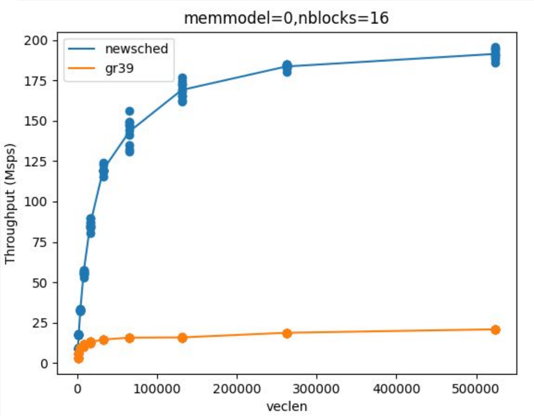
Custom Buffer Benchmarks

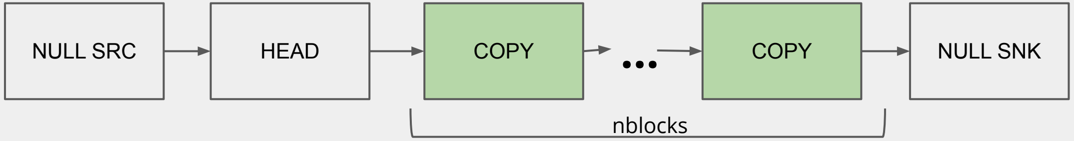
memmodel 0: H2D, D2D, D2H
veclen is batch_size into gpu

In the gr39 case, the H2D, D2H is done in every work() call

In the newsched case, custom buffers call the work() function assuming data is already accessible by gpu (either in device or pinned memory)







these copy blocks running on gpu accelerator

Going Forward

Next Steps

Refactoring to achieve design goals - simplify the code base

- Nested Schedulers → Hierarchical Schedulers
- Fully utilize ***port*** objects to handle messaging

Message ports and Messaging interfaces

- Some work done with re-imagining PMTs

Review / Contribute / Get Involved