Evolving the GNU Radio scheduler

Embracing and Breaking Legacy

Marcus Müller

2020-02-01

What'll happen in the next 40 minutes

Looking back at GNU Radio 3.8

Problems and Challenges

Taking Action

Marcus Müller

Bearer of a couple of roles

- Research assistant at CEL / KIT Karlsruher Institut für Technologie
  - I hold the exercise classes for KIT EEs' *Probability Theory* and *Communications Theory* courses ($>$ 300 students) and *Applied Information Theory* (ca 13 dB fewer students) and *Machine Learning and Optimization in Communications* (next semester)
- Ettus Research Support Grumpiness supplier
- Freelancing Engineer[1]
  - Technical Consulting
  - Contract Development
  - Seminars
- **Chief Architect of the GNU Radio project**

[1] Pretty time-limited

Marcus Müller

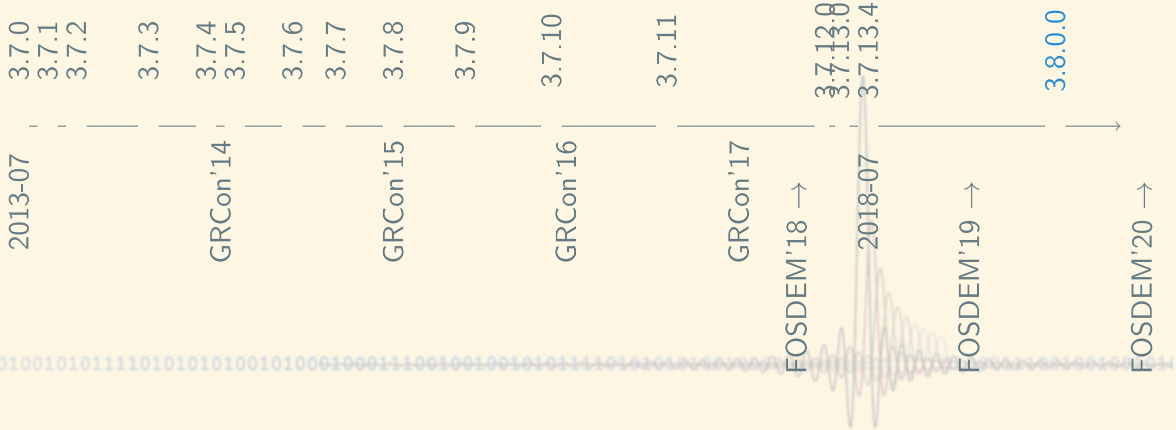
Contact

Depending on what you want to talk to me about, contact me using

- University Research & Teaching: `mueller@kit.edu`
- GNU Radio aspects: Preferably, `discuss-gnuradio@gnu.org`, for confident matters `mmueller@gnuradio.org`
- Freelancing & Private: `mueller@hostalia.de`

State of GNU Radio 2020

GNU Radio 3.7 released June 2013
GNU Radio 3.8 released August 2019



What we **changed** in GNU Radio 3.8

`--ignore-all-space`: 5220 files changed, 380586 insertions(+), 282592 deletions(-)

- Dependency cleanup: No choice, lots of benefits
  - •Qt4 → Qt5
  - •Cheetah/XML → YAML
- Language progression
  - •Py2 → Py2.7 OR Py3
  - •C++98 → C++11
- New functionality
  - Better `gr_modtool` (shoutout to Swapnil!)
  - C++ code generation (shoutout to Håkon)
  - Overall cooler GRC
- Code formatting (*way* better for development)

What we **didn't change** in GNU Radio 3.8

- no changes in the organization of modules
- no changes in the project scope
- no changes in the way we integrate new functionality
- **no changes in the "scheduler"**

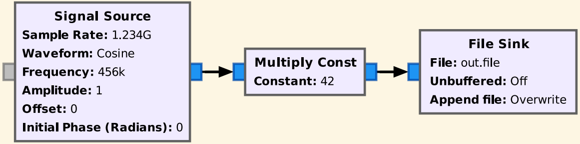
Why the quotation marks?

Everyone loves the GNU Radio "scheduler"?!

Bit of history (World's shortest intro to GNU Radio scheduling)

- Originally
  - Convert Flowgraph to flat, acyclic, directed graph
  - Find sources, call them
  - ripple the data through the very tree-ish structure
- Later: call that the *single-threaded scheduler*, introducing the
- *Thread-per-Block* (TPB) scheduler ca. 2009
  - Flatten flowgraph to determine data dependencies
  - Start a single thread per block
  - in that block executor, run `while(1){work(); notify_neighbors(); wait_for_notifications();};`

Signal Flow Architecture



- backpressure-driven parallel signal processing architecture
- Blocks produce as much output as they can at once, given
  - available input data ready at the start of processing
  - available output data memory
- asked to produce min(buffer size / 2, available output buffer)
- Block can start working again while downstream block is still consuming
- $\rightarrow$ high parallelism

Scheduling Mechanism – Abstracted

- *Scheduler* might be too strong a word
- *back pressure* limits processing speed
- great for throughput
- not so great for latency
- high parallelism stems from the ability to concurrently execute
- actual scheduling of threads done by OS
- no workload knowledge flows into OS → suboptimal . . .
- . . . but works surprisingly well.

Scheduling Mechanism in detail

- Each block gets its own executing thread[2]

When notified[3],

- ask the block (`forecast`) whether it can produce output, given the available input and output space.
  If *READY*
  - call `general_work` **DSP happening here** (this might take some time)
  - notify the upstream block(s) that we've consumed → free output buffer
  - notify the downstream block(s) that we've produced → new input

  If *blocked by lack of input*
  - go to sleep for a while and check back later

  If *blocked by lack of output space*
  - go to sleep until notification

[2] `tpb_thread_body.cc`, `block_executor.cc`
[3] ignoring asynchronous message passing

Problems and Challenges

- GNU Radio has ca. 21 years of history
- Not all decisions made in that period apply to the current architecture
  - to be completely honest, not even all decisions were good
- Use cases have evolved
  - Beginnings: Nearly only stream (TV, audio broadcast) processing
  - Nowadays: Real-time systems doing packetized data
- Environment has changed
  - SDR Hardware that supports bursting
  - Accelerators (GPUs, FPGAs, even network cards) widely available
- Audience has changed

Challenges for Scheduling

- Every block gets own thread getting scheduled on randomly (OS-)chosen CPU core
  - **clearly** suboptimal
- Memory-mapped Pseudo-Ring Buffers
  - Many blocks always consume all input → *no need* for ring buffer
  - hardware-def'ed DMA'd accelerator memory and doubly-mapped pages technologically mutually exclusive
  - Inefficient separate handling of tags, which logically belong to chunks of samples
- Confusion of block and scheduler state in practice
  - impossible to exchange block implementations at run time
  - or move them across scheduling domains (e.g. other server, or onto an accelerator)

Scheduling Shortcomings

We can do better than letting OS randomly decide which blocks gets executed when on which CPU core

- Memory locality is much more important than using large chunks of data
- Developer knows more about data dependencies than OS

We let the Single-Threaded Scheduler slowly die, because TPB scaled so well
Now:

- Stuck with Scheduler that works heuristically
- no way to feed in knowledge about data flow[4]
- no way of observing constraints

[4] Aside from pinning blocks to CPU cores. See: Kirby Cartwright's *A Case Study in Optimizing GNU Radio's ATSC Flowgraphs*, GRCon 2017

Better Scheduling

Short term design goals

We need a scheduler that

- we understand (and know how to fix when it breaks),
- is extensible,
- offers metrics, and
- clearly separates between block and scheduler state.

We need to reduce the block API, being

- as a stateful, but encapsulating data processor
- preserving and enhancing the purity of essence i.e. the ease of just writing a `work()` function,
- while flexible enough to fit accelerators and distributed systems

Taking Action

Prototype newsched 1/2

Implement a `block.h`

- For host CPU scheduling
- reduced API
  - remove scheduler-specific API components (esp. `estimate`)
  - replace inconsistent ways to communicate state modification (production of messages, tags and output samples) with clean object interface:

```
work_return_code_t
work(vector<block_work_input>&[5] work_input,
     vector<block_work_output>&[6] work_output)
```

  - represents both packetized data exchange (buffer pool) and stream data exchange (ring buffer)

⟶ separation of block and scheduler state

[5] data pointers, relevant tags
[6] captures write pointer advance, generated tags

Taking Action

Prototype newsched 2/2

*Worker*s replacing Thread-Per-Block *Block Executor*

- Groups all work to be done on a CPU
- Has multi-writer low-latency *update queue*
- receives messages, ring buffer pointer updates, status changes
- aggregates (aligns, eventually reorders) updates coming in while `work` was running

$\longrightarrow$ general interface for work done *anywhere*, not just on CPU

Expected Benefits

Higher Performance of CPU execution due to consecutive blocks being kept sequential on same core

Ability to *transparently* move blocks between execution hosts

- still requires efforts on serializing block state, but becomes pretty doable

Ability to allow for development of other/optimized schedulers

- . . . instead of hoping that any touch to the only scheduler doesn't break things (or decrease performance)
- cleaner API → Important metrics basically free via eBPF profiling

Scheduler API implementation

- up to now, accelerators are only superficially linked, iterate on API with accelerator working groups
- future: coordinate/check constraint (latency, throughput, max ops) between scheduling domains

Questions?

Backup Slides

Expected Concerns

- Wait! That breaks **all** the existing blocks!
  - easy to design shim to make old blocks work within new API
  - TPB is a special case: single-block worker
- Wait! This departs from two decades of practice and makes writing blocks harder!
  - Hopefully, with wrappers and easier API, this will not have long-term negative impact.
- Wait! You're spending time on redesigning a scheduler when you have 383 open issues on Github[7]?!
  - Hm, as long as none of the issues is *GNU Radio obsolete: archive project*, that sounds like a good idea.

[7] as of 2020-02-02 00:01

Backup Slides

Changes for 3.9

- removal of Python2
- gr-iio and gr-soapy upstream
- trying to replace SWIG with Pybind11
- VOLK fully out-of-tree