GNU Radio runtime

Reinventing a very fast wheel

Andrej Rode

FOSDEM 2018

Table of Contents

Introduction

- GNU Radio user and contributor since 2016
- EE Master student @ KIT
- GNU Radio CI & web infrastructure guy arode@gnuradio.org
- Interested in contributing and improving GNU Radio
- Has fun with SDRs sometimes

Table of Contents

State of the GNU Radio runtime (1/2)

- Upcoming GNU Radio 3.8 release contains no mentionable new runtime features aside of removing the single threaded scheduler
- GNU Radio 3.8 will contain a lot of dependency bumps which are a great load of work

This state leads to the question if development on the runtime is completed and GR runtime can be considered stable

State of the GNU Radio runtime (2/2)

Unfortunately no new features in the runtime don't necessarily mean it's done.

- DSP engineers are more attracted to DSP problems
- Hacking on the runtime is just a necessary evil

Table of Contents

History

- 2008-10-19: multithreaded scheduler (tpb) is merged into trunk
- 2009-08-15: message passing is merged into trunk
- 2010-11-29: tagging capability is merged into next
- 2012-12-04: ctrlport is merged into next
- 2013-03-17: tagged stream blocks are merged into next
- 2016-09-03: single threaded scheduler is removed from next

Lots of fixes were added and some restructuring happened during this period of time.

Achievements

- backpressure driven architecture
- concurrency (leave scheduling to the OS)
- strict block boundaries
- thread per block paradigm
- low block intercommunication overhead
- simple block API

These lead to a lot of new projects emerging and creating own blocks based on the GNU Radio runtime capabilities and existing in-tree blocks.

Shortcomings (1/2)

- Cache invalidation (high thread count)
- Missing tests for core features
- Missing runtime API docs
- Organic grown code
- Lacking thread-safety for direct method calls
- Varying code quality

Example of varying code quality: Test Coverage

Files					Coverage
gr-filter	3,440	2,732	0	708	79.42%
gr-analog	1,282	959	0	323	74.80%
gr-digital	5,780	3,856	0	1,924	66.71%
gr-blocks	5,813	3,752	0	2,061	64.54%
gnuradio-runtime	6,870	4,409	0	2,461	64.18%
gr-wavelet	112	71	0	41	63.39%
gr-fec	3,924	1,981	0	1,943	50.48%
gr-vocoder	3,193	1,445	0	1,748	45.26%
gr-fft	596	267	0	329	44.80%
gr-atsc	1,584	625	0	959	39.46%
gr-zeromq	501	191	0	310	38.12%
gr-qtgui	10,659	3,486	0	7,173	32.70%
gr-channels	517	86	0	431	16.63%
gr-trellis	1,212	157	0	1,055	12.95%
gr-uhd	867	27	0	840	3.11%
gr-noaa	171	3	0	168	1.75%
gr-pager	362	4	0	358	1.10%
gr-audio	1,223	9	0	1,214	0.74%
gr-video-sdl	317	2	0	315	0.63%
gr-dtv	14,092	53	0	14,039	0.38%
gr-fcd	867	1	0	866	0.12%
Project Totals (1363 files)	63,382	24,116	0	39,266	38.05%

Shortcomings (2/2)

- Lacking Integration with heterogeneous computing
  - GPU/FPGA/CPU
  - Cloud
- low latency
- packetized streams
- security
- flowgraph introspection
- loose API/impl boundary

Table of Contents

Comparison to Similar Projects (1/2)

- Do we need to improve the GR runtime itself?
- Or could it be possible to use already finished work in that domain?
- Should we learn from acomplishments and failures of similar projects?

Comparison to Similar Projects (2/2)

Similar projects include:

- Microsoft SoRa (BSD 2-clause)
- srsLTE (AGPLv3)
- RedHawk (various licenses)
- Pothos (Boost License)
- liquid-dsp (MIT)
- LuaRadio (GPLv3)
- OS kernels (various licenses)

Other projects don't really match GNU Radio in terms of goal and ecosystem.

Runtime design questions

Main questions:

- What are challenges we see today not being solved by the current runtime?
- What should an API look like for writing blocks extending GNU Radio?
- What has to be done to ease maintenance?
- How can already implemented signal processing blocks be kept around with little effort?

Runtime design theory

Is there a theoretical way to express to express the work done by the GNU Radio runtime?

- concurrent computing.

Design goals heavily depend on application. Example: Latency vs. Throughput.

Runtime design goals (1/2)

- API before implementation
- Configurable scheduling inside the runtime
- Buffer management (DMA/GPU/CPU buffers)
- flowgraph introspection
- remote computation
- Keep existing features: synchronuous streaming, safe async message passing, stream tags, packetized streams, multi-threading

Runtime design goals (2/2)

- Don't reinvent the wheel!
- Foster other FOSS projects' work to keep maintenance low

Development timeline

Steps to take:

1. Write down API
2. Agree on used technology for implementation
3. Implement new API
4. Port/glue existing ecosystem to new runtime

Questions?

- Please ask now!
- I'm available at mail@andrejro.de or using the nick *noc0lour* on the GNU Radio Slack and freenode IRC.
- We will keep the GNU Radio community in the loop about development on the runtime