GNU Radio runtime
Reinventing a very fast wheel

Andrej Rode

FOSDEM 2018
Table of Contents

1. Introduction

2. State of the GNU Radio runtime

3. GNU Radio runtime in retrospective
   - History
   - Achievements
   - Shortcomings

4. Future of the GNU Radio runtime
   - Comparison to Similar Projects
   - Runtime design
   - Development timeline
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

1 Introduction

2 State of the GNU Radio runtime

3 GNU Radio runtime in retrospective
   • History
   • Achievements
   • Shortcomings

4 Future of the GNU Radio runtime
   • Comparison to Similar Projects
   • Runtime design
   • Development timeline
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.
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
1 Introduction

2 State of the GNU Radio runtime

3 GNU Radio runtime in retrospective
   - History
   - Achievements
   - Shortcomings

4 Future of the GNU Radio runtime
   - Comparison to Similar Projects
   - Runtime design
   - Development timeline
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

<table>
<thead>
<tr>
<th>Files</th>
<th>Test</th>
<th>Cov</th>
<th>Code</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr-filter</td>
<td>3,440</td>
<td>2,732</td>
<td>0</td>
<td>708</td>
</tr>
<tr>
<td>gr-analog</td>
<td>1,282</td>
<td>959</td>
<td>0</td>
<td>323</td>
</tr>
<tr>
<td>gr-digital</td>
<td>5,780</td>
<td>3,856</td>
<td>0</td>
<td>1,924</td>
</tr>
<tr>
<td>gr-blocks</td>
<td>5,813</td>
<td>3,752</td>
<td>0</td>
<td>2,061</td>
</tr>
<tr>
<td>gnuradio-runtime</td>
<td>6,870</td>
<td>4,409</td>
<td>0</td>
<td>2,461</td>
</tr>
<tr>
<td>gr-wavelet</td>
<td>112</td>
<td>71</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>gr-fec</td>
<td>3,924</td>
<td>1,981</td>
<td>0</td>
<td>1,943</td>
</tr>
<tr>
<td>gr-vocoder</td>
<td>3,193</td>
<td>1,445</td>
<td>0</td>
<td>1,748</td>
</tr>
<tr>
<td>gr-fft</td>
<td>596</td>
<td>267</td>
<td>0</td>
<td>329</td>
</tr>
<tr>
<td>gr-atsc</td>
<td>1,584</td>
<td>625</td>
<td>0</td>
<td>959</td>
</tr>
<tr>
<td>gr-zeromq</td>
<td>501</td>
<td>191</td>
<td>0</td>
<td>310</td>
</tr>
<tr>
<td>gr-gtgsai</td>
<td>10,659</td>
<td>3,486</td>
<td>0</td>
<td>7,173</td>
</tr>
<tr>
<td>gr-channels</td>
<td>517</td>
<td>86</td>
<td>0</td>
<td>431</td>
</tr>
<tr>
<td>gr-trellis</td>
<td>1,212</td>
<td>157</td>
<td>0</td>
<td>1,055</td>
</tr>
<tr>
<td>gr-uhd</td>
<td>867</td>
<td>27</td>
<td>0</td>
<td>840</td>
</tr>
<tr>
<td>gr-noaa</td>
<td>171</td>
<td>3</td>
<td>0</td>
<td>168</td>
</tr>
<tr>
<td>gr-pager</td>
<td>362</td>
<td>4</td>
<td>0</td>
<td>358</td>
</tr>
<tr>
<td>gr-audio</td>
<td>1,223</td>
<td>9</td>
<td>0</td>
<td>1,214</td>
</tr>
<tr>
<td>gr-video-sdl</td>
<td>317</td>
<td>2</td>
<td>0</td>
<td>315</td>
</tr>
<tr>
<td>gr-dtv</td>
<td>14,092</td>
<td>53</td>
<td>0</td>
<td>14,039</td>
</tr>
<tr>
<td>gr-fcd</td>
<td>867</td>
<td>1</td>
<td>0</td>
<td>866</td>
</tr>
</tbody>
</table>

**Project Totals** (13,633 lines)

<table>
<thead>
<tr>
<th>Test</th>
<th>Cov</th>
<th>Code</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>63.382</td>
<td>24.116</td>
<td>0</td>
<td>39.266</td>
</tr>
</tbody>
</table>
Shortcomings (2/2)

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

2. State of the GNU Radio runtime

3. GNU Radio runtime in retrospective
   - History
   - Achievements
   - Shortcomings

4. Future of the GNU Radio runtime
   - Comparison to Similar Projects
   - Runtime design
   - Development timeline
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 accomplishments and failures of similar projects?
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.
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?
Is there a theoretical way to express the work done by the GNU Radio runtime?

- concurrent computing.

Design goals heavily depend on application. Example: Latency vs. Throughput.
API before implementation
- Configurable scheduling inside the runtime
- Buffer management (DMA/GPU/CPU buffers)
- Flowgraph introspection
- Remote computation
- Keep existing features: synchronous streaming, safe async message passing, stream tags, packetized streams, multi-threading
• 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.