GNU Radio Technical Update

Johnathan Corgan

Corgan Labs

Topics

- Release 3.8
  - Features and capabilities
  - Milestones and remaining tasks
  - How to get involved
- Future Directions
  - Distributed and Heterogeneous Computing
  - Client/Server Architecture
  - GRC as “online” design and debug tool

Release 3.8 Update

- Core feature of migrating to Python 3 has driven everything else
- Two major dependencies of GRC had to be replaced
  - PyGTK+2 (used for GUI) abandoned by upstream, now using PyGObject
  - Python Cheetah templates (used for code generation) abandoned by upstream, now using Python mako and yaml format
- Along the way, GRC has been completely refactored into GUI and non-GUI processing (code generation)
- Many thanks to Sebastian Koslowski and Seth Hitefield; this work is "mostly" done

Python 3 Support in Runtime and Components

- Core runtime now supports both Python 2 and Python 3
  - This is largely done through the Python “six” compatibility library
  - CMake handles dependency checking
  - Mostly automated conversion of non-SWIG generated code, lots of corner cases
- Huge thanks to Doug Anderson for most of the work implementing this

Major Changes to Expect

- The entire code generator for GRC has been rewritten
  - ".grc" and ".xml" files in-tree and in OOTs now replaced with simpler YAML format
  - We'll provide an automated tool for "simple" cases
- Your hand-written Python blocks/flowgraphs will need to use the six module and be Py2 or Py3 compatible
- Not much has changed on the C++/block API, other than deprecated block removal
- QT-GUI widgets will use QT5

Remaining Tasks

- QT-GUI porting
- Blocktree xml→yaml conversion
- gr-modtool/pybombs integration
- gr-digital deprecations/clean up
- Debugging QA failures and problems with numpy interaction
- Converting all python and GRC examples
- Documentation updates
- Testing, testing, testing, bug reporting

Future Directions

Adapting to Modern Compute Platforms

- GNU Radio was born in early 2000s on single core PCs
  - Originally, was even single-threaded
- Data-driven streaming architecture
  - “No shared state” communicating sequential processes
  - Parallelism through pipelining
  - Single process DSP flows combined with GUI
- Bolted on: thread-per-block, polymorphic types, stream tags, message passing, TSBs, RPC, GRC environment, ZMQ
- This has worked surprisingly well and for a long time

Distributed and Heterogeneous Computing

- Modern platforms are a complex mix of NUMA GPPs, GPUs, FPGAs, DSPs, and interconnects
  - Hybrid systems of SDR hardware, host software, and remote control/monitoring/GUI
  - Interaction with external (non-DSP) software
- So far we’ve been able to glue these kind of applications together “by hand” - not scalable
  - Everyone starts from scratch
- Calls for a fresh look at how GNU Radio can more easily abstract itself and fit into these environments

“Client/Server” Model

- DSP processing fits into a larger system in a small number of ways:
  - Bulk data streaming flows (w/metadata tags)
  - Message-based data (PDUs w/metadata)
  - Out of band configuration, monitoring, and UI
- The client/server model formalizes these as first-class abstractions in the runtime
  - Streams, message flows, and properties as named endpoints
  - Runtime responsible for resolving how to connect these
  - REST-like properties for configuration and querying
  - Blocks don’t “know” or care where others are running

GNU Radio in Heterogeneous Compute Platforms

- It's not possible for GNU Radio to take the role of *creating* native elements for execution on non-GPP.
- Instead, we should make it easier to integrate GNU Radio flows into the wider system
  - RFNoC model of having proxies for CEs running elsewhere
  - Standardized transport formats for stream/messages/config would let us abstract these
  - GNU Radio runtime takes on role of communicating over whatever interconnect (GPU, network, bus) to other compute elements

Remote Monitoring, Control, and UI

- Today’s GNU Radio applications have little support for non-local user interaction
  - ControlPort/Thrift provides limited RPC
  - ZMQ can be manually built to bridge flows
  - No support for building UIs that work locally or remotely
- The client/server design will allow connecting block flowgraph/stream/message/properties to UI elements
  - Can design UI independently from flowgraph
  - Same UI running locally or remotely; only endpoint changes
- Particularly useful for embedded systems

GRC "Online" Operation vs. Program Generator

- The GNU Radio Companion is a useful tool for describing simple flows and controls/displays
  - Much more rapid iteration than writing Python scripts, but at a cost of limited functionality
- The client/server design would allow GRC to become a dynamic, online runtime environment
  - Create/manipulate/probe flowgraphs while running locally or remotely
  - "GDB for flowgraphs"
  - Even more rapid iteration and exploration of design through immediate feedback of changes vs. regenerating and re-running application
  - Would also allow scripted creation of GNU Radio flows on-the-fly

Underlying Architecture

- ZMQ has become ubiquitous for fast, lightweight transport
  - Thin layer on top of TCP, Unix sockets, or shared memory
  - Useful message passing semantics (PUB/SUB, PUSH/PULL, etc.), no broker, self-healing
  - CurveCP authentication and encryption
- This can be used for network or local transport of GNU Radio stream/message/property flows
  - Runtime responsible for setting these up
  - Would *not* be used for same process block-to-block connections
- Does not solve the communication path needed for GPUs or DSPs

Message Serialization

- Serialization defines a versioned, extensible binary wire format for data interchange
  - Usually a message description file gets “compiled” into code that knows how to marshall/unmarshall these
- There are a number of serialization options, currently evaluating Google protobufs
  - Very popular, libraries in a dozen languages
  - Transport independent, often used over ZMQ

“I’m a roadmap, Jim, not a release plan.”

- Volunteer-driven, open-source software projects work very differently from commercial projects
  - Many different motivations for contributing...
  - ...but having a salary dependent on implementing a product specification is not one of them
  - Volunteers have real jobs and sometimes even real lives
- This is a vision for how GNU Radio can adapt to and take advantage of modern computing environments and capabilities
- We are also looking for directed-development sponsored funds to help realize the larger items that need more sustained, full-time effort.