GNU Radio 4.0: Hands-on Session
A guided tour over the construction site

Alexander Krimm\textsuperscript{2}, Josh Mormon\textsuperscript{1}, Ralph J. Steinhagen\textsuperscript{2}

on behalf of: the GR Architecture Team, Ralph Steinhagen\textsuperscript{2}, Björn Balazs\textsuperscript{3}, Giulio Camuffo\textsuperscript{3}, Ilya Doroshenko\textsuperscript{3}, Semën Lebedev\textsuperscript{2}, Ivan Čukić\textsuperscript{3}, Matthias Kretz\textsuperscript{2}, Frank Osterfeld\textsuperscript{3}, ...

\textsuperscript{1} GNU Radio 4.0 (lead) & Peraton Labs, Basking Ridge, NJ, USA
\textsuperscript{2} FAIR – Facility for Anti-Proton and Ion Research & GSI, Darmstadt, Germany
\textsuperscript{3} KDAB Berlin, Germany

GRCon23
TEMPE, ARIZONA
2023-09-07
Who I am and where I work

- Software Engineer for Beam Based Feedback Systems at GSI/FAIR particle accelerator facility near Frankfurt, Germany
- Visit us if you’re in the area!
Scope and Motivation

- GR4 will include some breaking changes while maintaining continuity to GR3
- For a complete rundown of the changes and the motivations, see yesterday’s talk: https://events.gnuradio.org/event/21/contributions/390/
Goals of the workshop

- short recap of changes and features
- assist you as users/gr-devs/oot-devs in getting into the code
- collect feedback and start discussions on
  - things that we have not yet figured out 100%
  - things we missed
  - things that need to be communicated more clearly
Short Recap: Major Differences

- C++ ≥ 20 → concepts, constexpr, NTTPs, std::simd
- codegen → compile-time reflection
  - previously: yaml → boilerplate c++ code, bindings, documentation, ...
  - new: code as single truth → bindings, documentation, ...
- additional prominent process_one API
- new features:
  - dataset based bulk processing
  - pmt based settings
  - automatic tag propagation
Entry Point

- currently in external repository
  - https://github.com/fair-acc/graph-prototype.git
  - to be merged into gnuradio’s dev 4.0 branch soon

- Requirements:
  - gcc > 12.2, clang > 15 or emscripten-latest
  - cmake
Getting started

- Pretty much the classic cmake workflow
  
  ```
  git clone https://github.com/fair-acc/graph-prototype.git
  
  # native linux
  cmake -S . -B build && cmake -build build -j
  cd build && ctest .
  
  # emscripten
  emcmake cmake -S . -B build_wasm && cmake -build build -j
  cd build && ctest .
  ```

- There is some meson support which unfortunately does not work at the moment

- Tried on Ubuntu, SuSe, Fedora, ...
What can you do in the current state

- Write simple blocks: 1:1, n:m
- Compose flow-graphs from either c++ code or load from yaml
- Generate .so library plugins (=OOT) from a set of blocks
- Run flow-graphs using a basic scheduler
- Use a profiler (perfetto) integration to analyse the flow-graph
- Manage node settings
- Add and propagate tags to sample streams
What’s not yet there?

- Python integration
- Implementation of more advanced schedulers
- Handling of message ports
- Dynamic number of ports
- Consistent naming
- https://github.com/fair-acc/graph-prototype/issues/148
Writing a block – Simplest possible example

```c++
struct MyBlock : public node<MyBlock> {
    IN<double> in;
    OUT<double> out;

    T process_one(T in) {
        return in * in;
    }
};

ENABLE_REFLECTION(MyBlock, in, out);
```
Writing a block – adding more features

- Templated types
- SIMD
- process_bulk
- settings
- history_buffer (stateful blocks with process_one)
- sources
- annotations
- dynamic number of ports (not implemented yet)
Writing a block – templated types

template<typename T>
requires (std::is_arithmetic<T>())
struct MyBlock : public node<MyBlock<T>> {
    IN<T> in;
    OUT<T> out;

    T process_one(T in) {
        return in * in;
    }
};

ENABLE_REFLECTION_FOR_TEMPLATE_FULL(((typename T),
    (MyBlock<T>), in, out));
Writing a block – SIMD

template<typename T>
requires (std::is_arithmetic<T>())
struct MyBlock : public node<MyBlock<T>> {
  IN<T> in;
  OUT<T> out;

  template<fair::meta::t_or_simd<T> V>
  constexpr auto process_one(V in) {
    return in * in;
  }
};

ENABLE_REFLECTION_FOR_TEMPLATE_FULL(((typename T), (MyBlock<T>), in, out);}
Writing a block – process_bulk

```cpp
struct MyBlock : public node<MyBlock> {
    IN<float> in;
    OUT<float> out;

    void process_bulk(
        std::span<const float> in,
        std::span<float> out
    ) const noexcept {
        for (int i=0; i < in.size(); i++) {
            out[i] = f(in[i]);
        }
    }

    ENABLE_REFLECTION(MyBlock, in, out);
};
```
struct MyBlock : public node<MyBlock> {
    IN<float>    in;
    OUT<float>   out;

    float factor; // setting

    void settings_changed(const property_map & /*old*/,
                          const property_map &updated) noexcept {
        if (settings_valid(updated)) {
            fmt::print("updated settings successfully: \n {}", updated);
        }
    }

    T process_one(T in) {
        return in * factor;
    }
};

ENABLE_REFLECTION(MyBlock, in, out, factor);
Writing a block – history buffer

```cpp
struct MyBlock : public node<MyBlock> {
    IN<float> in;
    OUT<float> out;

    history_buffer<float> history{ 8 };

    T process_one(T in) {
        history.push_back(in);
        return std::accumulate(history.begin(), history.end(), 0)/8;
    }
};

ENABLE_REFLECTION(MyBlock, in, out);
```

- just a small circular buffer to keep local node state
Writing a block – sources

```cpp
struct MyBlock : public node<MyBlock> {
    OUT<T> out;

    int count = 0;

    std::size_t available_samples() {
        auto result = samples - count;
        return result > 0 ? result : -1; // -1 → DONE
    }

    int process_one() {
        return count++;
    }
};

ENABLE_REFLECTION(MyBlock<T>, out);
```
struct MyBlok : public node<MyBlock, Doc<R""
@brief A super cool block doing important processing

Put a more extensive documentation here.
)""""> { 
  IN<float> in;
  OUT<float> out;

  Annotated<float, "myparam", Visible,
    Doc<"target size">, Unit<"mm">> myparam = 1f;

  T process_one(T in) {
    return f(in, myparam);
  }
};
ENABLE_REFLECTION(MyBlock, in, out, myparam);
Writing a block – plugin/OOT blocks

```cpp
GP_PLUGIN("Example Plugin", "A. Krimm", "LGPL3", "v1")

template<typename T>
requires (std::is_arithmetic<T>())
struct MyBlock : public node<MyBlock<T>> {
    IN<T>   in;
    OUT<T>  out;

    T process_one(T in) {
        return in * in;
    }
};

ENABLE_REFLECTION_FOR_TEMPLATE_FULL((typename T), (MyBlock<T>), in, out);

GP_PLUGIN_REGISTER_NODE(MyBlock, float, double);
```
Creating and running a fixed flowgraph from C++

```cpp
fg::graph graph{};
auto& source graph.make_node<source>();
auto& node grap.make_node<scale>({{{ "factor", 3.0 }}});
auto& sink graph.make_node<sink>();

graph.connect<"out">(source).to<"raw">(scale);
graph.connect<"scaled">(scale).to<"in">(sink);

fg::scheduler::simple<> scheduler{std::move(graph)};
scheduler.run();
```
Dynamic flowgraphs

```cpp
fg::graph graph{};

fg::node_registry registry;
fg::plugin_loader loader(&registry,
    std::vector<std::filesystem::path>{"/share/gr-plugins"});
GPREGISTER_NODE(registry, MySource, double, float);

// load builtin node
auto &node_source = loader.instantiate_in_graph(graph, "MySource", "double");
// load from .so file
auto &node_source = loader.instantiate_in_graph(graph, "MyBlock", "double");

node_source->dynamic_output_port(0)
    .connect(node_myblock(node_multiply->dynamic_input_port(0))

fg::scheduler::simple<> scheduler{std::move(graph)};
scheduler.run();
```
Compile time merging of nodes

```cpp
fg::graph graph{};
auto& source graph.make_node<source>();
auto& sink   graph.make_node<sink>();

auto merged = graph.add_node(fg::merge<"scaled", "raw">(
  scale({{"factor", 3.0 }}, square));

graph.connect<"out">(source).to<"raw">(merged);
graph.connect<"squared">(merged).to<"in">(sink);

fg::scheduler::simple scheduler{std::move(graph)};
scheduler.run();
```
Goals of the workshop

- assist you as users/gr-devs/oot-devs in getting into the code
- collect feedback and start discussions on
  - things that we have not yet figured out 100%
  - things we missed
  - things that need to be communicated more clearly

- Slides
  - [https://events.gnuradio.org/event/21/contributions/496/](https://events.gnuradio.org/event/21/contributions/496/)

- Repo
  - [https://github.com/fair-acc/graph-prototype](https://github.com/fair-acc/graph-prototype)