GNURadio 4.0 Block Development

Developer Tutorial Track

Alexander Krimm, Semen Lebedev
GSI Helmholtz Centre for Heavy Ion Research GmbH

Outline

- Modern C++ evolution
- Brief overview of `std::algorithm`
- Brief overview of `std::ranges`
- Block implementation examples:
  - Moving average block
  - Simple FIR block
  - Decimation block
  - Selector block
  - Even number filter block
- Benchmarking and Performance
  - Measure the performance
  - Examples of `bm_Buffer` and `bm_fft`
  - Performance monitoring block

Objectives of the Talk

- This is not intended to be a comprehensive overview of modern C++, `std::algorithm`, `std::ranges`, or `GNURadio4`.
- Instead, it aims to provide newcomers with some introductory examples of modern C++20 and its application in `GNURadio4` development.
- Discover how to use `std::algorithm` and `std::ranges` to write more readable and efficient code.
- Explore practical examples of block implementations that can serve as templates for future development.

Modern C++

Modern C++

- C++ has evolved significantly over the past decade to meet modern programming needs.
- Modern C++ evolution:
  - C++11: Modern C++ Foundation
  - C++14: Small but Impactful Improvements
  - C++17: Standardizing Modern Practices
  - C++20:The Most Comprehensive Update
  - C++23:Upcoming Features
- `GNURadio4` makes extensive use of modern C++ features.

C++11 - The Modern C++ Begins

- `auto` keyword: `auto x = 42; // x is deduced to be an int`
- Range-based for loops: `for (int n : {1, 2, 3}) { std::cout << n; }`
- Lambda expressions: `auto add = [](int a, int b) { return a + b; };`
- Smart pointers: `std::unique_ptr`, `std::shared_ptr`
- Move semantics and rvalue references
- Threading support: `std::thread`, `std::mutex`, `std::lock`
- Uniform initialization: `std::vector<int> v = {1, 2, 3};`
- Variadic templates

```
template<typename... Args>
void print(Args... args) {  (std::cout << ... << args) << std::endl; }
```

- and more https://en.cppreference.com/w/cpp/11

C++14, C++17 - Standardizing Modern Practices

- Generic lambdas: `auto multiply = [](auto x, auto y) { return x * y; };`
- Return type deduction: `auto square(int x) { return x * x; }`
- `if constexpr`: `if constexpr (sizeof(std::size_t) == 4) { /*...*/ }`
- `std::optional`, `std::variant`, `std::any`:
- Structured bindings: `auto [a, b] = std::pair<int, int>(1, 2);`
- `std::make_unique`: `auto p = std::make_unique<Block>();`
- Filesystem library: `std::filesystem::create_directory("new_folder");`
- Parallel algorithms: `std::sort(std::execution::par, v.begin(), v.end());`
- and more C++14 https://en.cppreference.com/w/cpp/14, C++17 https://en.cppreference.com/w/cpp/17

C++20 - The Most Comprehensive Update

- Concepts:

```cpp
// Define a concept for types with a size() method
template<typename T>
concept HasSize = requires(T t) {
    // Must have a size() method that returns something convertible to std::size_t
    { t.size() } -> std::convertible_to<std::size_t>;
};
template<HasSize T>
void printSize(const T& container) {...}
```

- Ranges: `auto squared = v | std::views::transform([](int n) { return n * n; });`

- Coroutines: `co_await some_async_function();`
- Modules
- Three-way comparison operator (<=>): `auto cmp = (a <=> b); // Returns std::strong_ordering`
- Calendar and time zone library
- Enhanced standard attributes: `[[nodiscard]] int compute();`

- and more https://en.cppreference.com/w/cpp/20

std::algorithm

`std::algorithm`

- A part of the C++ Standard Library.
- Provides a collection of functions for performing various operations on ranges of elements.
  - C++20 provides constrained versions of most algorithms in the namespace `std::ranges`.
- Standard algorithms help to manipulate data more effectively and concisely.
- Main points:
  - Learn what algorithms are available and use them.
  - Avoid usage of manual for loops.
  - Knowing the right std::algorithms improves coding speed and quality.
- https://en.cppreference.com/w/cpp/algorithm

`std::algorithm` categories

- Non-Modifying Sequence Operations:
  - Operate on elements without changing them. Used for searching, counting, and comparing.
  - `std::find`, `std::count`, `std::equal`, `std::mismatch`, `std::any_of` and many more
- Modifying Sequence Operations:
  - Change elements or their arrangement. Includes copying, transforming, swapping, and removing.
  - `std::copy`, `std::move`, `std::swap`, `std::transform`, `std::reverse` and many more
- Sorting and Related Operations:
  - Organize and manage sorted data. Involves sorting, partitioning, and set operations.
  - `std::partition`, `std::sort`, `std::lower_bound`, `std::upper_bound`, `std::min` and many more
- Numeric Operations:
  - Perform numerical calculations like summing and computing products.
  - `std::iota`, `std::accumulate`, `std::inner_product` and many more

Non-Modifying Sequence Operations

Do: Use standard algorithms for clarity.

```
bool hasEven = std::any_of(numbers.begin(), numbers.end(), [](int n) { return n % 2 == 0;});
```

Don’t: Write custom code for common tasks.

```
bool hasEven = false;
for (size_t i = 0; i < numbers.size(); ++i) {
    if (numbers[i] % 2 == 0) {
        hasEven = true;
        break;
    }
}
```

`std::for_each`, `std::ranges::for_each`, `std::for_each_n`, `std::ranges::for_each_n`, `std::all_of`, `std::any_of`, `std::none_of`, `std::ranges::all_of`, `std::ranges::any_of`, `std::ranges::none_of`, `std::ranges::contains`, `std::ranges::contains_subrange`, `std::find`, `std::find_if`, `std::find_if_not`, `std::ranges::find`, `std::ranges::find_if`, `std::ranges::find_if_not`, `std::ranges::find_last`, `std::ranges::find_last_if`, `std::ranges::find_last_if_not`, `std::find_end`, `std::ranges::find_end`, `std::find_first_of`, `std::ranges::find_first_of`, `std::adjacent_find`, `std::ranges::adjacent_find`, `std::count`, `std::count_if`, `std::ranges::count`, `std::ranges::count_if`, `std::mismatch`, `std::ranges::mismatch`, `std::equal`, `std::ranges::equal`, `std::search`, `std::ranges::search`, `std::search_n`, `std::ranges::search_n`, `std::ranges::starts_with`, `std::ranges::ends_with`, `std::ranges::fold_left`, `std::ranges::fold_left_first`, `std::ranges::fold_right`, `std::ranges::fold_right_last`, `std::ranges::fold_left_with_iter`, `std::ranges::fold_left_first_with_iter`

Modifying Sequence Operations

Do: Use standard algorithms for clarity.

```
auto it = std::remove_if(numbers.begin(), numbers.end(), [](int n) { return n % 2 == 0; });
numbers.erase(it, numbers.end());
```

Don’t: Write custom code for common tasks.

```
for (auto it = numbers.begin(); it != numbers.end(); ) {
    if (*it % 2 == 0) {
        it = numbers.erase(it);
    } else {
        ++it;
    }
}
```

`std::copy`, `std::copy_if`, `std::ranges::copy`, `std::ranges::copy_if`, `std::copy_n`, `std::ranges::copy_n`, `std::copy_backward`, `std::ranges::copy_backward`, `std::move`, `std::ranges::move`, `std::move_backward`, `std::ranges::move_backward`, `std::swap`, `std::swap_ranges`, `std::ranges::swap_ranges`, `std::iter_swap`, `std::transform`, `std::ranges::transform`, `std::replace`, `std::replace_if`, `std::ranges::replace`, `std::ranges::replace_if`, `std::replace_copy`, `std::replace_copy_if`, `std::ranges::replace_copy`, `std::ranges::replace_copy_if`, `std::fill`, `std::ranges::fill`, `std::fill_n`, `std::ranges::fill_n`, `std::generate`, `std::ranges::generate`, `std::generate_n`, `std::ranges::generate_n`, `std::remove`, `std::remove_if`, `std::ranges::remove`, `std::ranges::remove_if`, `std::remove_copy`, `std::remove_copy_if`, `std::ranges::remove_copy`, `std::ranges::remove_copy_if`, `std::unique`, `std::ranges::unique`, `std::unique_copy`, `std::ranges::unique_copy`, `std::reverse`, `std::ranges::reverse`, `std::reverse_copy`, `std::ranges::reverse_copy`, `std::rotate`, `std::ranges::rotate`, `std::rotate_copy`,

Sorting and Related Operations

Do: Use standard algorithms for clarity.

```
std::vector<int> numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // A sorted vector of integers
bool found = std::binary_search(numbers.begin(), numbers.end(), 6);
```

Don’t: Write custom code for common tasks.

```
bool manualBinarySearch(const std::vector<int>& numbers, int value) {
    int left = 0, right = numbers.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (numbers[mid] == value) { return true; }
        else if (numbers[mid] < value) { left = mid + 1; }
        else { right = mid - 1; }
    }
    return false;
}
bool found = manual_binary_search(numbers, 6);
```

Sorting and Related Operations

`std::is_partitioned`, `std::ranges::is_partitioned`, `std::partition`, `std::ranges::partition`, `std::partition_copy`, `std::ranges::partition_copy`, `std::stable_partition`, `std::ranges::stable_partition`, `std::partition_point`, `std::ranges::partition_point`, `std::sort`, `std::ranges::sort`, `std::stable_sort`, `std::ranges::stable_sort`, `std::partial_sort`, `std::ranges::partial_sort`, `std::partial_sort_copy`, `std::ranges::partial_sort_copy`, `std::is_sorted`, `std::ranges::is_sorted`, `std::is_sorted_until`, `std::ranges::is_sorted_until`, `std::nth_element`, `std::ranges::nth_element`, `std::lower_bound`, `std::ranges::lower_bound`, `std::upper_bound`, `std::ranges::upper_bound`, `std::equal_range`, `std::ranges::equal_range`, `std::binary_search`, `std::ranges::binary_search`, `std::includes`, `std::ranges::includes`, `std::set_union`, `std::ranges::set_union`, `std::set_intersection`, `std::ranges::set_intersection`, `std::set_difference`, `std::ranges::set_difference`, `std::set_symmetric_difference`, `std::ranges::set_symmetric_difference`, `std::merge`, `std::ranges::merge`, `std::inplace_merge`, `std::ranges::inplace_merge`, `std::push_heap`, `std::ranges::push_heap`, `std::pop_heap`, `std::ranges::pop_heap`, `std::make_heap`, `std::ranges::make_heap`, `std::sort_heap`, `std::ranges::sort_heap`, `std::is_heap`, `std::ranges::is_heap`, `std::is_heap_until`, `std::ranges::is_heap_until`, `std::max`, `std::ranges::max`, `std::max_element`, `std::ranges::max_element`, `std::min`, `std::ranges::min`, `std::min_element`, `std::ranges::min_element`, `std::minmax`, `std::ranges::minmax`, `std::minmax_element`, `std::ranges::minmax_element`, `std::clamp`, `std::ranges::clamp`, `std::lexicographical_compare`, `std::ranges::lexicographical_compare`, `std::lexicographical_compare_three_way`, `std::next_permutation`, `std::ranges::next_permutation`, `std::prev_permutation`, `std::ranges::prev_permutation`, `std::is_permutation`, `std::ranges::is_permutation`

Numeric Operations

Do: Use standard algorithms for clarity.

```
std::vector<int> numbers(10);
std::iota(numbers.begin(), numbers.end(), 1);
int sum = std::accumulate(numbers.begin(), numbers.end(), 0);
```

Don’t: Write custom code for common tasks.

```
std::vector<int> numbers(10);
int value = 1;
for (size_t i = 0; i < numbers.size(); ++i) {
     numbers[i] = value++;
}

int sum = 0;
for (size_t i = 0; i < numbers.size(); ++i) {
    sum += numbers[i];
}
```

`std::iota`, `std::ranges::iota`, `std::accumulate`, `std::inner_product`, `std::adjacent_difference`, `std::partial_sum`, `std::reduce`, `std::exclusive_scan`, `std::inclusive_scan`, `std::transform_reduce`, `std::transform_exclusive_scan`, `std::transform_inclusive_scan`

std::range

`std::range`

- A range is a concept which defines the requirements for a type that allows iteration over its elements.
- A valid range should provide a `begin()` and `end()` (end sentinel).
- Examples: `std::array`, `std::vector`, `std::string_view`, `std::span`, C-style array etc.
- `#include <ranges>`, `namespace std::ranges`

Ranges vs. Traditional Iterators

- Traditional STL uses pairs of iterators to denote ranges, which can be error-prone and verbose.

```
std::vector<int> numbers{1, 4, 2, 7, 9, 3, 5};
std::sort(numbers.begin(), numbers.end());
```

- `std::ranges` simplifies this by treating the sequence as a unified object.

```
std::vector<int> numbers{1, 4, 2, 7, 9, 3, 5};
std::ranges::sort(numbers);
```

Views and Range Adaptors

- Views are lightweight ranges, non-owning references to data.
  - `std::ranges::operation_view { range, arguments... }`

```
std::vector<int> numbers = {1, 2, 3, 4, 5, 6};
std::ranges::for_each(std::ranges::take_view{numbers, 3}, [](int n){ std::cout << n << " "; }); // output: 1 2 3
```

- Usually views are not created using their constructors, but instead using **range adapters**.
- Range adaptors are helper functions that creates views, such as `views::filter`, `views::transform`, and `views::take`.
- To create a pipeline of ranges and views use operator `|`:

```
std::vector<int> numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
auto result = numbers | std::views::filter([](const auto& n) { return n % 2 == 0; })
                      | std::views::transform([](const auto& n) { return n * n; })
                      | std::views::take(4)
                      | std::views::reverse; // Output: 64, 36, 16, 4
```

Lazy Evaluation, prime numbers example

- A view is just the description of a processing.
- The processing does not happen when one defines the view.
- The actual processing is performed element by element when the next value is asked for.

```
bool isPrime(int num) {  // predicate returns true if num is prime
    if (num <= 1) return false;
    if (num % 2 == 0) return false;
    for (int i = 2; i * i <= num; ++i) {
        if (num % i == 0) return false;
    }
    return true;
}

int main() {
    auto numbers = std::ranges::iota_view{2}; // Generate an infinite range starting from 2
    // Filter the numbers to keep only prime numbers and take the first 20
    auto primeNumbers = numbers | std::views::filter(isPrime) | std::views::take(20);
    for (int prime : primeNumbers) {
        std::cout << prime << " "; // Print the first 20 prime numbers
    }
}
```

Projections

Sort struct by different fields.

```
struct Person{
    std::string name{};
    std::size age{};
};
```

```
std::vector<Person> persons{ {"A", 10}, {"B", 2},{"C", 30},
                             {"D", 67}, {"E", 23}, {"F", 42} };

std::ranges::sort(persons, {}, &Person::name); // ascending by name
std::ranges::sort(persons, std::ranges::greater(), &Person::name); // descending by name
std::ranges::sort(persons, {}, &Person::age); // ascending by age
std::ranges::sort(persons, std::ranges::greater(),&Person::age); // descending by age
```

Example with std::variant

- Create a vector containing numbers and strings
- Filter out the strings using `std::views::filter`

```
int main() {
    std::vector<std::variant<int, std::string>> mixedNumbers = {1, 2, 3, "four", "five"};

    auto stringValues = mixedData | std::views::filter([](const auto& val) {
        return std::holds_alternative<std::string>(val);
    });

    for (const auto& str : stringValues) {
        std::cout << std::get<std::string>(str) << " ";
    }
    // Output: four five
}
```

GNURadio4 uses modern C++

- While developing `GNURadio4`, we extensively utilize modern C++ features:
  - **Concepts** allow for more expressive and safer templates.
  - **std::ranges** and **std::algorithm** make code more concise and easier to understand.
    - Altought syntax might seem unfamiliar to newcomers.
  - **constexpr** enables more computations at compile-time improving runtime performance.
  - and many more ...

Compiler Support:

- Not all features are fully implemented in all compilers yet.
- Stay tuned: https://en.cppreference.com/w/cpp/compiler_support

Conclusion:

- Learn and use modern C++ and standard algorithms whenever feasible.
- It improves your code quality, safety, readability, and performance.

Block implementation examples

Block examples

We will implement several example blocks

- Moving average block
- Simple FIR block
- Decimation block
- Selector block
- Even Number filter block

Example 1

Moving average block

- Implement a block which calculates moving average of the last N samples

Implementation Checklist:

- `HistoryBuffer` to store samples
- Implement both `processBulk` or `processOne`
- Use Non-Type Template Parameter (NTTP) to select between `processBulk` and `processOne`
- `std::accumulate`

```
enum class ProcessFunction {
    USE_PROCESS_BULK     = 0,
    USE_PROCESS_ONE      = 1
};
```

Example 1

```
template<typename T, std::size_t bufferSize, ProcessFunction UseProcessVariant>
struct MovingAverageBlock : public Block<MovingAverageBlock<T, bufferSize, UseProcessVariant>> {
    PortIn<T>  in;
    PortOut<T> out;
    HistoryBuffer<T, bufferSize> _inputHistory{};

    constexpr T processOne(const T& input) noexcept
    requires(UseProcessVariant == ProcessFunction::USE_PROCESS_ONE)
    {
        _inputHistory.push_back(input); // Update history buffer with new input sample
        T sum = std::accumulate(_inputHistory.begin(), _inputHistory.end(), T{});
        return sum / static_cast<T>(_inputHistory.size());
    }

    constexpr work::Status processBulk(std::span<const T> input, std::span<T> output) noexcept
    requires(UseProcessVariant == ProcessFunction::USE_PROCESS_BULK)
    {
        for (std::size_t i = 0; i < input.size(); ++i) {
            _inputHistory.push_back(input[i]); // Update history buffer with new input sample
            T sum = std::accumulate(_inputHistory.begin(), _inputHistory.end(), T{});
            output[i] = sum / static_cast<T>(_inputHistory.size());
        }
        return work::Status::OK;
    }
};
ENABLE_REFLECTION_FOR_TEMPLATE_FULL((typename T, std::size_t bufferSize, ProcessFunction pf),
                                    (MovingAverageBlock<T, bufferSize, pf>), in, out);
```

Example 2

Simple FIR filter

- Extend the previous implementation and implement a simple FIR filter

Implementation Checklist

- `HistoryBuffer` to store samples
- Add coefficients
- Implement only processBulk
- Use `std::transform_reduce``
- Execution policy https://en.cppreference.com/w/cpp/algorithm/execution_policy_tag
  - `std::execution::seq`
  - `std::execution::par`
  - `std::execution::par_unseq`
  - `std::execution::unseq`

Example 2

```
template<std::floating_point T, std::size_t bufferSize>
struct FIRBlock : public Block<FIRBlock<T, bufferSize>> {
    PortIn<T>  in;
    PortOut<T> out;
    HistoryBuffer<T, bufferSize> _inputHistory{};
    std::vector<T> _coefficients{};

    constexpr work::Status processBulk(std::span<const T> input, std::span<T> output) noexcept {
        for (std::size_t i = 0; i < input.size(); ++i) {
            _inputHistory.push_back(input[i]);  // Update history buffer with new input sample

            output[i] = std::transform_reduce(
                std::execution::seq, // Execution policy
                _coefficients.cbegin(), _coefficients.cend(), // Range of coefficients
                _inputHistory.cbegin(),
                T{}, // Initial value for reduction
                std::plus<>{}, // Binary operation to accumulate results
                std::multiplies<>{} // Binary operation to multiply elements
            );
        }
        return work::Status::OK;
    }
};
ENABLE_REFLECTION_FOR_TEMPLATE_FULL((typename T, std::size_t bufferSize), (FIRBlock<T, bufferSize>), in, out);
```

Example 3

Decimation block

- Implement block that decimate N samples to 1 by averaging them

Implementation Checklist:

- Add `Resampling<>` to allow decimation
- Use `input_chunk_size` and `output_chunk_size` to adjust input and output `std::span` size in processBulk
- Implement `processBulk`
- `std::views::chunk`
- `std::views::transform`
- `std::accumulate`
- `std::ranges::copy`

Example 3

```
template<typename T>
struct DecimationBlock : public Block<DecimationBlock<T, Resampling<>>> {
    PortIn<T>  in;
    PortOut<T> out;

    constexpr work::Status processBulk(std::span<const T> input, std::span<T> output) noexcept
    {
        auto decimatedView = input
            | std::views::chunk(input_chunk_size)  // Create chunks of N samples
            | std::views::transform([](auto chunk) {   // Average each chunk
                return std::accumulate(chunk.begin(), chunk.end(), 0.0f) / chunk.size();
            });
        std::ranges::copy(decimatedView, output.begin()); // Copy the averaged results to the output
        return work::Status::OK;
    }
};
ENABLE_REFLECTION_FOR_TEMPLATE_FULL((typename T), (DecimationBlock<T>), in, out);
```

Example 4

Selector block

- Implement a block with multiple input ports and a single output port. It should be capable of transferring data from any chosen input port to the output port.

Implementation Checklist:

- Use array of ports
- `gr::Annotated`
- Implement `processBulk` with array of ports
- `std::ranges::copy`

Example 4

```
template<typename T, std::size_t nPorts = 2>
struct SelectorBlock : public Block<SelectorBlock<T, nPorts>> {
    std::array<PortIn<T>, nPorts>  inputs;
    PortOut<T>                     out;
    // selected_port will be auto updated via Tags
    Annotated<gr::Size_t, "selected_port", Visible, Doc<"selected port index">>  selected_port  = 0U;

   template<ConsumableSpan TInSpan>
    work::Status processBulk(const std::span<TInSpan> &ins, std::span<T> &out) {
        // we assume that selected_port was checked already for limits
        std::ranges::copy(ins[selected_port], out.begin());
        return work::Status::OK;
    }
};
ENABLE_REFLECTION_FOR_TEMPLATE_FULL((typename T, std::size_t nPorts),
                                    (SelectorBlock<T, nPorts>), inputs, out, selectedPort);
```

Example 5

Event Number filter block

- Implement block that can filter even numbers

Implementation Checklist:

- Use `Async` output ports
- Implement `processBulk` with signature for `Async` ports
- `PublishableSpan`
- `out.publish(nPublished)`
- `std::ranges::views::filter`

Example 5

```
template<typename T>
struct EvenNumberFilterBlock : public Block<EvenNumberFilterBlock<T>> {
    PortIn<T>  in;
    PortOut<T, Async> out;

    constexpr work::Status processBulk(std::span<const T> &in, PublishableSpan auto &out) noexcept {
        auto evenNumbers = in | std::views::filter([](const T& item) {
            return item % 2 == 0;
        });

        // Copy the even numbers to the output span
        std::ranges::copy(evenNumbers, out.begin());
        std::size_t nPublished = std::ranges::distance(evenNumbers);
        out.publish(nPublished); // we must publish for Async ports

        return work::Status::OK;
    }
};
ENABLE_REFLECTION_FOR_TEMPLATE_FULL((typename T), (EvenNumberFilterBlock<T>), in, out);
```

Performance and benchmarking

Performance and benchmarking

- Performance is crucial, which is why we are using C++.
- Here are some key lessons we've learned:
  - Do not guess; measure.
  - Premature optimization is the root of all evil.
- For benchmarking, we use our extension of the `boost-ut` library https://github.com/fair-acc/gnuradio4/tree/main/bench
- Profiling tools to identify hot spots in the code
  - `perf` + clion IDE
  - `Valgrind` + clion IDE

Benchmark example 1

bm_fft

- It tests the performance of FFT implementations
- https://github.com/fair-acc/gnuradio4/blob/main/core/benchmarks/bm_fft.cpp

```
std::vector<float> signal = generateSinSample<float>(1024, sampleRate, frequency, amplitude);
gr::blocks::fft::FFT<T, DataSet<float>, FFTw> fft1({ { "fftSize", 1024 } });
std::ignore = fft1.settings().applyStagedParameters();
std::vector<DataSet<float>> result(1);
::benchmark::benchmark<nRepetitions>("fftw") = [&fft1, &signal, &result] {
        std::ignore = fft1.processBulk(signal, result);
};
```

FFT benchmark results

benchmark:		CPU branch misses	ops/s	mean	<CPU-I>	CPU cache misses	stddev	#N	CTX-SW	total time	min	median	max
std::complex<float> - fftw	PASS	34.0k / 6.19M = 0.5%	30.7k	33 us	378k	36.4k / 468k = 7.8%	49 us	100	0	3 ms	27 us	27 us	515 us
std::complex<float> - fft	PASS	20.0k / 7.31M = 0.3%	29.4k	34 us	467k	12.0k / 411k = 2.9%	9 us	100	0	3 ms	32 us	33 us	121 us
std::complex<float> - fftCT	PASS	9.62k / 20.4M = 0.0%	8.0k	125 us	1.1M	4.13k / 61.3k = 6.7%	1 us	100	0	12 ms	124 us	124 us	134 us
std::complex<double> - fftw	PASS	24.8k / 7.16M = 0.3%	26.2k	38 us	423k	43.4k / 1.08M = 4.0%	35 us	100	0	4 ms	34 us	34 us	384 us
std::complex<double> - fft	PASS	14.5k / 8.29M = 0.2%	21.8k	46 us	546k	41.2k / 1.05M = 3.9%	4 us	100	0	5 ms	45 us	45 us	88 us
std::complex<double> - fftCT	PASS	22.1k / 23.7M = 0.1%	5.3k	189 us	1.6M	43.1k / 641k = 6.7%	14 us	100	0	19 ms	179 us	184 us	244 us
float - fftw	clang-18	27.1k / 4.23M = 0.6%	42.7k	23 us	249k	38.3k / 384k = 10.0%	61 us	100	0	2 ms	15 us	15 us	633 us
float - fft		15.1k / 4.79M = 0.3%	41.1k	24 us	342k	6.87k / 288k = 2.4%	4 us	100	0	2 ms	23 us	24 us	61 us
double - fftw	PASS	25.2k / 4.72M = 0.5%	40.0k	25 us	275k	33.7k / 665k = 5.1%	59 us	100	0	3 ms	19 us	19 us	612 us
double - fft	PASS	11.2k / 5.18M = 0.2%	31.1k	32 us	395k	9.57k / 617k = 1.6%	4 us	100	0	3 ms	31 us	32 us	75 us

benchmark:		#N	CTX-SW	CPU cache misses	CPU branch misses	<CPU-I>	min	mean	stddev	median	max	total tim	ops/s
std::complex<float> - fftw	PASS	100	0	34.3k / 457k = 7.5%	32.5k / 6.11M = 0.5%	362k	32 us	36 us	41 us	32 us	439 us	4 m	27.6k
std::complex<float> - fft	PASS	100	0	10.8k / 422k = 2.6%	25.3k / 7.38M = 0.3%	435k	62 us	63 us	8 us	62 us	140 us	6 m	15.8k
std::complex<float> - fftCT	PASS	100	0	3.77k / 71.5k = 5.3%	6.57k / 14.6M = 0.0%	929k	71 us	72 us	2 us	72 us	80 us	7 m	13.9k
std::complex<double> - fftw	PASS	100	0	59.0k / 1.10M = 5.3%	32.8k / 7.20M = 0.5%	414k	39 us	44 us	39 us	40 us	434 us	4 m	22.5k
std::complex<double> - fft	PASS	100	0	29.5k / 1.06M = 2.8%	17.9k / 8.41M = 0.2%	481k	69 us	70 us	5 us	69 us	111 us	7 m	14.3k
std::complex<double> - fftCT	PASS	100	0	15.5k / 443k = 3.5%	10.0k / 17.5M = 0.1%	1.3M	93 us	93 us	2 us	93 us	104 us	9 m	10.7k
float - fftw	gcc-13	100	0	29.3k / 359k = 8.2%	28.0k / 4.25M = 0.7%	242k	18 us	24 us	59 us	18 us	614 us	2 m	41.7k
float - fft		100	0	6.97k / 282k = 2.5%	15.1k / 4.80M = 0.3%	316k	49 us	50 us	3 us	50 us	83 us	5 m	19.9k
double - fftw	PASS	100	0	30.3k / 676k = 4.5%	30.0k / 4.82M = 0.6%	270k	21 us	28 us	61 us	22 us	636 us	3 m	35.6k
double - fft	PASS	100	0	16.5k / 612k = 2.7%	14.8k / 5.26M = 0.3%	342k	53 us	54 us	4 us	53 us	88 us	5 m	18.5k

Benchmark example 2

bm_Buffer

- It tests the performance of `CircularBuffer` with different number of producers and consumers
- https://github.com/fair-acc/gnuradio4/blob/main/core/benchmarks/bm_Buffer.cpp

benchmark:		CPU branch misses	ops/s	mean	<CPU-I>	CPU cache misses	stddev	#N	CTX-SW	total time	min	median	max
	SKIP												
POSIX: 1 producers -< 1 >-> 1 consumers	PASS	61.1k / 524k = 11.6%	24.1M	42 ms	306m	20.2k / 118k = 17.1%	8 ms	8	106	333 ms	35 ms	38 ms	55 ms
POSIX: 1 producers -< 1 >-> 2 consumers	PASS	71.7k / 621k = 11.5%	18.7M	53 ms	363m	18.4k / 123k = 14.9%	841 us	8	112	427 ms	52 ms	54 ms	55 ms
POSIX: 1 producers -< 1 >-> 4 consumers	PASS	74.8k / 650k = 11.5%	12.4M	81 ms	378m	14.0k / 118k = 11.9%	1 ms	8	120	647 ms	80 ms	80 ms	84 ms
POSIX: 2 producers -< 1 >-> 1 consumers	PASS	90.2k / 783k = 11.5%	6.6M	151 ms	457m	13.4k / 130k = 10.2%	30 ms	8	139	1 s	115 ms	178 ms	182 ms
POSIX: 2 producers -< 1 >-> 2 consumers	PASS	76.7k / 668k = 11.5%	5.7M	177 ms	389m	9.20k / 124k = 7.4%	7 ms	8	136	1 s	158 ms	180 ms	180 ms
POSIX: 2 producers -< 1 >-> 4 consumers	PASS	83.0k / 718k = 11.6%	5.6M	179 ms	418m	11.5k / 127k = 9.1%	936 us	8	136	1 s	178 ms	179 ms	181 ms
POSIX: 4 producers -< 1 >-> 1 consumers	PASS	87.1k / 679k = 12.8%	5.8M	174 ms	394m	71.6k / 198k = 36.1%	7 ms	8	146	1 s	160 ms	176 ms	182 ms
POSIX: 4 producers -< 1 >-> 2 consumers	PASS	86.7k / 752k = 11.5%	5.6M	179 ms	437m	9.58k / 131k = 7.3%	7 ms	8	147	1 s	164 ms	180 ms	192 ms
POSIX: 4 producers -< 1 >-> 4 consumers	PASS	106k / 931k = 11.4%	3.8M	265 ms	541m	9.38k / 205k = 4.6%	17 ms	8	229	2 s	227 ms	275 ms	280 ms
POSIX: 1 producers -<1024>-> 1 consumers	PASS	24.6k / 227k = 10.8%	11.5G	87 us	132m	1.50k / 5.38k = 27.8%	154 us	8	3	695 us	26 us	26 us	494 us
POSIX: 1 producers -<1024>-> 2 consumers	PASS	35.8k / 324k = 11.1%	8.9G	113 us	189m	1.38k / 5.74k = 24.1%	150 us	8	4	902 us	39 us	45 us	496 us
POSIX: 1 producers -<1024>-> 4 consumers	PASS	52.6k / 453k = 11.6%	4.2G	236 us	265m	1.82k / 12.6k = 14.5%	213 us	8	11	2 ms	154 us	156 us	799 us
POSIX: 2 producers -<1024>-> 1 consumers	PASS	56.9k / 493k = 11.5%	1.2G	855 us	288m	1.68k / 28.1k = 6.0%	155 us	8	33	7 ms	771 us	806 us	1 ms
POSIX: 2 producers -<1024>-> 2 consumers	PASS	62.2k / 540k = 11.5%	1.1G	924 us	315m	1.78k / 29.7k = 6.0%	197 us	8	34	7 ms	806 us	811 us	1 ms
POSIX: 2 producers -<1024>-> 4 consumers	PASS	56.6k / 491k = 11.5%	1.2G	865 us	287m	1.82k / 28.7k = 6.3%	151 us	8	33	7 ms	804 us	809 us	1 ms
POSIX: 4 producers -<1024>-> 1 consumers	PASS	61.7k / 532k = 11.6%	1.1G	907 us	310m	2.81k / 33.5k = 8.4%	207 us	8	34	7 ms	755 us	795 us	1 ms
POSIX: 4 producers -<1024>-> 2 consumers	PASS	53.5k / 467k = 11.5%	1.1G	943 us	273m	4.19k / 33.6k = 12.5%	350 us	8	34	8 ms	806 us	809 us	2 ms
POSIX: 4 producers -<1024>-> 4 consumers	PASS	52.7k / 461k = 11.4%	1.1G	926 us	269m	1.45k / 28.2k = 5.1%	201 us	8	34	7 ms	789 us	813 us	1 ms

PerformanceMonitor block

- This block is used to track and report performance metrics
  - Memory usage
  - Sample rate
- Printout or CSV file
- Qa example: https://github.com/fair-acc/gnuradio4/blob/main/core/test/qa_PerformanceMonitor.cpp

```
3 optional settings are available: qa_PerformanceMonitor <run_time>[in sec] <test_case_id>[1:no tags,2:moderate,3:1-to-1] <output_file_path>
<run_time>:120 s, <test_case_id>:1, <output_file_path>:
Performance at 2024-08-26T08:45:04.000439, #0 dT:1.0001254 s, rate:281 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:05.000439, #1 dT:1.0003349 s, rate:287 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:06.000439, #2 dT:1.0000144 s, rate:287 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:07.000440, #3 dT:1.0002021 s, rate:287 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:08.000440, #4 dT:1.0001345 s, rate:290 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:09.000440, #5 dT:1.0002948 s, rate:267 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:10.000440, #6 dT:1.0002573 s, rate:287 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:11.000441, #7 dT:1.000321 s, rate:287 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:12.000441, #8 dT:1.0002369 s, rate:289 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:13.000441, #9 dT:1.0002449 s, rate:286 MS/s, memory_resident:242 Mb
Performance at 2024-08-26T08:45:14.000441, #10 dT:1.000342 s, rate:288 MS/s, memory_resident:242 Mb
```

Thank you for your attention!