EQUINOX

Equinox: A C++11 platform for realtime SDR applications

FOSDEM 2019

Manolis Surligas
surligas@csd.uoc.gr
Libre Space Foundation & Computer Science Department, University of Crete

Introduction

Software Radio Platforms

- Every SDR needs a software platform
- The platform is responsible for:
  - Orchestration and scheduling of processing tasks
  - Data management and transfer
  - Provide set of commonly used processing tasks
- The platform can be application **specific** or **generic**

Software Radio Platforms

Application specific platforms:

- ✓ Tend to outperform the generic platforms
- ✓ They adapt better to the computational requirements
- ✓ Low latency
- ✗ No code re-use
- ✗ Less flexibility, longer development times

Software Radio Platforms

Generic platforms:

- ✓ All in one solution
- ✓ Reusable, flexible and extensible
- ✓ Fast development cycles
- ✓ Effort on the algorithm not at the platform
- ✓ Visual Programming Language (VPL) interface ➔ better designs
- ✗ Latency

Software Radio Platforms

Generic Purpose SDR platforms: The VPL paradigm

- Each processing task is represented with a graphical block
- Connections represent data transfers
- Executable is auto-generated based on the design

Existing SDR Platforms

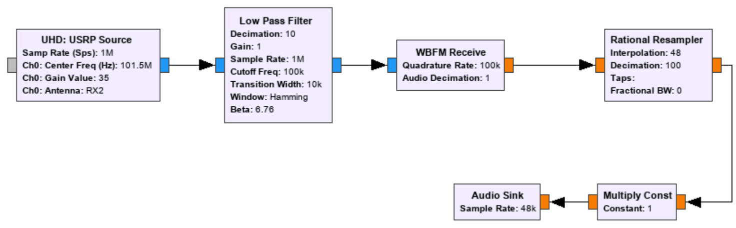
- GNU Radio
- LabView
- Matlab Simulink
- Pothos-SDR

Note!

All of these platforms are VPL based

Software Radio Platforms

Generic Purpose SDR platforms: The VPL paradigm



UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 101.5M
Ch0: Gain Value: 35
Ch0: Antenna: RX2

Low Pass Filter
Decimation: 10
Gain: 1
Sample Rate: 1M
Cutoff Freq: 100k
Transition Width: 10k
Window: Hamming
Beta: 6.76

WBFM Receive
Quadrature Rate: 100k
Audio Decimation: 1

Rational Resampler
Interpolation: 48
Decimation: 100
Taps:
Fractional BW: 0

Multiply Const
Constant: 1

Audio Sink
Sample Rate: 48k

GNU Radio FM receiver application in 30 seconds!

The Equinox SDR Platform

Equinox

- **Equinox** is a C++11 based SDR platform
- Based on message passing rather than streaming

Goals

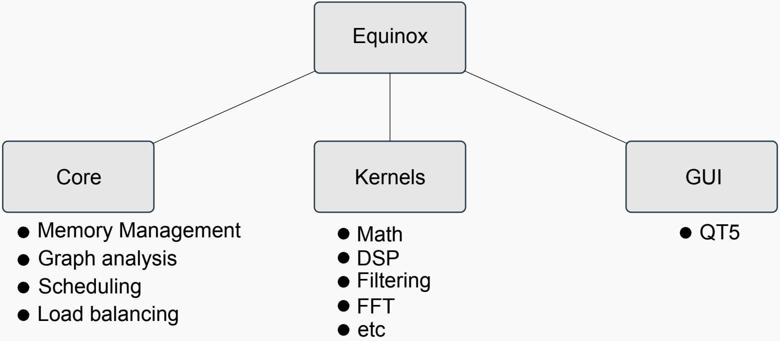
- Generic platform
- Extendable via plugins
- Adapt to application requirements
- **Proper handling of bursty transmissions**
- **Reduce latency**

Equinox

Why C++11?

- Modern, fast, complete
- Range based loops
- Shared pointers
- Integrated threading library
- Bye-bye Boost!!!

Architecture



Equinox

Core

- Memory Management
- Graph analysis
- Scheduling
- Load balancing

Kernels

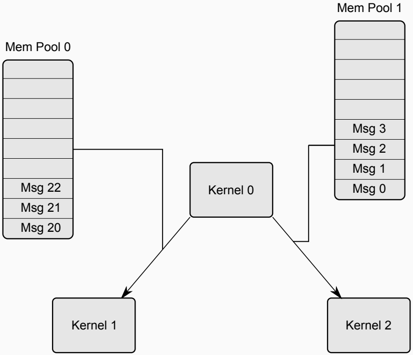
- Math
- DSP
- Filtering
- FFT
- etc

GUI

- QT5

Memory Management

- **NO** dynamic memory allocation
- Memory pools with pre-allocated memory
- Each output port holds a memory pool



Memory Management

- Kernels exchange messages of fixed size
- Each message is a *std::shared_ptr* pointer to a memory location at the memory pool
- Each output port is a message queue holding message pointers
- No memory copy, just pass the pointers (Zero-Copy)
- Automatic garbage collection, through the *std::shared_ptr* based messages

Equinox: Graph analysis & Load balancing

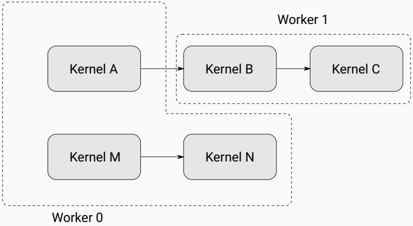
- Platforms like GNU Radio, follow a one-thread-per-block approach
- This is fine, as soon as the number of blocks is small
- Modern telecommunications systems require a large number of processing tasks
  - E.g IEEE 802.11 transceiver has about 40 blocks
- Thread synchronization, preemption and cache misses overhead starts to exceed the actual computation

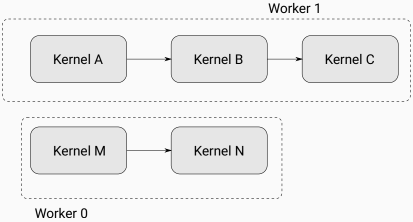
Graph analysis & Load balancing

- Equinox tries to balance these overheads
- Use minimum number of threads
- Exploit graph topology
- Assign efficiently the processing tasks into the available worker threads

Graph analysis & Load balancing

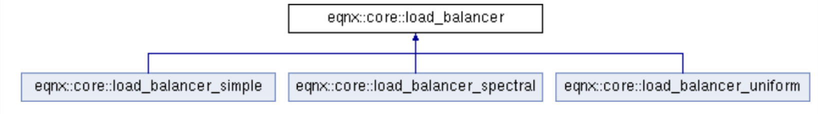
- The first task is to identify the connected components of the graph
- Use a slightly altered version of the DFS
- Different components should be assigned to different workers to avoid indirect data dependencies





Graph analysis & Load balancing

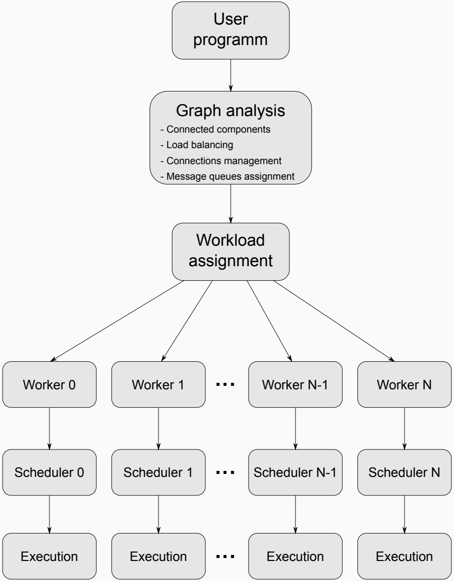
- Then split the graph into N sub-graphs, where N is the number of workers
- Equinox provides different ways to split the graph
- The most interesting is the spectral method
  - Split the graph based on the eigenvalues of the adjacency matrix
  - Minimizes the connections between sub-graphs


eqnx::core::load_balancer
eqnx::core::load_balancer_simple
eqnx::core::load_balancer_spectral
eqnx::core::load_balancer_uniform

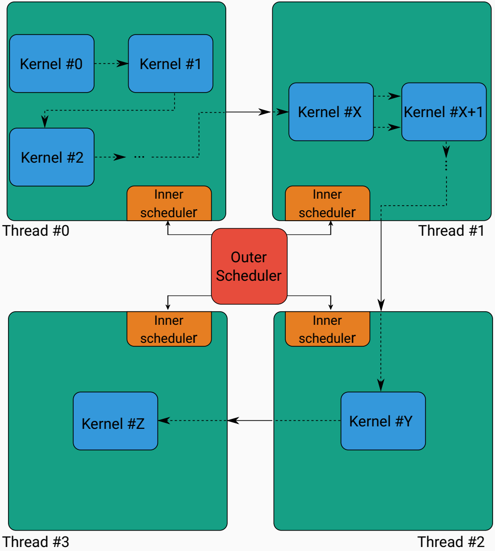
Scheduling

- The Equinox platform has two different scheduler types
  - **Inner Scheduler:** Operates for every worker
  - **Outer Scheduler:** Orchestrates the deployed inner schedulers
- Support of different inner schedulers through templates
- Currently we use the Round Robin inner scheduler

Scheduling


User programm
Graph analysis
- Connected components
- Load balancing
- Connections management
- Message queues assignment
Workload assignment
Worker 0
Worker 1
...
Worker N-1
Worker N
Scheduler 0
Scheduler 1
...
Scheduler N-1
Scheduler N
Execution
Execution
...
Execution
Execution

Scheduling



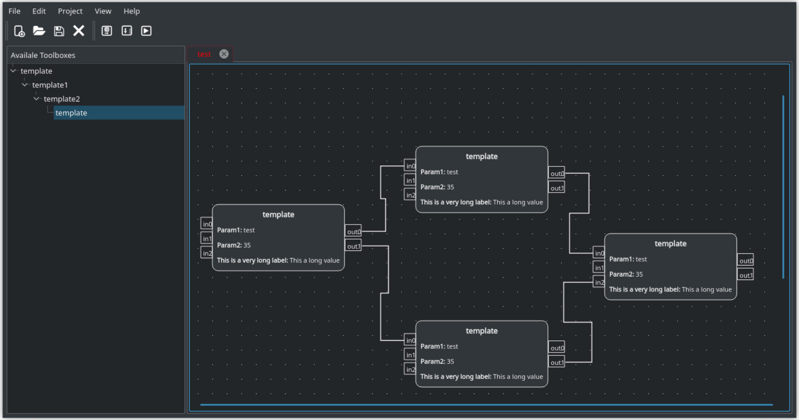
- #Threads == #Cores
- ┈┈➤: Virtual data connection
- ──➤: Shared memory connection

Delay comparison

# of blocks	GNU Radio	Equinox
4	55	58
8	105	81
12	131	59
16	158	67
24	262	189
32	378	182
48	2795	220
64	9384	233
72	16958	242
96	67716	268

Table 1: Delay comparison on i7-2600K @ 3.4 GHz

User interface



template

Param1: test

Param2: 35

This is a very long label: This a long value

template

Param1: test

Param2: 35

This is a very long label: This a long value

template

Param1: test

Param2: 35

This is a very long label: This a long value

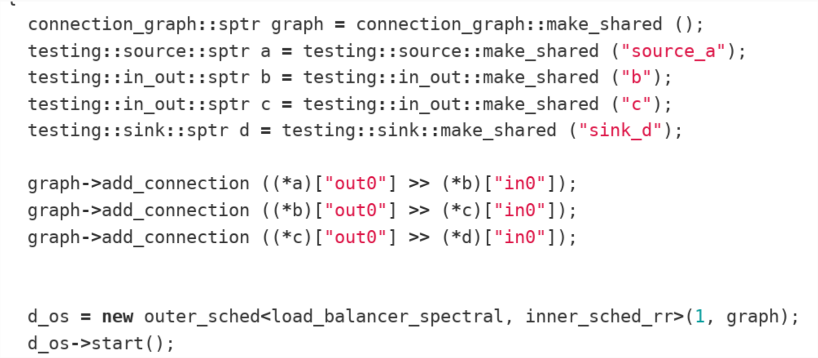
template

Param1: test

Param2: 35

This is a very long label: This a long value

A simple program

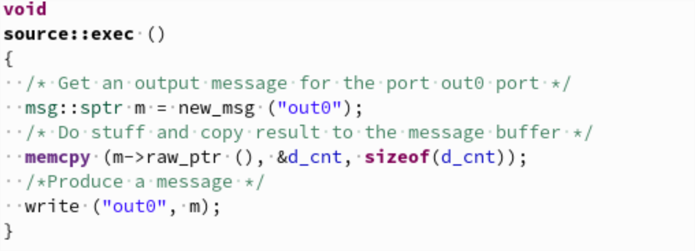


```
  connection_graph::sptr graph = connection_graph::make_shared ();
  testing::source::sptr a = testing::source::make_shared ("source_a");
  testing::in_out::sptr b = testing::in_out::make_shared ("b");
  testing::in_out::sptr c = testing::in_out::make_shared ("c");
  testing::sink::sptr d = testing::sink::make_shared ("sink_d");

  graph->add_connection ((*a)["out0"] >> (*b)["in0"]);
  graph->add_connection ((*b)["out0"] >> (*c)["in0"]);
  graph->add_connection ((*c)["out0"] >> (*d)["in0"]);


  d_os = new outer_sched<load_balancer_spectral, inner_sched_rr>(1, graph);
  d_os->start();
```

A simple program



```
void
source::exec ()
{
  /* Get an output message for the port out0 port */
  msg::sptr m = new_msg ("out0");
  /* Do stuff and copy result to the message buffer */
  memcpy (m->raw_ptr (), &d_cnt, sizeof(d_cnt));
  /*Produce a message */
  write ("out0", m);
}
```

Other applications?

Is Equinox only for SDR applications?

- Audio processing
- Video processing
  - Handle frames as messages ☺
- Network applications
  - Packet tagging
  - Filtering
  - DPI

Join the party!

`https://gitlab.com/equinox-sdr/equinox`

EQUINOX

Questions?