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
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
Generic Purpose SDR platforms: The VPL paradigm

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
Why C++11?

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

- 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
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
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
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

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

<table>
<thead>
<tr>
<th># of blocks</th>
<th>GNU Radio</th>
<th>Equinox</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>55</td>
<td>58</td>
</tr>
<tr>
<td>8</td>
<td>105</td>
<td>81</td>
</tr>
<tr>
<td>12</td>
<td>131</td>
<td>59</td>
</tr>
<tr>
<td>16</td>
<td>158</td>
<td>67</td>
</tr>
<tr>
<td>24</td>
<td>262</td>
<td>189</td>
</tr>
<tr>
<td>32</td>
<td>378</td>
<td>182</td>
</tr>
<tr>
<td>48</td>
<td>2795</td>
<td>220</td>
</tr>
<tr>
<td>64</td>
<td>9384</td>
<td>233</td>
</tr>
<tr>
<td>72</td>
<td>16958</td>
<td>242</td>
</tr>
<tr>
<td>96</td>
<td>67716</td>
<td>268</td>
</tr>
</tbody>
</table>
A simple program

```cpp
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

```c
void source::exec() {
    /* Get an output message for the port out0: */
    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
Questions?