GNURadio and CEDR: Runtime Scheduling to Heterogeneous Accelerators
Who are we?

Arizona State University
The University of Arizona
University of Michigan
The University of Texas at Austin
Wisconsin University of Wisconsin–Madison
ARM

GENERAL DYNAMICS
Mission Systems
Who are we?

Serhan Gener
PhD Student

Joshua Mack
PhD Student

Ali Akoglu

Anish NK
PhD Student

Jacob Holtom
PhD Student

Dan Bliss

Umit Ogras

Chaitali Chakrabarti
Motivation

- Heterogeneous computation holds a lot of potential, but it is typically difficult to effectively use.
- One approach: domain-specific processors
  - Restrict the scope of the problem while still enabling potential large gains
- Goal: develop a useable, domain-specific, coarse-scale heterogeneous processor
CEDR - Compiler-Integrated, Extensible DSSoC Runtime

- Runtime for heterogeneous systems that enables:
  - Hardware agnostic application development
  - Flexible integration of various software and hardware schedulers
  - Support for dispatching tasks to arbitrary hardware IPs

- Portable
  - Runs in Linux userspace
  - Daemon-based runtime
  - Validated across numerous FPGA/GPU & arm/aarch64/x86 systems

- Scalable
  - Supports arbitrary mixtures of dynamically submitted workloads

CEDR - [https://ua-rcl.github.io/CEDR/](https://ua-rcl.github.io/CEDR/)
CEDR - Runtime Model
CEDR for Application Developers - API-based Development

- Operational principles:
  - Users write code using hardware-agnostic APIs
  - CEDR dynamically loads a set of compatible API implementations at startup
  - Each API internally calls into CEDR
  - CEDR schedules each of the incoming API tasks dynamically to the system’s resources & signals completion to user application when done

```c
#include "dash.h"
int main()
{
    double *input = (double*) malloc(...)
    double *output = (double*) malloc(...)
    // ------------------ Replaced portion ------------------
    pthread_barrier_t kernel_1_barrier;
    pthread_barrier_init(&kernel_1_barrier, NULL, 2);
    enqueue_kernel("FFT", &input, &output, &size, &forwardTrans, &kernel_1_barrier);
    pthread_barrier_wait(&kernel_1_barrier);
    // ------------------
}
```

User-written Code

CEDR-equivalent code
Integration With GNURadio

- Developed an OOT module: gr-cedr\textsuperscript{1}
- Blocks in gr-cedr make calls to CEDR APIs
  - Flowgraphs using these blocks are either written directly in C++ or converted via Cython
- When run in CEDR, these flowgraphs are dynamically scheduled and dispatched to heterogeneous resources
- These same binaries are portable – without changes – to other heterogeneous systems running CEDR

\textsuperscript{1} gr-cedr: \url{https://github.com/WISCA/gr-cedr}
Demo - Setup

- **Goal:** demonstrate dynamic, heterogeneous scheduling of GNURadio blocks in CEDR via gr-cedr
- **Scenario:** execute a simple correlator application with and without GPU-acceleration on an Nvidia Jetson AGX Xavier
- **Validation:** monitor waterfalls generated from the standalone GNURadio & CEDR-based executions
Demo - Presentation

GNU Radio in CEDR
Summary & Future Directions

Summary

- Presented CEDR, a runtime for use on any Linux-based heterogeneous system along with an OOT module that illustrates how to integrate GNURadio applications into this runtime

- Demonstrated the ability to make dynamic, heterogeneous scheduling decisions for GNURadio blocks via a basic correlator flowgraph

Future Directions

- Expand the scope of supported blocks within the OOT module
- Work with the community to strive towards integration with GR4.0 newsched
Thank you, GRCon!

Questions?
Backup
GNURadio + DASH Runtime Integration Demo

- **Goal:**
  - Demonstrate Radar Correlator implemented with DASH APIs in GNURadio using CEDR

- **Scenario:**
  - Execute 3 applications concurrently: **Radar Correlator** runs as a continuous process along with **WiFi Tx** and **SAR**

- **Process:**
  - Show API based Radar Correlator in GNURadio
  - Generate binary for CEDR and execute
  - Show ARM Cores and FFT accelerators shared by three applications for FFT tasks

- **Validation:**
  - Monitor waterfalls generated from CPU and CEDR based execution of Radar Correlator

**SoC Configuration:**
- 2 FFT accelerators and 3 ARM Cores on Xilinx ZCU102 MPSoC
- Workload: 90 WiFi, 2 SAR, 1 Radar Correlator

**College of Engineering**
**Electrical & Computer Engineering**
CEDR - Performance Counters + Workload Profiling

- Integrated Performance Application Programming Interface (PAPI) counters

- Enables low-level performance profiling and workload characterization without changes in the user code at the granularity of individual kernels/DAG nodes

- Xilinx ZCU102: 113 different performance counters
  - perf::INSTRUCTIONS, perf::CACHE-MISSES, perf::BRANCHES, perf::STALLED-CYCLES-FRONTEND ...

<table>
<thead>
<tr>
<th>Applications</th>
<th>Instructions</th>
<th>Branches</th>
<th>Branch Misses</th>
<th>L1 Cache Loads</th>
<th>L1 Cache Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radar Correlator</td>
<td>158341</td>
<td>6273</td>
<td>958</td>
<td>69348</td>
<td>1435</td>
</tr>
<tr>
<td>Temporal Mitigation</td>
<td>3543527</td>
<td>349478</td>
<td>11944</td>
<td>1351507</td>
<td>4063</td>
</tr>
<tr>
<td>Pulse Doppler</td>
<td>15016980</td>
<td>686875</td>
<td>80525</td>
<td>6484258</td>
<td>192936</td>
</tr>
<tr>
<td>WiFi-TX</td>
<td>9861806</td>
<td>1102819</td>
<td>60703</td>
<td>3339442</td>
<td>11475</td>
</tr>
</tbody>
</table>
Large Scale Design Space Explorations

- 3480 configurations
- Scheduling 10 million total tasks on an off-the-shelf SoC < 3 hours
- Orders of magnitude faster than cycle-accurate and discrete-event simulators
Is acceleration always the best choice?

DSSoC should provide users with a development environment where application programmers can design their applications in a hardware-agnostic manner.

**MET**

High latency workload, 3 CPUs and 1 FFT, oversubscribed system (injection rate 2000 Mbps), total of 2610 FFT tasks

**EFT**
Is a scheduler with the best “cumulative execution time” performance always the best choice †?

- There is a trade-off between quality and complexity of scheduling decisions
  - ETF makes good decisions
  - When system is oversubscribed with high injection rate simple scheduler such as round robin becomes desirable

Portability

Verified CEDR across a number of different platforms

- Xilinx ZCU102
- Xilinx VCU128
- HTG-960 (Xilinx VU19P)
- Nvidia Jetson AGX Xavier
- Avnet Ultra96-v2
- Various x86 systems (CPU + GPU)
Sample Gantt Charts