GNURadio and CEDR: Runtime Scheduling to Heterogeneous Accelerators

COLLEGE OF ENGINEERING
Electrical & Computer Engineering

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?

The University of Arizona

Serhan Gener

PhD Student

Joshua Mack

PhD Student

Ali Akoglu

Arizona State University

Jacob Holtom

PhD Student

Dan Bliss

Chaitali Chakrabarti

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

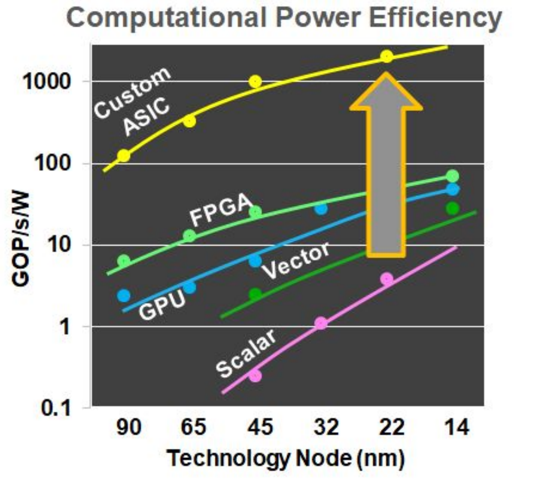
Anish NK

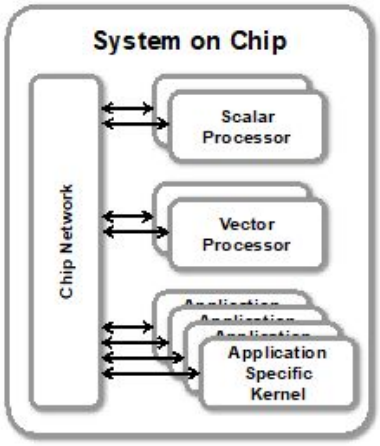
PhD Student

Umit Ogras

COLLEGE OF ENGINEERING
Electrical & Computer Engineering

Motivation

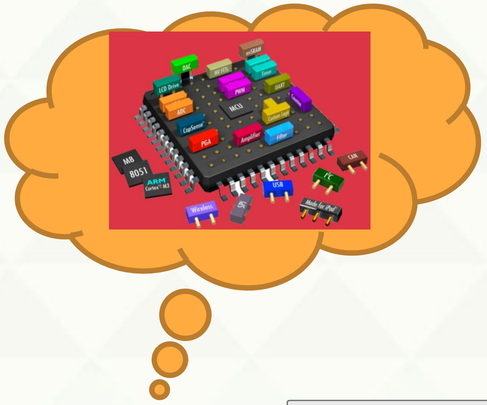


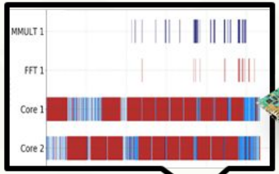


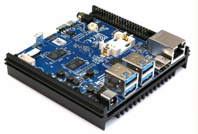
- Heterogeneous computation holds a lot of potential, but it is typically difficult to effectively use
- One approach: domain-specific processors
  - Restrict the scope of the problem while still enabling potential large gains
- Goal: develop a useable, domain-specific, coarse-scale heterogeneous processor

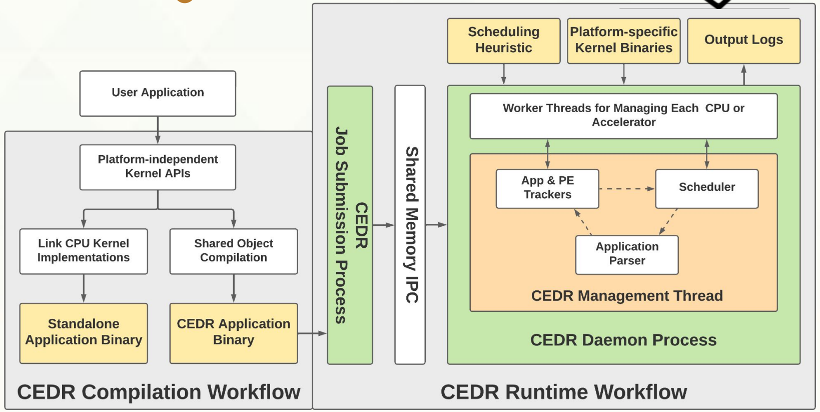
CEDR - Compiler-Integrated, Extensible DSSoC Runtime

- 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



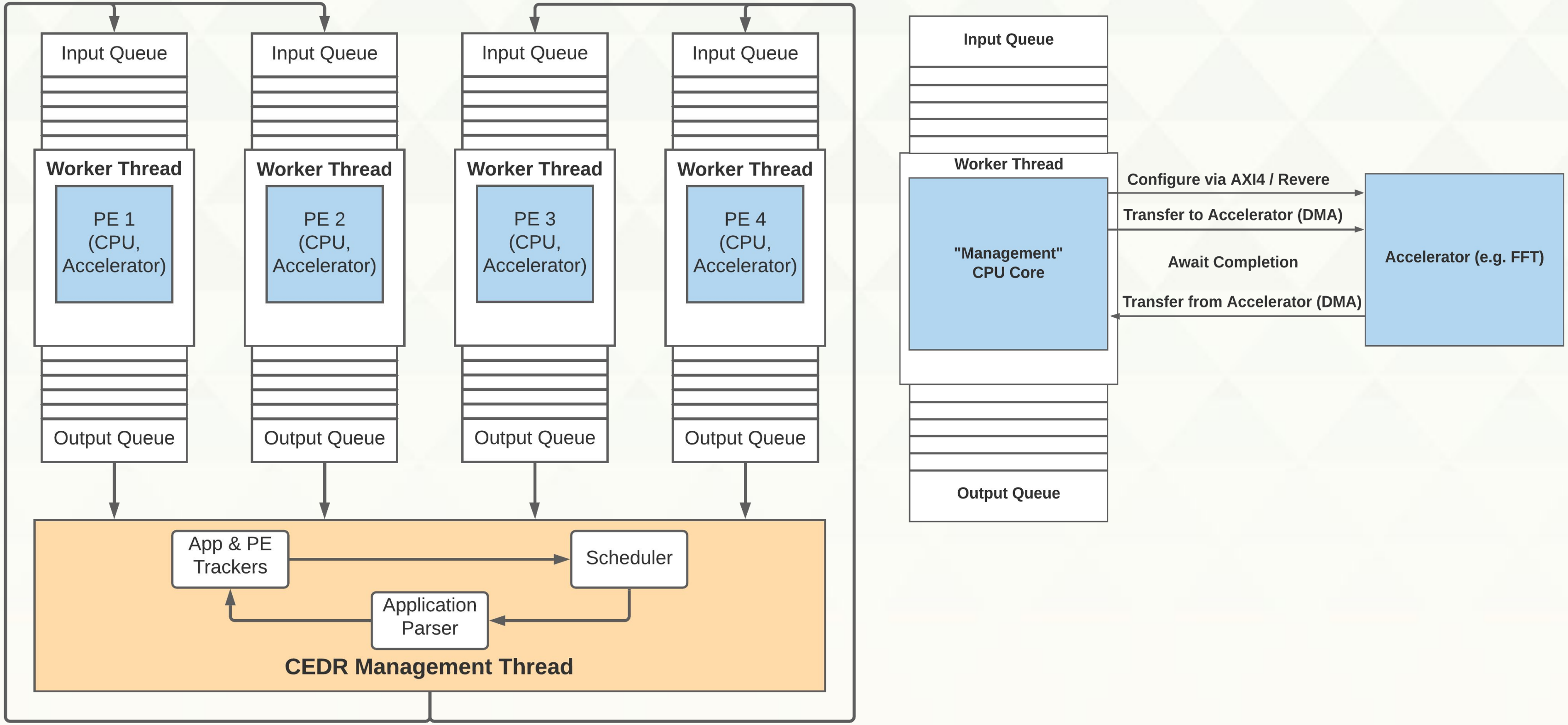






CEDR - https://ua-rcl.github.io/CEDR/
J. Mack et al "CEDR - A Compiler-integrated, Extensible DSSoC Runtime," ACM Transactions on Embedded Computing Systems (TECS), April 2022, https://doi.org/10.1145/3529257

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

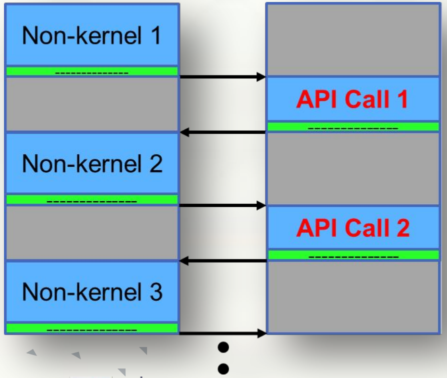
```
void DASH_FFT(double* input, double* output, size_t size, bool isForwardTransform)

void DASH_GEMM(double* A_re, double* A_im,
               double* B_re, double* B_im,
               double* C_re, double* C_im, size_t Row_A, size_t Col_A, size_t Col_B)

void DASH_FIR(double* input_re, double* input_im,
              double* filter_re, double* filter_im,
              double* output_re, double* output_im, size_t input_len, size_t filter_len)

void DASH_SpectralOpening(double* input, double* output, size_t io_len, size_t window_len);

void DASH_CIC(int* input_re, int* input_im,
              int* output_re, int* output_im,
              size_t input_len, unsigned int rate_change,
              unsigned int dif_delay, unsigned int len, bool is_up);
```




```
#include "dash.h"
int main(){
  double *input = (double*) malloc…
  double *output = (double*) malloc…
  DASH_FFT(input, output, size, forwardTrans);
}
```

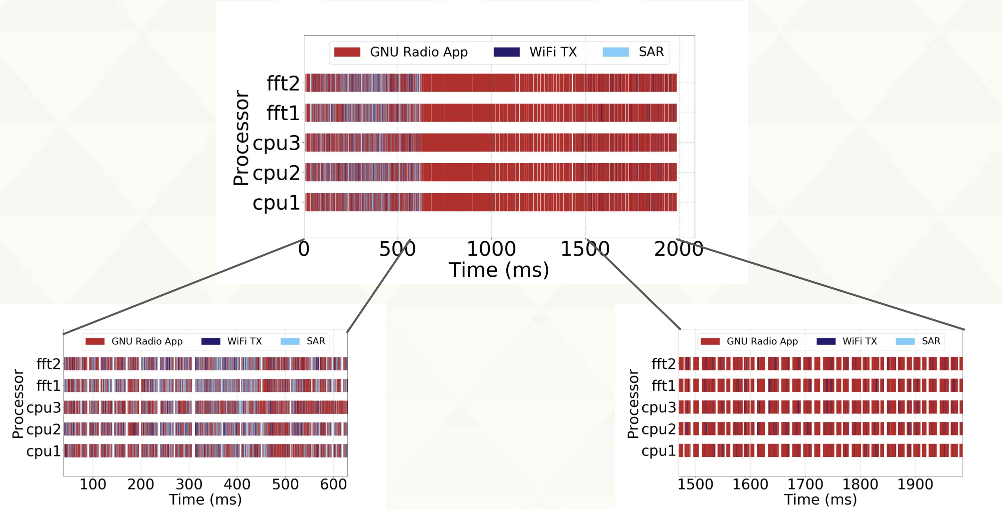
User-written Code

```
#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, nullptr, 2);
  enqueue_kernel("FFT", &input, &output, &size,
                 &forwardTrans, &kernel_1_barrier);
  pthread_barrier_wait(&kernel_1_barrier);
  // ---------------------------------------------
}
```

CEDR-equivalent code

Integration With GNURadio

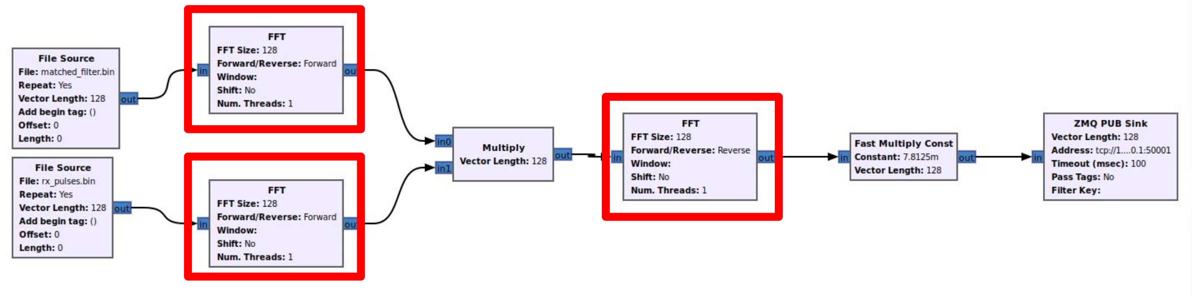
- Developed an OOT module: `gr-cedr`[1]
- Blocks in `gr-cedr` make calls to CEDR APIs
  - Flowgraphs using these blocks are either written directly in C++ or converted via Cython
- When run in CEDR, these flowgraphs are dynamically scheduled and dispatched to heterogeneous resources
- These same binaries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

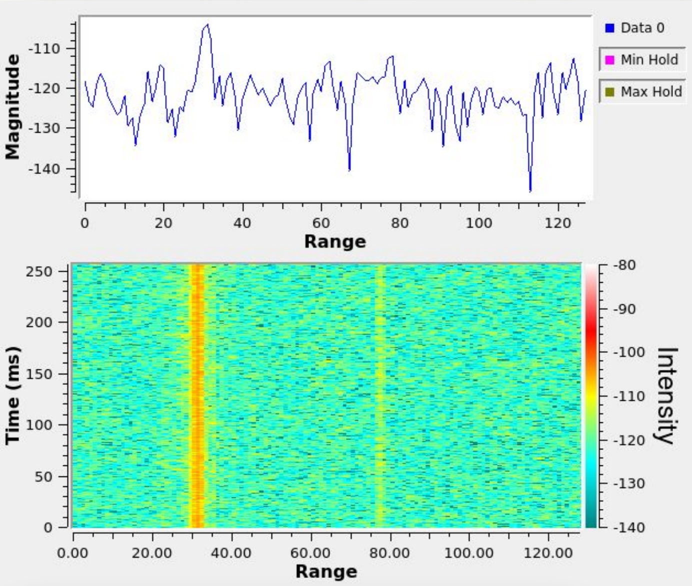


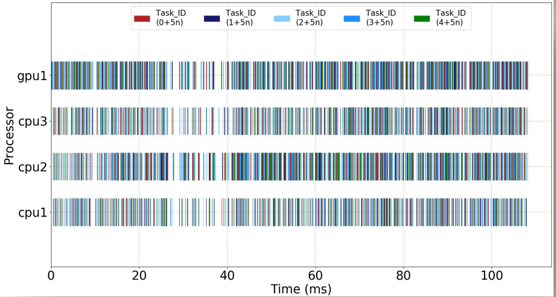
1. `gr-cedr`: https://github.com/WISCA/gr-cedr

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







COLLEGE OF ENGINEERING
Electrical & Computer Engineering

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`

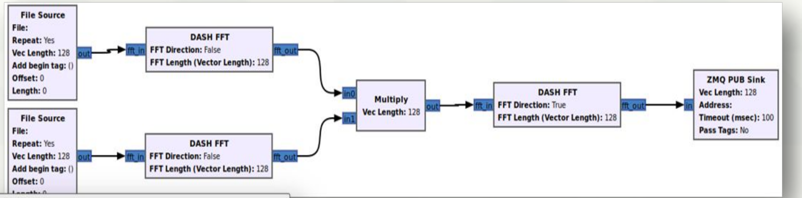
Thank you, GRCon!

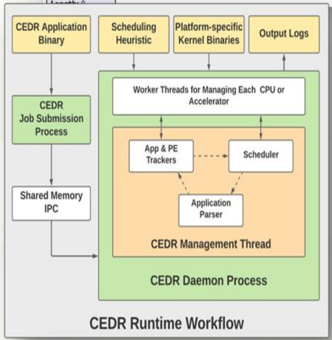
Questions?

Backup

GNURadio + DASH Runtime Integration Demo

- Goal:
  - Demonstrate Radar Correlator implemented with DASH APIs in GNURadio using CEDR
- Scenario:
  - Execute 3 applications concurrently: <u>**Radar Correlator**</u> runs as a continuous process along with <u>**WiFi Tx**</u> and <u>**SAR**</u>
- 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

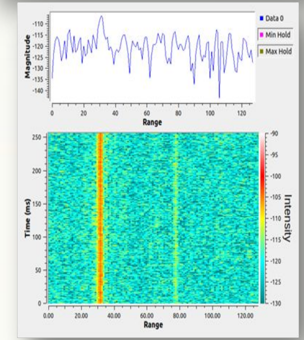


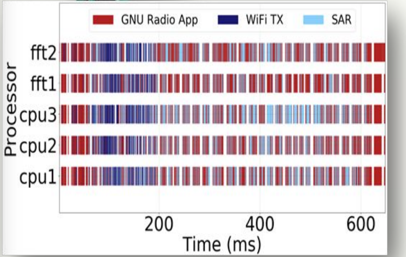


SoC Configuration:

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 …`

Applications	Instructions	Branches	Branch Misses	L1 Cache Loads	L1 Cache Misses
Radar Correlator	158341	6273	958	69348	1435
Temporal Mitigation	3543527	349478	11944	1351507	4063
Pulse Doppler	15016980	686875	80525	6484258	192936
WiFi-TX	9861806	1102819	60703	3339442	11475

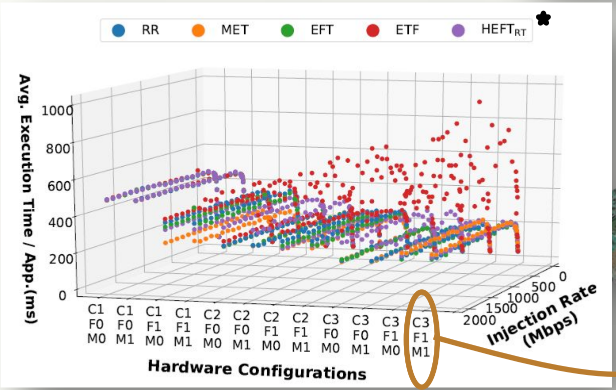
Task Name	Instructions	Branches	Branch Misses	L1 Cache Loads	L1 Cache Misses
Head Node	728	65	43	476	38
Linear Frequency Modulation	13417	875	110	6146	189
FFT_0	33411	1299	204	14781	384
FFT_1	47703	1398	126	21029	317
Multiplication	23607	382	54	10499	176
IFFT	23556	667	64	10010	195
Find maximum	15919	1587	357	6407	136

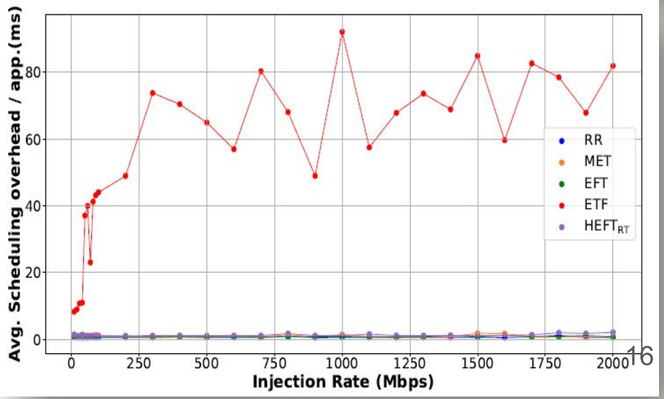
Large Scale Design Space Explorations

- 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

Input Configurations	12 Hardware configurations 3 CPUs (C1-C3), 1 FFT (F0-F1), 1 MMULT (M0-M1)
	5 Schedulers (SIMPLE, MET, EFT, ETF, $HEFT_{RT}$)
	2 Workloads (High latency, Low latency) (Table 2)
	29 Injection rates High latency (29 points between 10-2000 Mbps) Low latency (29 points between 1-1000 Mbps)
Output Metrics	Average cumulative execution time/ application
	Average execution time / application
	Average scheduling overhead / application
	Average resource utilization ratio

Application	Avg. Exec. Time CPU (ms)	Task Count	FFT Support	MMULT Support
Radar Correlator	0.82	7	✓	
Temporal Mitigation	4.39	11		✓
WiFi TX	16.12	93	✓	
Pulse Doppler	95.83	1027	✓	

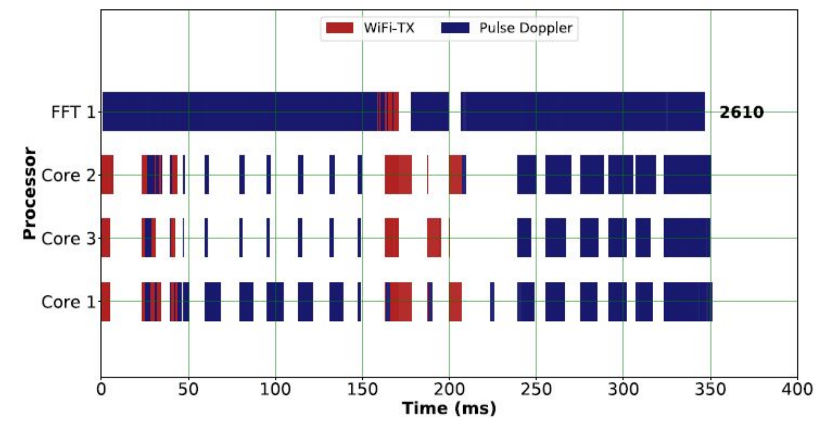




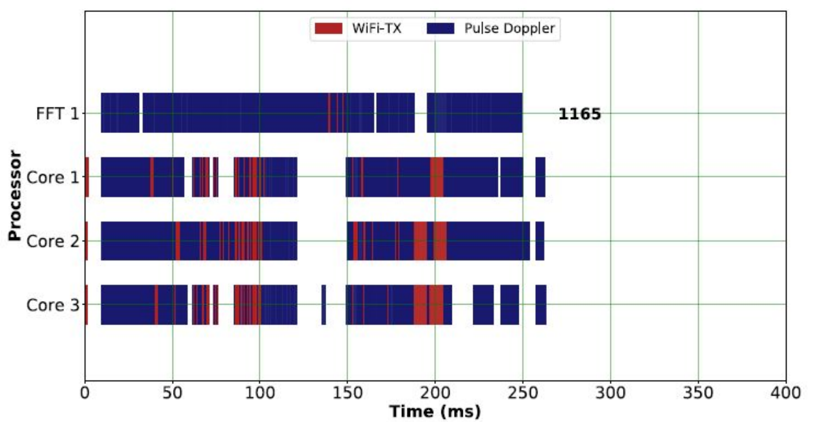
COLLEGE OF ENGINEERING
Electrical & Computer Engineering

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

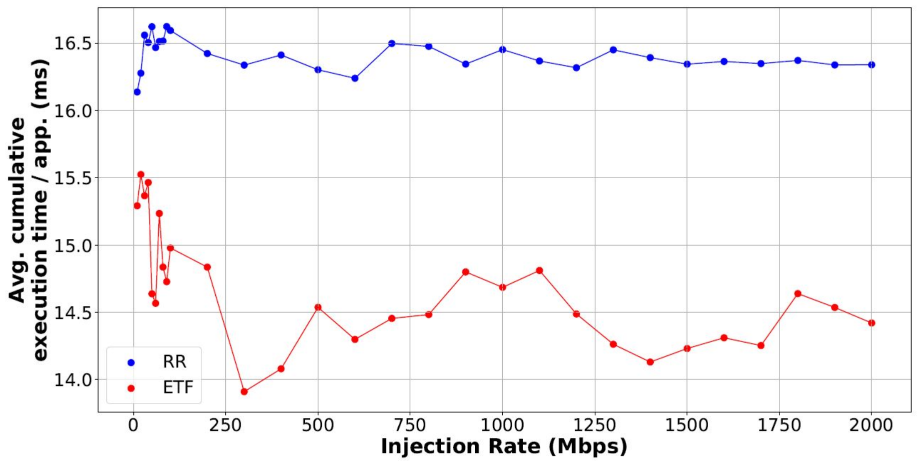


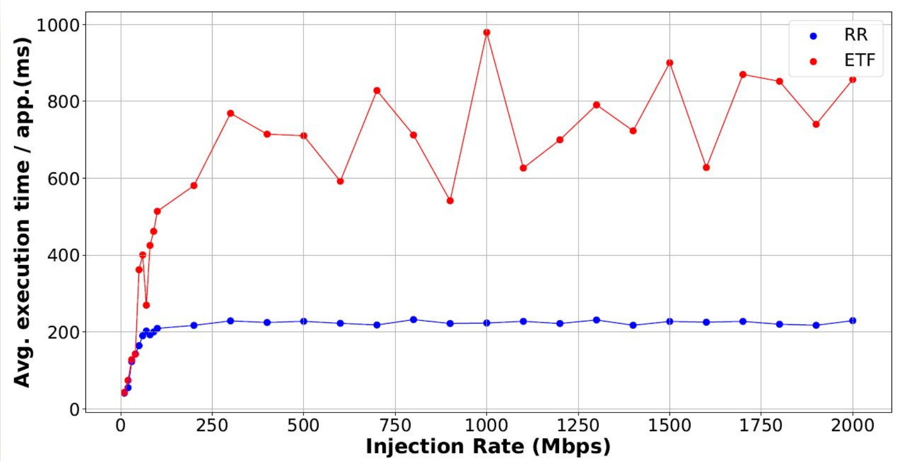
EFT

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

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





High latency workload, 3 CPUs and 1 FFT

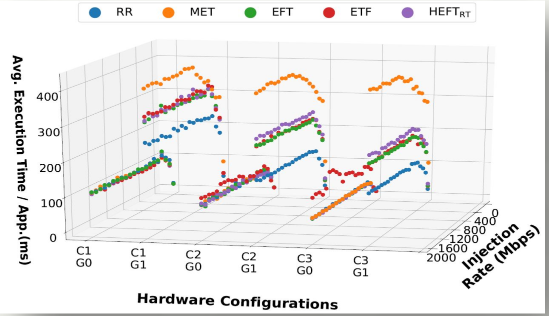
COLLEGE OF ENGINEERING
Electrical & Computer Engineering

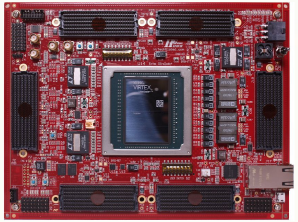
† A. Goksoy, A. Krishnakumar, S. Hassan, A. Farcas, A. Akoglu, R. Marculescu and U. Ogras, "DAS: Dynamic Adaptive Scheduling for Energy-Efficient Heterogeneous SoCs," *Embedded Systems Letters, vol 14, no 1, pp. 51-54,* March 2022. https://doi.org/10.1109/LES.2021.3110426

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)





COLLEGE OF ENGINEERING
Electrical & Computer Engineering

Sample Gantt Charts