A Unified Portable and Programmable Framework for Task-Based Execution and Dynamic Resource Management on Heterogeneous Systems

Serhan Gener[1], Sahil Hassan[1], Liangliang Chang[2],
Chaitali Chakrabarti[2], Tsung-Wei Huang[3], Umit Ogras[3], Ali Akoglu[1]

[1]University of Arizona, Tucson, AZ, USA
{**gener**,sahilhassan,akoglu}@arizona.edu

[2]Arizona State University, Phoenix, AZ, USA
{lchang21,chaitali}@asu.edu

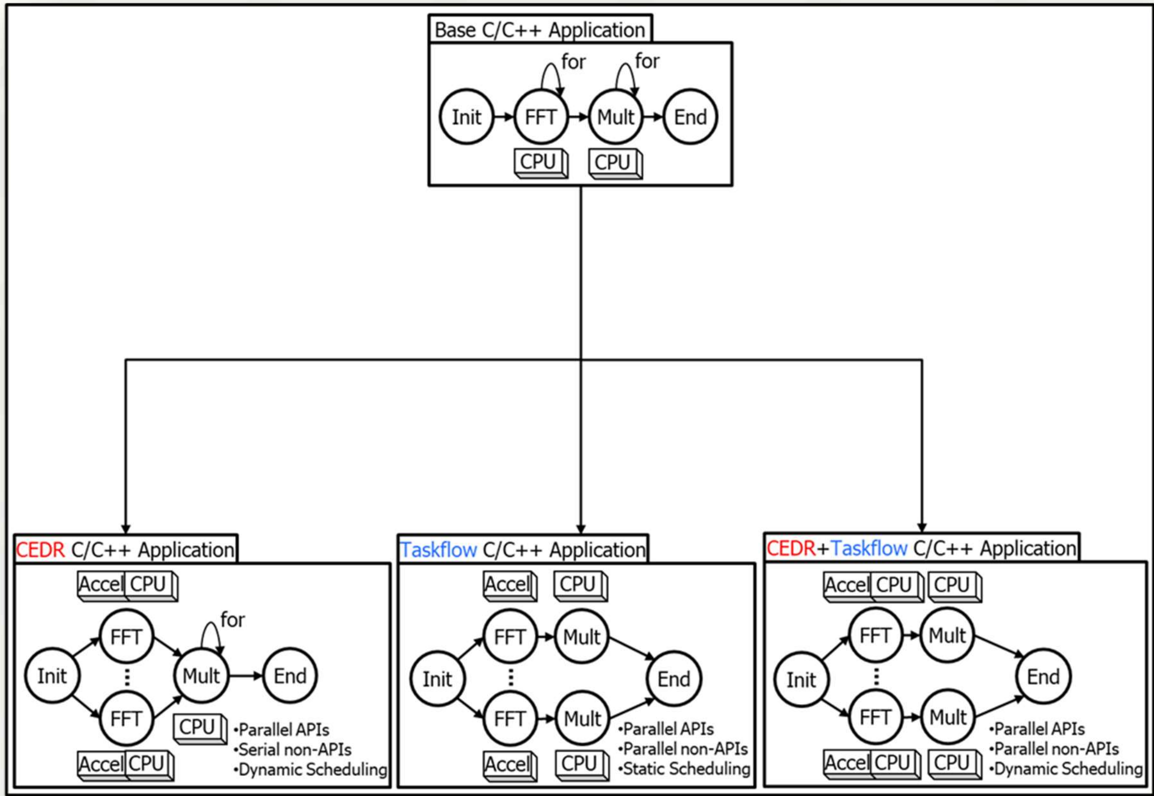
[3]University of Wisconsin at Madison, Madison, WI, USA
{tsung-wei.huang,uogras}@wisc.edu

COLLEGE OF ENGINEERING
Electrical & Computer Engineering

Reconfigurable Computing Lab

Motivation

- Heterogenous systems are widely used on SoC to HPC scale
- Optimizing performance while maintaining programmability remains difficult
  - API-based Runtime (**CEDR**): Dynamic scheduling improves programmability but misses parallelization in non-API regions
  - Parallel programming (**Taskflow**): Full task parallelization but relies on static scheduling ➔resource contention under dynamic workloads

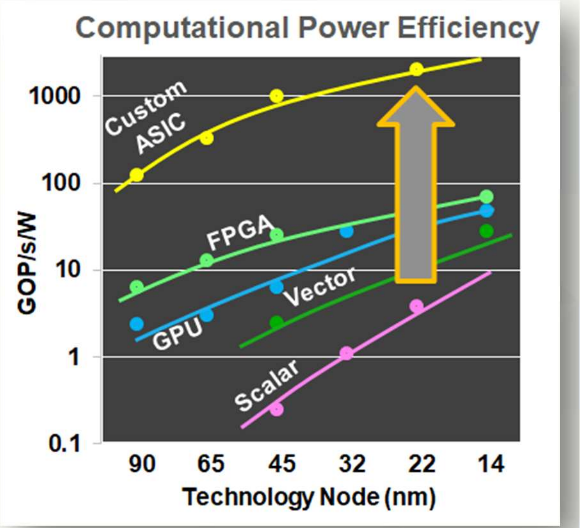


Goal: *A system that combines parallelization with dynamic scheduling*

Contributions

- **Generalizable methodology** for building communication protocols
  - Allows integrating runtime systems and task-based programming frameworks
- **Runtime integrated task-level programming framework**
  - **portable** on any given commercial off-the-shelf heterogenous SoC platform
- **Robust framework**
  - **hardware-agnostic application development** and deployment
  - **exploit parallelism** from task-to-application levels on heterogeneous systems
  - **balance** programmability, dynamic resource management, and performance
- **Resolve limitations of task-level programming framework**
  - replace **static** with **dynamic** scheduling and **single application** at a time-based execution with **multiple application** instances on heterogeneous systems

Motivation for a Heterogeneous Runtime



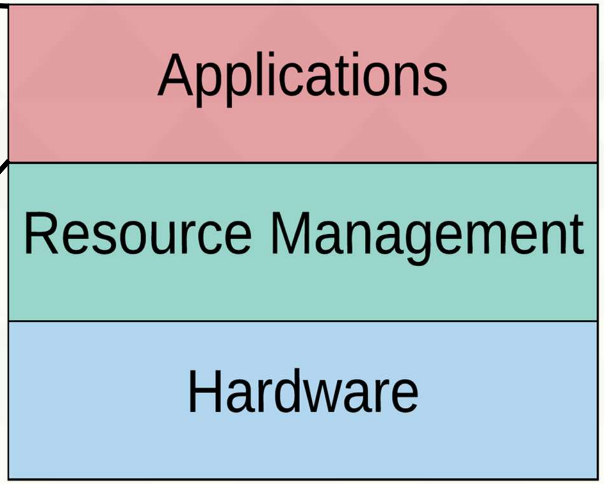
- It's easy to build fast processors that no one can program
- Programming environments for heterogenous systems should share key characteristics:
  - Enable use of system resources by users with limited hardware knowledge
  - Do so while allowing performant, energy-efficient execution
  - Ideally: allow for *portably performant* code
    - Run on multiple heterogeneous platforms without degradation

Users and Challenges

How do you represent or integrate new applications?

What is the most effective resource management strategy?

Type of User	Algorithmic Background? Willing to Modify Code?	Strict power or execution constraints?
Application Programmer	✓	X
Application User	X	X
Performance Programmer	✓	✓
Performance User	X	✓



How to physically invoke tasks on accelerators?

Which accelerators should be included?

How do you evaluate and compare architectures?

Need: *Productive and Hardware Agnostic Application Development and Deployment on Heterogenous SoCs*

Productive and Hardware Agnostic Application Development and Deployment on Heterogenous SoCs

- **Domain-Specific System on Chip** (DSSoC)
  *Domain-Focused Advanced Software-Reconfigurable Heterogeneous System on Chip (**DASH-SoC**) (2019-current)*

- **Space-Based Adaptive Communications Node** (Space-BACN)
  *Configurable Communications via Heterogeneous-processing Optimized Node (**COCHON**) (2022-current)*

- **Processor Reconfiguration for Wideband Sensor Systems** (PROWESS)
  *Dynamic Runtime Domain-Focused Software-Reconfigurable Heterogeneous (**DR-DASH**) Processor (2023-current)*

Outcomes:
- ✓ Coarse-scale heterogenous and programmable SoC
- ✓ > 5+ simultaneous applications
- ✓ >90% resource utilization
- ✓ 5ns scheduling latency
- ☐ 50 ns context switching

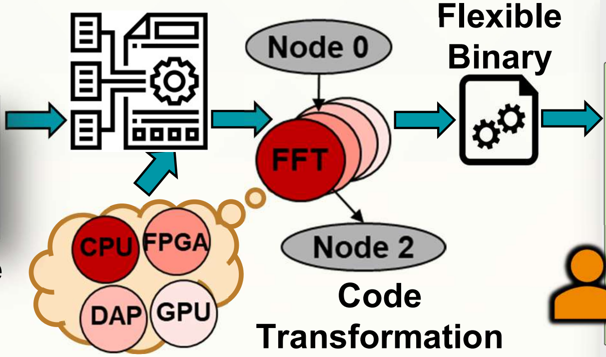
Portable
– Validated on COTS platforms

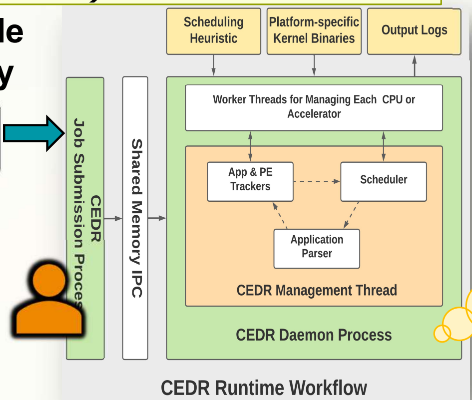
Flexible
– Seamlessly execute applications on the SoC

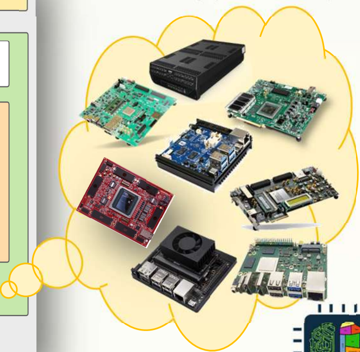
Scalable
– Dynamic workload scenarios

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

Platform Independent Code







Compiler Integrated Extensible DSSoC Runtime (CEDR)

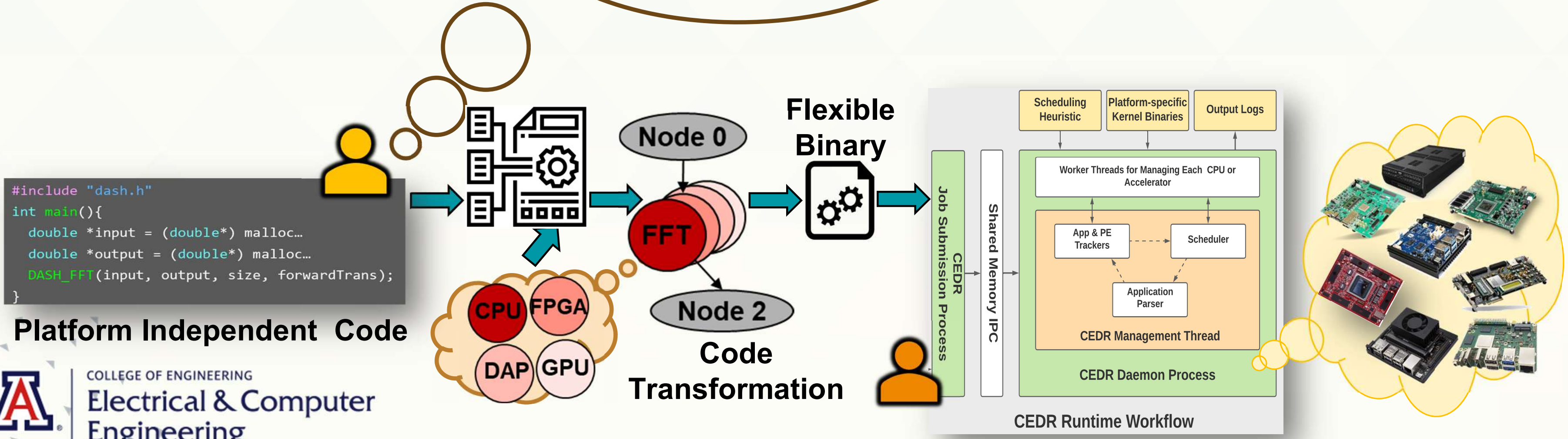
COLLEGE OF ENGINEERING
Electrical & Computer Engineering

UA-RCL

Research Group & Commercial Partners

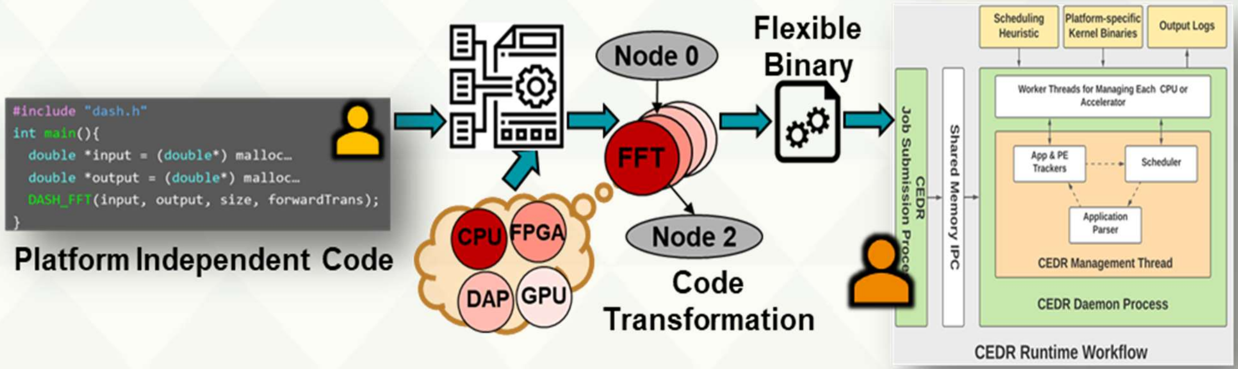
Collins Aerospace
GENERAL DYNAMICS Mission Systems
ARM
Carnegie Mellon University Software Engineering Institute
APL Johns Hopkins Applied Physics Laboratory
DASH Tech IC

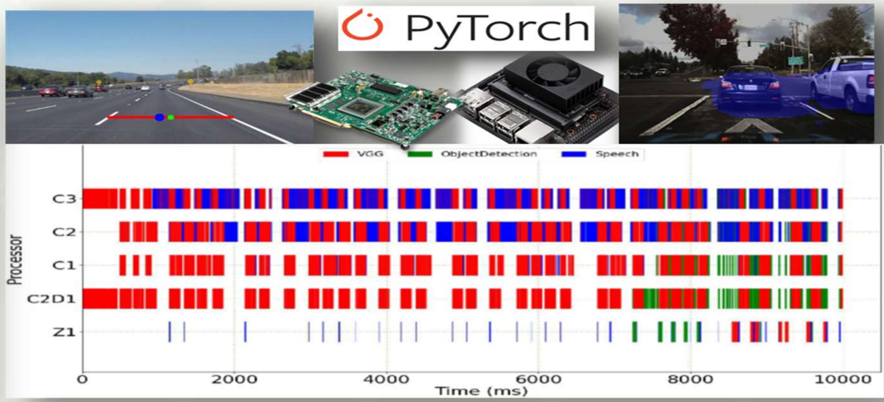
Arizona State University
University of Michigan
Wisconsin University of Wisconsin–Madison
Arizona

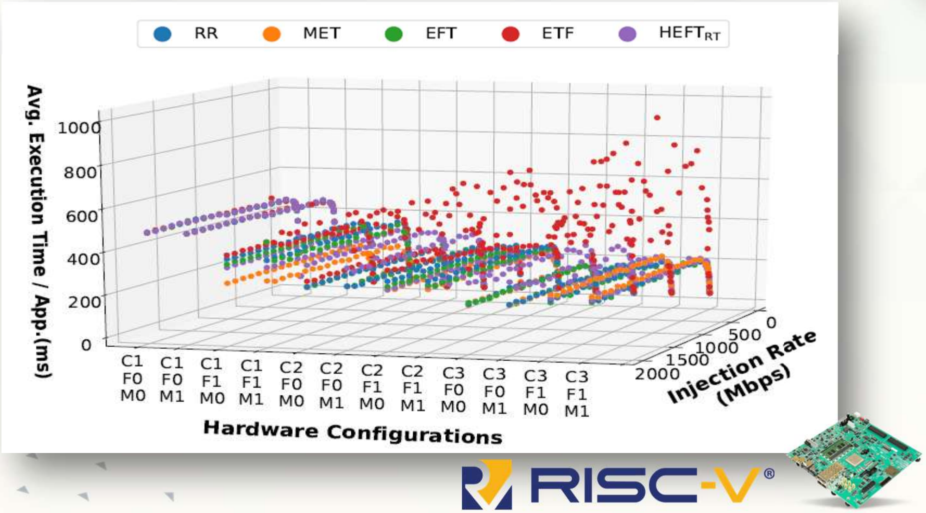


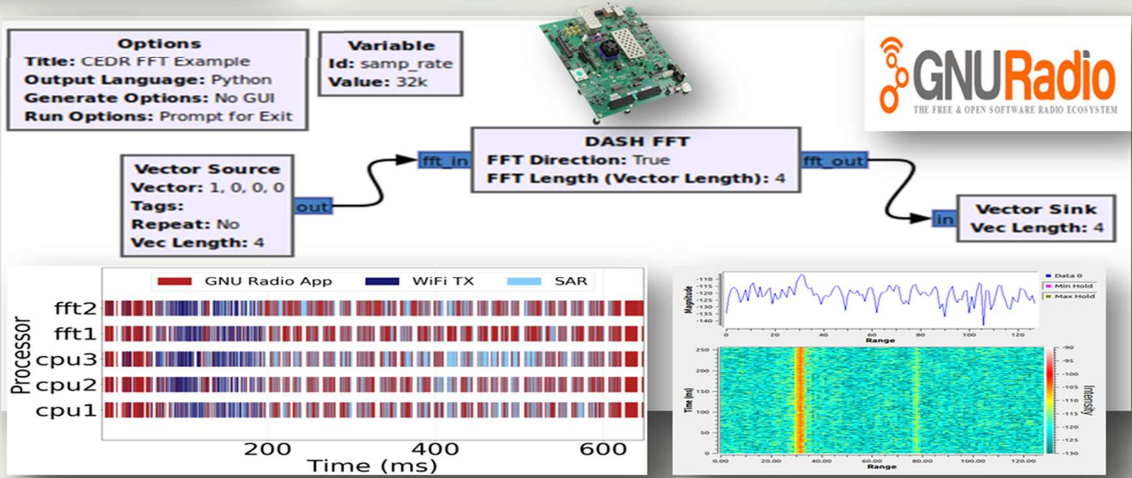
CEDR https://ua-rcl.github.io/CEDR/

CEDR* – A Compiler-Integrated, Extensible DSSoC Runtime





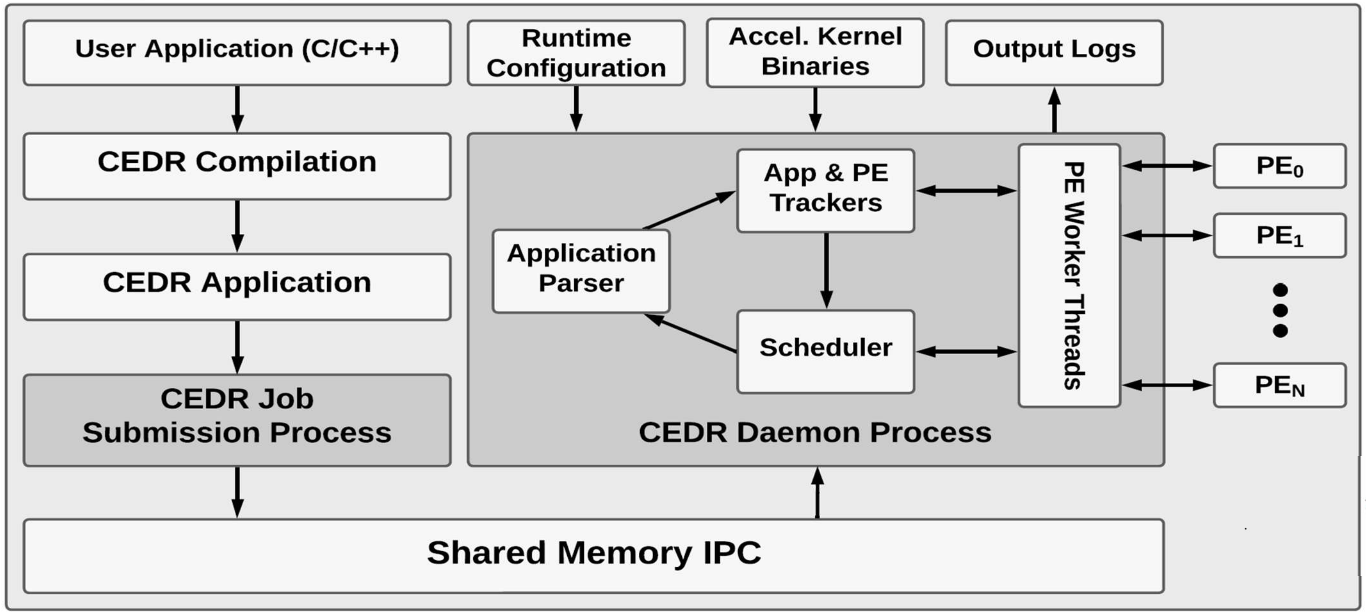


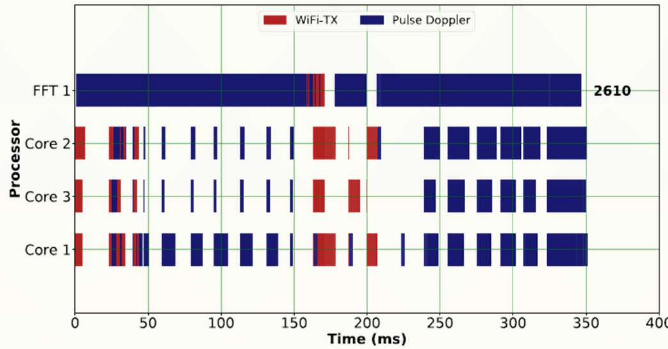


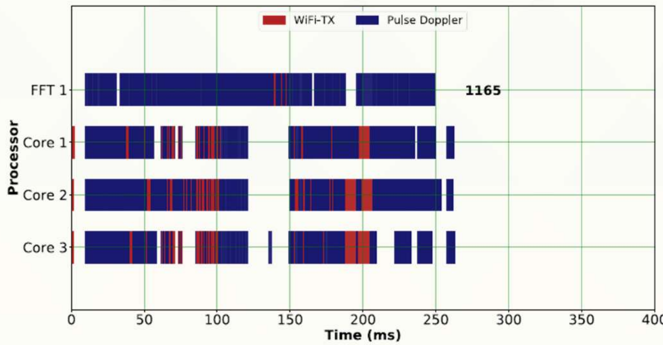
CEDR https://ua-rcl.github.io/CEDR/

*J. Mack, S. Hassan, N. Kumbhare, M. Castro Gonzalez, and A. Akoglu, “CEDR: A compiler-integrated, extensible DSSoC runtime,” *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 2, Jan. 2023, issn: 1539-9087. doi: 10.1145/3529257

CEDR[1,2] – A Compiler-Integrated, Extensible DSSoC Runtime







[1]J. Mack, **S. Hassan**, N. Kumbhare, M. Castro Gonzalez, and A. Akoglu, "CEDR: A compiler-integrated, extensible DSSoC runtime," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 2, Jan. 2023, issn: 1539-9087. doi: 10.1145/3529257

[2]J. Mack, **S. Gener, S. Hassan**, H. U. Suluhan and A. Akoglu, "CEDR-API: Productive, Performant Programming of Domain-Specific Embedded Systems," *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2023, pp. 16-25, doi: 10.1109/IPDPSW59300.2023.00016.

CEDR-API: Application Development

Baseline C/C++ Application:

```
...
int start=0,end=512,size=128;
bool forward=true;
complex input=allocate(512);
complex output=allocate(512);
// FFT for loop


for (int i=start; i<end; i++){
  FFT(input[i],
    output[i],
    size,
    forward);
}
// Multiplication for loop


for (int i=start; i<end; i++){
  for (int j=0; j<size; j++){
    output[i][j] =
        output[i][j] * 2;
  }
}
...
deallocate(input);
deallocate(output);
```

Easy to use header-only APIs

Hardware-agnostic APIs

CEDR C/C++ Application:

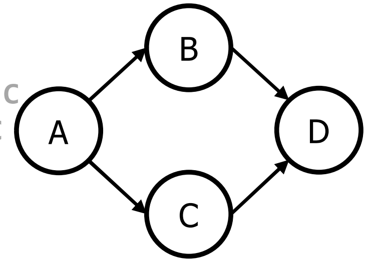
```
#include <libcedr.h>

...
int start=0,end=512,size=128;
bool forward=true;
complex input=allocate(512);
complex output=allocate(512);
// FFT for loop


for (int i=start; i<end; i++){
  CEDR_FFT(input[i],
    output[i],
    size,
    forward);
}
// Multiplication for loop


for (int i=start; i<end; i++){
  for (int j=0; j<size; j++){
    output[i][j] =
        output[i][j] * 2;
  }
}
...
deallocate(input);
deallocate(output);
```

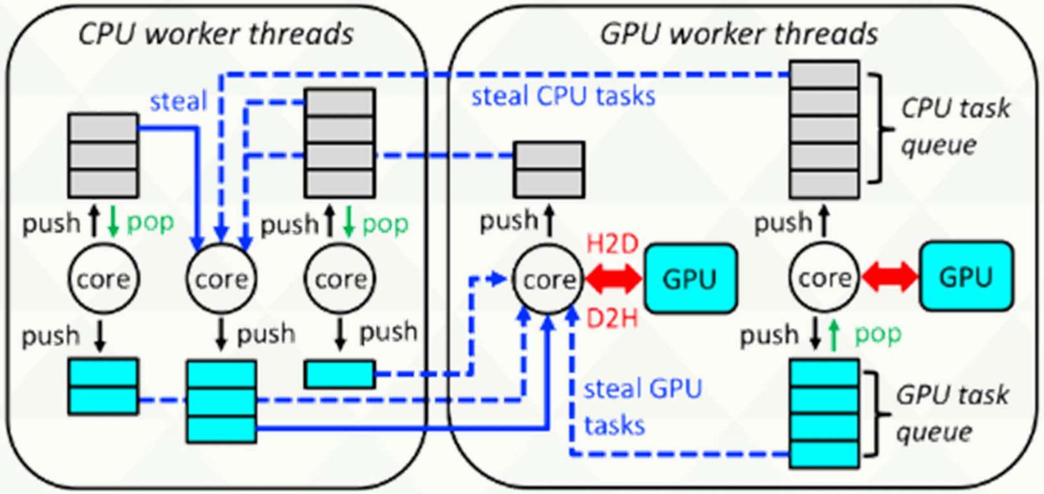
Taskflow: Fast Task-based Parallel Programming using Modern C++*

"Hello World" in Taskflow

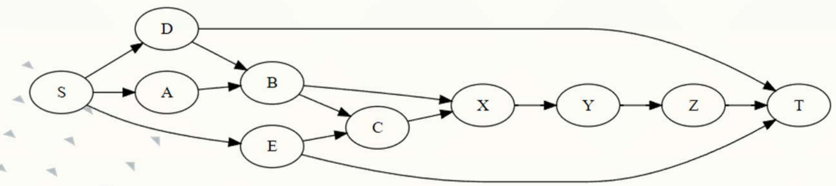
```cpp
#include <taskflow/tasflow.hpp> // Taskflow is header-only
int main(){
  tf::Taskflow taskflow;
  tf::Executor executor;
  auto [A, B, C, D] = taskflow.emplace(
    [](){std::cout<<"TaskA\n";},
    [](){std::cout<<"TaskB\n";},
    [](){std::cout<<"TaskC\n";},
    [](){std::cout<<"TaskD\n";}
  );
  A.precede(B, C); // A runs before B and C
  D.succeed(B, C); // D runs after B and C
  executor.run(taskflow).wait();
  return 0;
}
```

Only **15 lines** of code to get a parallel task execution!

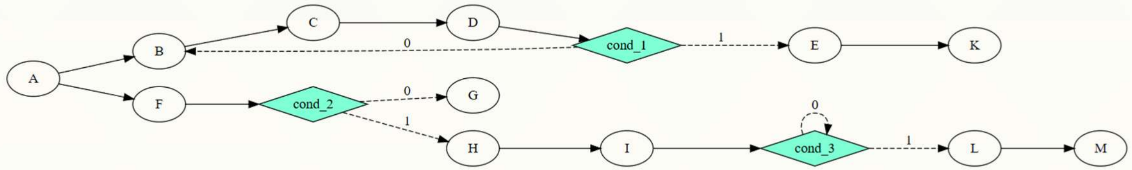




- Static Tasking:



- Conditional Tasking:



*Huang et al. , "Cpp-Taskflow: Fast Task-Based Parallel Programming Using Modern C++," IPDPS'19 DOI:10.1109/IPDPS.2019.00105

Taskflow: Application Development

Baseline C/C++ Application:

```
...
int start=0,end=512,size=128;
bool forward=true;
complex input=allocate(512);
complex output=allocate(512);
// FFT for loop


for (int i=start; i<end; i++){
  FFT(input[i],
    output[i],
    size,
    forward);
}
// Multiplication for loop


for (int i=start; i<end; i++){
  for (int j=0; j<size; j++){
    output[i][j] =
        output[i][j] * 2;
  }
}
...
deallocate(input);
deallocate(output);
```

Easy to use header-only APIs

Task-based Parallel programming using APIs

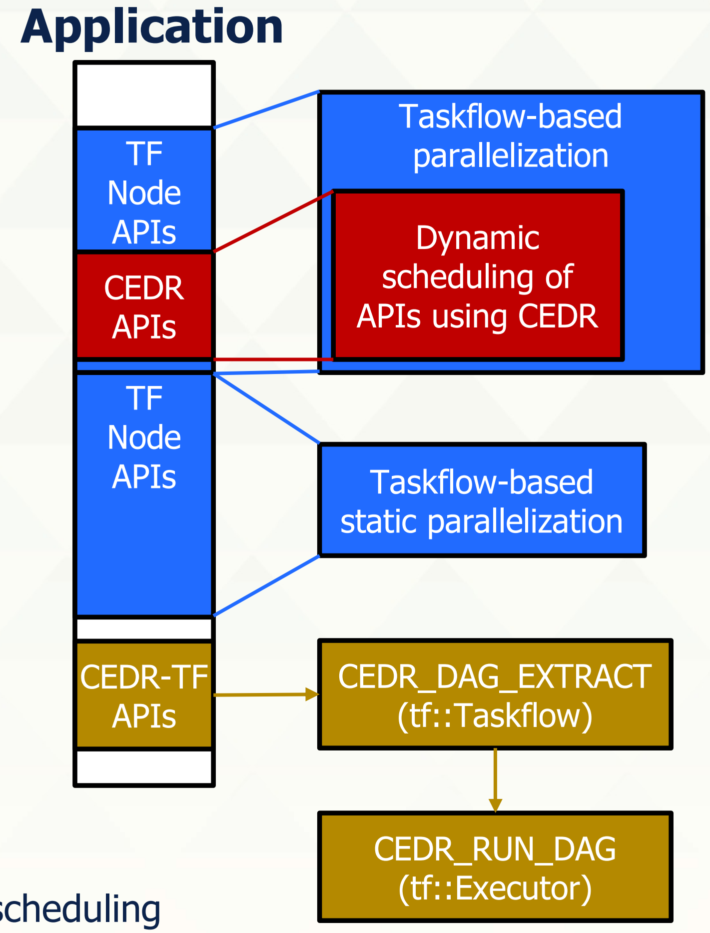
Taskflow C++ Application:

```
#include <taskflow.hpp>
...
int start=0,end=512,size=128;
bool forward=true;
complex input=allocate(512);
complex output=allocate(512);
// FFT for loop
task0=taskflow.for_each_index(
  ref(start), ref(end), 1,
  [input, &output,
  size, forward])(int i){
  FFT(input[i],
    output[i],
    size,
    forward);
}
// Multiplication for loop
task1=taskflow.for_each_index(
  ref(start), ref(end), 1,
  [&output,
  size])(int i){
  for (int j=0; j<size; j++){
    output[i][j] =
        output[i][j] * 2;
  }
}
...
deallocate(input);
deallocate(output);
```


Huang et al. , "Cpp-Taskflow: Fast Task-Based Parallel Programming Using Modern C++," IPDPS'19 DOI:10.1109/IPDPS.2019.00105

CEDR-Taskflow

- Taskflow:
  - Parallelization of the applications
  - Generation of the DAG for the applications
  - Static task execution
- CEDR:
  - Dynamic scheduling of APIs during runtime
  - Parallel API regions (serial non-API regions)
- CEDR-Taskflow:
  - Parallelization of both API and non-API regions
  - Dynamic scheduling of APIs during runtime
  - New APIs:
    - CEDR_DAG_EXTRACT: The whole application view to be used by CEDR for scheduling
    - CEDR_RUN_DAG: Control over DAG execution handed to CEDR



Application using CEDR and Taskflow

Baseline C/C++ Application:

```
...
int start=0,end=512,size=128;
bool forward=true;
complex input=allocate(512);
complex output=allocate(512);
// FFT for loop


for (int i=start; i<end; i++){
  FFT(input[i],
    output[i],
    size,
    forward);
}
// Multiplication for loop


for (int i=start; i<end; i++){
  for (int j=0; j<size; j++){
    output[i][j] =
        output[i][j] * 2;
  }
}
...
deallocate(input);
deallocate(output);
```

CEDR+Taskflow C++ Application:

```
#include <libcedr.h>
#include <taskflow.hpp>
...
int start=0,end=512,size=128;
bool forward=true;
complex input=allocate(512);
complex output=allocate(512);
// FFT for loop
task0=taskflow.for_each_index(
  ref(start), ref(end), 1,
  [input, &output,
  size, forward])(int i){
  CEDR_FFT(input[i],
    output[i],
    size,
    forward);
}
// Multiplication for loop
task1=taskflow.for_each_index(
  ref(start), ref(end), 1,
  [&output,
  size])(int i){
  for (int j=0; j<size; j++){
    output[i][j] =
        output[i][j] * 2;
  }
}
...
deallocate(input);
deallocate(output);
```

Easy to use header-only APIs

Hardware-agnostic APIs

Task-based Parallel programming using APIs

Experimental Setup

Platforms

- ZCU102
  - 3 CPUs
  - 2 FFTs
  - 2 GEMMs
  - 1 ZIP
- Jetson AGX
  - 7 CPUs
  - 1 GPU

Workload Composition

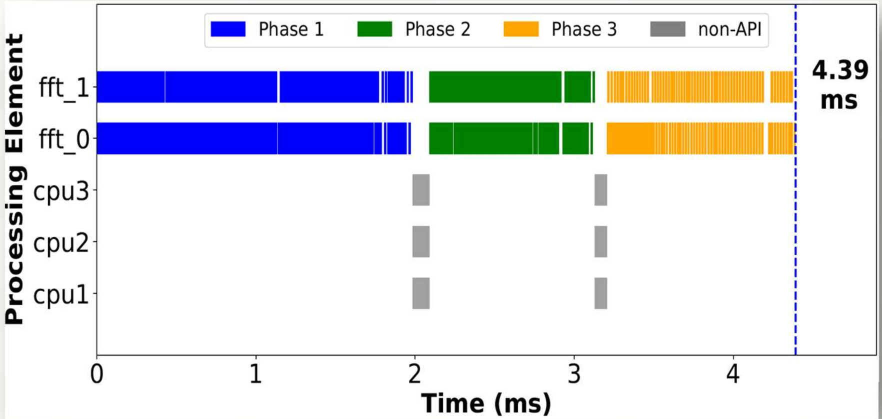
- Radar Correlator (RC)
- Temporal Mitigation (TM)
- WiFi-TX
- Pulse Doppler (PD)
- Synthetic Aperture Radar (SAR)

Scheduling Heuristics

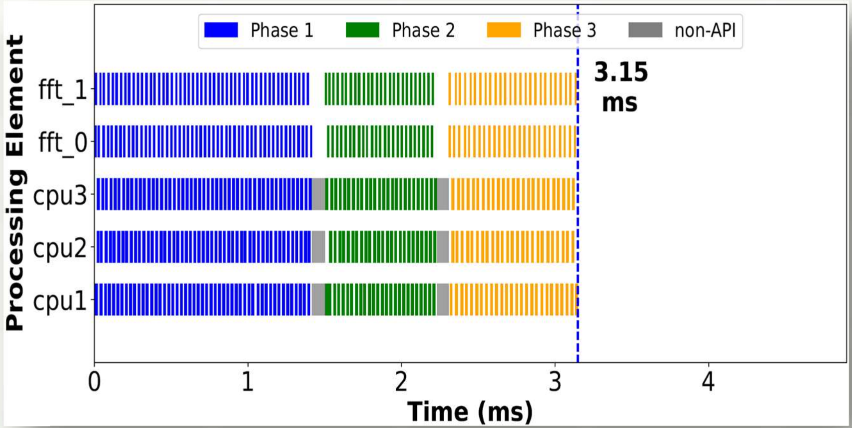
- Round Robin (RR)
- Earliest Finish Time (EFT)

Results: Application Performance

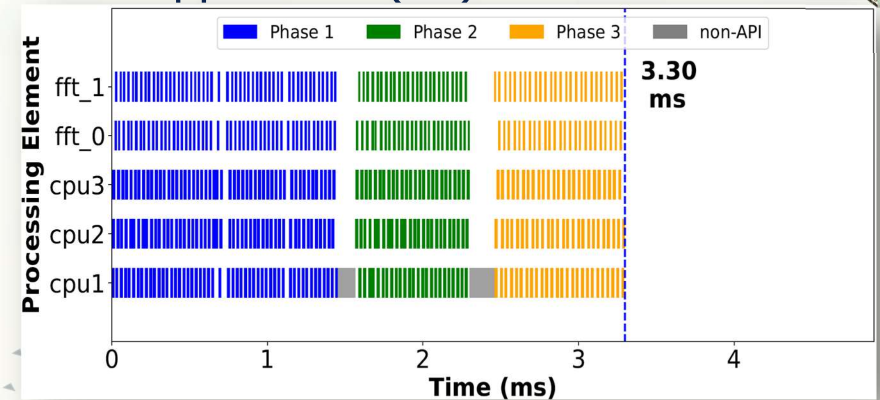
Taskflow Application (PD)

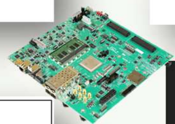


CEDR+Taskflow Application (PD)



CEDR Application (PD)

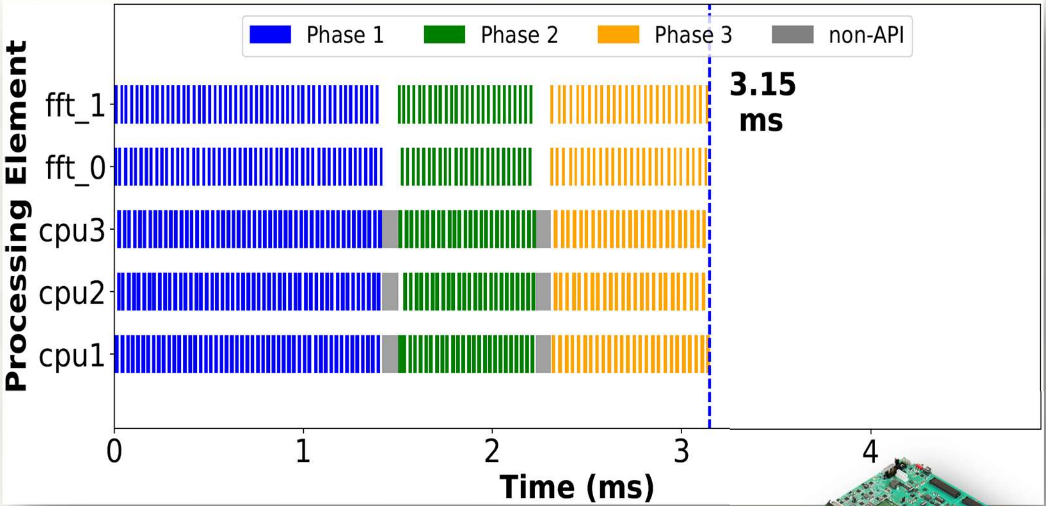


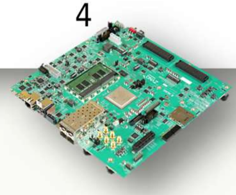


App Name	Taskflow only (ns)	CEDR only (ns)	CEDR and Taskflow (ns)
RC	120,972	120,612	120,162
TM	2,597,770	2,575,658	1,762,166
WiFi-TX	714,621	712,721	651,685
PD	5,144,564	3,868,427	3,790,988
SAR	37,351,722	38,111,968	28,980,316

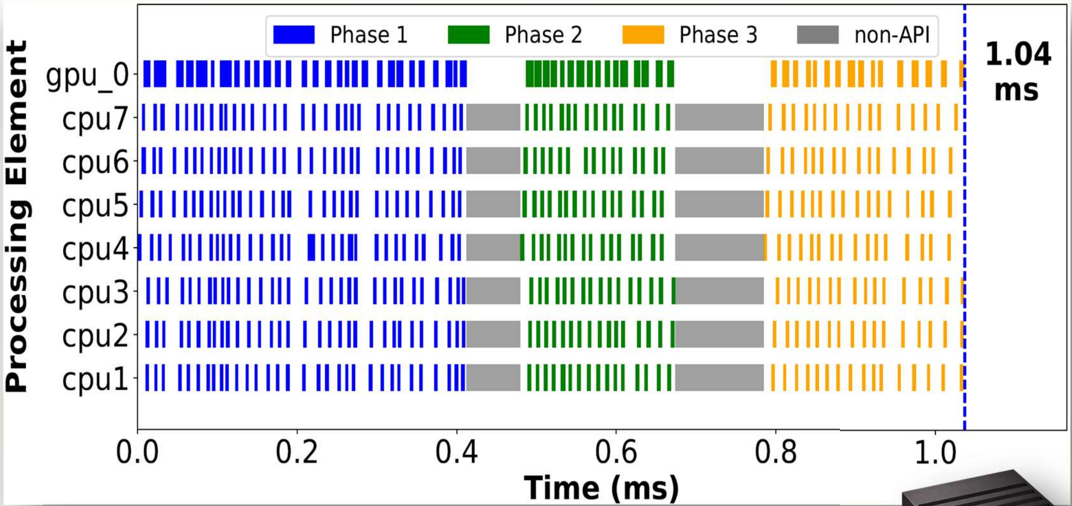
Results: Portability Using PD

CEDR+Taskflow Application on ZCU102





CEDR+Taskflow Application on Jetson AGX

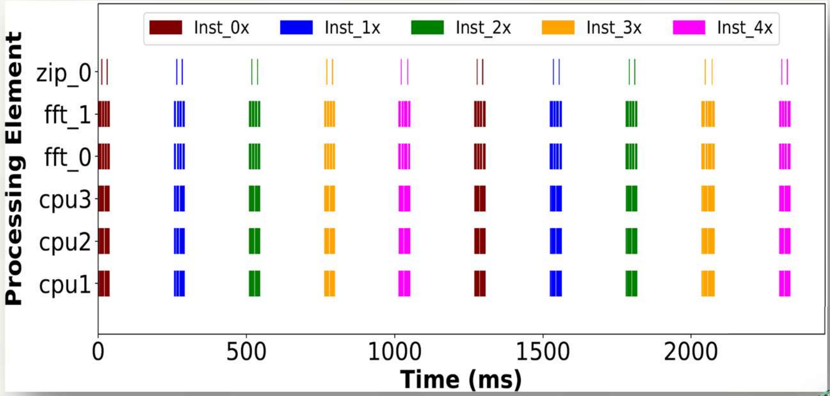


The same application runs on different platforms without changing anything on the application code

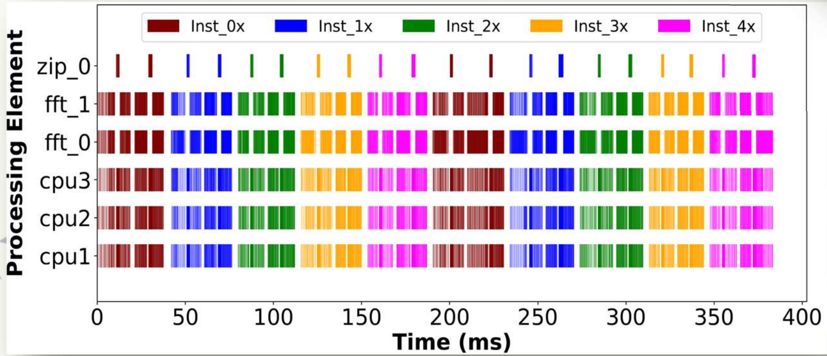
Results: Features Enabled by CEDR-Taskflow Integration

Streaming Input Processing:

10 SAR applications as instances



10 SAR applications as streams



Cached Scheduling:

App Name	API Count	Stream (us)	Cached (us)	Improvement
RC	3	2,376	283	8.37x
TM	5	3,759	643	5.84x
WiFi-TX	10	7,662	723	10.59x
PD	512	291,790	10,769	27.09x
SAR	2,305	1,405,034	47,475	29.60x

More improvement in time spent on scheduling as the number of APIs increases

Less time spent on initializations and allocations

Conclusions and Future Work

- **CEDR-Taskflow integration**
  - **portable** and **scalable** framework for heterogeneous systems
  - **hardware-agnostic application development** while exploiting parallelism and improving **resource utilization**
  - Combines **Taskflow's task dependency representation** with **CEDR's dynamic scheduling**, reducing execution time **without increasing developer complexity**
  - Demonstrated applicability across various platforms and applications, leading to **better resource management** and **lower execution latency**
- **Future Work:**
  - Automatic pipelined execution for improved application execution
  - Merging DAGs of multiple applications for a global system-wide optimization

Continuous Community Outreach & Live Demos

ARM'19 → GNU Radio'20 → FOSDEM '20 → FOSDEM '21 → GNU Radio'22 → ESWEEK '23 → ISFPGA' 24 → ISPFGA' 25

CEDR: A Holistic Software and Hardware Design Environment for Hardware Agnostic Application Development and Deployment on FPGA-Integrated Heterogeneous Systems
- Tutorial @ ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, March 2025
- https://ua-rcl.github.io/presentations/fpga25/

CEDR: A Holistic Software and Hardware Design Environment for FPGA-Integrated Heterogeneous Systems
- Tutorial @ ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, March 2024
- https://ua-rcl.github.io/presentations/fpga24/

CEDR: A Novel Runtime Environment for Accelerator-Rich Heterogeneous Architectures
- Lecture @ ESWEEK Education, September 2023
- https://www.youtube.com/watch?v=nMWDFAChcFI&list=PLMohsHZ1Urxvq9ZXyDenPMtbodupJaoZw&index=10

GNURadio and CEDR: Runtime Scheduling to Heterogeneous Accelerators, GNU Radio Conference, September 2022
- https://pubs.gnuradio.org/index.php/grcon/article/view/124
- https://archive.org/details/youtube-MR6h6e60-V4

Runtime Strategies and Task Scheduling of Software-Defined Radio on Heterogeneous Hardware, Is an accelerator always the best option?
- Free and Open source Software Developers' European Meeting (FOSDEM), February 2021
- https://archive.fosdem.org/2021/schedule/event/fsr_runtime_strategies_and_scheduling_of_sdr_on_heterogeneous_hw/

Automating Programming and Development of Heterogeneous SoCs with LLVM Tools
- Free and Open source Software Developers' European Meeting (FOSDEM), February 2020
- https://archive.fosdem.org/2020/schedule/event/llvm_aut_prog_het_soc/

Thank you!

Contact:
{gener,sahilhassan,akoglu}@arizona.edu

UA-RCL

COLLEGE OF ENGINEERING
Electrical & Computer Engineering

Reconfigurable Computing Lab