

Adrastea: An Efficient FPGA Design Environment for Heterogeneous Scientific Computing and Machine Learning

Architecture and Performance Group

CCSD

Aaron Young, **Narasinga Rao Miniskar**, Frank Liu, Willem Blokland, Jeffrey Vetter

Aug 25, 2022

SMC 2022 Conference

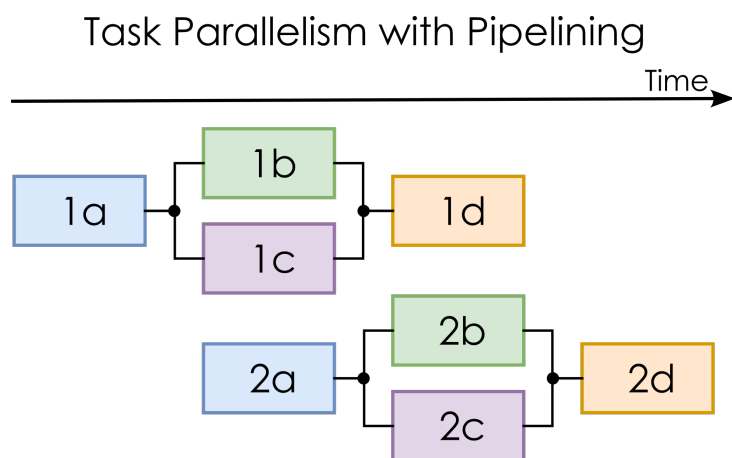


Overview

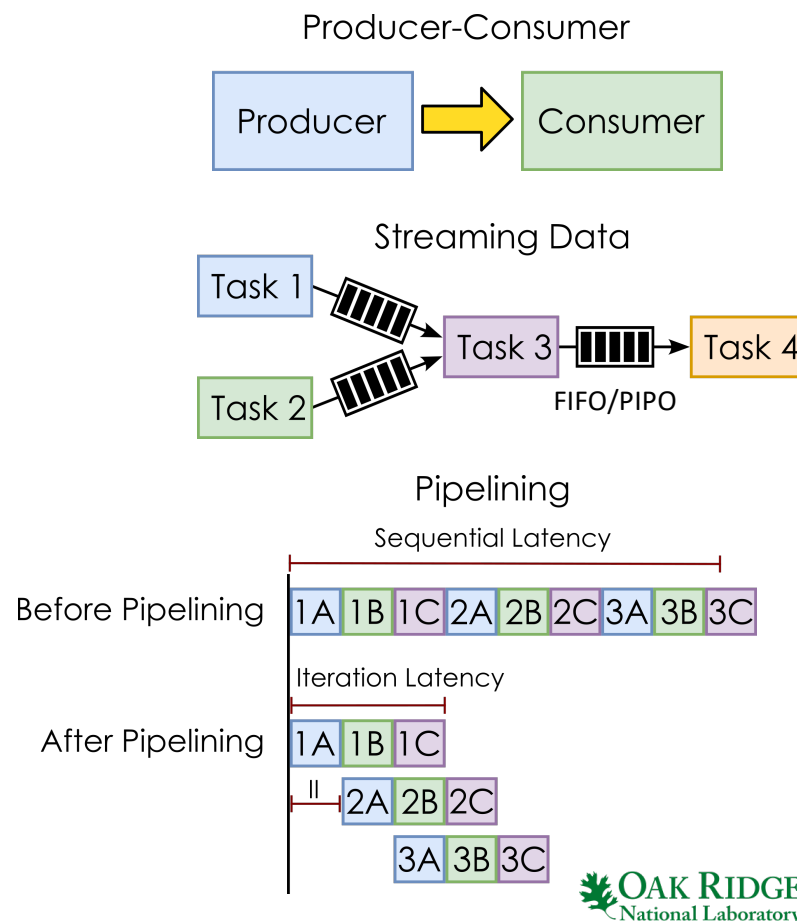
- FPGA Design and Strengths
- FPGA Development Challenges
- Adrastea Design Environment
 - Build Environment
 - Kernel Interfacing
 - Overall flow (Including Design Space Exploration)
- Use-cases
 - SNS Random Forest Application
 - Fast Fourier Transform

FPGA Design and Strengths

- FPGAs excel at applications that run close to the edge, for workloads that require low-latency solutions, and when the application can be expressed as a data-flow in a processing pipeline.



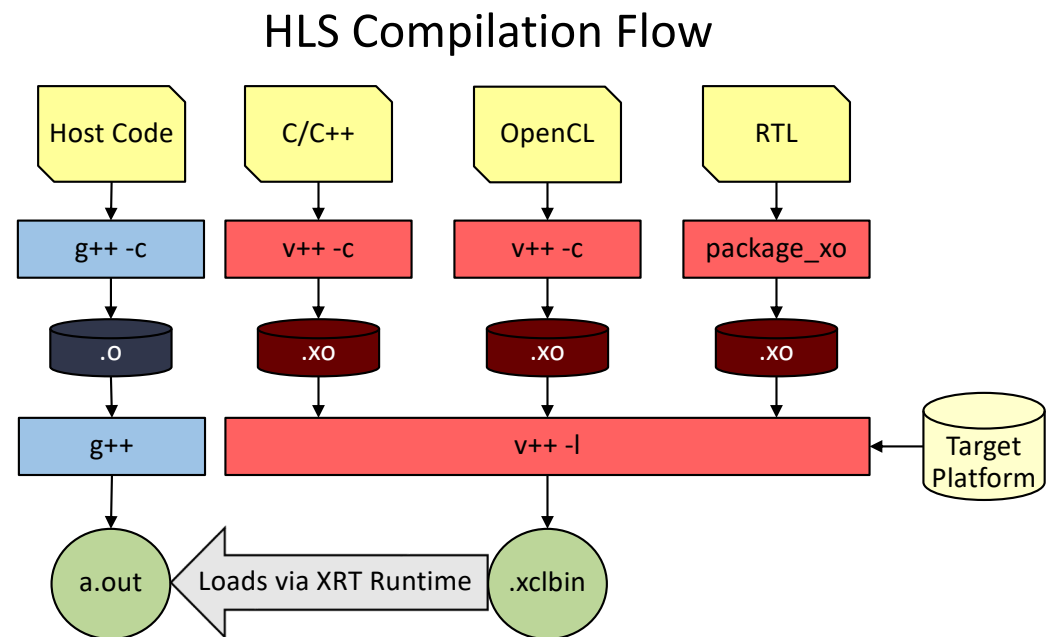
Three paradigms for FPGA kernel design:



Xilinx Vitis HLS Development Environment

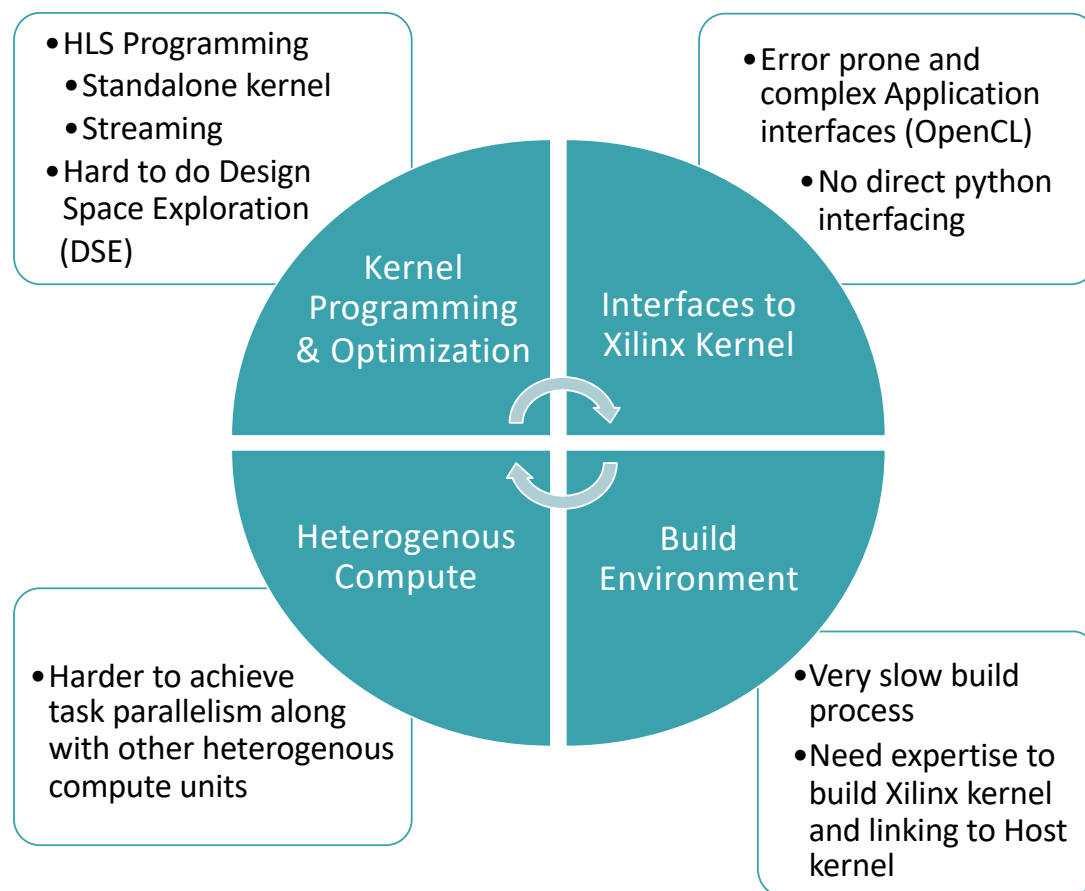
High-Level Synthesis (HLS)
programs consist of:

- Kernel code
 - Written in C/C++, OpenCL, or RTL and built using the Vitis compiler (v++)
- Host Code
 - Leverages OpenCL or XRT API to execute on the acceleration kernel.

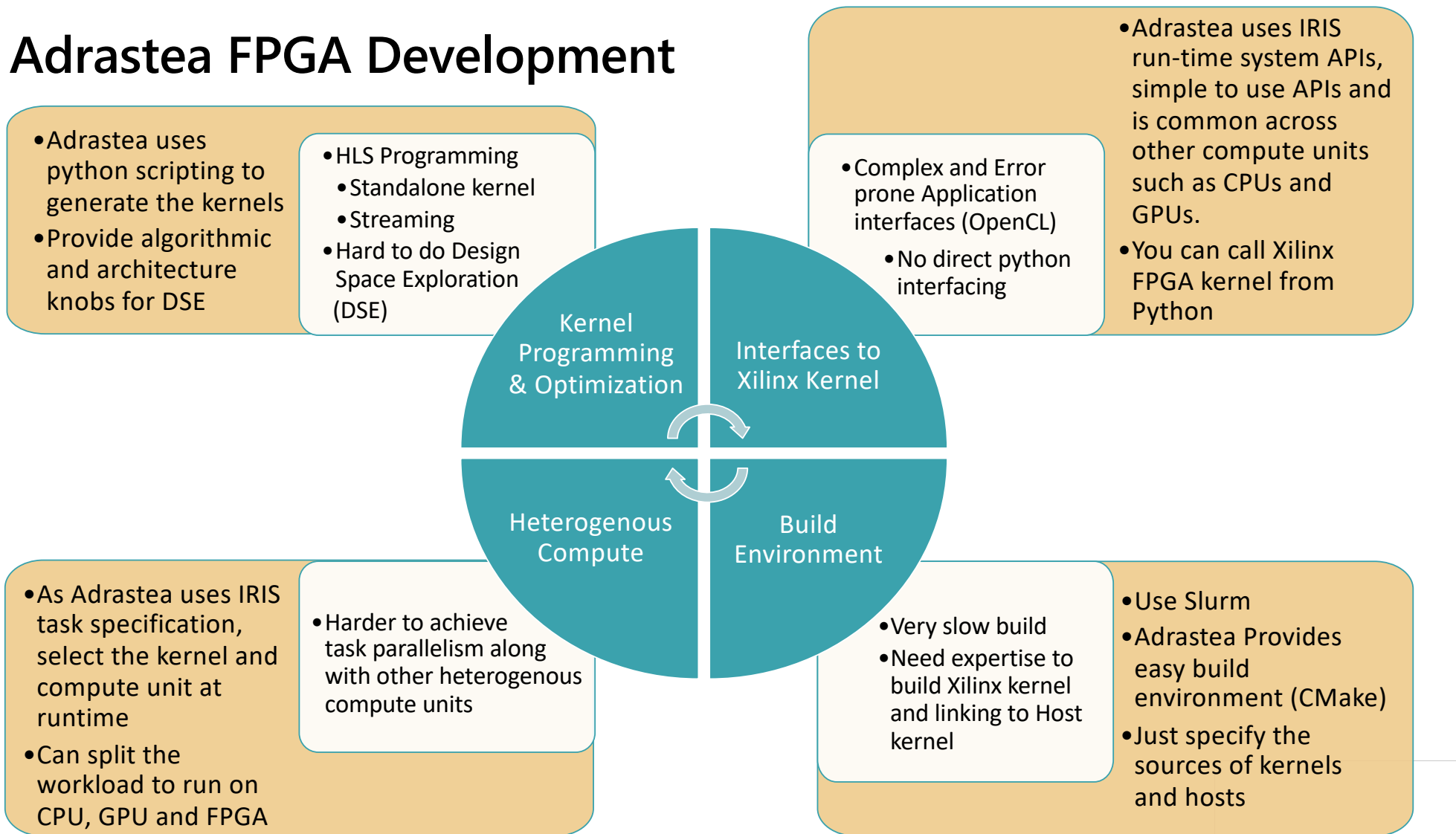


FPGAs for Edge and FPGA Development Challenges

- Push scientific AI to the edge
- FPGA is an ideal platform for low-latency computational workloads, widely deployed at the "edge"
- Development of FPGA kernels and host interfaces tend to be slow and tedious
- Proposal: Automatic turn-key FPGA development environment (incl. interface)
- Objective:
 - Efficient FPGA implementation
 - Rapid FPGA development

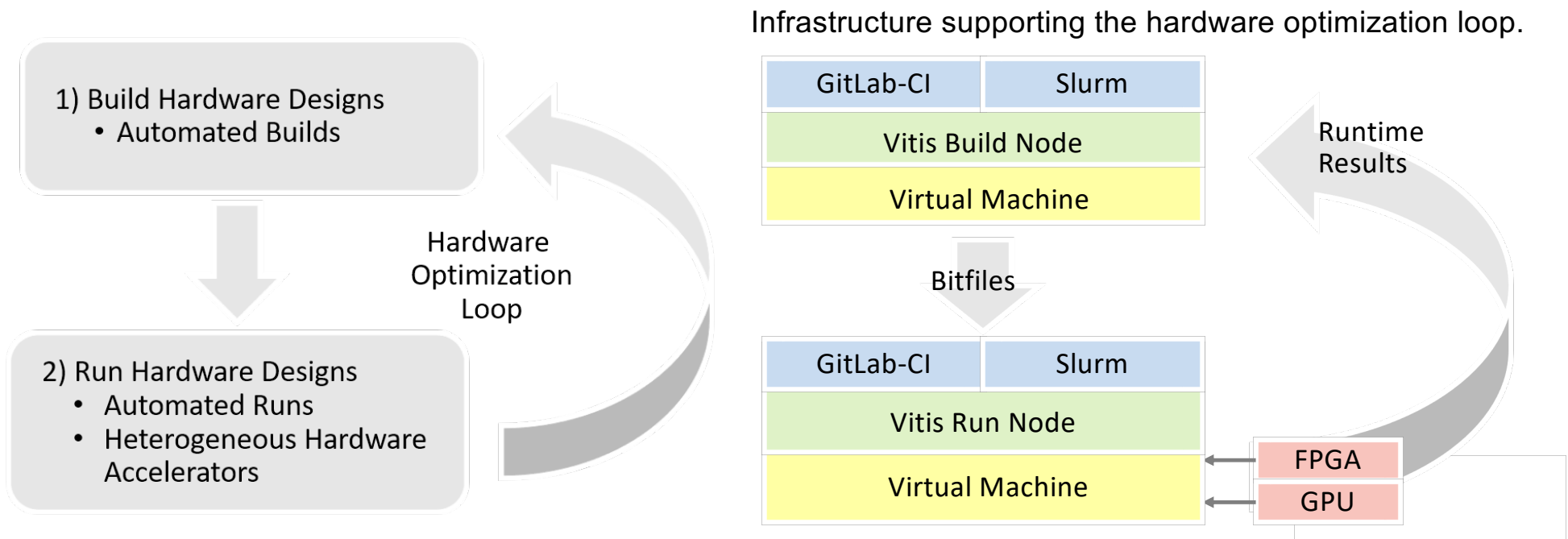


Adrastea FPGA Development

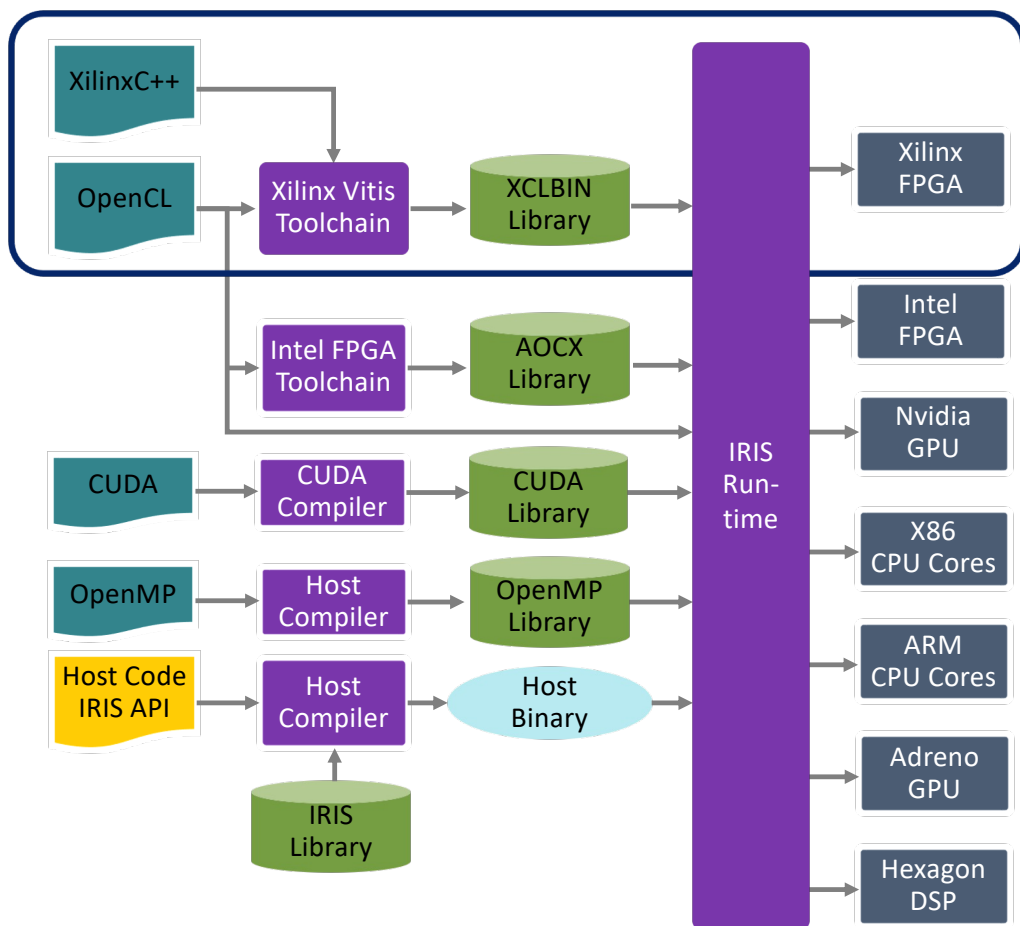


Adrastea Build/Compile Environment

- Long running FPGA builds (~hrs) can be accelerated by building multiple design in parallel.
- FPGA Runs are faster (~mins) so a fewer number of run nodes can be shared by multiple build nodes.
- VMs simplify toolchain installation and scaling up of cluster.



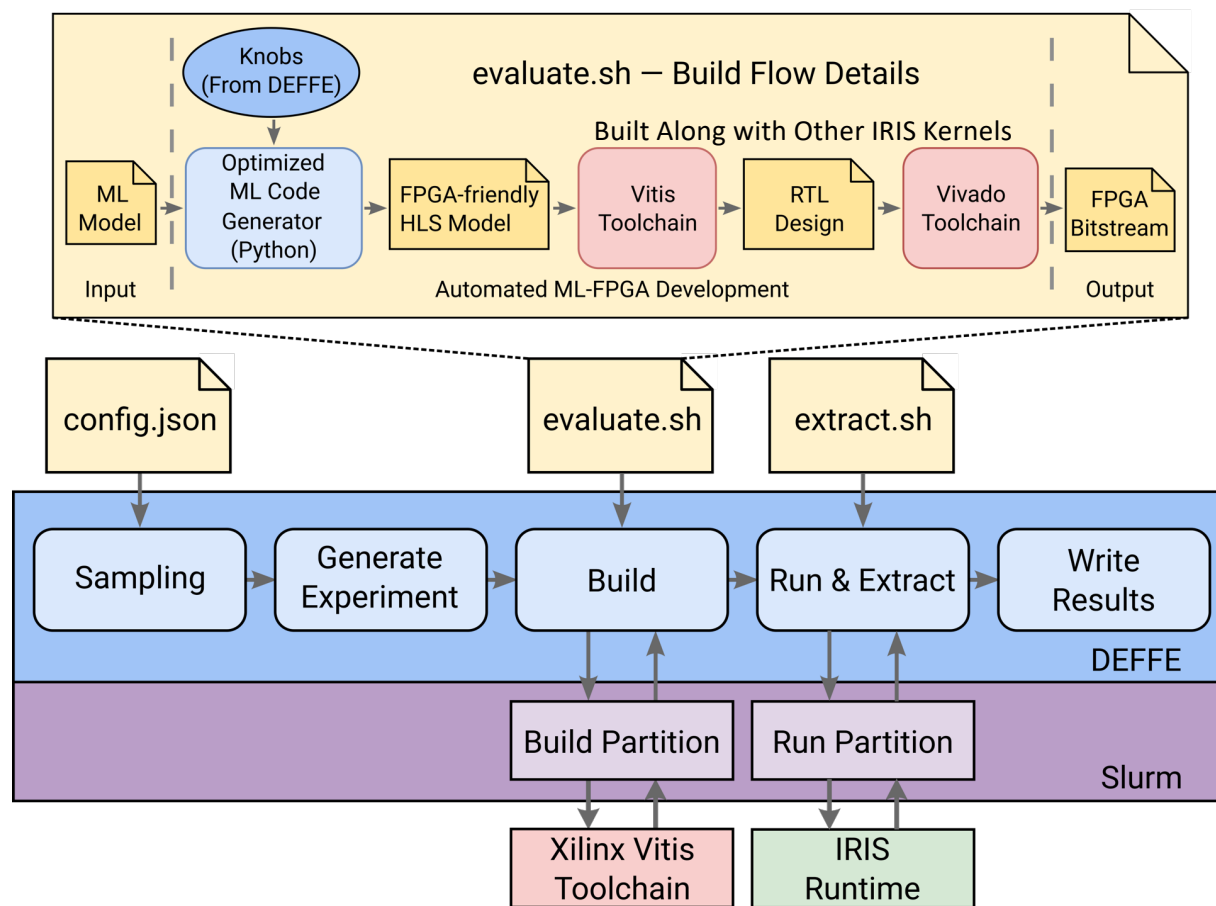
FPGA Development with Python Interface through IRIS



```
import iris
def run_random_forest(args, test_data):
    features= test_data.data
    Y = test_data.target
    Y_predict = np.zeros(Y.size, dtype=np.int8)
    SIZE = Y.size
    iris.init() # Initialize IRIS Run-time
    mem_features = iris.mem(features.nbytes)
    mem_Y_predict = iris.mem(Y_predict.nbytes)
    task = iris.task() # Create IRIS task
    task.h2d(mem_features, 0, features.nbytes, features)
    task.kernel("rf_classifier", 1, [0], [1], [1],
               [mem_features, SIZE, mem_Y_predict], #Parameters
               [iris.iris_r, 4, iris.iris_w] ) # Parameters information
    task.d2h(mem_Y_predict, 0, Y_predict.nbytes, Y_predict)
    cu = iris.iris_cpu
    if args.cu == 'fpga':
        cu = iris.iris_fpga
    task.submit(cu)
    iris.finalize()
```


Overall Adrastea Build Flow

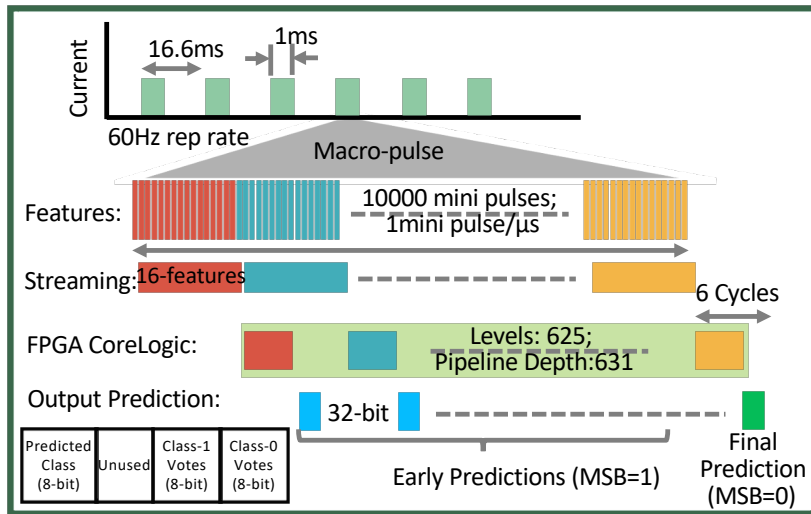
- Complete build flow leverages:
 - Build VMs
 - Slurm
 - DEFFE (Design space Exploration)
 - Vitis and Vivado Toolchains
 - Custom HSL generator script
 - IRIS



Example: SNS Random Forest Application

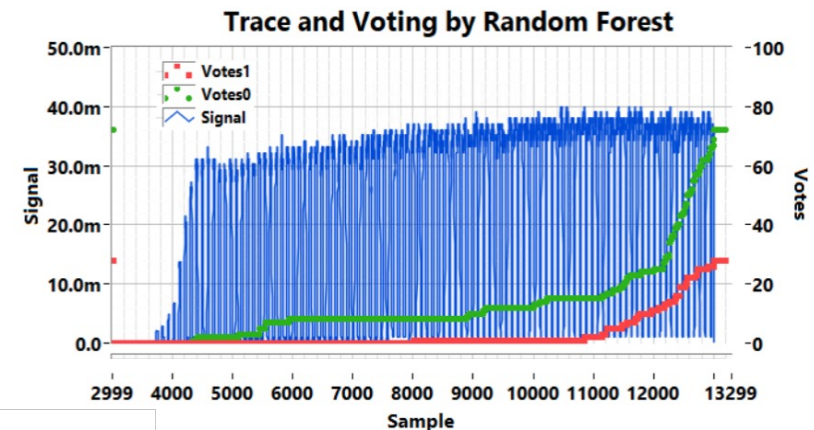
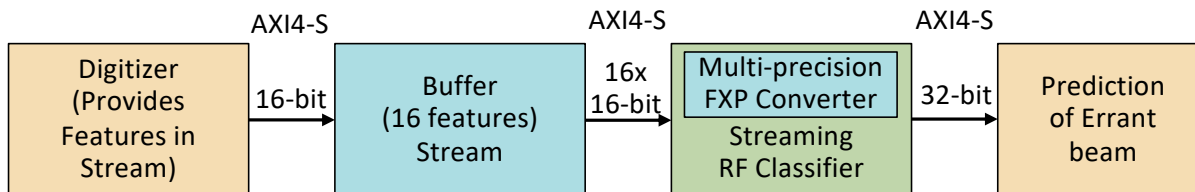


Adrastea for Ultra-low latency Random Forest ML deployment on Edge



- SNS Background:
 - Ill-formed SNS beam pulses must be aborted quickly to avoid equipment damage
 - Intra-pulse abort signal only has response time of a 1 μ s
- ML methods have shown to be effective, but have large computational latency
- Proposal: Ultra-Low latency Streaming based Random Forest Classifier on FPGA
- Measured latency of **60 ns** (100MHz) and early prediction

At edge, integration of FPGA Classifier with Beam Line



FPL 2022, Ultra Low Latency Machine Learning for Scientific Edge Applications, Narasinga Rao Miniskar, Aaron Young, Frank Liu, Willem Blokland, Anthony Cabrera and Jeffery S. Vetter

SNS Random Forest Parameters for Design Space Exploration

Parameters used for experiments

Parameter	Type	# of Values	Values
Frequency (MHz)	Build	5	50, 100, 200, 300*, 500
Decision tree	Generator	2	Conditional (conditional_trees), Flattened*
Bitwise optimization	Generator	2	Yes*, No (no_bitwise)
Feature datatype	Generator	12	Float (FP32), Fixed 8-bit(FX8), FT100*, FT90, FT80, FT70, FT60, FT50, FT40, FT30, FT20, FT10
Fixed point type	Generator	2	Quantization*, Power2 (no_quantization)
Voting	Generator	3	Array (array_votes), SSA*, No-SSA (no_votes_ssa)

1440 design points *: Base Parameters (The best)
With Avg 4 hrs/design point, sequential build takes 240 days

Design Space Exploration of SNS

Parameter

Frequency (MHz)

Decision tree

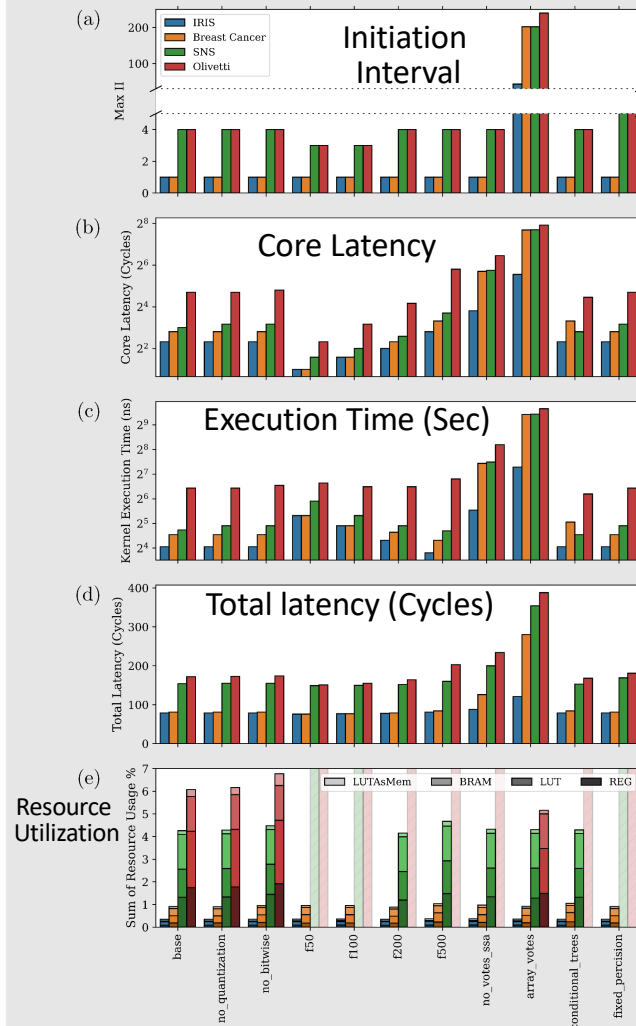
Bitwise optimization

Feature datatype

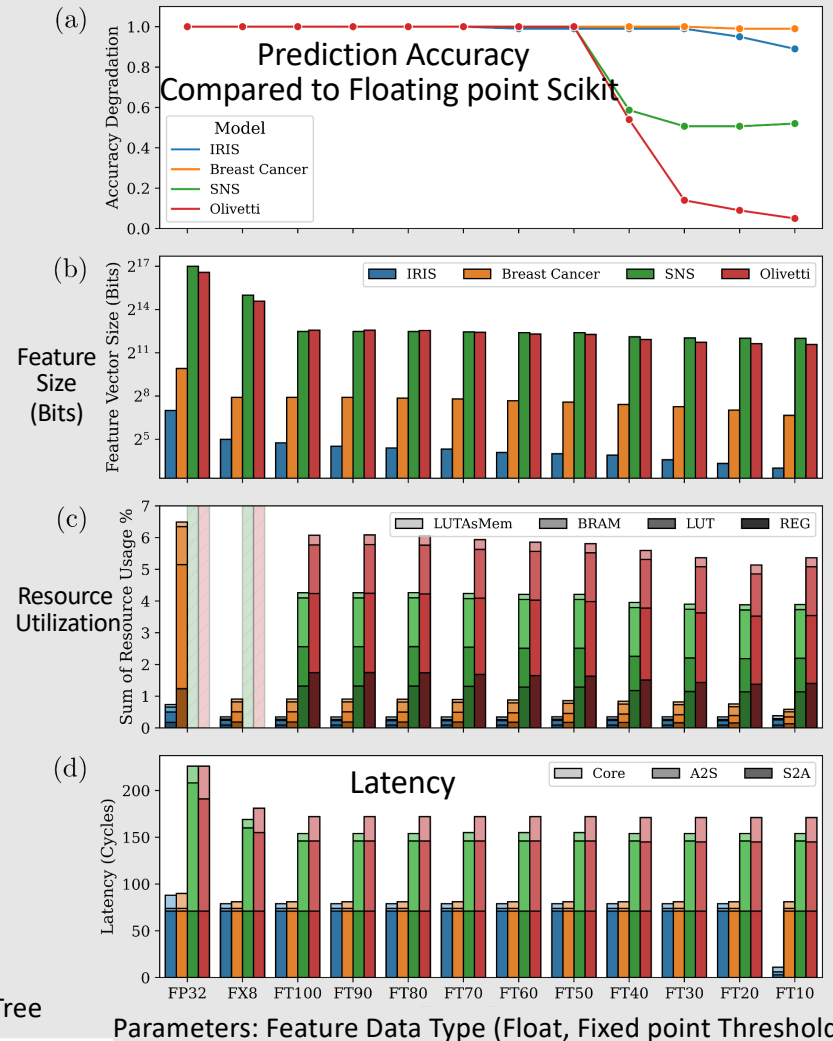
Fixed point type

Voting

Comparison with different build options



Optimization metrics of interest for different input feature datatypes

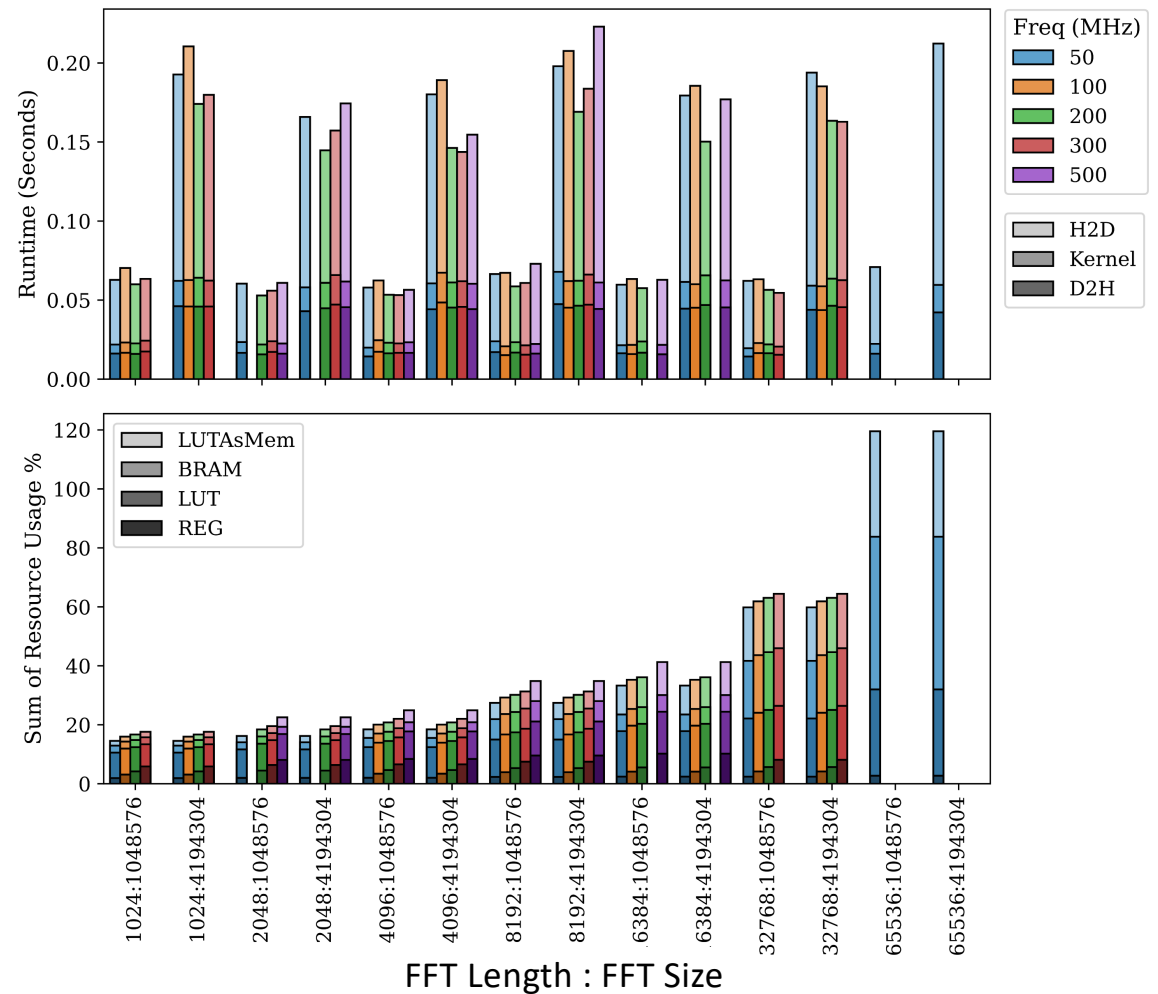


Example: Fast Fourier Transform Application



Example: Design Space Exploration of FFT

- AFM Analyzer uses FFT to remove noise in the scan line
- FFT Design Parameters
 - FFT Length: 1024 - 65536
 - FFT Size: 1M to 4M
 - Frequency of FPGA: 50 to 500 MHz
 - Datatype 32bit float



Effectiveness of Adrastea

	Traditional FPGA Design	Adrastea Based SNS Random Forest	Adrastea Based FFT
Adrastea Integration Time	-	1 hr (Easy Setup)	1 hr
Interface Programming Time	Weeks (Needs Expertise)	1 hr (Very easy, similar to a function call)	1 hr
Build Script Writing	2 Days (Needs Expertise)	1 hr (Very easy, as configuring a CMake variable)	1 hr
Average Build Time	~ 4hrs	~4 hrs	~9 hrs
Number of Builds	90 (Manual)	90	54
Design Space Exploration Time	Months ~13 Days (SNS) ~21 Days (FFT) (Sequential Build Time)	~15 hrs	~26 hrs
Graphing Time	6 hrs	3 hrs (Data is already tabulated)	3 hrs

Conclusion

- Adrastea is an automated and efficient design environment to design, implement, and optimize complex FPGA kernels and their interfaces.
 - This is done by leveraging the DEFFE design space exploration tool across multiple identical VMs with the toolchain's setup and by using IRIS APIs to simplify FPGA kernel writing.
- We demonstrated the potential of Adrastea to reduce FPGA development time with two example applications, Random Forest and Fast Fourier Transforms.
- Adrastea performed the design space exploration and optimization in days in parallel whereas traditional sequential methods would have taken months.

Thank you