OAK RIDGE National Laboratory

IRIS workshop
Ranger and Adrestea

Narasinga Rao Miniskar, Aaron Young, Anthony Cabrera, Frank Liu, Jeffrey Vetter

Architecture & Performance Group

04-Jan-2022

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

ORNL is managed by UT-Battelle
for the US Department of Energy

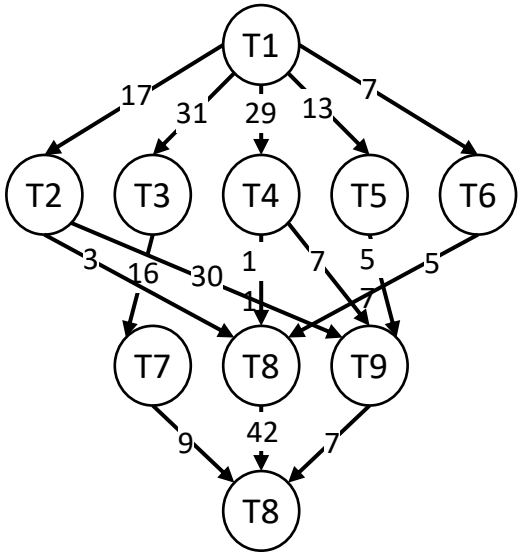
https://csmd.ornl.gov/profile/narasinga-rao-miniskar
miniskarnr@ornl.gov

RANGER PLATFORM

About RANGER

- Future of HPC: Heterogenous Computing
- Proposal
  - RANGER Platform: Hardware assisted task scheduling framework
  - RISC-V cores with Accelerators
  - Extended GEM5 simulator for RANGER platform
  - Hierarchical Task Scheduler (TS)
    - Local level: Task into fine grained sub-tasks ; Accelerator Specific TS
    - Global level: Coarse grain task specification with programming portability; Coarse grain TS
- Task Scheduling Challenges
  - Balance application performance and programmability
  - Varying data communication time between tasks
  - Varying computation time across different variants of accelerators

State of the art for Heterogenous Task Schedulers



Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
P1	22	22	32	7	29	26	14	29	15	13
P2	21	18	27	10	27	17	25	23	21	16
P3	36	18	43	4	35	24	30	36	8	33

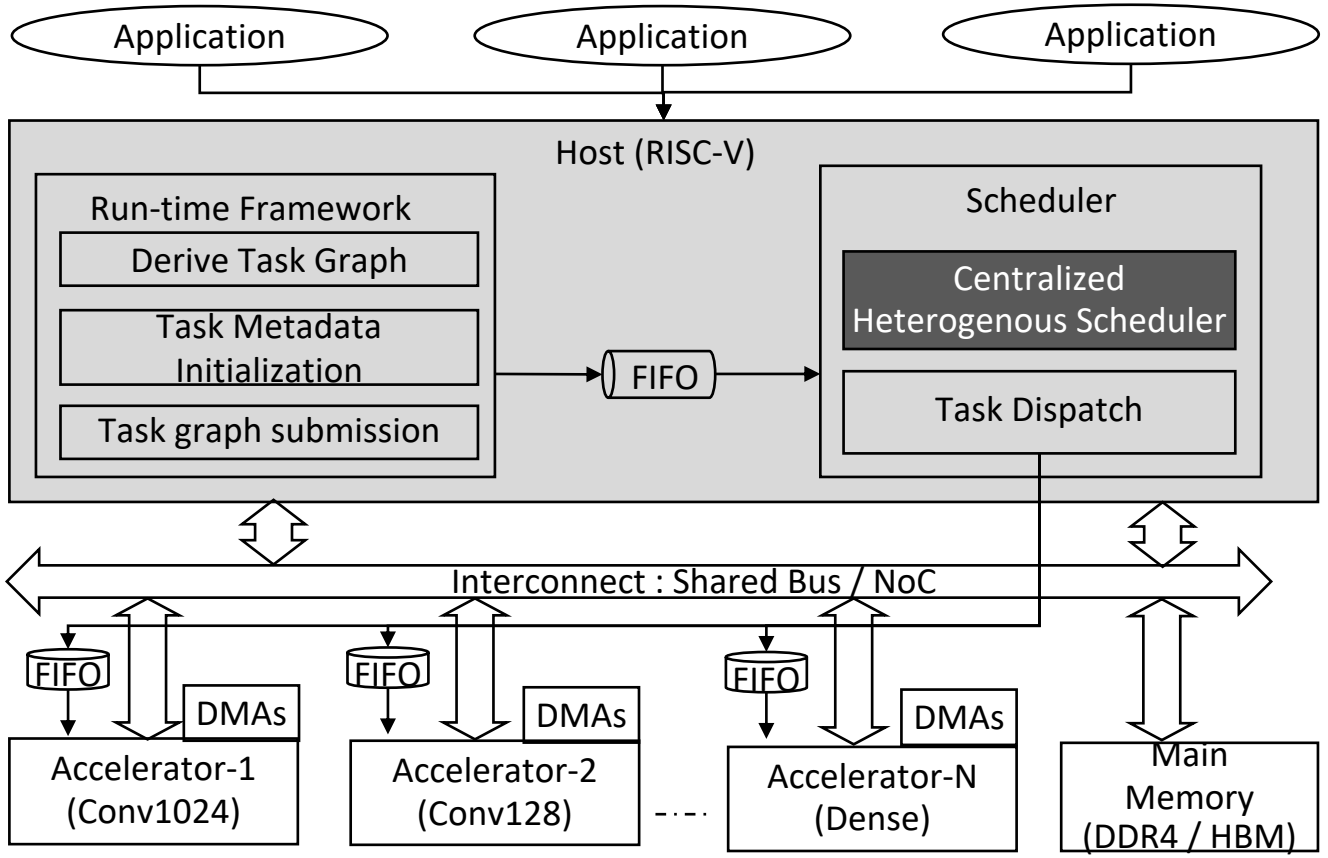
CCM* for each task on 3 devices

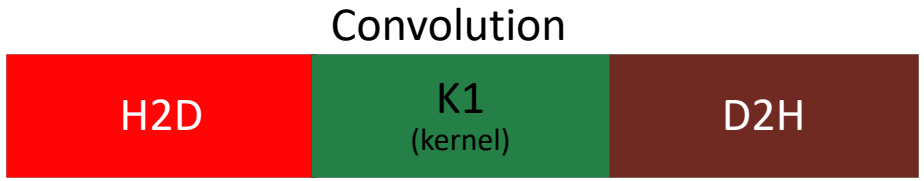
- Popular Static Task Scheduler: PEFT (Predict Earliest Finish Time)
  - Considers impact of current scheduling decision on subsequent decisions
- Task: Atomic unit for data transfer and computation
  - Can be portioned into finer sub-tasks
  - Smaller fine grain sub-tasks (streaming)
    - Lesser scratch pad memory
    - More scheduling opportunity but increases complexity
    - Less efficient (Reasonable) data transfers due to large number of sub-tasks
    - Accelerator friendly
    - Data transfers overlaps with computation

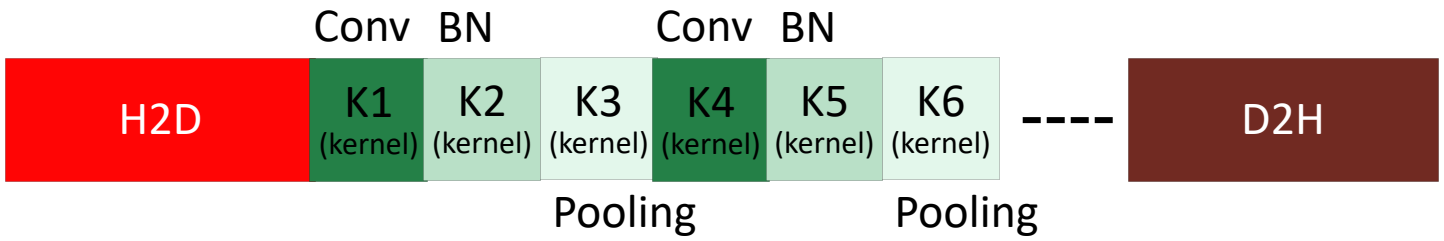
* CCM: Cost Computation Matrix

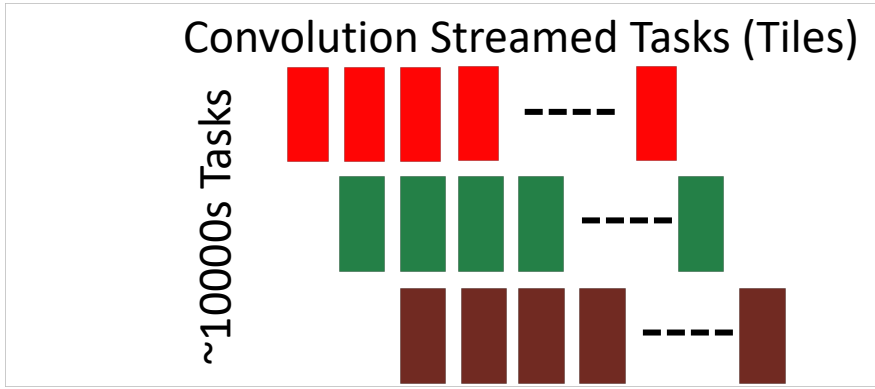
Vanilla Platform with Heterogenous Accelerators

- Centralized Heterogenous Scheduler: PEFT
- Complexity depends on the number of tasks
- Accelerators and DMAs are controlled by Host
- Needs bigger local memories (scratchpads)



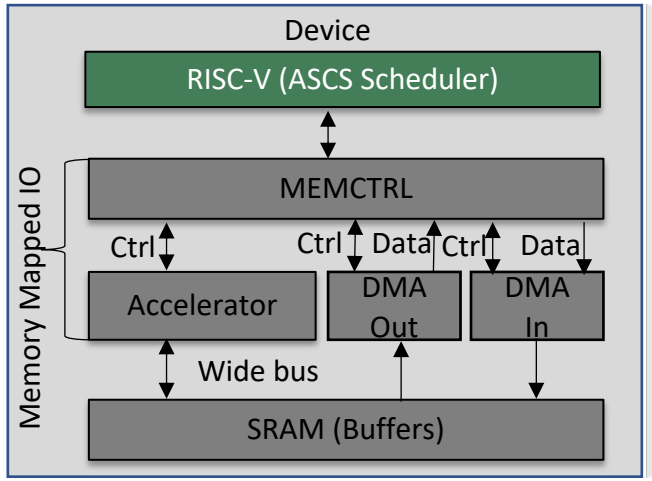


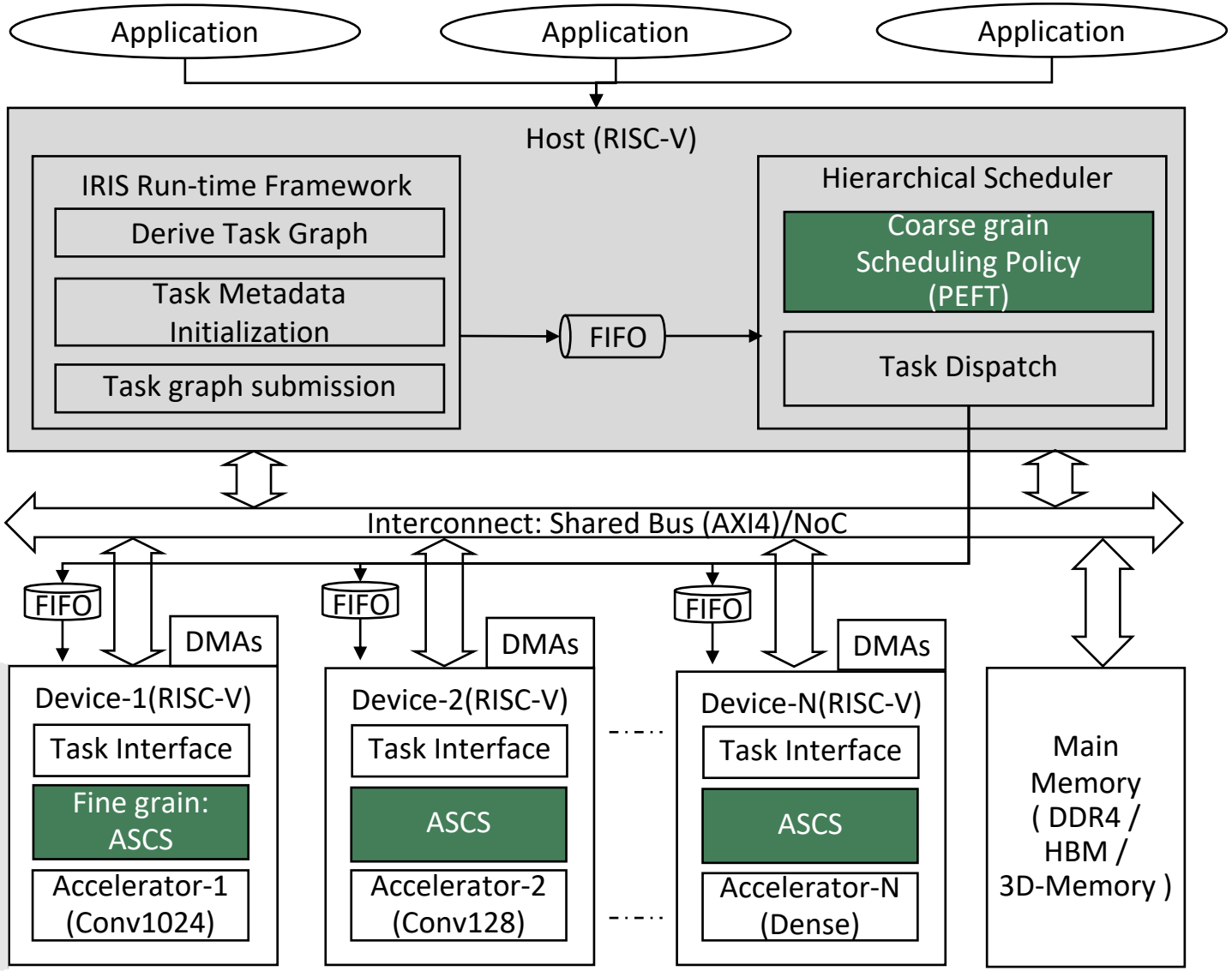




RANGER Platform with Heterogenous Accelerators

- Accelerator Interface: Memory-mapped IO
- RISC-V for host and accelerator devices
- GEM5 based modeling and simulation
- Perfect for stream dataflow tasks



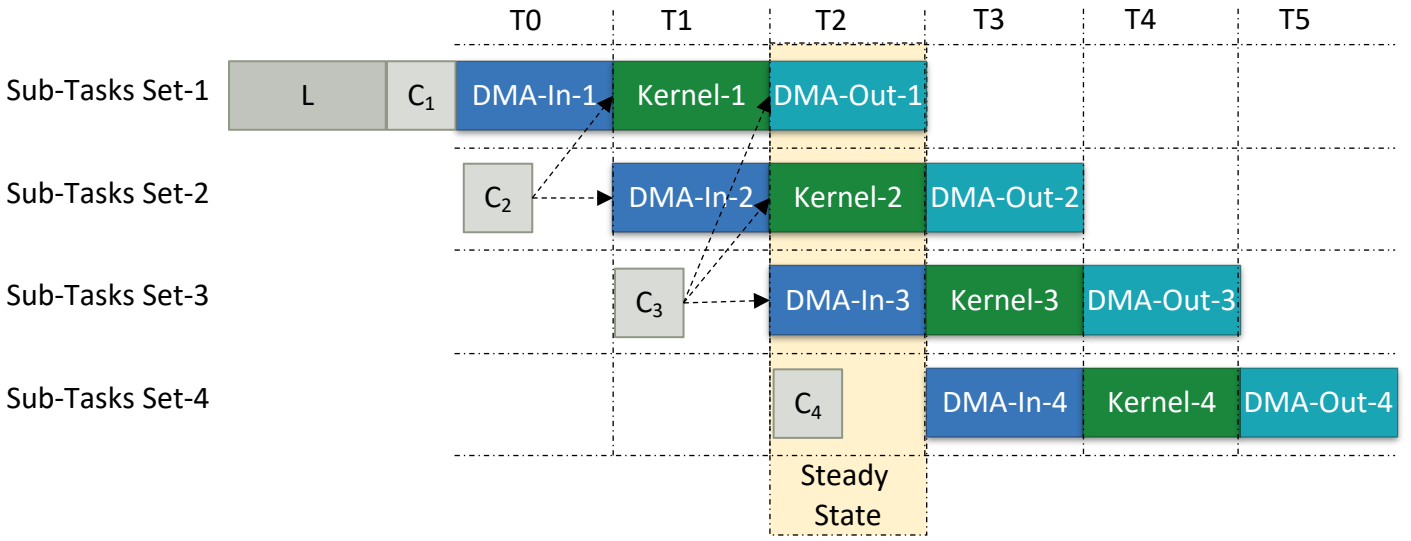


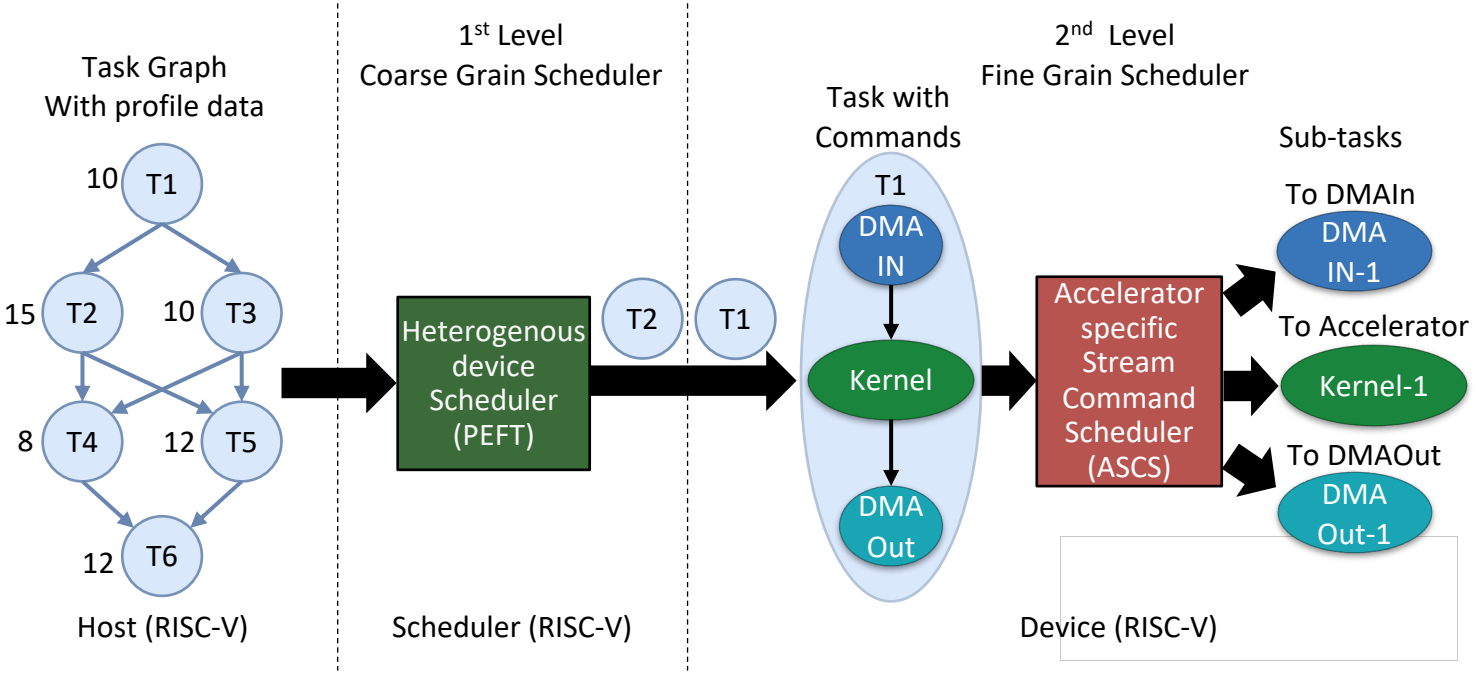
OAK RIDGE National Laboratory

Hierarchical Task Scheduler Flow (ISC-Paper)

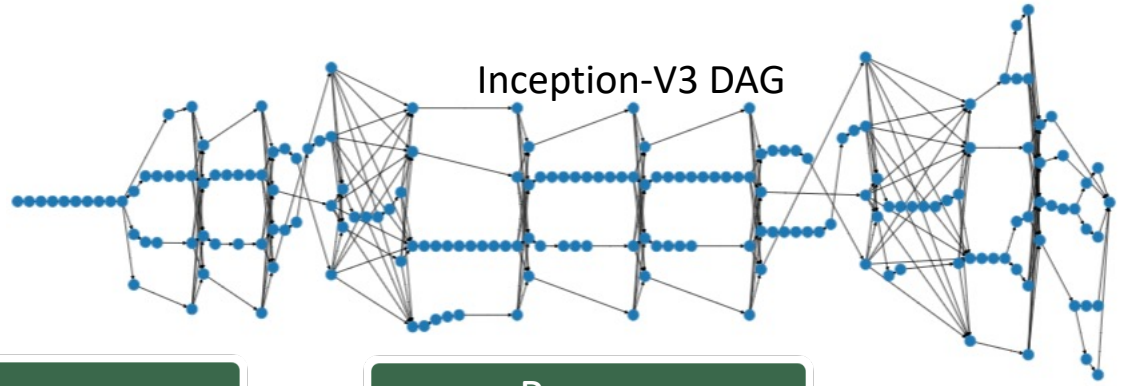
L: Task initialization time
C_i: Configuration of DMAs & Accelerator

- OpenCL supports data flow pipelines
  - Use cases: FPGA, GPU
  - Needs support in IRIS (TBD)
- For custom accelerators
  - Accelerator specific command scheduler
- Why Accelerator Specific?
  - Configurable Tile size
  - Configurable Data reuse
  - Synchronize DMAs and Accelerators
  - Programmable is a best solution
  - Generic solution is possible, but cannot satisfy timing constraints
    - Hardware task scheduler solution can be an option





Experimental Setup



- Resnet, VGG
- MobileNet, UNet
- Inception-v3

TensorFlow

- Accelerators: Convolution (CONV), Batch Normalization (BN), Dense

	Accelerator	Functionality	MAC Units	SRAM Size (KB)	Area (mm^2)
1	CONV1024	2D Convolution	1024	256.0	1.81
2	CONV512	2D Convolution	512	256.0	1.28
3	CONV256	2D Convolution	256	256.0	1.01
4	CONV128	2D Convolution	128	256.0	0.88
5	CONV64	2D Convolution	64	128.0	0.59
6	BN1024	Batch Normilization	1024	8.0	1.09
7	BN512	Batch Normilization	512	4.0	0.55
8	BN256	Batch Normilization	256	2.0	0.28
9	BN128	Batch Normilization	128	1.0	0.14
10	BN64	Batch Normilization	64	0.5	0.08
11	DENSE1024	Dense	1024	128.0	1.30
12	DENSE512	Dense	512	128.0	0.76
13	DENSE256	Dense	256	128.0	0.50
14	DENSE128	Dense	128	128.0	0.37
15	DENSE64	Dense	64	128.0	0.30

List of kernel accelerators and their area estimations in a TSMC 16nm technology

	Accelerators																Area mm^2			
	2D Convolution					Batch Normilization					Dense					Total				
	1024	512	256	128	64	1024	512	256	128	64	1024	512	256	128	64		RANGER	Baseline	Overhead
Design																				
Design 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	15	11.290	10.945	3.15 %
Design 2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	30	22.581	21.891	3.15 %
Design 3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	35	28.266	27.461	2.93 %
Design 4	4	4	4	4	4	2	2	2	2	2	2	2	2	2	2	40	33.951	33.031	2.79 %
Design 5	5	5	5	5	5	2	2	2	2	2	2	2	2	2	2	45	39.636	38.601	2.68 %
Design 6	6	6	6	6	6	2	2	2	2	2	2	2	2	2	2	50	45.321	44.171	2.60 %
Design 7	7	7	7	7	7	2	2	2	2	2	2	2	2	2	2	55	51.006	49.741	2.54 %
Design 8	8	8	8	8	8	2	2	2	2	2	2	2	2	2	2	60	56.691	55.311	2.49 %
Design 9	9	9	9	9	9	2	2	2	2	2	2	2	2	2	2	65	62.376	60.881	2.46 %
Design 10	10	10	10	10	10	2	2	2	2	2	2	2	2	2	2	70	68.061	66.451	2.42 %
Design A	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	15	11.290	10.945	3.15 %
Design B	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	30	22.581	21.891	3.15 %
Design C	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	45	33.871	32.836	3.15 %
Design D	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	60	45.161	43.781	3.15 %
Design E	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	75	56.452	54.727	3.15 %
Design F	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	90	67.742	65.672	3.15 %
Design G	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	105	79.032	76.617	3.15 %
Design H	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	120	90.323	87.563	3.15 %
Average																			2.74 %

Various heterogeneous designs, the number of kernel accelerators, and the estimated area. Design A and B are the aliases of Design 1 and 2, respectively.

Experimental Evaluation: Makespan

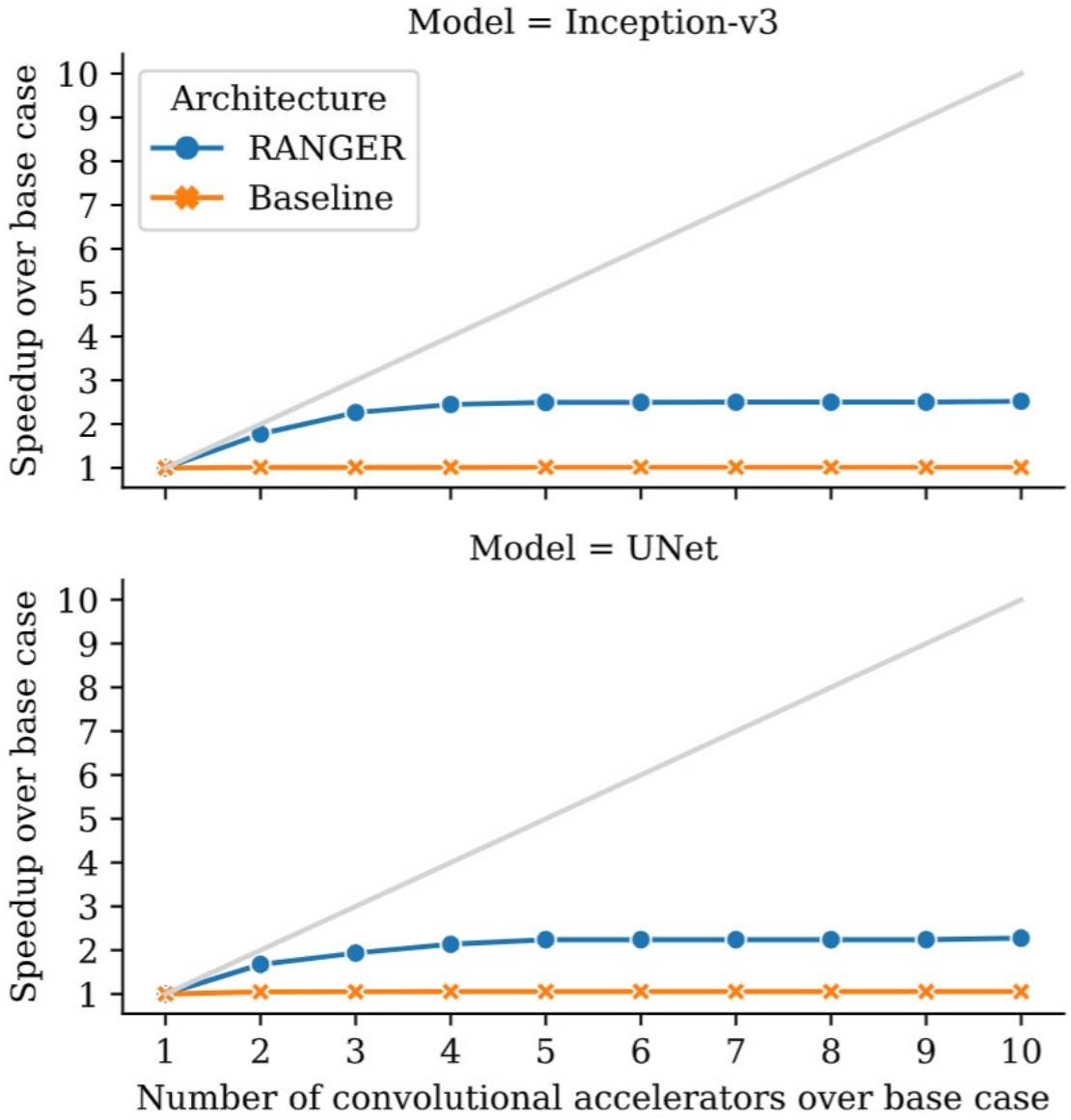
Architecture Model	RANGER	Baseline	Increase
Inception-v3	189	20,469	108×
Resnet-50	107	6,824	64×
UNet	17	12,372	728×
VGG16	16	38,064	2,379×

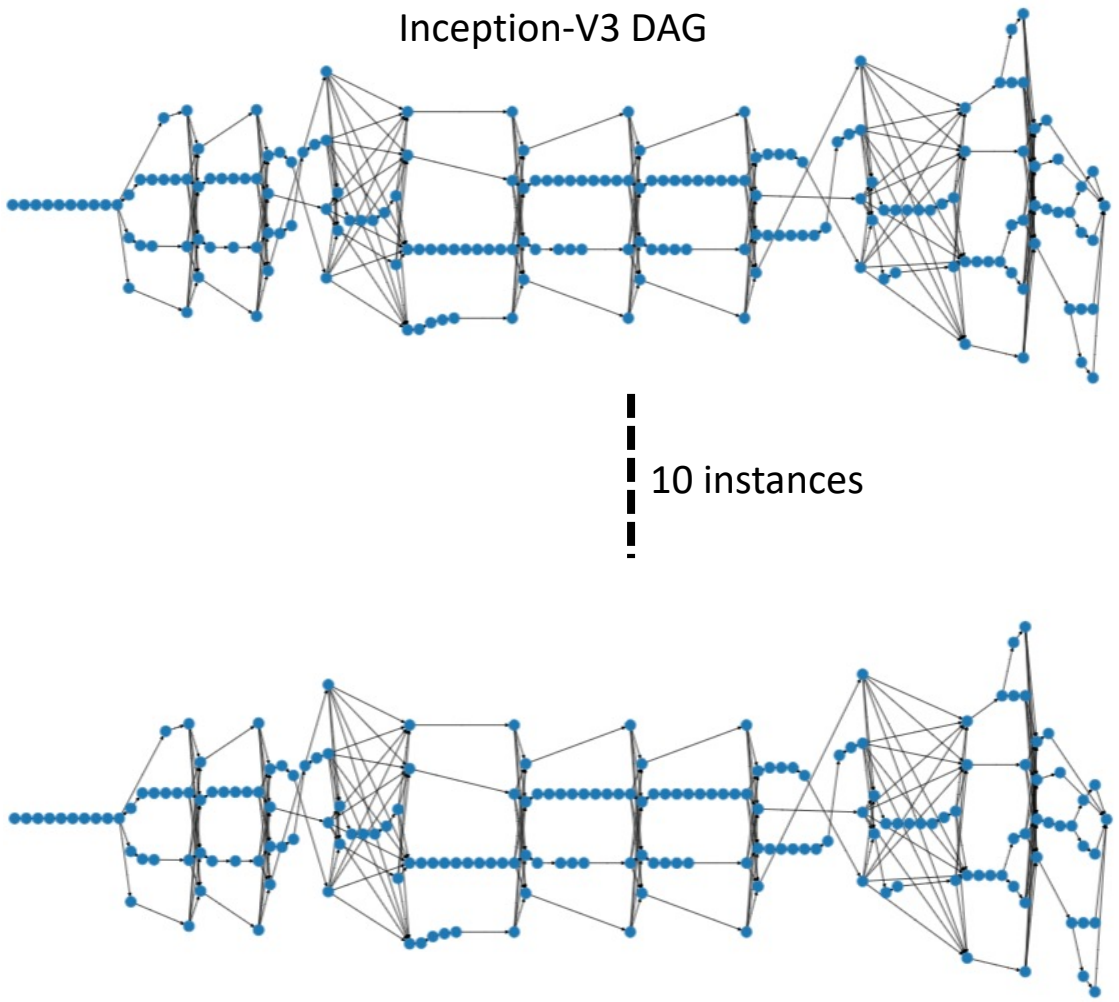
Baseline: Vanilla architecture with flattened stream tasks for centralized scheduler

Model Architecture Design	Makespan Inception-v3 RANGER	Baseline	Speedup	Resnet-50 RANGER	Baseline	Speedup	VGG16 RANGER	Baseline	Speedup	UNet RANGER	Baseline	Speedup
Design 1	94,788,333	762,320,311	8.04×	88,346,488	504,482,689	5.71×	389,202,487	3,193,921,550	8.21×	336,496,490	1,233,341,834	3.67×
Design 2	53,422,061	752,853,720	14.09×	57,531,844	500,829,541	8.71×	209,315,117	3,149,123,579	15.04×	201,002,387	1,177,865,323	5.86×
Design 3	41,881,254	752,544,223	17.97×	52,782,400	500,433,710	9.48×	191,533,321	3,147,951,787	16.44×	173,986,414	1,171,882,097	6.74×
Design 4	38,749,897	752,450,123	19.42×	52,195,186	500,340,314	9.59×	181,632,752	3,147,767,996	17.33×	157,652,345	1,169,892,512	7.42×
Design 5	37,999,017	752,407,810	19.80×	52,191,088	500,295,143	9.59×	180,193,518	3,147,649,984	17.47×	150,250,684	1,169,590,055	7.78×
Design 6	37,988,551	752,393,040	19.81×	52,191,088	500,288,595	9.59×	180,193,518	3,147,724,843	17.47×	150,250,684	1,169,578,215	7.78×
Design 7	37,914,589	752,387,551	19.84×	52,190,081	500,282,221	9.59×	180,193,518	3,147,642,459	17.47×	150,250,684	1,169,575,296	7.78×
Design 8	37,898,912	752,382,660	19.85×	52,190,081	500,275,555	9.59×	180,193,518	3,147,724,163	17.47×	150,250,684	1,169,574,480	7.78×
Design 9	37,891,335	752,382,660	19.86×	52,190,081	500,272,549	9.59×	180,193,518	3,147,669,094	17.47×	150,250,684	1,169,574,480	7.78×
Design 10	37,618,903	752,367,084	20.00×	52,190,081	500,264,005	9.59×	178,630,640	3,147,724,117	17.62×	147,870,807	1,169,574,480	7.91×
Design A	94,788,333	762,320,311	8.04×	88,346,488	504,482,689	5.71×	389,202,487	3,193,921,550	8.21×	336,496,490	1,233,341,834	3.67×
Design B	53,422,061	752,853,720	14.09×	57,531,844	500,829,541	8.71×	209,315,117	3,149,123,579	15.04×	201,002,387	1,177,865,323	5.86×
Design C	42,241,318	752,541,894	17.82×	52,447,643	500,420,375	9.54×	174,801,184	3,144,474,922	17.99×	173,986,414	1,171,816,637	6.74×
Design D	38,570,566	752,444,207	19.51×	51,672,158	500,332,941	9.68×	155,344,163	3,144,196,072	20.24×	157,652,123	1,169,797,015	7.42×
Design E	37,793,805	752,406,134	19.91×	51,362,009	500,254,081	9.74×	153,596,950	3,144,134,148	20.47×	150,247,240	1,169,471,954	7.78×
Design F	37,793,452	752,401,577	19.91×	51,162,596	500,254,081	9.78×	153,596,950	3,144,134,148	20.47×	150,247,240	1,169,470,736	7.78×
Design G	37,708,203	752,386,686	19.95×	51,162,596	500,254,081	9.78×	153,596,950	3,144,134,148	20.47×	150,247,240	1,169,468,346	7.78×
Design H	37,704,900	752,386,686	19.95×	51,161,299	500,254,081	9.78×	153,596,950	3,144,134,148	20.47×	150,247,240	1,169,468,346	7.78×
Average			17.66×			9.10×			16.96×			6.96×

Comparison of makespans for various RANGER and baseline designs. On average, RANGER achieves a 12.7×speedup

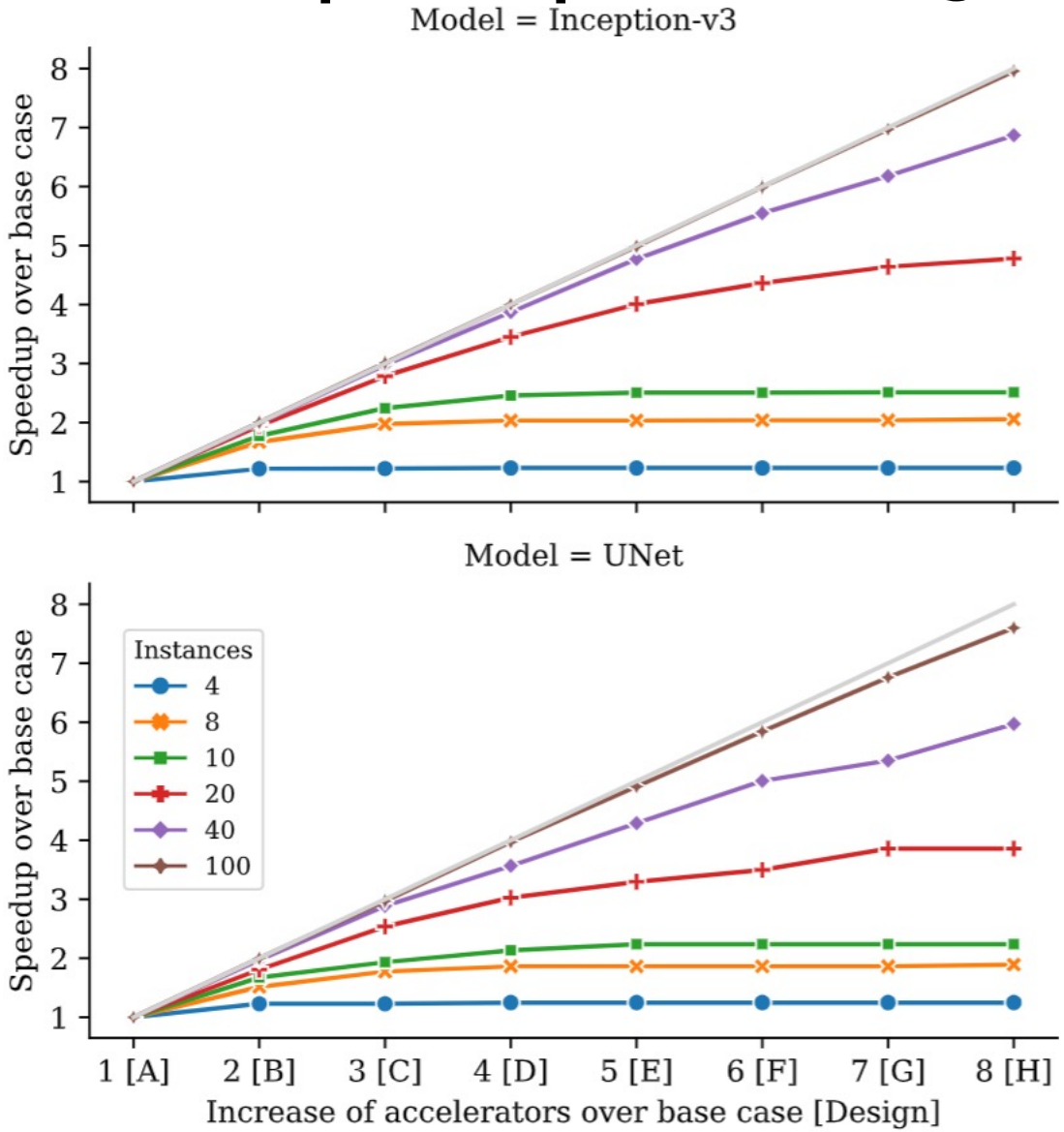
RANGER Speedup (Fixed 10 parallel instances of each application)

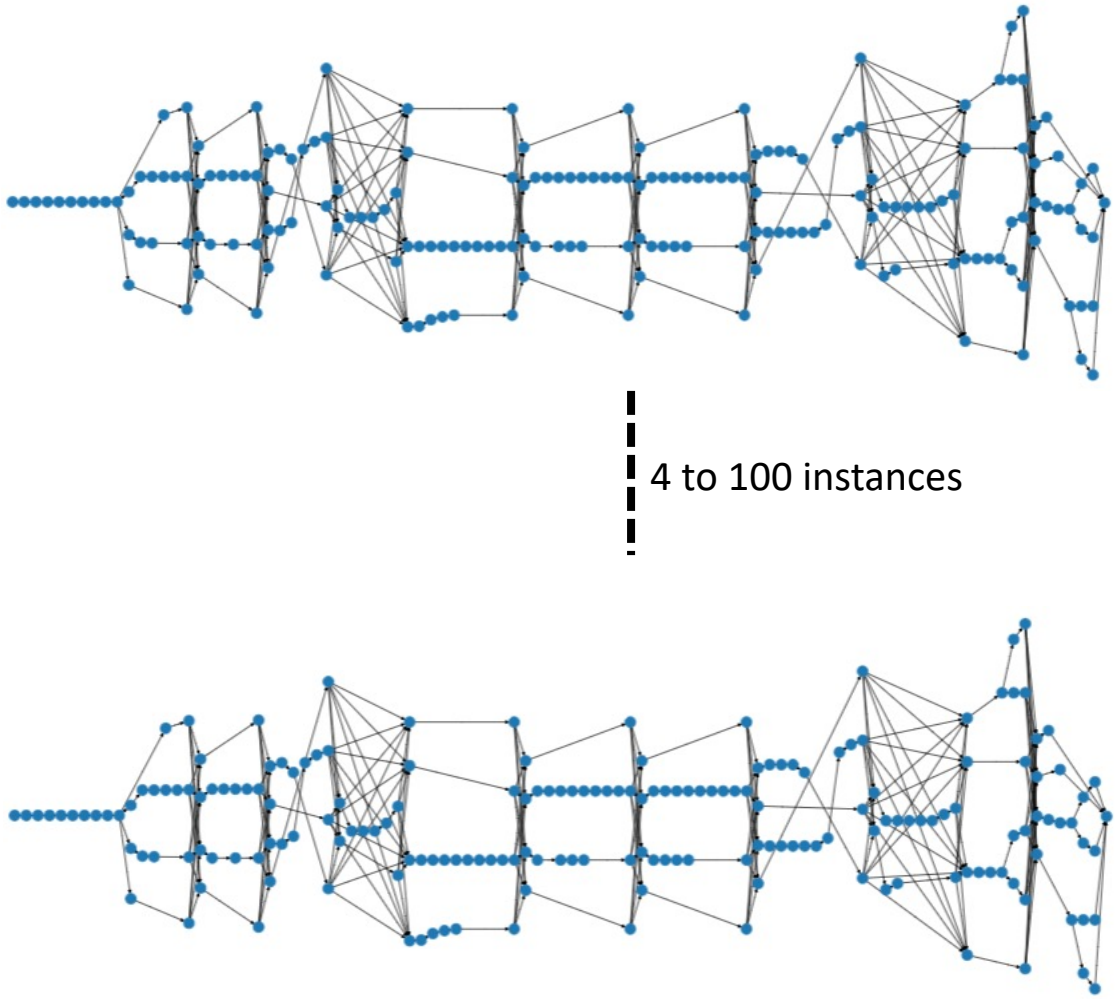




Measured speedup of RANGER by running 10 parallel instances of each application with respect to Designs 1–10, which contain an increasing number of kernel accelerators. The speedup plateaus at Design 3 due to an insufficient number of tasks for the available kernel accelerators

RANGER Speedup (Increasing number of application instances)





Measured speedup of RANGER by running an increasing number of instances of the same application on Designs A–H.
RANGER demonstrates excellent scalability with 100 instances of application running in parallel

Area and Makespan Compared to Reference

- Reference: (Hypothetical) Identical number of Accelerators as its RANGER counterpart, but has huge scratchpad memory
- Has large scratchpad to hold complete task I/O
- Requires no local task scheduler

Model Architecture Design	Area mm^2 RANGER	Reference	Difference
Design 1	11	165	14.61×
Design 2	23	275	12.18×
Design 3	28	337	11.93×
Design 4	34	440	12.96×
Design 5	40	445	11.23×
Design 6	45	484	10.67×
Design 7	51	512	10.04×
Design 8	57	546	9.63×
Design 9	62	603	9.67×
Design 10	68	662	9.73×

Model Architecture Design	Makespan Inception-v3 RANGER	Reference	Difference	Resnet-50 RANGER	Reference	Difference	VGG16 RANGER	Reference	Difference	UNet RANGER	Reference	Difference
Design 1	94,788,333	95,707,644	-0.96%	88,346,488	88,621,038	-0.31%	389,202,487	214,684,328	81.29%	336,496,490	216,796,889	55.21%
Design 2	53,422,061	56,538,771	-5.51%	57,531,844	63,717,300	-9.71%	209,315,117	165,634,098	26.37%	201,002,387	156,806,696	28.18%
Design 3	41,881,254	45,388,469	-7.73%	52,782,400	58,538,562	-9.83%	191,533,321	150,559,308	27.21%	173,986,414	145,184,318	19.84%
Design 4	38,749,897	40,891,867	-5.24%	52,195,186	56,076,426	-6.92%	181,632,752	144,136,809	26.01%	157,652,345	134,955,240	16.82%
Design 5	37,999,017	39,601,255	-4.05%	52,191,088	55,193,132	-5.44%	180,193,518	141,111,694	27.70%	150,250,684	129,117,096	16.37%
Design 6	37,988,551	38,921,698	-2.40%	52,191,088	54,871,507	-4.88%	180,193,518	139,536,510	29.14%	150,250,684	128,893,489	16.57%
Design 7	37,914,589	38,412,723	-1.30%	52,190,081	54,847,500	-4.85%	180,193,518	138,354,082	30.24%	150,250,684	127,945,051	17.43%
Design 8	37,898,912	38,260,460	-0.94%	52,190,081	54,561,879	-4.35%	180,193,518	137,473,689	31.07%	150,250,684	127,915,851	17.46%
Design 9	37,891,335	38,260,436	-0.96%	52,190,081	54,561,879	-4.35%	180,193,518	137,473,689	31.07%	150,250,684	127,915,851	17.46%
Design 10	37,618,903	37,782,552	-0.43%	52,190,081	53,711,730	-2.83%	178,630,640	134,278,895	33.03%	147,870,807	121,184,411	22.02%
Average			-2.95%			-5.06%			31.01%			20.53%

Comparison of makespans for RANGER and reference. On average, RANGER shows only 10.88% of penalty, which is the measurement of performance overhead of the local ASCS.

RANGER Summary

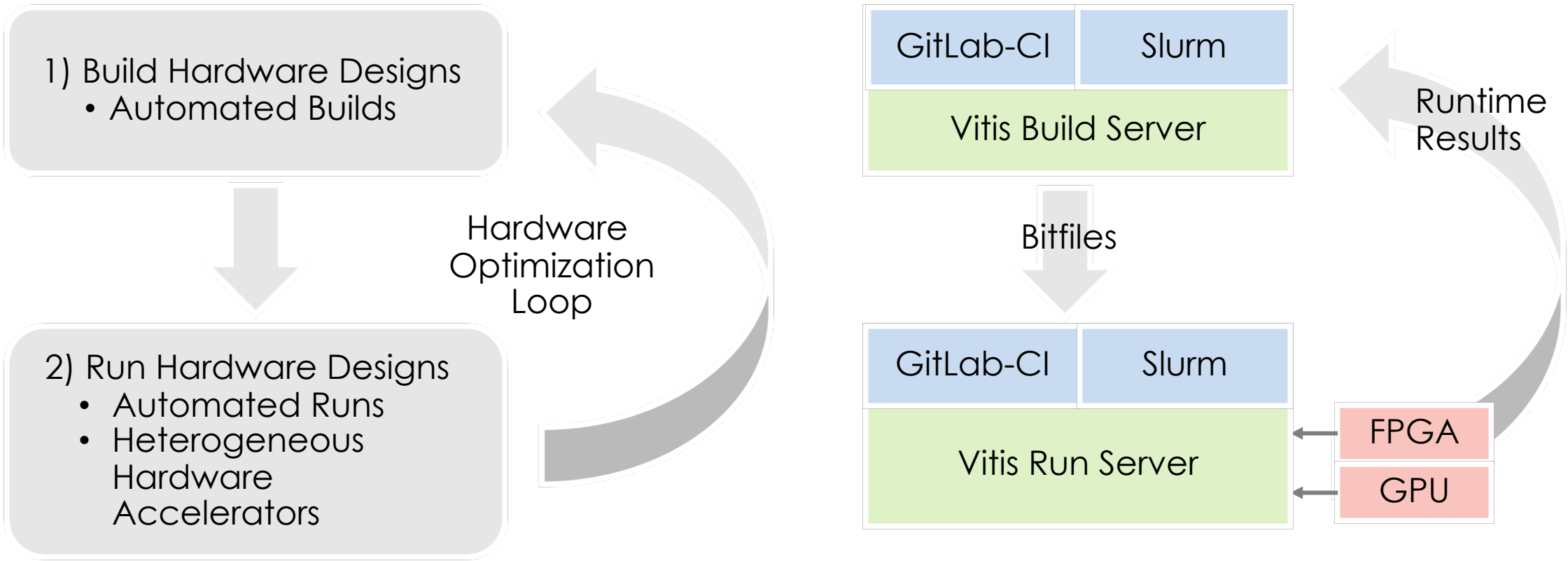
- Presents RANGER framework (Extremely heterogenous computing)
  - An architecture design for hierarchical task scheduling
- Hierarchical Task Scheduling
  - Requires coarse grained task dependency specification
  - Fine grain accelerator specific scheduling at lower level
- RANGER uses customized RISC-V cores
- Achieves 12.7x performance gain in terms of makespan
- Area Overhead: +2.7% in a 16nm technology
- Future work:
  - Experiment different scheduling policies with HUNTER and DAGGER
  - Hardware task scheduler

Adrestea: An efficient FPGA design environment for scientific machine learning

Why Adrestea and What is it?

- FPGA Programming Challenges
  - Writing interface program to communicate with FPGA kernels
  - Writing kernels along with other heterogenous compute units
  - Design space exploration
- Adrestea: An efficient FPGA design environment for scientific machine learning applications to run at the edge
  - Simplified call to FPGA kernel; Extended IRIS Runtime Framework (https://iris-programming.com)
  - Enables to write kernel targeting other compute units as well with run-time selection
  - An intelligent design space exploration environment for FPGAs
  - Proof of concept:
    - Application: Random forest classifier ; FPGA: Xilinx Alveo U250
- SC 2022 Demo: https://ornl.sharepoint.com/:v:/s/CSMDSTAFF-ACSR-APG/EYN-bqYLgB5Msg_pd0oZJnUBbBTp-bkE4ba_K1CflpeGrA?e=jahFX0

Automated Adrastea Development Environment

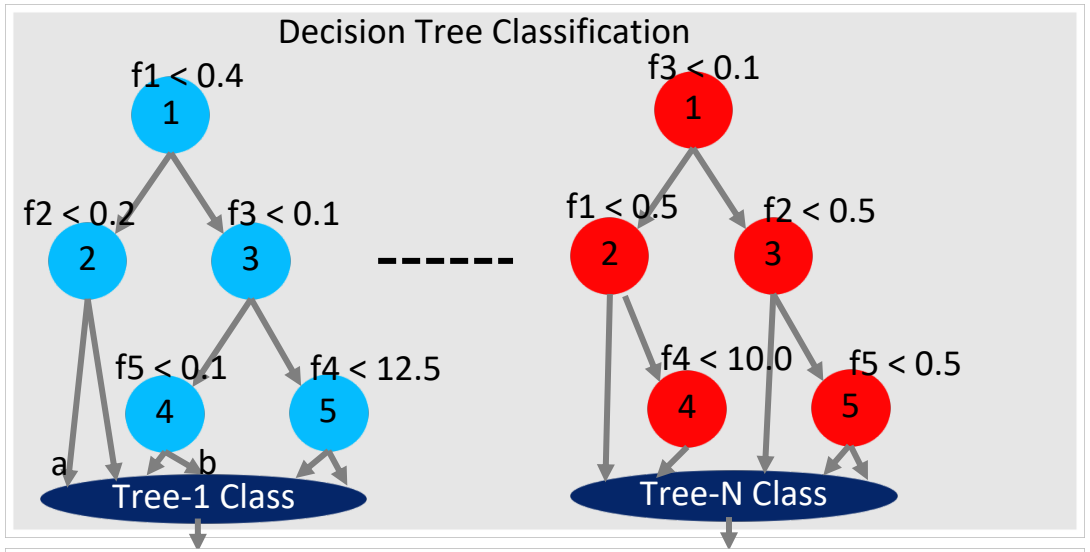


Note: Servers are VMs with the Vitis toolchain installed and hardware added with PCIe Passthrough.

This enables:

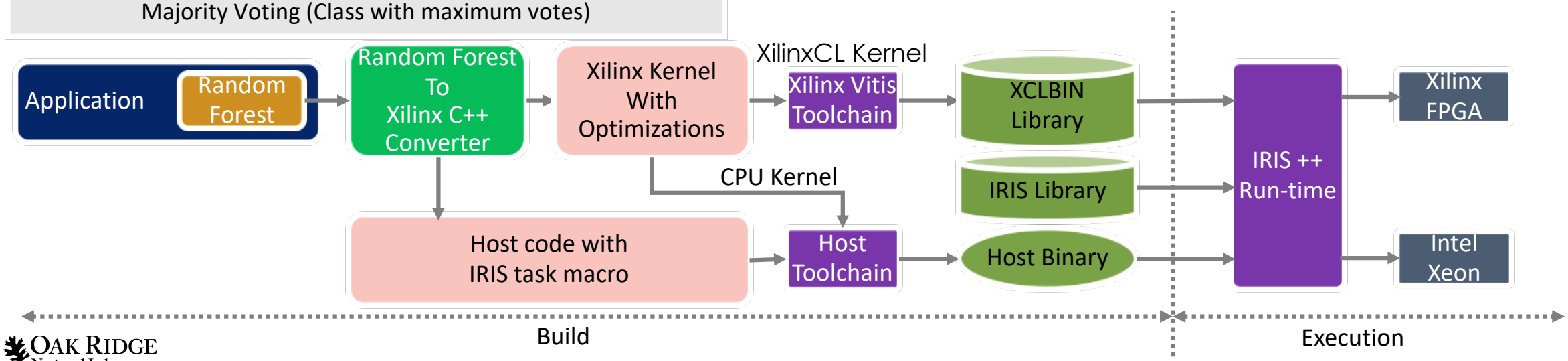
- Easy configuration of development environment.
- Easy migration of VMs to new hosts for additional resources.

Application: Random Forest (RF) Classifier With IRIS



RF Models

RF Model	IRIS	Breast Cancer	Olivetti	SNS
Input feature size	4	30	4096	10000
# of classes	3	2	40	2
# of trees in forest	20	100	100	100
# of comparison nodes in forest	161	2081	6237	6044
# of terminal nodes in forest	181	2181	6337	6144
Max/Min depth of tree	7 / 4	11 / 5	20 / 20	26 / 10



Application build with IRIS

CMakeLists.txt

```
#set(XILINX_TARGET  "$ENV{XCL_EMULATION_MODE}")  # sw_emu
set(XILINX_LANGUAGE "xilinx")
#set(XILINX_FREQUENCY "1000000000")
set(XKERNEL  rf_100.0_d20_t100_uint8_t)
set(XKERNEL_DIR  breast_cancer)
set(XKERNEL_NAME rf_classifier)

set(XILINX_OPENCL_SOURCES
    ${XKERNEL_NAME} kernel_stage src/kernel.cl
   )
set(XILINX_KERNEL_SOURCES
   ${XKERNEL_NAME} kernel_stage src/kernel.xilinx.cpp
   )

set(OPENMP_KERNEL_SOURCES
   src/kernel.openmp.c
   )

set(APP_SOURCES
   ${XKERNEL_DIR}/${XKERNEL}.c
   )
```

FPGA (OpenCL/XilinxCL), CPU, GPU task specification

Host Code

```
#ifdef ENABLE_BRISBANE
        BRISBANE_SINGLE_TASK(task0, "rf_classifier", target_dev, 1,
                NULL_OFFSET, GWS(SIZE), LWS(SIZE),
                IN_BRISBANE(features, IRISFeatures *, IRISFeatures,
                    features, features_size),
                PARAM(batch_size, int32_t),
                OUT_BRISBANE(class, INT8_T *, INT8_T,
                    goutput, goutput_size));
#else
        rf_classifier(features,
                batch_size,
                goutput);
#endif
```

CPU call

kernel.cl
Kernel.xilinx.cpp
kernel.cpu.cpp

```
OPENCL_KERNEL void rf_classifier(
        BATCH_DTYPE *batch,
        int32_t batch_size,
        OPENCL_PARAM INT8_T *goutput BRISBANE_OPENMP_PARAMS) {
    main_loop: for(INT32_T i = 0; i<batch_size; i++)
    {
        #pragma HLS PIPELINE II=1
        FEATUTRES_DTYPE features = BATCH_PARAM(batch, FEATURE_SIZE, i);
        INT8_T *output = &goutput[i];
        UINT8_T votes[3] = {0,};
        INT8_T _class = -1;
        _class = rf_classifier_tree_0(features); votes[_class] += 1;
        _class = rf_classifier_tree_1(features); votes[_class] += 1;
        _class = rf_classifier_tree_2(features); votes[_class] += 1;
        _class = rf_classifier_tree_3(features); votes[_class] += 1;

        INT8_T most_voted_class = -1;
        UINT8_T most_voted_votes = 0;

        for(INT8_T j=0; j<3; j++) {
            if (votes[j] > most_voted_votes) {
                most_voted_class = j;
                most_voted_votes = votes[j];
            }
        }
        *output = most_voted_class;
    }
}
```

Build Commands

```
$ export XCL_EMULATION_MODE=sw_emu
$ cmake –DUSE_XILINX=ON ../
$ make –j install
$ cd install
$ ./irf   3    # RUN on FPGA
```

BRISBANE Macros and Interface Code Generation

```
#ifdef ENABLE_BRISBANE
        BRISBANE_SINGLE_TASK(task0, "rf_classifier", target_dev, 1,
                NULL_OFFSET, GWS(SIZE), LWS(SIZE),
                IN_BRISBANE(features, IRISFeatures *, IRISFeatures,
                    features, features_size),
                PARAM(batch_size, int32_t),
                OUT_BRISBANE(class, INT8_T *, INT8_T,
                    goutput, goutput_size));
#else
        rf_classifier(features,
                batch_size,
                goutput);
#endif
```

IRIS Task Calls Extractor And
Kernel Interface Generator (Python)

Interface code for OpenMP/Hexagon kernels:

- brisbane_rf_classifier_setarg
- brisbane_rf_classifier_setmem
- brisbane_setarg
- brisbane_setmem
- brisbane_kernel
- brisbane_launch

```
brisbane_task task0;
brisbane_mem __iris_features;
brisbane_mem __iris_class;
brisbane_mem_create((features_size), &__iris_features);
brisbane_mem_create((goutput_size), &__iris_class);
void* __task_params[] = { __iris_features,
    &batch_size, __iris_class, };
int __task_params_info[] = { -1, sizeof(batch_size), -2, };
size_t *__st_offset = (size_t *)0;
size_t __st_gws[] = { (size_t) SIZE, };
size_t __st_lws[] = { (size_t) SIZE, };
brisbane_task_create(&task0);
brisbane_task_h2d(task0, __iris_features, 0,
        ((features_size)), features);
brisbane_task_kernel(task0, "rf_classifier", 1,
        __st_offset, __st_gws, __st_lws,
        sizeof(__task_params_info)/sizeof(int),
        __task_params, __task_params_info);
brisbane_task_d2h(task0, __iris_class, 0,
        ((goutput_size)), goutput);
brisbane_task_submit(task0, target_dev, __null, 1);
brisbane_task_release(task0);
brisbane_mem_release(__iris_features);
brisbane_mem_release(__iris_class);
```

RF Xilinx Generator Knobs and Optimizations

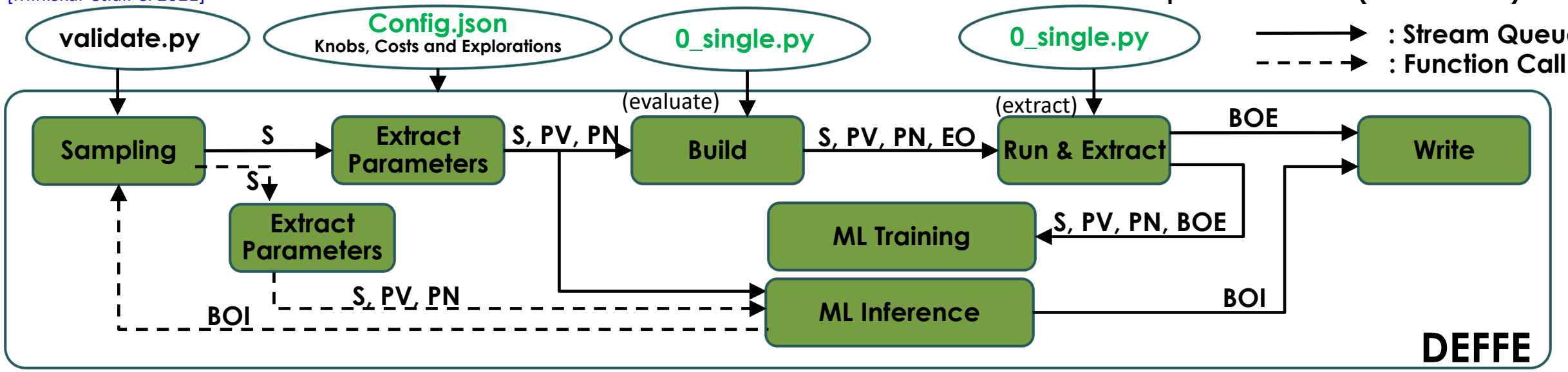
- Algorithmic Knobs
  - Fixed point knobs
  - Data type (8-bit, 16-bit, 32-bit)
  - Bitwise optimizations
  - Data layout optimizations
  - Features pruning
  - Voting arithmetic fusion
  - Static single assignment of Voting
  - Early prediction with confidence
  - Configurable number of M-AXI interfaces and bitwidth
  - Streaming tasks
- Build knobs
  - Clock Frequency
- Compiler pragmas
  - #pragma HLS PIPELINE II
  - #pragma HLS INTERFACE axis port=classify_stream
  - #pragma HLS INTERFACE m_axi port=batch_g0 bundle=maxiport0 offset=slave
  - #pragma HLS INTERFACE ap_ctrl_none port=return
  - #pragma HLS DATAFLOW
  - #pragma HLS array_partition variable=
  - #pragma HLS inline
  - #pragma HLS unroll

Automated Design Space Exploration

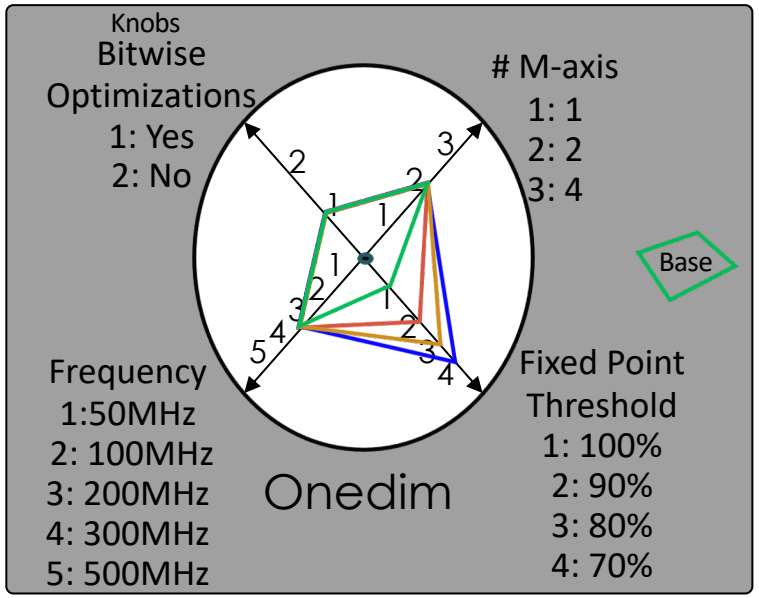
DEFFE: https://github.com/miniskar/deffe/tree/release.v1

[Miniskar et.al. CF2021]

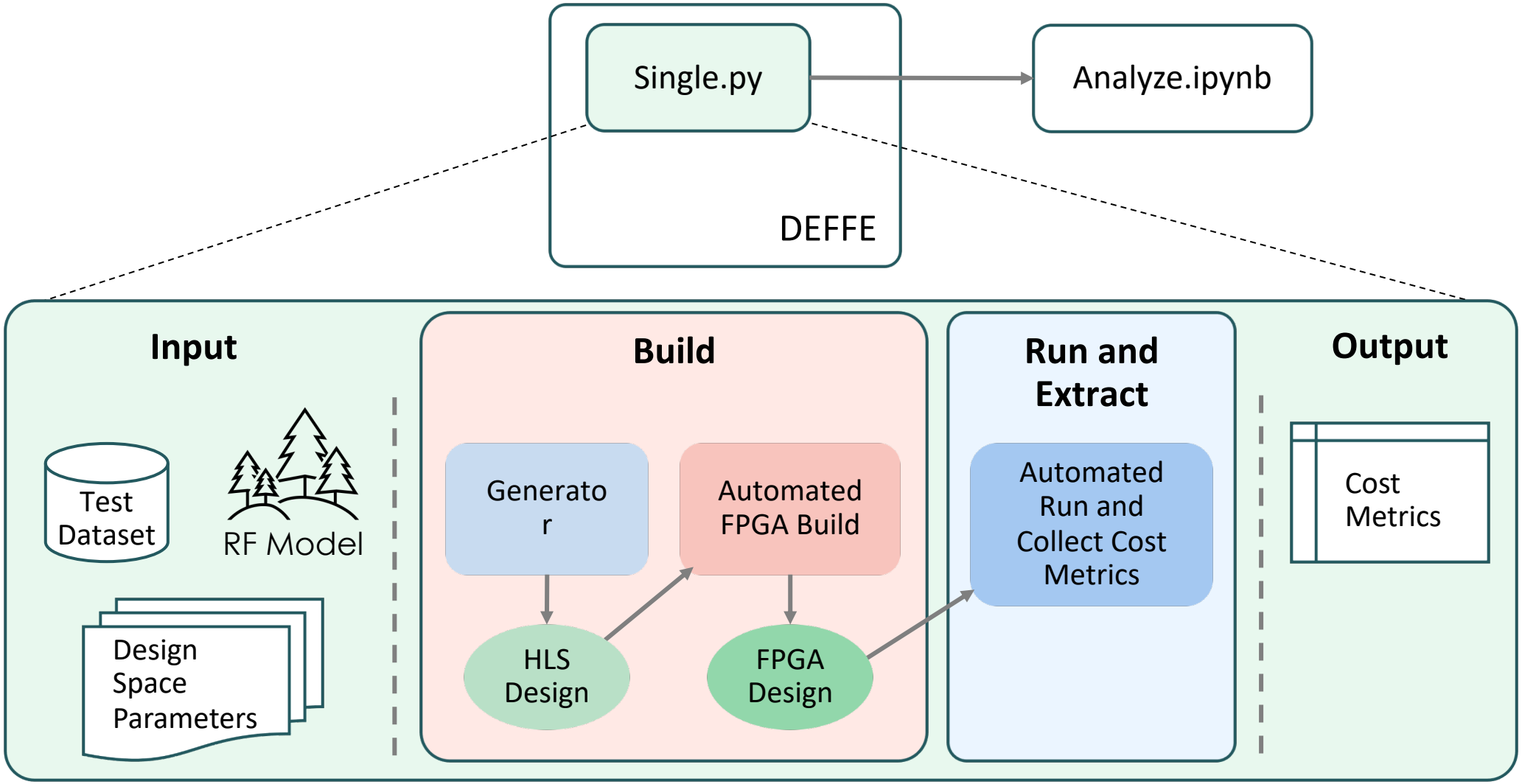
S: Sampling (Indexes)
PV: Parameter Values
PN: Parameter Normalized Values
EO: Evaluation Output
BOE: Batch Output of Extraction (Cost metrics)
BOI: Batch Output of Inference (Cost metrics)



- Sequential and Parallel threads with Streams
- Sampling techniques: Random, DOEPY sampling techniques: Fractional factorial, Plackett Burman, Sukharev Grid, Box-Wilson, Latin hyper cube (LHC), Space filling LHC, Random K-mean cluster, Maximin, Halton, Uniform random matrix, DEFFE-Smart, DEFFE-One-dimension (Onedim)
- ML: Keras/PyTorch-DNN, Random Forest, SVM
- Evaluate: Multi-Thread, Slurm

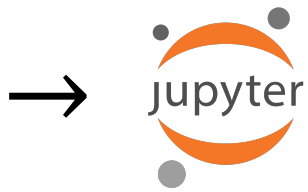


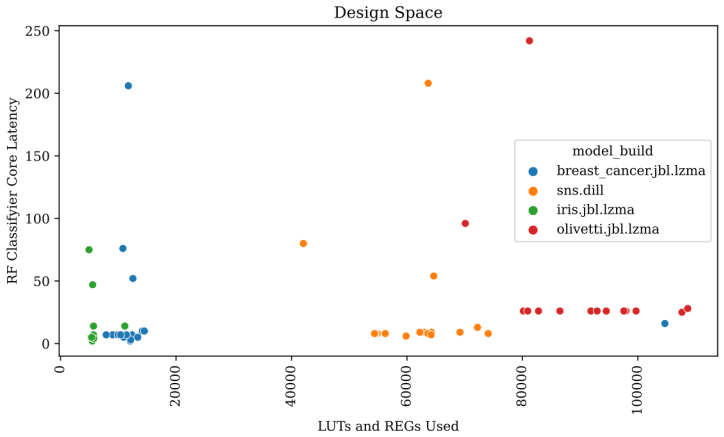
Single Run Script

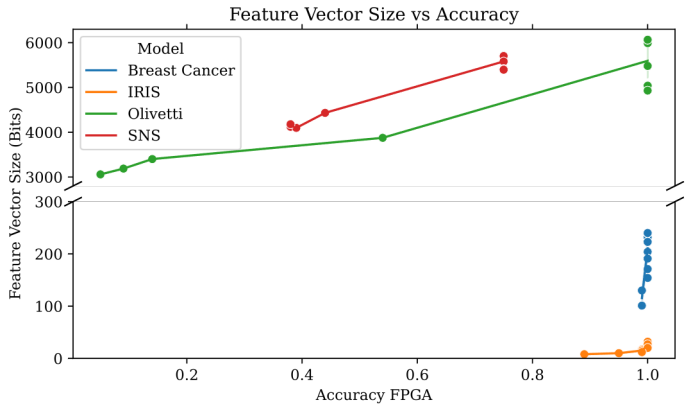


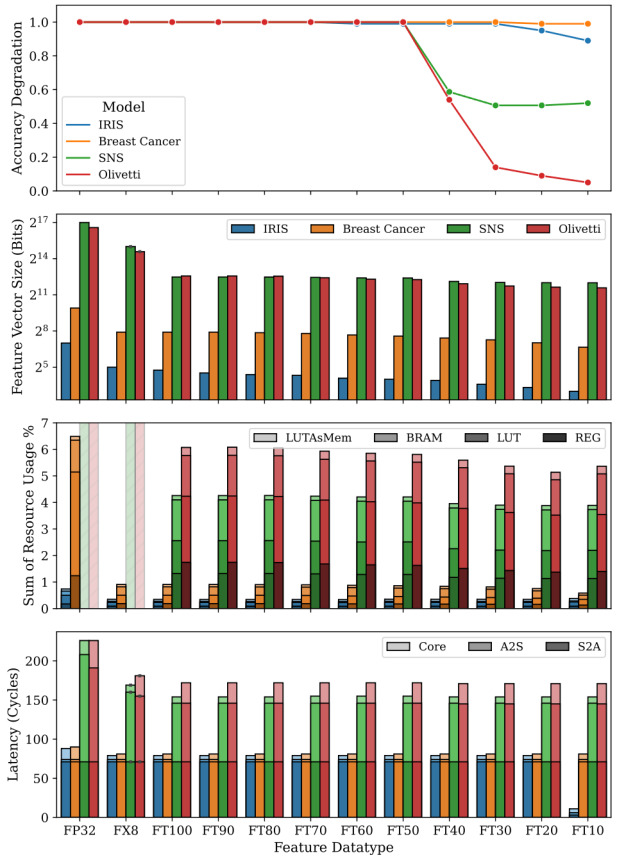
Analyze Jupyter Notebook

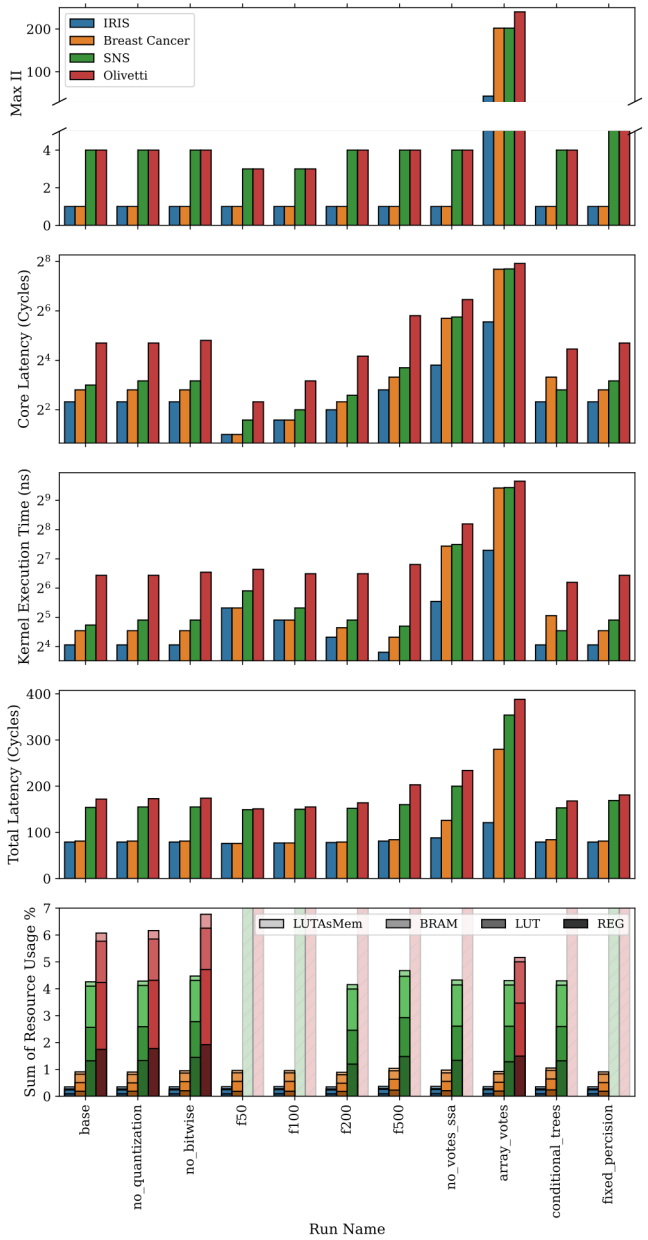
Cost Metrics











Thank you

- Resnet, VGG
- MobileNet
- Inception-v3