Leveraging Vendor Tools for AI Acceleration

Joshua Einstein-Curtis

RadiaSoft LLC

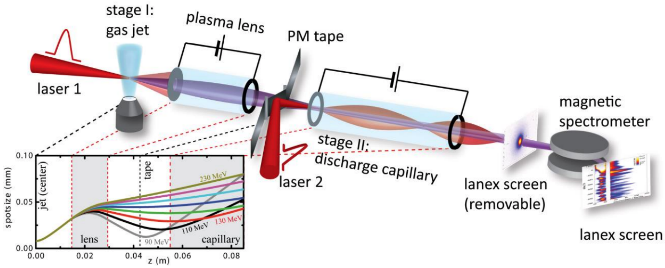
March 6, 2024

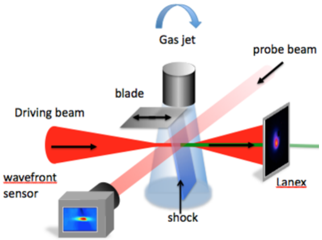
This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, under Award Number(s) DE-SC0021680.

Introduction

High intensity lasers are a critical technology for present-day and future accelerators

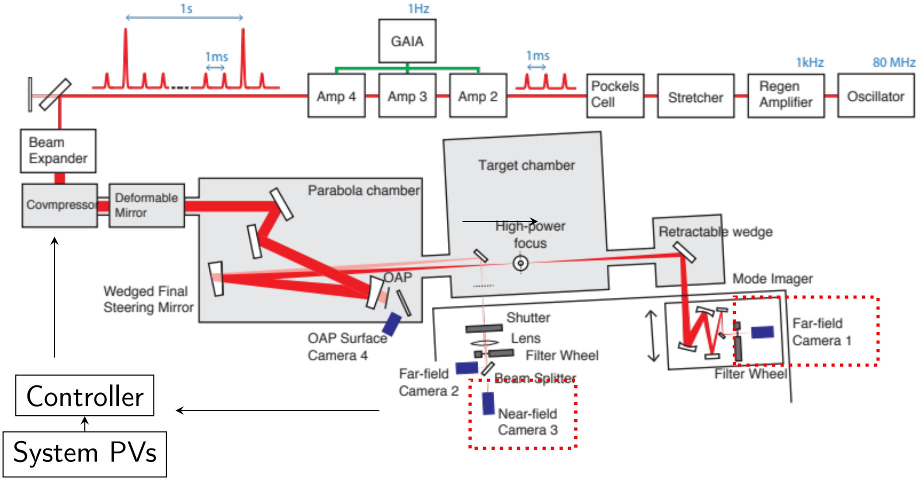
Laser focal position (and temporal beam profile) is a critical figure of merit for plasma accelerator applications





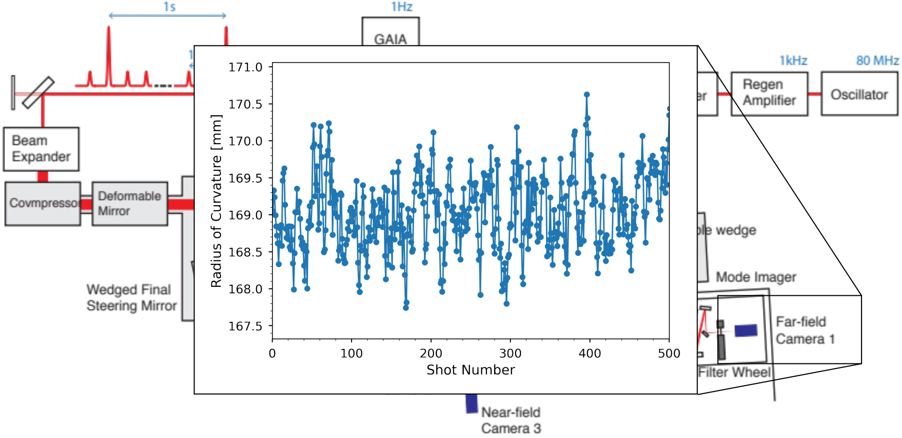
Introduction

Objective: Utilize a fast non-perturbative wavefront sensor to predict focal position with high accuracy. A motorized beam expander will permit rapid corrections to the focus



Focal position variation is a concern

Systems exhibit significant shot-to-shot fluctuations in focal position, as evidenced by high-quality laser wavefront measurements taken at the beamline

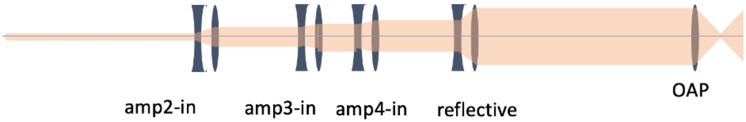


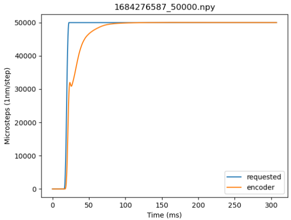
Beam Telescope and Motion Stage

A transmissive, telescopic beam expander enables flexible focal position adjustment

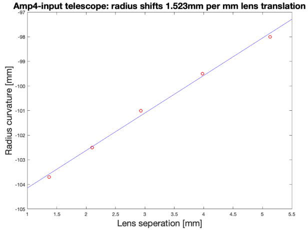
We chose a Zaber X-LDA025A-AE53D12 for prototype testing due to its availability and features

Built-in PID controller and serial communication baud rate limits control bandwidth





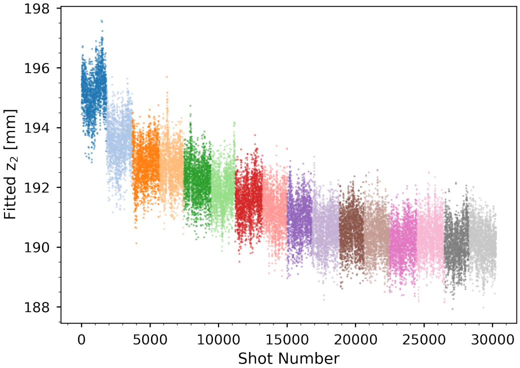
Linear stage response



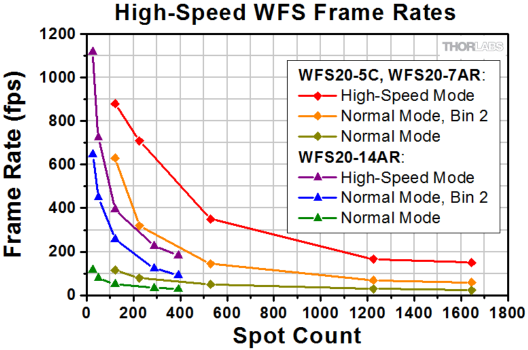
Focal position sensitivity

Speed and fidelity tradeoffs motivate processing pipeline

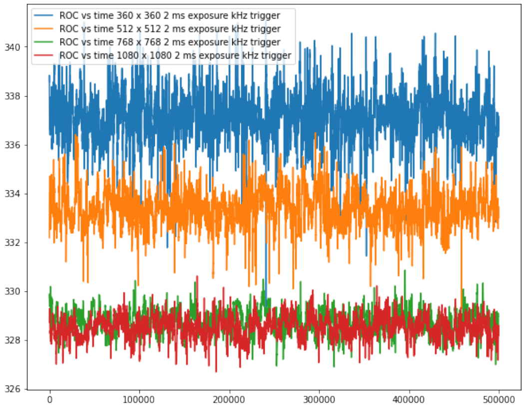
Dataset includes 30k shots across separate runs from a HASO4 Wavefront sensor as the ground truth

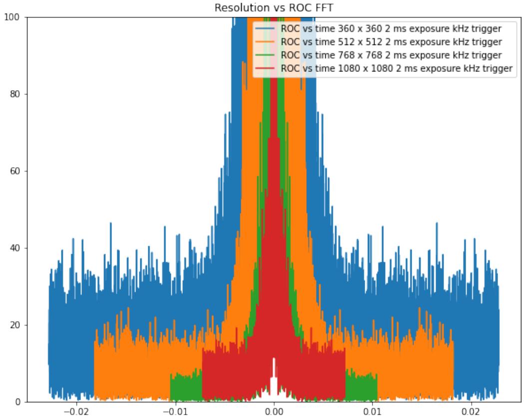


Thorlabs WFS20-7AR Wavefront Sensor



Systematic Measurement Effects





Thorlabs WFS20 image resolution setting vs calculated Radius of Curvature Left: Calculated ROC in time domain, Right: FFT of calculated ROC

So why use vendor tools?

We don't often talk about it in our (scientific-focused) domain, but it helps to go back to why machine learning accelerators exist: to either decrease *latency* or *power*[1]

Edge (embedded NPU, FPGA, or bluefield-style) vs 'edge' (deployed containers) vs edge (centralized data streams)

Don't open source solutions exit? – this is a misnomer given that the cost of development comes from somewhere. Open source does not mean free

[1] Watt per bit, Watt per computation

‘Mainstream’ Accelerators

Defining ML accelerators (by order of specialization):

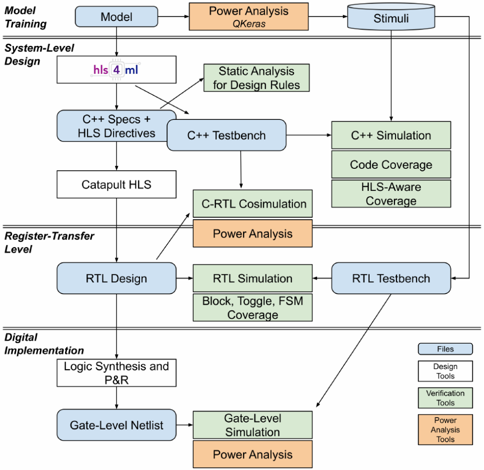
- CPUs (Arm NN, Intel optimizers)
- GPUs (Zink, OpenCL, CUDA, ROCm)
- TPUs/NN/array processing cores
- Reconfigurable AI accelerators/CPUs (graphcore, Xilinx AI Engine, Cerberas)
- In-fabric programmable hardware (custom processing)
- Custom processors (e.g., IBM accelerated application processors)
- ASIC dedicated hardware (asic-design-complexity)

Complexity of Implementation

CPU

```
x = model(data)
```

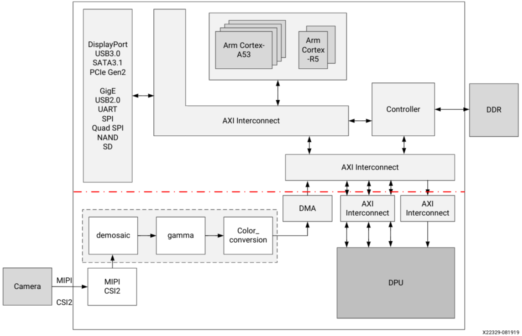
ASIC



Deployment at LBNL: Toolchains in use

MLops	Model training, deployment, optimization, quantization
FPGA	Co-accelerator lives in FPGA fabric; minimizes power, easy hardware interfacing
Labview	Thorlabs uses NI CVI vision drivers
Operating System	Cameras and motion stages (RS232, RS422, GPIO)
Python	Camera interface, user application, and communication
Control System	EPICS or GEECS (BELLA LabVIEW system)

Example Xilinx Implementation



Device	DPUCZDX8G Configuration	Frequency (MHz)	Peak Theoretical Performance (GOPS)
Z7020	B1152x1	200	230
ZU2	B1152x1	370	426
ZU3	B2304x1	370	852
ZU5	B4096x1	350	1400
ZU7EV[2]	B4096x2	330	2700
ZU9	B4096x3	333	4100

[2]Used in ZCU104

Model Development Process

- Experimentation
  - PyTorch, PyTorch Lightning, Tensorflow, Keras
- Optimization / Quantization
- Compilation
- Runtimes
- Performance Testing

Model Development Process: Steps

- Model was developed in PyTorch or Tensorflow
- Optimization and pruning occurs in the vendor toolkit
- Model can be exported to compliant format (i.e., PyTorch, ONNX)
- Xilix Vitis AI Docker container provides model conversion, quantization, and compilation tooling
- Compiled model is loaded on to ZCU104 with bitstream including two DPUs
- Model is loaded with Xilinx utilities in image
- Application will interface with DPU and camera data

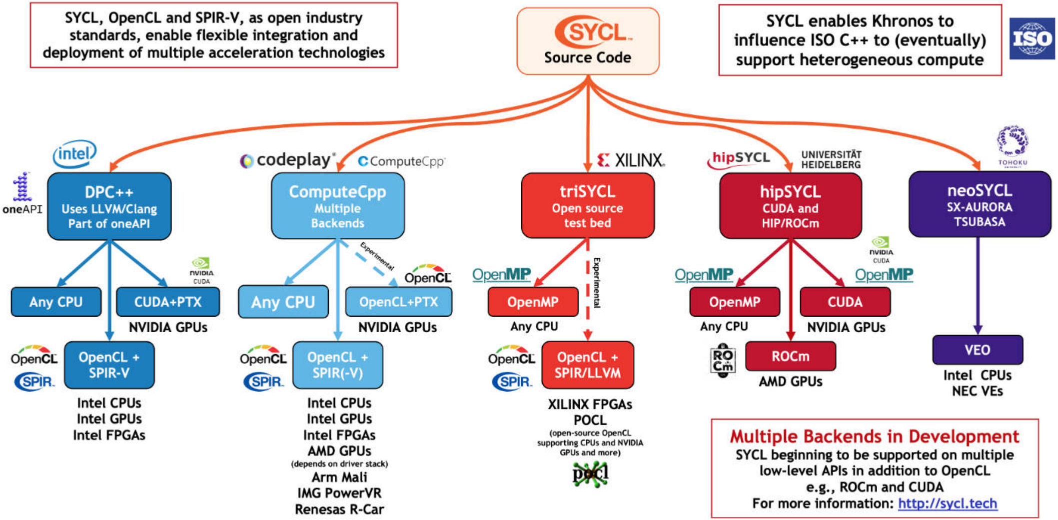
Model Development Process: Optimization Tooling

- Tensorflow (and Tensorflow Lite)
  - tensorflow_model_optimization (separate package)
  - https://www.tensorflow.org/model_optimization/guide/pruning/comprehensive_guide
- Pytorch
  - torch.nn.utils.prune
  - https://pytorch.org/tutorials/intermediate/pruning_tutorial.html#global-pruning
- Qkeras – Keras extension for quantization
- Manufacturer-specific tooling (e.g., Xilinx Optimizer, Intel OpenVINO)

Model Development Process: Compilation

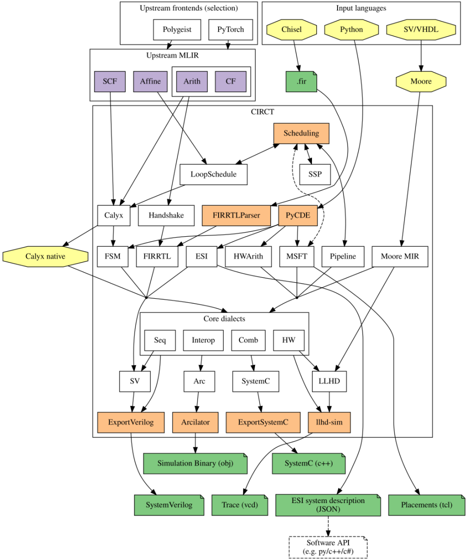
- For co-accelerators (DPU, TPU, GPU)
  - Necessary to compile model to an assembly language and operations that the hardware understands
- Often requires specific input artifacts and generates custom build artifacts
  - Tensorflow Lite (TPU)
  - SYCL/OpenCL programs
  - Xilinx .xmodel files
  - Device bitstreams
  - GPU-specific architectures (CUDA, AMD ROCm)
  - ONNXruntime
    - Microsoft-developed framework
    - Almost every toolchain now takes in ONNX models and the ONNXruntime itself can run models on almost any device (ONNXruntime)
    - CPU, OpenCL devices, GPU, TPU, FPGA accelerators (DPU), OpenVINO (Intel)

Shared Languages and Compilers



Shared Languages and Compilers

Significant development has been occurring within the LLVM ecosystem to develop intermediate representations (IRs) that can map to hardware



Model Development Process: Runtimes

- SYCL
  - Necessary to compile model to an assembly language and operations that the hardware understands
- ONNXruntime
  - Microsoft-developed framework
  - Almost every toolchain now takes in ONNX models and the ONNXruntime itself can run models on almost any device (ONNXruntime)
  - CPU, OpenCL devices, GPU, TPU, FPGA accelerators (DPU), OpenVINO (Intel)

Xilinx DPU on ZCU104 Workflow

1. Develop and save model
2. Build python script to run in Vitis AI docker container to quantize model and save the model
    - This script should also check performance for a production deployment as performance IS lost in the quantization process
    - Quantization is necessary to run a model using integer types instead of float types, due to accelerator data format requirements
3. Compile the model in to the Xilinx DPU .xmodel format
4. Deploy the xmodel, run script, and dataset to the device
5. Run the performance test

Xilinx DPU on ZCU104 Workflow

Requires use of several files chained together to create, compile, optimize, and deploy a model

Stage 1

ffnn.py → docker_run.py → runme.sh → runme_tf2.sh → compile_for_zcu104.sh

Stage 2

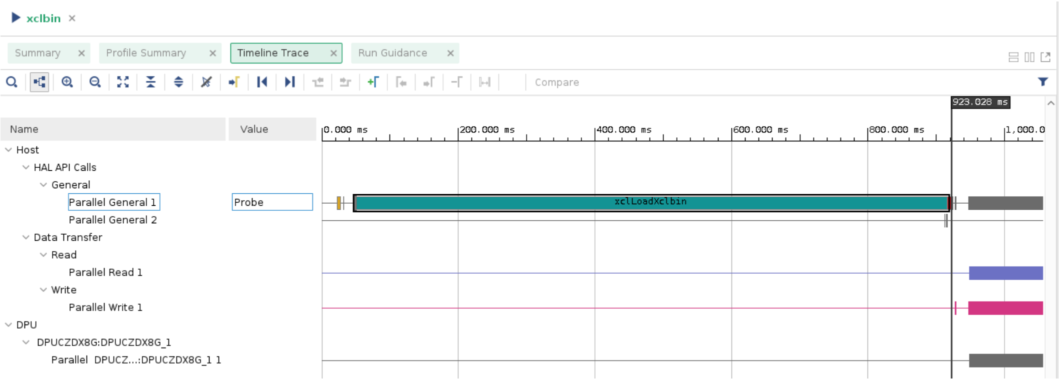
deploy.sh → radiasoft.py

Model Performance

```
(image) xilinx-zcu104-2021_1:~# vaitrace -p python /tmp/radiasoft.py -d /tmp/data.hdf5 -m /tmp/radiasoft
      .xmodel
input_fixpos=7 input_scale=128
SPS=5047.43, total samples = 360.00 , time=0.071323 seconds
Closing remaining open files:/tmp/data.hdf5...done
```

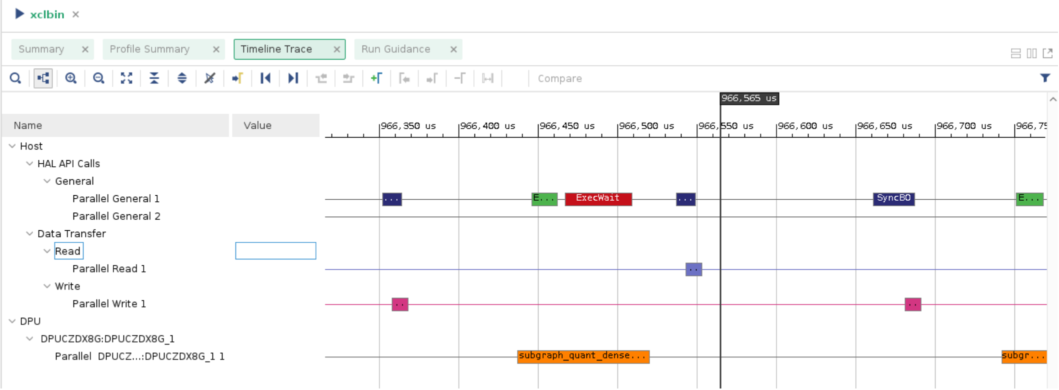
Listing 1: Tracing Performance

Xilinx Performance Tracing



Xilinx Performance Tracing

Xilinx Performance Tracing



Final Thoughts

There's almost no way to get around using vendor tools when deploying ML

Open source does not mean not proprietary

Integrating tools for custom accelerators requires some thought as to how it would map to infrastructure and support needs

References

1. ARM ML: https://www.mlplatform.org/
2. Google ecosystem: Tensorflow, TFX, Tensorflow Lite, and coral.ai
3. ONNXruntime: https://onnxruntime.ai/
4. Vitis AI: https://github.com/Xilinx/Vitis-AI

Thank you!

Disclaimer

Code

runme.sh

```
#!/bin/bash

# Activate conda environment
conda activate vitis-ai-tensorflow2

# Install tables
pip install tables

# Run model quantizer
python3 runme_tf2.py

# Compile the model
./compile_for_zcu104.sh

# Generate graph
xir svg build/compiled_model_zcu104/radiasoft.xmodel build/compiled_model_zcu104/out.svg
```

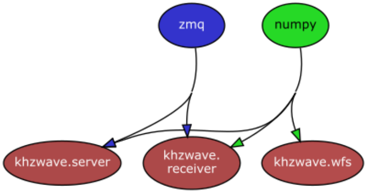
Code

runme_tf2.py

```python
#!/usr/bin/env python3

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['VAI_LOG_LEVEL']='-1'

import tensorflow as tf
import pandas as pd

tf.keras.backend.set_learning_phase(0)

# Load dataset
storer = pd.HDFStore("data.hdf5", mode="r")
x_train = storer.get("/input_train").to_numpy()
x_val = storer.get("/input_validate").to_numpy()
y_train = storer.get("/output_train").to_numpy()
y_val = storer.get("/output_validate").to_numpy()
storer.close()

# Create tensorflow dataset for ease of management
ds = tf.data.Dataset.from_tensor_slices((x_val,
     y_val))
ds = ds.batch(5, drop_remainder=True)
```

```python
# Load model
mm = tf.keras.models.load_model("model_tuned/")
mm.summary()
model = mm

# Quantize model
from tensorflow_model_optimization.quantization.
     keras import vitis_quantize
quantizer = vitis_quantize.VitisQuantizer(model)
quantized_model = quantizer.quantize_model(
     calib_dataset=ds)

# Quantization-aware training
quantized_model.save('quantized_model.h5')
```

Python Camera Driver



Index

Super-module

- khzwave

Classes

- **Wfs20Instrument**
  - DEVOFFSET
  - MAXLINE
  - STATUS
  - buildSpotfieldImageArray
  - calcBeamCentroidDia
  - calcBeamInformation
  - calcSpotToReferenceDeviations
  - calcZernikeLSF
  - close
  - configureCamera
  - displaySpotDeviations
  - displaySpotIntensities
  - getDriverRevision
  - getHighSpeedWindows
  - getInstrumentCount
  - getInstrumentInfo
  - getInstrumentListInfo
  - getLine
  - getLineView
  - getMLAData
  - getMlaCount
  - getSpotCentroids
  - getSpotDeviations
  - getSpotIntensities
  - getSpotfieldImage
  - getStatus
  - handleError
  - initInstrument
  - printStatus
  - reset
  - selfTest
  - setExposureGain
  - setHighSpeedMode
  - setReferencePupils
  - takeSpotfieldImage

Module `khzwave.wfs`

Interface to WFS driver based on sample applications (Windows-only)

This code includes several major classes, of which two are used only by the WfsInstrument class for data storage (WfsImage, WfsInfo)

WfsInfo : data required for managing instrument settings and parameters WfsImage : data buffers and image information

The other classes in this file are used to provide a direct interface to the Thorlabs NI CVI camera driver and to wrap the functions themselves with Python methods

WfsInterface : interface to driver Wfs20Instrument : Instrument management and interaction methods

▸ EXPAND SOURCE CODE

Classes

```
class Wfs20Instrument (dll, *args, instrIdx=0, mlaIdx=0, doInit=True, ignoreErrors=False,
        **kwargs)
```

Instrument management class

Management methods and interface to get data and configure instrument

Initializes Wfs20Instrument

Args

`dll` : `WfsInterface`
interface to driver

`instrIdx` : `int`, optional
index of which instrument to use. Defaults to 0.

`mlaIdx` : `int`, optional
index of which MLA array to use. Defaults to 0.

`doInit` : `bool`, optional
perform initialization of camera. Defaults to True.

`ignoreErrors` : `bool`, optional
disables printing from handleError method. Defaults to False.

Raises

`RuntimeError`