Real Time Server Monitoring and Fixed Point Inference in FPGA

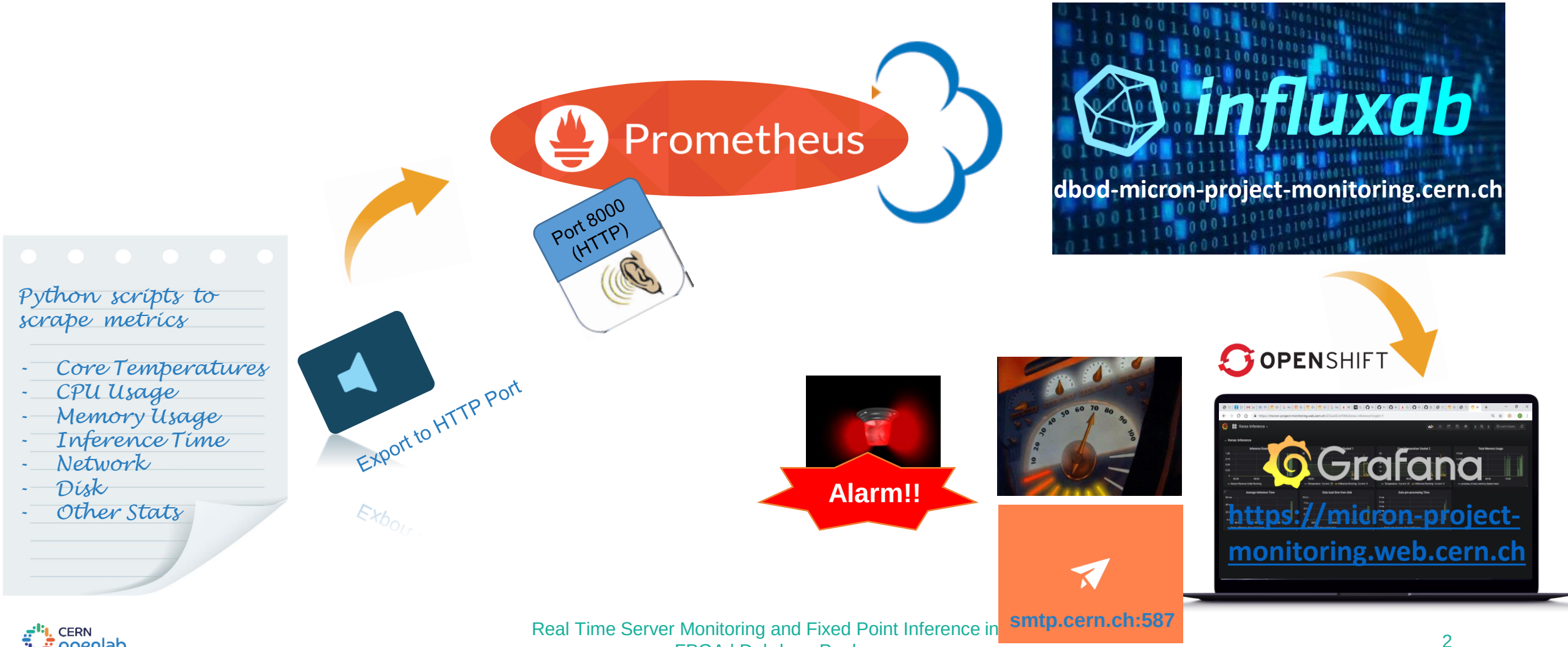
CERN Openlab Lightning Talk

Debdeep Paul

Supervisor :
Paola Sala

13/08/2019

Server Monitoring Architecture

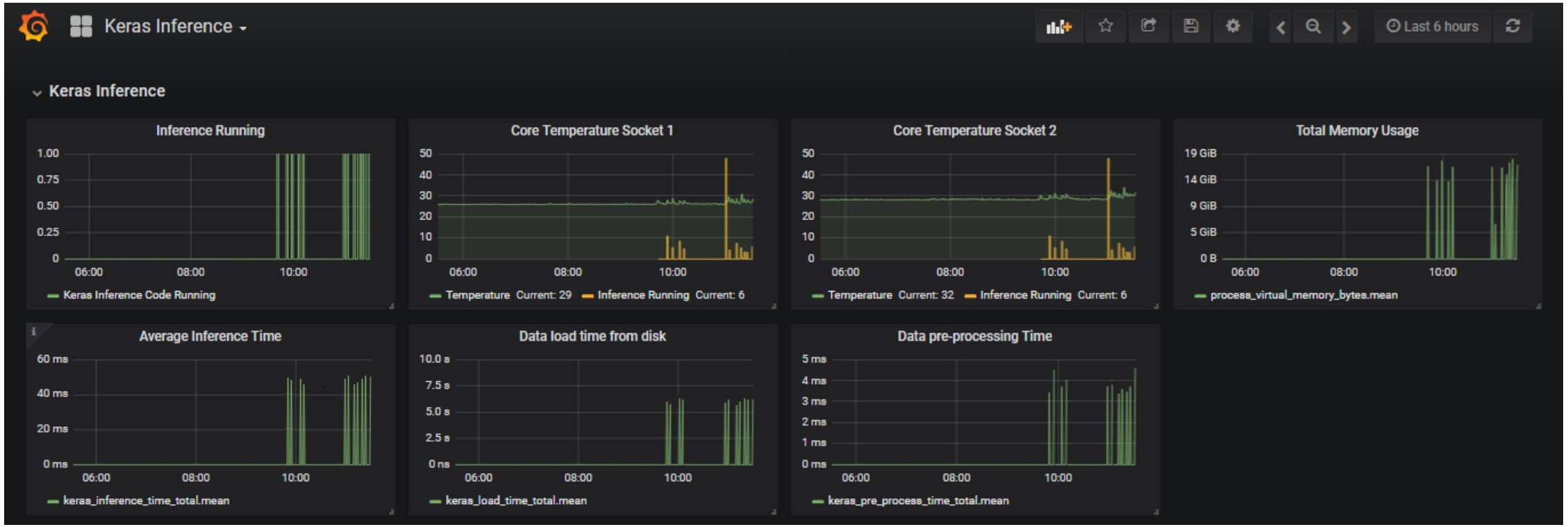


Server Monitoring Home Dashboard

- ✓ CPU
- ✓ Memory
- ✓ Disk
- ✓ Network
- ✓ Temperature
- ✓ Power

Temperature Monitoring Dashboard

CNN Inference Dashboard



Inference Workflow

FPGA vs CPU

Compared latency of CPU and FPGA :

CPU

========== **Timing Benchmark** ==========
Average inference time for a single image : 49.52706972757975 ms
Time taken to load model from disk : 2.501638174057007 s
Time taken for image pre-processing : 3.3588409423828125 ms
==========

FPGA

========== **Timing Benchmark** ==========
Average inference time for a single image : 190.1567 ms
Time taken to load input to main memory: 0.4680 ms
Time taken for converting float to int : 12.1521 ms
==========

- Inference time of CPU < FPGA : Because CPU used batch processing.
- We concatenate the entire 150 images into a single matrix; So, matrix multiplication becomes a lot faster.
- This is not the case with the FPGA. FPGA process 1 image at a time.

✓ Reducing the batch size to 1 in CPU gave similar timing results.

Working with Fixed Point Numbers

The Overflow problem

```
WARNING: overflows in node 18: 4.18 % (1340/32042)
WARNING: overflows in node 19: 0.09 % (49/57330)
WARNING: overflows in node 21: 1.98 % (503/25354)
WARNING: overflows in node 23: 14.40 % (3479/24160)
WARNING: overflows in node 24: 14.78 % (4477/30287)
WARNING: overflows in node 25: 74.77 % (15950/21332)
WARNING: overflows in node 26: 17.28 % (5843/33806)
WARNING: overflows in node 27: 13.11 % (5290/40341)
WARNING: overflows in node 28: 57.73 % (11732/20321)
WARNING: overflows in node 31: 33.33 % (5/15)
```

Q8.8
Fixed Point Format
in FPGA

```
type conv id=18 in=(H32,W32,P256) out=(H32,W32,P256) k=1x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 18->19
type conv id=19 in=(H32,W32,P256) out=(H32,W32,P256) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 19->20
type resconv id=20 in=(H32,W32,P256) out=(H32,W32,P256) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 20,18->21
type conv id=21 in=(H32,W32,P256) out=(H32,W32,P256) k=1x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 21->22
type conv id=22 in=(H32,W32,P256) out=(H16,W16,P512) k=1x1 stride=2x2 dilation=1x1 pad=(0,0,0,0) 21->23
type conv id=23 in=(H32,W32,P256) out=(H16,W16,P512) k=3x3 stride=2x2 dilation=1x1 pad=(0,1,0,1) 22->24
type resconv id=24 in=(H16,W16,P512) out=(H16,W16,P512) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 24,23->25
type conv id=25 in=(H16,W16,P512) out=(H16,W16,P512) k=1x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 25->26
type conv id=26 in=(H16,W16,P512) out=(H16,W16,P512) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 26->27
type resconv id=27 in=(H16,W16,P512) out=(H16,W16,P512) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 27,25->28
type conv id=28 in=(H16,W16,P512) out=(H16,W16,P512) k=1x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 28->29
type intlv id=29 in=(H16,W16,P512) out=(H16,W16,P512) 29->30
type avgpool id=30 in=(H16,W16,P512) out=(H1,W1,P512) k=16x16 stride=1x1 pad=(0,0,0,0) ceil=0 30->31
type conv id=31 in=(H4,W1,P128) out=(H1,W1,P13) k=4x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 31->32
type softmax id=32 in=(H1,W1,P13) out=(H1,W1,P13) 32->33
```

- ✓ The overflow keeps accumulating in each layer starting from Node 18 (Convolution Layer).
- ✓ Max. overflow at Node:25 (Convolution Layer) – 74.77 %
- ✓ Training done in keras using single precision floating point numbers (32 bit)

Possible Solutions

- **Normalization of the Images –** Already implemented; still there is overflow ✓ *Done*
- **Use Dynamic Quantization -** Both layer-wise and different quantization schemes for weights ϵ [-1,1] and activations (usually large mantissa) ✗ *Limitation of the Micron Library*
- **Training in Fixed Points –** Quantized both Weights and Activations during training itself to prevent overflow during inference on FPGA. ✓ *Tried*
- **Change Activation Function –** Use tanh or sigmoid instead of ReLU after the layers that have values overflowing. ? *have t o try; ReLU is more popular*

Fixed Point Training

Using Ristretto

Ristretto is an automated CNN-approximation tool which condenses 32-bit floating point networks.

- ✓ An extension of **Caffe** and allows to test, train and fine-tune networks with limited numerical precision.
- ✓ Trained smaller Networks like LeNet and Cifar-10 to get familiar with the workflow of Ristretto: (q8.8)

CNN Architectures	Base Line Accuracy	Fixed Point Accuracy (Training)	Fixed Point Accuracy (Inference)
LeNet (MNIST)	98.76 %	98.76 %	97.61 %
Cifar-10 Full (Cifar-10)	81.62 %	80.84 %	79.26 %

- ✓ Problems with our network:
  - ✓ We used batch normalization layers after each layer to bring the activation values within -1<0<1. Currently, Ristretto doesn't support batch normalization.
  - ✓ Removing the layers and changing the network degraded accuracy steeply. So, the proposed solution failed.

Using qpytorch

QPyTorch is a low-precision arithmetic simulation package in PyTorch.

- ✓ Converted Model to Pytorch Framework.
- ✓ Trained VGG-16 with Cifar-10 dataset to test the working of the library.

CNN Architectures	Base Line Accuracy	Fixed Point Accuracy (Training)
VGG-16 (Cifar-10)	92.15 %	92.04 %

- ✓ Script to convert ResNet-18 model with the Dune dataset done. Currently training.

❖ Expecting some good results !!

FUTURE WORK

- Do FPGA Monitoring as well
- Test the quantized network on the entire DUNE Dataset once we get satisfactory results.

Thank You !

QUESTIONS?

debdeep.paul@cern.ch

pauldebdeep2009@gmail.com

www.linkedin.com/in/debdeep-paul-iit2019