Contribution ID: **15** Type: **not specified**

Quantized ONNX (QONNX)

Monday, 3 October 2022 14:15 (15 minutes)

One of the products of the cooperation between the hls4ml and FINN groups is Quantized ONNX (QONNX), a simple but flexible method to represent uniform quantization in ONNX. Its goal is to provide a high-level representation that can be targeted by training frameworks while minimizing reliance on implementation-specific details. It should also be lightweight, only adding a small number of operators. QONNX accomplishes this by being in the fused quantize-dequantize (QDQ) style. The main operator is the Quant operator, which takes a bitwidth, scale, and zero-offset to quantize an input tensor and then immediately dequantizes it, undoing the scale and zero offset. The resulting values are (quantized) floating point numbers, which can be used by standard ONNX operators. There is also a BipolarQuant operator, which is like the regular Quant operator but specialized for binary quantization. Finally there is a Trunc operator to truncate the least significant bits. Currently Brevitas, a PyTorch research library for quantization-aware training (QAT), and QKeras, a Keras library for QAT, can produce QONNX. HAWQ support is being added, and is the focus of a separate abstract.

The FINN and hls4ml groups also worked on a common set of utilities to ease the ingestion of QONNX by the FINN and hls4ml software. These utilities simplify the ONNX graphs by doing such things as folding constants, inferring dimensions, making sure nodes are named—commonly referred to as cleaning. FINN and hls4ml also prefer convolution data to be in a channels-last format, so we have a common pass to convert the ONNX graphs to a channels-last format using custom operators. We also have some common optimizers to, for example, change Gemm operators to lower level MatMul and Add operators so that FINN and hls4ml do not need to handle Gemm explicitly.

We will also present how hls4ml ingests QONNX. Given the high-level nature of QONNX, a direct implementation, dequantizing right after quantizing, does not map well to hardware. Instead, hls4ml makes use of optimizers to convert the abstract graph to something that can be more easily implemented on an FPGA or ASIC. In particular, the scale and zero-point in a quantization and in dequantization, if not one and zero respectively, are logically stripped from the quantization operation, resulting in three operations: scale and offset, quantization, and unscale and de-offset. The unscaling can then often be propagated down across linear operations like matrix multiplies or convolutions, to produce quantized dense or convolution layers. As an optimization, for power-of-two scales and zero offsets, we can offload the scale propagation to the HLS compiler by using fixed precision numbers, and for quantized constant weights, we can merge the scale/offset and quantization into the weights, only leaving an unscale and de-offset node if needed.

We also introduce a QONNX model zoo to share quantized neural networks in the QONNX format.

Primary authors: PAPPALARDO, Alessandro (AMD Adaptive and Embedded Computing Group (AECG) Labs); HAWKS, Benjamin (Fermi National Accelerator Lab); BORRAS, Hendrik (Uni Heidelberg); DUARTE, Javier Mauricio (Univ. of California San Diego (US)); MITREVSKI, Jovan (Fermi National Accelerator Lab. (US)); MUHIZI, Jules (Fermilab/Harvard University); TRAHMS, Matthew (UW ACME Lab); BLOTT, Michaela (AMD Adaptive and Embedded Computing Group (AECG) Labs); TRAN, Nhan (Fermi National Accelerator Lab. (US)); HAUCK, Scott; HSU, Shih-Chieh (University of Washington Seattle (US)); SUMMERS, Sioni Paris (CERN); LONCAR, Vladimir; UMUROGLU, Yaman

Presenter: MITREVSKI, Jovan (Fermi National Accelerator Lab. (US))

Session Classification: Contributed Talks