Work-in-Progress – A Vertically Integrated Framework to Deploy Deep Neural Networks on Extreme Edge Devices

Francesco Conti[1,2], Alessio Burrello[1], Angelo Garofalo[1],
Davide Rossi[1] and Luca Benini[1,2]

[1] Energy-Efficient Embedded Systems Laboratory, University of Bologna, Bologna, Italy
[2] Integrated Systems Laboratory, ETH Zürich, Zürich, Switzerland
{f.conti,alessio.burrello,angelo.garofalo,davide.rossi,luca.benini}@unibo.it

Abstract. Running state-of-the-art Deep Neural Network (DNN) models on top of a cheap and low-power microcontroller can be considered the *holy grail* of current research on edge embedded devices. This requires beating fundamental obstacles: 1) achieving a sufficient compute throughput within the allotted power envelope; 2) squeezing the DNN model footprint to better fit within restricted (< 1MB) on-chip memory resources; 3) if the model still requires off-chip memory, minimizing its required memory bandwidth. In this Work-in-Progress paper, we describe the vertical integration framework we are designing to deploy DNNs on top of SoC's based on the PULP (Parallel Ultra-Low-Power) platform, from a DNN specification in an open-source framework (PyTorch) down to execution on optimized DSPs and hardware accelerators, passing through memory-aware quantization and tiling.

Keywords: Extreme Edge Computing, Deep Neural Networks, PULP.

1 Introduction and Target Architecture

Future devices operating at the extreme edge of the Internet-of-Things (IoT) will have to deliver high performance within a very low power envelope to run data analytics pipelines directly on-device, avoiding expensive communication of raw data. This kind of sophisticated behavior is considered necessary for the next generation of biomedical devices, autonomous insect-sized drones, cheap smart sensors for structural monitoring, and many other applications. Running state-of-the-art Deep Neural Networks (DNNs) such as MobileNet [1] or EfficientNet [2] can, therefore, be considered the *holy grail* of current research on extreme edge devices. Due to the staggering limitations of these platforms, fundamental obstacles have to be beaten before the execution of Deep Neural Networks on such platforms is enabled. Achieve a good peak compute throughput (> 1 GMAC/s) within a restricted amount power envelope (< 100 mW at peak) is a necessary condition, but it is not sufficient: weight and activation data have to be fit within the on-chip memory hierarchy [3] or brought from off-chip memory at a high enough bandwidth not to hinder computational efficiency.

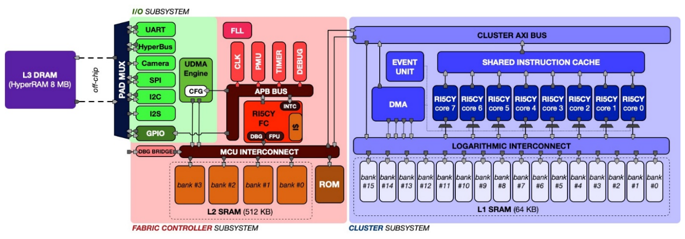

Fig. 1. PULP architecture template

Among the various architectural templates targeting extreme edge data analytics, tightly-coupled clusters of in-order cores have attracted attention as they offer significantly improved performance and energy-efficiency with respect to conventional single-core microcontrollers without the drop-in flexibility typical of application-specific accelerators. The PULP platform (http://pulp-platform.org) is an embodiment of this template combining a single-core microcontroller unit with a cluster of 8 RISC-V cores [4] with ISA optimizations dedicated to signal processing and DNN inference [5]. The template of the PULP platform is schematically shown in **Fig. 1**. The cluster cores work concurrently on a small high-bandwidth L1 scratchpad (64-128kB, up to 30 GB/s at 250 MHz), while the microcontroller hosts a larger but slower L2 (512kB, 1.5 GB/s at 100 MHz). PULP-based platforms have been demonstrated to achieve a performance of ~1GMAC/s with 8-bit data using pure software [6], the minimum to support the real-time execution of "meaningful" real-world DNNs. Further improvements can be achieved by targeting low-precision data representation in Quantized Neural Networks (QNNs) [3], which, however, are known to require specialized tools for retraining and/or data-free quantization [7] and unique techniques for deployment.

In this Work-in-Progress contribution, we highlight our efforts towards the construction of a vertically integrated framework to deploy real-world Deep Neural Networks on top of an extreme edge node based on PULP, starting directly from a DNN trained in a standard framework (PyTorch) and performing quantization (with/without retraining), export of deployable weights, automatic memory-aware tiling and generation of source code optimized to used the PULP ISA to achieve high throughput and energy efficiency. We share preliminary results showing 1) the generation of a hardware-equivalent model for a real-world DNN topology (MobileNet), achieving 69% with 8-bit weights and activations or 65.5% with 6-bit weights and 4-bit activations.

2 Deployment Framework

2.1 Quantization and Graph Transform

The first step of the deployment procedure concerns the transformation of an input Deep Neural Network, selected and pretrained for a specific task such as object detection or image classification. State-of-the-Art Deep Neural Network frameworks such as PyTorch and TensorFlow are primarily targeted at the training of *float32* (i.e., full precision) models, although they also typically support "fake-quantized" training and export of 8-bit models. To fully exploit the capabilities of the underlying PULP platform, which includes extensions to accelerate also low-precision and mixed-precision networks, we developed a custom library for layer-wise quantization called *NEMO* (Neural Minimizer for pytOrch) capable of 1) fake-quantized retraining of Deep Neural Networks, based on an extension of the PACT model [8], and 2) progressively transforming the network from the *Quantized-Fake* (QF) representation to a more realistic representation where all nodes in the network are replaced with entirely quantized nodes (*Quantized-Deployable* or QD) and, finally, to a version where all tensors are replaced by integer equivalents (*Integer-Deployable* or ID). Differently from most state-of-the-art tools, the *NEMO* transformations target not only 8-bit deployment but also lower (e.g., 4-bit and 2-bit) and mixed-precision deployment.

The *NEMO* QF retraining exploits the simple heuristic observation that a "good region" where to look for optimally quantized DNNs is in the vicinity of a full-precision or more loosely quantized solution. The backend approximation methodology utilized for quantization is a variant of PACT [8], relying on linear quantization of both weights (asymmetric signed) and activations (unsigned). The tool applies a "local search" heuristic, which progressively relaxes the value of the precision ε (i.e., the minimum representable value) when the total loss/accuracy is enough, alternating precision relaxation with "annealing" / retraining on the original dataset. In **Fig. 2**, we show an example of the precision relaxation procedure operated on a 1.0-MobileNet-224 network trained on ImageNet.

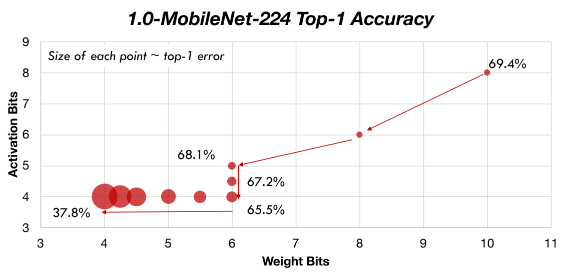

Fig. 2. Example of *NEMO* precision relaxation on 1.0-MobileNet-224 / ImageNet

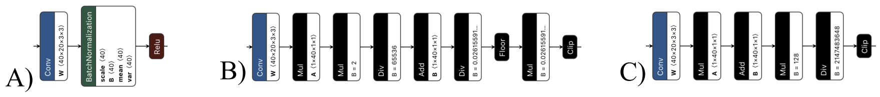

Fig. 3. Example of quantization and transformation from a Quantized-Fake representation (A) to Quantized-Deployable (B) and then to Integer-Deployable (C)

The second operation performed by *NEMO*, transformation to a full integer ID representation can be operated on a QF network obtained by the relaxation procedure described above or directly on a pretrained full-precision. **Fig. 3** shows an example of this transformation operating upon a "super-layer" composed by a linear node followed by batch-normalization and activation. The transformation to a QD version implies changing layers such as batch-norm (untouched in QF representation) so that all tensors involved are *quantized*, i.e., they are representable in the form $\boldsymbol{T} = \boldsymbol{T}_{\boldsymbol{int}} \cdot \varepsilon_{\boldsymbol{T}}$ where $\boldsymbol{T}_{\boldsymbol{int}}$ is an integer-valued tensor and $\varepsilon_{\boldsymbol{T}}$ a real-valued scalar. The example in **Fig. 3B** shows the batch-norm layer being replaced by a quantized version implemented as an affine transformation. By propagating the input ε, *NEMO* computes the ε of all nodes in its internal DNN graph representation, enabling the representation of the network in a format where the ε is left out: the *Integer-Deployable* ID. The transformation from QF to ID representation is typically non-destructive; for example, transforming the 8-bit version of 1.0-MobileNet-224 from QF to ID causes a loss of ~0.3% in top-1 accuracy.

2.2 Memory-Aware Tensor Tiling

Running a DNN model on top of a device optimized for the extreme edge of the IoT requires not only making it "small," as targeted by the *NEMO* tool but also map the DNN so that it maximally exploits the available hardware, in particular in terms of the memory hierarchy. PULP systems lack a coherent hardware cache to save energy at the cost of labor-intensive explicit memory management, making it practically challenging to achieve high bandwidth utilization rates. We automatize this by performing *tiling*, i.e., by dividing the tensors in the work set of each graph in blocks (tiles) that can be operated upon independently from one another.

We developed a tool called *DORY* (Deployment Oriented to memory) [9] to this end. Thanks to the predictable nature of each node's computation, DORY abstracts the optimization of tile sizes as an integer Constraint Programming problem. From a practical viewpoint, *DORY* takes as input the ONNX ID representation generated by *NEMO*, constructs an internal graph of super-nodes (composed of a single linear layer followed by pointwise operations and pooling), and calculates the optimal tiling of one super-node at a time, targeting minimization of the overall layer-wise traffic. We subject this minimization to several constraints:

- the combined size of all tiles, taking into account also double buffering schemes if present, must be smaller than a given budget (e.g., 64 kB for the L1);
- the relationships between weight, input, and output tile dimensions are mandated by the characteristics of the layer (convolutional vs. fully-connected, etc.);
- the tiles should be sized in such a way to provide a well-parallelizable input to the backend library (PULP-NN [6]), maximizing its efficiency.

DORY uses the open-source OR-tools constraint solver from Google AI to derive a solution (in terms of tile sizes) that is compatible with all constraints; using these, it generates the C code of the DNN running on PULP, including data movement and double buffering, according to these tile sizes. **Fig. 4A** shows the L2/L1 data movement scheme targeted by *DORY.*

We verified the efficiency of L2/L1 tiling [9] by comparing the performance of an L2/L1 tiled DNN with an identical one without tiling, executed on the virtual platform simulating actual PULP chips. **Fig. 4B** shows the L2/L1 tiling efficiency of the DNN code produced by *DORY* in the case of a small (142 kB) network. The code generated by *DORY* acts as a very specialized software cache to effectively hide the fact that execution happens on L1 instead of L2 for convolutional layers. In fully-connected layers, whose arithmetic intensity is 100x lower, the "caching" mechanism is less efficient – however, performance is still ~2x that achieved with direct execution on the L2 memory. Overall, the execution of this small network with tiling consumes 3.2x less time and 1.9x less energy than direct L2 execution.

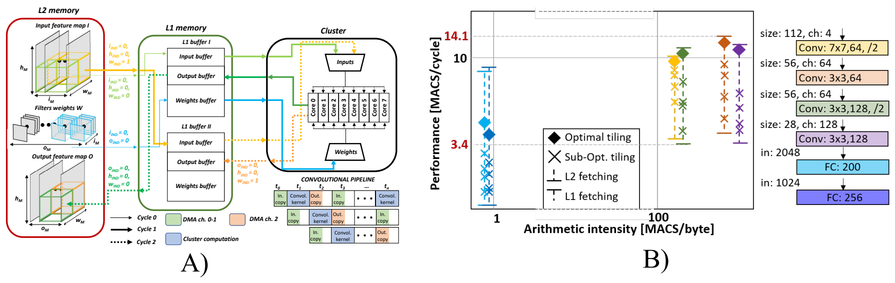

Fig. 4. A) DORY memory tiling scheme between L2 and L1; B) example performance [9]

At the time of this writing, full L3/L2 tiling (with activation data movement) is not yet entirely supported; therefore, *DORY* organizes execution layer-wise and assumes that activations never have to be moved between on- and off-chip, while weights for the following layer can be fully or partially preloaded while the cluster is performing the current layer's execution. Using a PULP-based GreenWaves Technologies GAP8 SoC and a Cypress 8MB HyperRAM as testbench, we measured an L3/L2 bandwidth of 180 MB/s at the peak, which is sufficient to support real-time computation for a real-world, non-toy network.

References

[1] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *ArXiv170404861 Cs*, Apr. 2017.

[2] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *ArXiv190511946 Cs Stat*, May 2019.

[3] M. Rusci, A. Capotondi, and L. Benini, "Memory-Driven Mixed Low Precision Quantization For Enabling Deep Network Inference On Microcontrollers," *ArXiv190513082 Cs Stat*, May 2019.

[4] A. Pullini, D. Rossi, I. Loi, A. D. Mauro, and L. Benini, "Mr. Wolf: A 1 GFLOP/s Energy-Proportional Parallel Ultra Low Power SoC for IOT Edge Processing," in *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, 2018, pp. 274–277.

[5] M. Gautschi *et al.*, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.

[6] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating Quantized Neural Networks on Parallel Ultra-Low-Power RISC-V Processors," *ArXiv190811263 Cs*, Aug. 2019.

[7] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling, "Data-Free Quantization Through Weight Equalization and Bias Correction," *ArXiv190604721 Cs Stat*, Sep. 2019.

[8] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized Clipping Activation for Quantized Neural Networks," *ArXiv180506085 Cs*, May 2018.

[9] A. Burrello, F. Conti, A. Garofalo, D. Rossi, and L. Benini, "Work-in-Progress: DORY: Lightweight Memory HierarchyManagement for Deep NN Inference on IoT Endnodes," in *2019 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2019, pp. 1–2.