An On-the-Fly Feature Map Compression Engine for Background Memory Access Cost Reduction in DNN Inference

Georg Rutishauser[1], Lukas Cavigelli[1], and Luca Benini[1,2]

[1] ETH Zürich, 8092 Zürich, Switzerland
[2] University of Bologna, 40136 Bologna, Italy
{georgr, lukasc, lbenini}@iis.ee.ethz.ch

Abstract. Specialized hardware architectures and dedicated accelerators allow the application of Deep Learning directly within sensing nodes. With compute resources highly optimized for energy efficiency, a large part of the power consumption of such devices is caused by transfers of intermediate feature maps to and from large memories. Moreover, a significant share of the silicon area is dedicated to these memories to avoid highly expensive off-chip memory accesses. Extended Bit-Plane Compression (EBPC), a recently proposed compression scheme targeting DNN feature maps, offers an opportunity to increase energy efficiency by reducing both the data transfer volume and the size of large background memories. Besides exhibiting state-of-the-art compression ratios, it also has a small, simple hardware implementation. In post-layout power simulations, we show an energy cost between 0.27 pJ/word and 0.45 pJ/word, 3 orders of magnitude lower than the cost of off-chip memory accesses. It allows for a reduction in off-chip access energy by factors of 2.2× to 4× for MobileNetV2 and VGG16 respectively and can reduce on-chip access energy by up to 45%. We further propose a way to integrate the EBPC hardware blocks, which perform on-the-fly compression and decompression on 8-bit feature map streams, into an embedded ultra-low-power processing system and show how the challenges arising from a variable-length compressed representation can be navigated in this context.

Keywords: Edge AI · Feature Map Compression · Deep Learning · Hardware Acceleration

1 Introduction

Deep Learning has become the dominant technique in computer vision and its applications to other data processing tasks, such as speech recognition and biomedical signal processing, are rapidly gaining prominence. To meet the demand for computational efficiency generated by the rapid proliferation of these algorithms, specialized hardware accelerators are being developed and studied, by academia and industry alike, at an exceptional rate [1, 12, 16, 19, 27, 37]. A trend, commonly summarized under the term "Edge AI", has been to move neural network

inference from remote cloud resources to the source of sensor data in order to improve latency, reduce transmission energy costs and alleviate privacy concerns. Edge devices developed for these applications may rely on low-power general-purpose processing cores [29] or dedicated accelerators [11] to perform the data processing.

Power efficiency is the primary target metric in the design of such devices, and with the elimination of energy-intensive long-range communication, data transfer to and from central memories becomes the main contributor to power consumption [3, 13]. In systems relying on external memory (DRAM), off-chip communication dominates system energy consumption by a wide margin [4]. Disregarding off-chip communication, on-chip (commonly SRAM-based) memories are responsible for a major part - often more than half - of the power consumption [3, 13]. At a given operating frequency, on-chip SRAM power consumption is proportional to the frequency of accesses and static (leakage) power consumption rises with the area allocated to memory macros. Consequently, reducing the number of memory accesses and the size of on-chip memories holds great potential for improving energy efficiency. One natural approach to compressing both model representation and result data is reducing operand precision, which is a subject of intensive investigation in the research community [17, 24, 38, 39].

Another way to reduce memory size and the number of accesses is to compress data located in large, energy-intensive memories and decompress it only at the time of use. Compressing the model size in this way is also common practice, with most works using approaches based on sparsity, which may be induced in the model with negligible accuracy loss by techniques such as weight pruning [20, 23, 25]. With model size shrinking thanks to these methods, intermediate feature maps become a more significant contributor to data transfer volume. However, the compression of feature maps has been less researched, for several reasons: While weight precision can be reduced to extreme degrees while retaining good accuracy [24, 38], reducing activation precision to e.g. ternary representations leads to significant accuracy drops [2, 10]. Furthermore, while static weights can be compressed offline, compressing intermediate feature maps requires custom hardware, and the offline pruning approach to induce sparsity is not applicable to them as they are input-dependent. However, the transfer of intermediate feature maps is responsible for a major part of total data transfer. With recent applications working on high-resolution input data, such as pose estimation [6, 22] and object detection [26, 32], the share of feature map transfers increases even further, making feature map compression even more promising for power efficiency gains. In this work, we present the following contributions:

1) Implementation of a feature map compression and decompression engine [8] in an advanced technology node (GlobalFoundries 22 nm FDX) with post-layout and energy efficiency results;
2) A scheme for the integration of on-the-fly feature map compression into an embedded computing platform, with a detailed discussion of the challenges presented by the compressed representation in this regime and how to address them;

3) An analysis of the potential impact of EBPC integration on the energy cost of on-chip and off-chip data transfers, showing a energy cost compression and decompression between 0.27 pJ and 0.45 pJ per 8-bit word, which enables a reduction in off-chip data transfer energy between 2.2× and 4× - improving on the previous state of the art - and on-chip transfer energy savings of up to 45%.

2 Related Work

A number of works present accelerators for the efficient computation of convolutional layers with sparse inputs, among them Eyeriss v2 [9], SCNN [30] and NullHop [1]. While reducing memory size and bandwidth requirements is an important benefit of the schemes proposed in these works, their focus is mostly on reducing the number of computations by directly using the compressed representations in the compute units to skip multiplications by zero. The compression methods used in these works rely exclusively on data sparsity, which allows them to be applied to both model parameters (which can be sparsified further by applying network pruning) and activations.

We focus on compressing activations with the intent of reducing data transfer volume, an approach that allows the use of a compression method that exploits properties exclusive to feature map data. The approach of compressing only feature maps is shared by the Compressing DMA Engine (cDMA) [33]. It exploits only sparsity in its compression method, by using a zero-value compression (ZVC) scheme, which achieves an average compression ratio of 2.6× on feature maps of set of benchmark networks including VGG16 and SqueezeNet [18]. ZVC-encoded data is represented by a binary mask, indicating for each word whether it is zero or not, and a sequence comprised of the unencoded nonzero values. cDMA targets sparse feature maps in 32-bit precision and provides a hardware solution to increase the effective bandwidth over a PCIe link connecting GPU memory space to the system memory, providing an average performance increase of 32% on virtualized DNNs (vDNN). While cDMA offers attractive advantages for backplane data transfers of full-precision data, the performance of its ZVC scheme does not improve with lower arithmetic precision as employed in state-of-the-art embedded applications. For 8-bit and 16-bit data, it is surpassed by the recently introduced method of Extended Bit-Plane Compression.

3 EBPC: Extended Bit-Plane compression

Extended Bit-Plane Compression (EBPC), first introduced in [7] and extended in [8], combines zero run-length encoding (ZRLE) with bit-plane compression [21], an algorithm originally intended for texture compression. It is a lossless compression algorithm uniquely suited to the compression of neural network feature maps as it exploits not only data sparsity but also its smoothness and achieves state-of-the-art compression rates between 2.2× (MobileNetV2) and 5.1× (AlexNet) on 8-bit data. EBPC encodes an input data stream into two output streams: 1) A

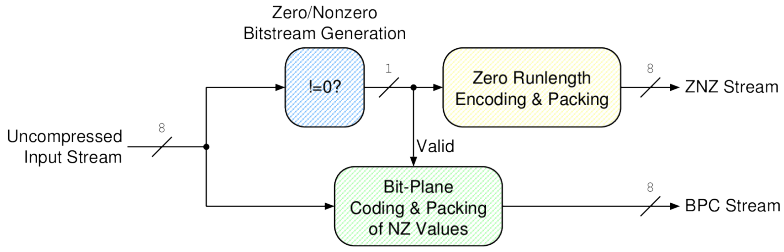

Fig. 1: Extended bit-plane compression for 8-bit data: The ZNZ stream encodes for each input word whether it is zero, the BPC stream encodes the nonzero words.

zero-runlength encoded (ZRLE) zero/nonzero stream (ZNZ) analogous to the binary mask in ZVC, exploiting data sparsity and providing additional compression for longer sequences of zeros, and 2) a BPC stream, generated by applying BPC to the nonzero values of the input stream. Figure 1 shows the process for 8-bit data.

Due to its simplicity and inherent symmetry, the encoder and decoder blocks, which operate on streams of 8-bit data, can both be implemented in hardware with a very small resource footprint of about 3000 gate-equivalents each in UMC 65 nm, giving a single compressor-decompressor pair an area footprint equivalent to that of a 32-bit multiplier. As EBPC is applied only to intermediate feature maps and the feature maps are decompressed before being processed, its use is orthogonal to that of model compression schemes which act on the network weights, such as Deep Compression [14]. The only condition is that feature map data be consumed and produced by the compute units in raw form rather than some compressed representation.

In comparison to the ZVC scheme employed by cDMA, EBPC achieves significantly higher compression ratios for fixed-point data for 8-bit and 16-bit data types, which have been found to provide sufficient arithmetic precision to retain baseline accuracy, i.e., deliver equivalent results to 32-bit floating point data [5]: For VGG16, MobileNetV2 and ResNet34, EBPC achieves average compression ratios of 4×, 2.2× and 2.4× respectively on 8-bit fixed-point data, comparing favorably to the compression ratios of 2.3×, 1.5× and 1.6× attained by ZVC, making it the best-performing lossless compression algorithm in literature for DNN feature maps to the best of our knowledge. More detailed comparisons between EBPC and other compression methods can be found in [8].

3.1 Applications in Computing Systems

There are different goals which can be achieved by the integration of EBPC hardware into an existing system, shown in Figure 2:

1) Memory size reduction: In order to reduce silicon area (in the case of on-chip memory) and power consumption, it is desirable to keep the size of the large

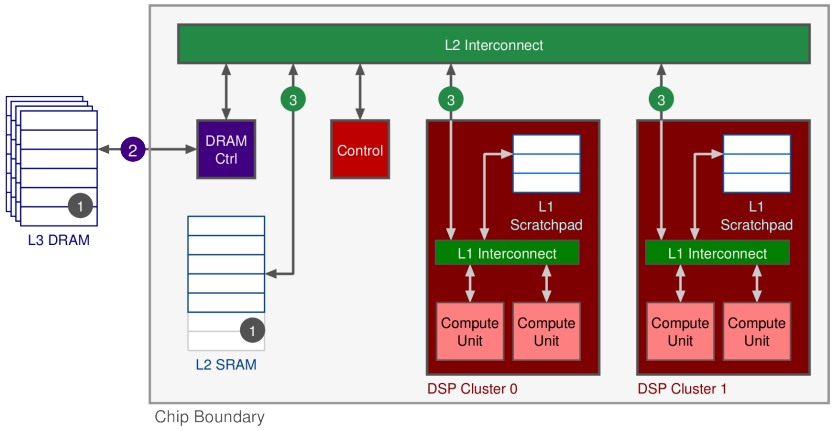

Fig. 2: Opportunities for power efficiency gains from EBPC: 1) Memory size reduction, 2) Off-chip data transfer volume reduction, 3) On-chip data transfer volume reduction

background memory (L2 in Figure 2) small. EBPC lowers the required memory capacity by consistently providing state-of-the-art compression ratios with no accuracy penalty due to its lossless nature: For instance, output feature maps from MobileNetV2's fifth layer, which are the largest in terms of data volume and thus set the lower bound for memory size in layerwise processing, can reliably be compressed by a factor of more than 1.7, theoretically enabling a reduction in L2 memory size by that factor.

2) More efficient use of off-chip bandwidth: Systems which rely on off-chip memory to store intermediate feature maps can benefit from a significant reduction in off-chip data traffic volume or, equivalently, an increase in effective bandwidth. As off-chip communication is typically responsible for a large share of system power consumption [1, 4], this holds major potential for energy efficiency improvements.

3) Reducing on-chip data access energy: While on-chip memory accesses are generally about 2 orders of magnitude more energy-efficient than off-chip accesses, they are nonetheless the main contributor to the power consumption of neural network accelerators [3, 13]. If the added energy cost of compressing and decompressing the feature map data is lower than the energy saved as a result of the reduced memory accesses, EBPC offers a way to reduce system energy consumption.

4 System Integration

Integrating EBPC into an embedded system architecture brings several challenges:

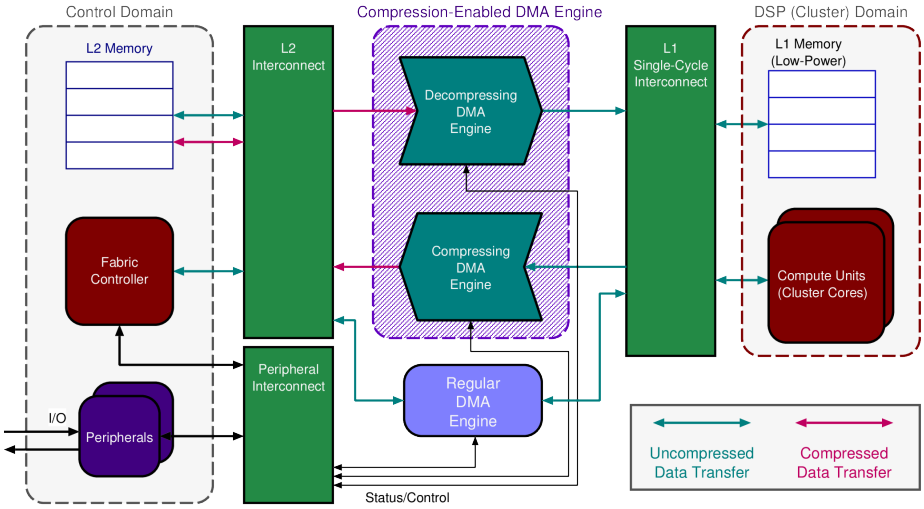

Fig. 3: Proposed System Integration. The extension to the basic system is underlaid in purple.

1. The EBPC-encoded representation of a data stream has a data-dependent length not known ahead of the compression process. The compressed streams in memory must thus be managed actively;
2. Due to its variable-length nature, randomized access into the compressed space is in general not possible;
3. As the encoded representation consists of two separate streams and the compressing/decompressing hardware blocks process only one word of a single stream at a time, memory accesses must be time-multiplexed between the different compressed streams and the respective BPC and ZNZ streams they comprise.

In this section, we will address these challenges as they relate to integrating EBPC into a parallel ultra-low-power (PULP), RISC-V based system architecture based on Mr. Wolf [31]. In such a system, there is a tiered, user-managed memory hierarchy with a large SRAM-based central L2 memory and one or multiple smaller, low-latency and low-power L1 scratchpads based on standard-cell memories (SCM). The system operation is orchestrated by a Fabric Controller (FC), which in Mr. Wolf is implemented as a low-complexity, low-power RISC-V microcontroller. The L2 memory and the FC are located in the *Control Domain*, which contains all the components that make the system work, such as peripherals, clock generation and power management. Intensive data processing is performed by then compute units in the *DSP Domain* (called the *Cluster Domain* in the context of Mr. Wolf and other PULP systems) working on the scratchpad memories, so high-frequency memory accesses happen on low-power memories. In the case of Mr. Wolf, the compute units are implemented as a cluster of high-efficiency RISC-V cores optimized for embedded DSP tasks, but

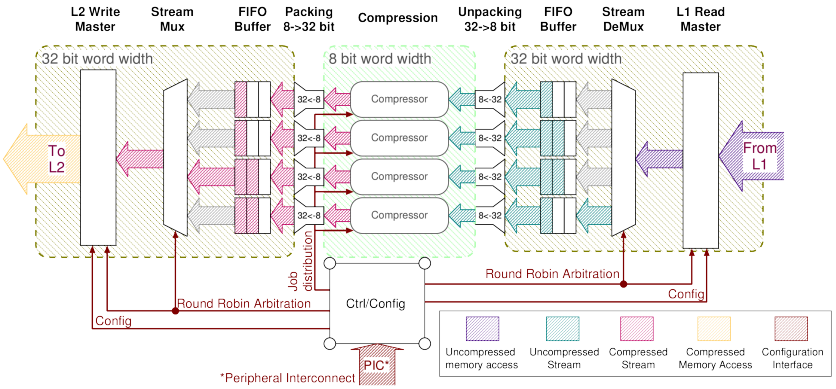

Fig. 4: Compressing DMA Engine

they might also take the form of more specialized accelerators in a generic system. A system overview is shown in Figure 3. Data is transferred between the Control and DSP domains via DMA transfers. Following the pattern laid out by cDMA, we propose a compression-enabled DMA engine to compress (decompress) feature map data being transferred to (from) L2 on-the-fly. A regular DMA controller is retained for transferring data not amenable to compression, and in general, no major system reorganization is needed to integrate the compression functionality. A typical computation flow would then proceed as follows:

1. Compressed data is transferred from L2 to L1 using the proposed DMA engine, which decompresses them on-the-fly;
2. Compute units perform calculations on the uncompressed data in the L1 scratchpad;
3. The uncompressed results are transferred back to L2 and compressed on-the-fly.

As the data in L1 is uncompressed, compute units see the same input data as when using a regular, non-compressing DMA engine and inference time is not affected by our proposed scheme. Likewise, time spent on data transfer is not noticeably impacted as our DMA engine transfers data at a rate of 1 uncompressed word/cycle.

4.1 Compression-Enabled DMA Controller Architecture

The DMA engine has a compressing and a decompressing submodule, both of which are attached to interconnects connecting to the L1 and L2 memories. A block diagram of the compressing (i.e., L1 to L2) DMA engine is shown in Figure 4. As each compressor and decompressor block has a maximum throughput of one uncompressed 8-bit word per cycle and most systems use wider words, multiple streams are processed in parallel. After the start addresses of input and

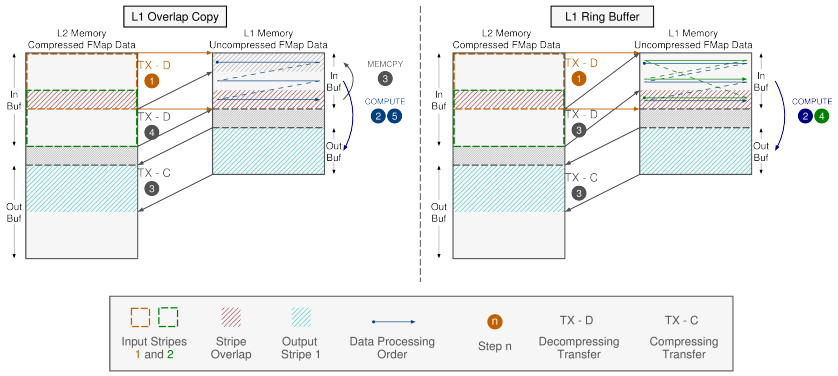

Fig. 5: Dealing with overlap regions: In the "L1 Overlap Copy" scheme, applications or accelerators can assume feature maps stored contiguously but an additional copy operation is required. In the "L1 Ring Buffer" scheme, compute elements must be able to accommodate address wrap-around but no additional operations are required.

output streams and the length of each input stream are configured, the unit starts reading from the configured input stream addresses in short bursts. The data read from memory is fed into a FIFO queue for the corresponding stream, from which the packed words are removed and unpacked into their constituent 8-bit words, which are fed into the (de)compressor unit assigned to the stream. Its output is packed into wide data words again, passes through another FIFO queue and is finally written to the target address in short bursts. The other functionality of the compression-enabled DMA engine is identical to that of a simple memory-to-memory DMA controller.

4.2 Tiling Compressed Feature Maps

Intermediate feature maps of networks that perform complex tasks, e.g. autonomous navigation of nano-drones as in PULP-DroNet [29], may not fit into the limited space available in L1 memory scratchpads. To enable their execution on small embedded systems, a commonly used approach is *tiling* of input or output data into multiple portions (tiles) and processing them individually. Three-dimensional feature map tensors can be tiled in each of their dimensions or any combination thereof. In the following, we will assume tiling in only one dimension. In *channel-wise-tiling*, the tensor is split into multiple sets of complete feature maps. As separate feature maps are compressed separately, channel-wise tiling is natively supported if the individual feature maps are transferred to L1 in their entirety. In *spatial tiling*, the tensor is divided along spatial dimensions. Following [29], we will consider tiling into horizontal stripes. In this case, the EBPC-encoded representation of a block of as two variable-length streams introduces additional sources of complexity relative to the uncompressed case.

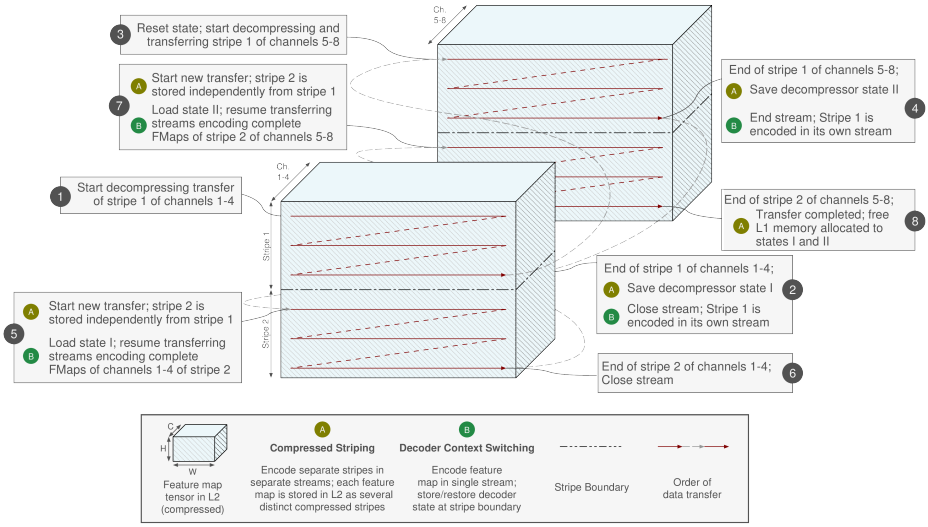

Fig. 6: Approaches for horizontal tiling of compressed feature maps, illustrated for L2-to-L1 transfers: Stripes can be individually compressed in L2 (A), or the feature maps can be encoded in their entirety (B), requiring context switching capability of the decoder.

The first is the overlap between two successive tiles: a convolutional kernel of dimension $k \times k$ requires an overlap of $k-1$ pixels between tiles. When using uncompressed data, a possible (but non-optimal) solution is to transfer each tile, including the overlap with the previous one, from L2 to L1 in its entirety. With EBPC-encoded feature maps however, the beginning of the overlap region cannot be found by simple address manipulation. Thus, the only option is to keep the overlap region from the previous tile in the L1 memory. Two ways to deal with this issue are shown in Figure 5. As the overlap region will be at the end of the memory buffer allocated in L1 for the input feature maps, the overlap region may be either: 1) copied to the beginning of the input buffer (shown on the left), or 2) left in place (as shown on the right). The first option incurs the cost (in energy and available computation time) of an additional L1-to-L1 copy operation but provides the data in a linear organization to the computation implementation, while the second option requires that the application or accelerator performing the calculations is capable of dealing with a ring buffer memory. Note that in [29], as in other optimized implementations, a double buffering scheme is used for the L1 input buffer, which is not shown in Figure 5. The underlying principle still applies, with the input buffer simply having a larger size.

The second consideration is how to tile the compressed feature maps. We propose two approaches, shown in Figure 6 for L2-to-L1 transfers, to this challenge, each with its respective trade-offs: A) *Compressed striping* entails compressing each channel of a computed output tile into a distinct set of BPC and ZNZ

streams. This approach has the advantage of being native to the architecture shown in Figure 3. Its drawbacks are a potential degradation of the compression ratio due to edge effects (which will be more severe for smaller tiles) and an added memory overhead for book-keeping metadata in L2 to keep track of the size and location of the different compressed streams. This approach also has the drawback of imposing a fixed tiling scheme on the compressed data. B) *Decoder context switching*, on the other hand, stores each feature map as a single set of compressed streams in L2 memory. To illustrate the decoder context switching approach, consider the transfer from L2 to L1 memory of $M = 8$ feature maps divided into $N_T = 2$ stripes by a decompressing DMA engine which processes 4 streams in parallel. The engine starts transferring the first 4 feature maps, keeping track of the number of uncompressed words produced. When it reaches the tile boundary, it raises an interrupt to notify a processor core in the DSP domain (or the fabric controller). The internal state (context) of the unit is then stored to L1 memory and the transfer of the second set of 4 channels is initiated. On reaching the tile boundary again, the unit's context is again saved and the feature maps can be processed. Once processing of the first tile is completed, the transfer of the second tile is started by restoring the first saved context. Compressing transfers from L2 to L1 memory are performed analogously.

This approach has the advantage of avoiding the degradation of the compression ratio through edge effects and eliminating the book-keeping overhead inherent to compressed striping. Furthermore, it allows for different (one-dimensional) striping schemes to be applied at will throughout the network, as the feature maps are stored contiguously in L2 and can be split arbitrarily. The drawback is the additional L1 memory capacity required to store one decoder and encoder state (approximately 200 bit each) per feature map. This fixed overhead makes the context switching approach suitable for spatially large tiles with few feature maps, where it would make up only a small portion of the uncompressed data size. Taking as an example the second layer of PULP-DroNet with a feature size of 50×50 pixels and 32 channels, the feature maps would need to be divided into 5 stripes to allow for a partition of a 64 KiB L1 memory space into a double-buffered input and an output buffer while also accommodating the layer's weights. A stripe of one feature map would then take up approximately 500 B, making the relative overhead of the two 200 bit transfer contexts a mere 10%. This approach requires hardware support for saving and restoring the internal state, but this would likely introduce only an insignificant area overhead.

5 Results

5.1 Evaluation Setup

Physical implementation: To evaluate the energy which can be saved on memory accesses by integrating an EBPC-enabled DMA into an integrated processing system, we synthesized the compressor and decompressor hardware designs detailed in [8]. We generated silicon layouts in GlobalFoundries 22 nm FDX, with timing closure at 556 MHz at $V_{DD} = 0.65$ V. Using these implementations, we

performed a post-layout power analysis at the nominal clock speed and supply voltage to determine the average energy cost of (de)compressing a word.

Stimuli data: We simulated the compression and decompression of intermediate feature maps extracted from inferences on the VGG16 [36], ResNet34 [15] and MobileNetV2 [35] network architectures. To generate the feature maps, 4 random images from the ILSVRC2012 [34] validation dataset were run through the network. 5% of the output feature maps at each layer were selected at random and used as stimuli, and the sparsity and resulting compression ratio were confirmed to be in line with the statistics reported in [8]. In addition to the feature map datasets, we also evaluated a sequence of zero words, representing the best case, and a sequence of uniform randomly generated data to represent the worst, uncompressible case. No stalls were introduced at the interfaces, i.e., the compressor and decompressor were operating at full throughput.

Energy cost modelling: Accesses to central memory were modeled as having fixed energy costs of 106 fJ/bit and 112 fJ/bit for read and write accesses respectively. These numbers were obtained from the datasheet of a 4096x32 bit SRAM macrocell for the same process. The SRAM macrocell was characterized at $V_{DD} = 0.65\,\mathrm{V}$ and $V_{CS} = 0.8\,\mathrm{V}$. Having determined for a given dataset the compression ratio C, the write/read memory access energy cost per word E_{acc} and the cost for (de)compression of a word E_{ebpc}, we define the total transfer cost (memory access and (de)compression) E_{xfer} of the compression-enabled transfer of one word between the control and DSP domains and the transfer energy efficiency η_{xfer} as:

$$E_{\mathrm{xfer}} = \frac{1}{C} E_{\mathrm{acc}} + E_{\mathrm{ebpc}} \tag{1}$$

$$\eta_{\mathrm{xfer}} = \frac{E_{\mathrm{acc}}}{E_{\mathrm{xfer}}} \tag{2}$$

We denote the relative transfer efficiency by $\eta_{\mathrm{xfer,S}}$ for on-chip SRAM accesses and by $\eta_{\mathrm{xfer,D}}$ for off-chip DRAM acceses.

5.2 Power Analysis and Discussion

Table 1 shows the results of our evaluation. At less than 60 fJ/bit for 8-bit feature map data, the energy cost of compressing and decompressing the data to be transferred is around 3 orders of magnitude lower than the cost of transferring it to or from an external LPDDR2 DRAM, which has been estimated to consume 40 pJ/bit [28]. Consequently, the energy efficiency gains from storing data off-chip in EBPC-compressed form scale almost linearly with the compression rate and even for a modest compression rate of $C = 1.2$, memory accesses can be reduced by 16.5% under the simplifying assumptions that each result is stored to and loaded from off-chip memory once. For the more typical cases of MobileNetV2 and VGG16, which exhibit average compression ratios of $C = 2.2$

Table 1: Compressing/Decompressing Data Transfer Costs vs. Uncompressed Transfers

Dataset	$\boldsymbol{C}$	**Compressor**		**Decompressor**	
	($\approx \eta_{\text{xfer,D}}$)	E_{ebpc}	$\eta_{\text{xfer,S}}$	E_{ebpc}	$\eta_{\text{xfer,S}}$
VGG16	4.00	271 fJ	1.81	308 fJ	1.69
ResNet34	2.27	408 fJ	1.12	425 fJ	1.10
MobileNetV2	2.11	424 fJ	1.06	452 fJ	1.02
Random	0.74	696 fJ	0.47	581 fJ	0.50
All Zeros	25.60	110 fJ	6.20	149 fJ	4.89

and $C = 4$ respectively, off-chip access energy would be reduced by 55% and 75% respectively. The ZVC scheme used by cDMA has a simpler hardware implementation which can thus be assumed to consume even less energy than our EBPC blocks. However, due to the superior compression ratios achieved by EBPC, the energy efficiency gains for off-chip memory transfers are correspondingly higher.

The energy cost of accessing data stored in on-chip SRAM is much lower, at around 100 fJ/bit for both read and write accesses. Our results show that even for on-chip transfers of feature maps between large central memories and local scratchpads, substantial energy savings are possible: For the well-compressible VGG16 feature maps, write energy can be reduced by 45%, while the read energy decreases by 39%. Networks whose feature maps exhibit lower compression ratios, represented by MobileNetV2 in our analysis, naturally show smaller gains - this is exacerbated by the fact that the energy cost of the compression and decompression processes increases with lower compression ratios. The break-even point in energy consumption for on-chip data transfers in our evaluation setup is around a compression ratio of 2.

6 Conclusion

In this paper, we have shown that EBPC compression and decompression can be implemented at an energy cost of less than 60 fJ/bit. This makes it possible to reduce the cost of off-chip data transfer nearly linearly with the compression ratio, i.e., by factors between 2.2× and 4×, depending on the network in question. We have further demonstrated that EBPC offers a viable method for reducing the energy consumption resulting from the transfer of feature maps between different levels of an on-chip memory hierarchy by up to 45% for read accesses and 39% for write accesses. To realize this potential, we have presented an approach to integrating EBPC into ultra-low-power edge computing systems in the form of a compression-enabled DMA engine capable of tiling compressed feature maps horizontally.

References

1. Aimar, A., Mostafa, H., Calabrese, E., Rios-Navarro, A., Tapiador-Morales, R., Lungu, I.A., Milde, M.B., Corradi, F., Linares-Barranco, A., Liu, S.C., Delbruck, T.: NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps. IEEE Transactions on Neural Networks and Learning Systems **30**(3), 644–656 (2019)
2. Alemdar, H., Leroy, V., Prost-Boucle, A., Petrot, F.: Ternary neural networks for resource-efficient AI applications. In: Proceedings of the International Joint Conference on Neural Networks. vol. 2017-May, pp. 2547–2554 (2017)
3. Andri, R., Cavigelli, L., Rossi, D., Benini, L.: Hyperdrive: A systolically scalable binary-weight CNN Inference Engine for mW IoT End-Nodes. Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI **2018-July**, 509–515 (2018)
4. Andri, R., Cavigelli, L., Rossi, D., Benini, L.: Yoda NN: An architecture for ultralow power binary-weight CNN acceleration. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(1), 48–60 (2018)
5. Banner, R., Hubara, I., Hoffer, E., Soudry, D.: Scalable methods for 8-bit training of neural networks. Advances in Neural Information Processing Systems **2018-Decem**(NeurIPS), 5145–5153 (2018)
6. Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S.E., Sheikh, Y.A.: OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. IEEE Transactions on Pattern Analysis and Machine Intelligence (Xxx), 1–1 (2019)
7. Cavigelli, L., Benini, L.: Extended Bit-Plane Compression for Convolutional Neural Network Accelerators pp. 279–283 (2019)
8. Cavigelli, L., Rutishauser, G., Benini, L.: EBPC: Extended Bit-Plane Compression for Deep Neural Network Inference and Training Accelerators. arXiv preprint arXiv:1908.11645 pp. 1–13 (aug 2019), `http://arxiv.org/abs/1908.11645`
9. Chen, Y.H., Yang, T.J., Emer, J.S., Sze, V.: Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **9**(2), 292–308 (2019)
10. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1 (2016)
11. Google Inc.: Google Edge TPU. `https://cloud.google.com/edge-tpu/`, accessed: 11-16-2019
12. Guo, K., Sui, L., Qiu, J., Yao, S., Han, S., Wang, Y., Yang, H.: From model to FPGA: Software-hardware co-design for efficient neural network acceleration. In: 2016 IEEE Hot Chips 28 Symposium, HCS 2016. pp. 1–27. IEEE (aug 2017)
13. Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A., Dally, W.J.: EIE: Efficient Inference Engine on Compressed Deep Neural Network. Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016 **16**, 243–254 (2016)
14. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In: 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings. pp. 1–14 (2016)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition **2016-Dec**, 770–778 (2016)

16. Huawei Technologies Co., Ltd.: Huawei Kirin 990 Series. https://consumer.huawei.com/en/campaign/kirin-990-series/, accessed: 11-16-2019
17. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. Journal of Machine Learning Research **18**, 1–30 (sep 2017)
18. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-Level Accuracy with 50x fewer Parameters and ¡0.5MB Model Size pp. 1–13 (2016)
19. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.L., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T.V., Gottipati, R., Gulland, W., Hagmann, R., Richard Ho, C., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., Yoon, D.H.: In-datacenter performance analysis of a tensor processing unit. In: Proceedings - International Symposium on Computer Architecture. vol. Part F1286, pp. 1–12. ACM Press, New York, New York, USA (2017)
20. Karnin, E.D.: A Simple Procedure for Pruning Back-Propagation Trained Neural Networks. IEEE Transactions on Neural Networks **1**(2), 239–242 (jun 1990)
21. Kim, J., Sullivan, M., Choukse, E., Erez, M.: Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures. Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016 pp. 329–340 (2016)
22. Kocabas, M., Karagoz, S., Akbas, E.: MultiPoseNet: Fast Multi-Person Pose Estimation Using Pose Residual Network. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **11215 LNCS**, 437–453 (2018)
23. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal Brain Damage (Pruning). Advances in neural information processing systems pp. 598–605 (1990)
24. Li, F., Zhang, B., Liu, B.: Ternary Weight Networks. ArXiv **abs/1605.0** (may 2016), http://arxiv.org/abs/1605.04711
25. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning Filters for Efficient ConvNets. In: Proceedings - International Conference on Learning Representations. pp. 1–13. No. 2017 (2017)
26. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9905 LNCS, pp. 21–37. Springer, Cham (2016)
27. Lu, L., Liang, Y.: SpWA: An Efficient Sparse Winograd Convolutional Neural Networks Accelerator on FPGAs. 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC) pp. 1–6 (2018)
28. Malladi, K.T., Nothaft, F.A., Periyathambi, K., Lee, B.C., Kozyrakis, C., Horowitz, M.: Towards energy-proportional datacenter memory with mobile DRAM. Proceedings - International Symposium on Computer Architecture (Figure 1), 37–48 (2012)

29. Palossi, D., Conti, F., Benini, L.: An Open Source and Open Hardware Deep Learning-Powered Visual Navigation Engine for Autonomous Nano-UAVs. 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS) pp. 604–611 (2019)
30. Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S.W., Dally, W.J.: SCNN: An accelerator for compressed-sparse convolutional neural networks. Proceedings - International Symposium on Computer Architecture **Part F1286**, 27–40 (2017)
31. Pullini, A., Rossi, D., Loi, I., Tagliavini, G., Benini, L.: Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing. IEEE Journal of Solid-State Circuits **54**(7), 1970–1981 (jul 2019)
32. Redmon, J., Farhadi, A.: YOLOv3: An Incremental Improvement (2018)
33. Rhu, M., O'Connor, M., Chatterjee, N., Pool, J., Kwon, Y., Keckler, S.W.: Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks. In: Proceedings - International Symposium on High-Performance Computer Architecture. vol. 2018-Febru, pp. 78–91. IEEE (feb 2018)
34. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision **115**(3), 211–252 (2015)
35. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition **81**(2), 4510–4520 (feb 2018)
36. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: Proceedings - 2015 International Conference on Learning Representations. pp. 1–14 (2015)
37. Zhang, C., Sun, G., Fang, Z., Zhou, P., Pan, P., Cong, J.: Caffeine: Towards Uniformed Representation and Acceleration for Deep Convolutional Neural Networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **PP**(X), 1 (2018)
38. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. Proceedings - 2017 International Conference on Learning Representations pp. 1–14 (feb 2017)
39. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained Ternary Quantization. In: Proceedings - 2017 International Conference on Learning Representations. pp. 1–10 (2017)