Sparsity-Aware Memory Interface Architecture using Stacked XORNet Compression for Accelerating Pruned-DNN Models

Younghoon Byun [*1] Seungsik Moon [*1] Baeseong Park [2] Se Jung Kwon [2] Dongsoo Lee [2] Gunho Park [1] Eunji Yoo [1] JungGyu Min [1] Youngjoo Lee [1]

Abstract

This paper presents a new algorithm-hardware co-optimization approach that maximizes memory bandwidth utilization even for the pruned deep neural network (DNN) models. Targeting the well-known model compression approaches, for the first time, we carefully investigate the memory interface overheads caused by the irregular data accessing patterns. Then, the sparsity-aware memory interface architecture is newly developed to regularly access all the data of pruned-DNN models stored with the state-of-the-art XORNet compression. Moreover, we introduce the novel stacked XORNet solution for minimizing the number of data imbalances, remarkably relaxing the interface costs without slowing the effective memory bandwidth. As a result, experimental results show that our co-optimized interface architecture can achieve almost the ideal model-accessing speed with reasonable hardware overheads, successfully allowing the high-speed pruned-DNN inference scenarios.

1 Introduction

In the last few years, the size of deep neural networks (DNNs) has been continuously increased to achieve better performance, and now we can easily find a number of large-scale transformer models even, including more than several billions of trained weights for various applications, e.g., GPT-3 (Brown et al., 2020), PaLM (Chowdhery et al., 2022), and Dall-E (Ramesh et al., 2021). Due to the impractical memory requirements, therefore, model compression approaches have been essentially applied for deploying practical services associated with the recent large-scale networks (Kwon et al., 2022), i.e., building the memory-reduced compact model from the original one without degrading the application performances (Wang et al., 2020; Cheng et al., 2017). After applying the weight quantization considered as a very beginning model compression step (Jacob et al., 2018; Bondarenko et al., 2021), more precisely, the pruning-based network sparsification is commonly followed as the next technique to allow sparse matrix operations by eliminating less important weights (Han et al., 2015). Then, we only store the compressed network in the memory to reduce the total memory footprint, where the compressed format includes the extra information denoting the survived weight positions (Buluç et al., 2009; Moon et al., 2019a).

[*]Equal contribution [1]Department of Electrical Engineering, Pohang University of Science and Technology, Pohang, Republic of Korea [2]NAVER Cloud, Seongnam, Republic of Korea. Correspondence to: Youngjoo Lee <youngjoo.lee@postech.ac.kr>.

Proceedings of the 6th MLSys Conference, Miami Beach, FL, USA, 2023. Copyright 2023 by the author(s).

However, accelerating pruned-DNN models is difficult as we have to find the survived elements in sparse matrices from the compressed data structure. Some commercialized GPUs may support software kernels that perform the on-the-fly sparsity-aware computations directly from the compressed forms, such as compressed sparse row (CSR) and coordinate (COO) formats; however, those kernels are only effective for the extremely-spare matrix operations generally developed for the super-computing applications (Gale et al., 2020). Comparing two well-known software kernels as a case study, i.e., NVIDIA cuBLAS and cuSPARSE for dense and sparse matrices, respectively, it is clearly shown in Fig. 1 that the cuSPARSE kernel with the CSR format provides a meaningful performance improvement under the sparsity ratio of more than $S = 0.97$. Note that the current DNN accelerators have similarly offered the sparsity-aware dynamic data scheduling with the dedicated matching hardware; however, the state-of-the-art designs still accept the sparse but dense-format matrices in general for maximizing the processing efficiency of pruned-DNN computations (Cai et al., 2022). Considering the practical sparsity ratio of pruned large-scale DNN models, which ranges $0.5 \leq S \leq 0.8$ not to severely degrade the application performances of original dense models (Gale et al., 2019), as a result, it is reasonable to decompress the compressed model by utilizing the dedicated memory interface architecture in prior to activating the massive-parallel processing engines (Park et al., 2020).

Suppose we focus on the achievable system performance during the design or selection of computing platforms, which can be analyzed using the well-known roof-line anal-

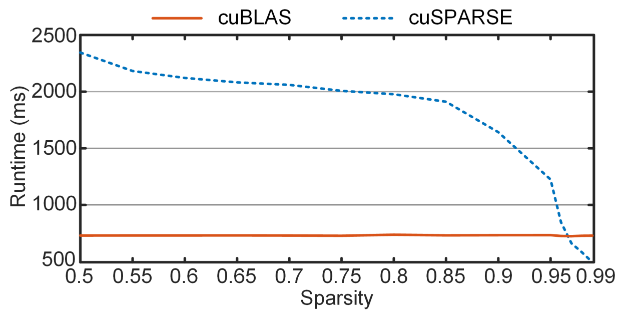

Figure 1. Runtime evaluation of previous software kernels for multiplying two 2048×2048 FP32-valued matrices with different sparsity ratios (Tested on an Nvidia RTX 3090 GPU with CUDA 11.3).

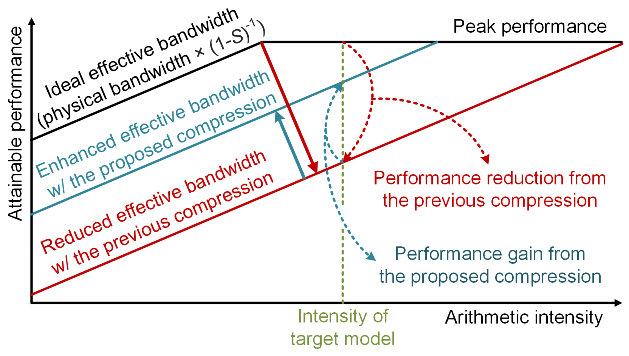
Ideal effective bandwidth
(physical bandwidth × (1-S)$^{-1}$)
Peak performance
Enhanced effective bandwidth
w/ the proposed compression
Reduced effective bandwidth
w/ the previous compression
Performance reduction from
the previous compression
Performance gain from
the proposed compression
Attainable performance
Intensity of
target model
Arithmetic intensity

Figure 2. Roofline analysis describing the performance reduction from pruned-DNN models with previous compression methods.

ysis in Fig. 2. First, an ideal effective bandwidth denoted as a black line in the upper left represents ideally obtainable data per time from the view of the processing part. The ideal effective bandwidth is only attainable when the pruned model is ideally stored by collecting only the survived values. In that case, it is natural to decide the number of processing engines for the given ideal bandwidth as well as the intrinsic arithmetic intensity of the target DNN model. However, due to the limitation of the model compression technique, we can observe that the effective memory bandwidth is always lower than the ideal one. In other words, the original computational-bounded system dedicated to the ideal model compression becomes the memory-bounded system after applying the practical pruning-based model compression as depicted in Fig. 2, causing the unwanted performance degradation as we cannot transfer a sufficient number of weights to the parallel processing engines designed for the ideal bandwidth. When the conventional CSR format is used to compress the sparse weight matrices of pruned-DNN models, for example, reconstructing the dense-format matrices should manage the irregular access patterns on the survived weights, making the effective memory bandwidth far from the ideal value (Lee et al., 2018). Achieving a similar compression quality, the recent works from (Kwon et al., 2020; Park et al., 2021) have reported that the model compression with XOR gates (XORNet) can be an alternative option to store the pruned-DNN models by forcing the regular memory interface for compressed weights, increasing the effective memory bandwidth close to the ideal bandwidth. However, the interface complexity for the previous XORNet impractically increased due to the irregular patch distribution.

In this work, for the first time, we quantitatively investigate the interface overheads of different compression techniques for pruned-DNN models. Then, we present a novel sparsity-aware memory interface architecture with vertically-arranged patches for realizing the cost-efficient parallel decompressors, supporting the on-the-fly decompressing of compressed pruned-DNN models to increase the effective memory bandwidth. In addition, the stacked XORNet (sXORNet) is newly developed by modifying the previous XORNet compression, which makes more imbalance-aware compression results to relax the extra on-chip buffer costs. Various experiments were performed to reveal that the proposed algorithm-hardware co-design approach allows the most attractive interface architecture in terms of effective memory bandwidth and hardware complexity. Targeting the practical compressed transformer model whose sparsity ratio is $S = 0.6$ (Gale et al., 2019), for example, the proposed sparsity-aware memory interface adopting the sXORNet compression reduces the area costs by 82.3% while utilizing 1.34 times faster effective memory bandwidth when compared with the conventional CSR-based interface architecture. As a result, we can fully enjoy the achievable performance of the original computational-bounded system by maximally utilizing the given memory bandwidth even for the pruned-DNN processing.

2 Evaluating Interface Overheads from Model Compression

2.1 Previous Model Compression Techniques

In this paper, the previous compression methods, including CSR (Buluç et al., 2009), Viterbi (Ahn et al., 2019), and XORNet (Kwon et al., 2020) approaches are firstly considered to investigate their interface-level overheads to support the on-the-fly decompression procedures. The conventional CSR format provides a reasonable compression ratio with an acceptable amount of extra information to represent the given sparse matrix by using index pointers and index values as depicted in Fig. 3(a) (Capra et al., 2020). More precisely, the index pointer represents an accumulated number of survived weights in each row, and the index value represents its corresponding position. As the pruned-DNN models generally have locally different sparsity ratios (Moon et al., 2019b), it is impossible to guarantee the fixed decompression latency for finding the exact loca-

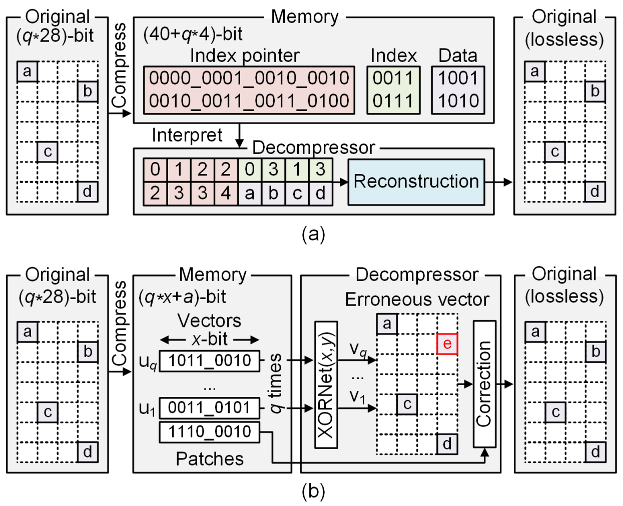

Figure 3. Decompression steps of compressed data using (a) CSR (Buluç et al., 2009) and (b) XORNet formats (Kwon et al., 2020)

tions of survived non-zero weights under the given physical memory bandwidth, i.e., the size of decoded data from the same amount of compressed data varies depending on the local sparsity of the original data. Therefore, the memory interface should access weights irregularly, and the effective memory bandwidth is naturally reduced for supporting the pruned-DNN, as shown in Fig. 2.

In contrast to the CSR case, the recent Viterbi-based compression supports the fixed decompression latency by utilizing the bit-by-bit sequence generation mechanism from the fixed-size encoded seeds (Lee et al., 2018; Ahn et al., 2019). The Viterbi compression draws a trellis path and back-trace it to choose the best match input. However, the compression technique shows a limited level of compression ratio and accuracy despite of retraining process. Therefore, the previous Viterbi-based approach cannot be the practical solution when we have to exploit massive-parallel decompressors to increase the effective bandwidth demanded by the advanced large-scale ML models.

For q-bit quantized DNN model, the recent XORNet compression in (Kwon et al., 2020; Park et al., 2021) introduces a bit-level look-up table (LUT) to encode sparse weight matrices of arbitrary ML models, which can be realized by fixed-level XOR gates for providing a deterministic decompression latency. As detailed in Fig. 3(b), to decompress the stored DNN model with the pruning ratio of S, the bit-level LUT denoted as XORNet(x, y) accepts x-bit vector $\mathbf{u}$ from memory and generates y-bit vector $\mathbf{v}$ in every cycle, where the ratio between x and y is carefully chosen to follow $x/y \simeq 1 - S$. In other words, we sequentially generate y decompressed weights bit by bit from x-bit compressed data, and each q-bit weight is potentially obtained by appending q consecutive LUT outputs. When we target the same physical memory bandwidth as the CSR compression generating the word-level decompression results, note that the bit-level LUT of XORNet compression manages a q times larger dense matrix at a time, performing the decompression q times for generating the same dense-format results as shown in Fig. 3(b). Due to the limited number of representable outputs, the LUT results can sometimes differ from the original weights. Therefore, it is required to utilize the correction step shown in Fig. 3(b), accessing the extra patches from the memory to denote the error positions. To increase the compression ratio approach to the pruning ratio, reducing the number of LUT errors is important, i.e., minimizing the extra patches as many as possible. The recent study from (Park et al., 2021) shows that including the previous LUT inputs with additional shift, registers can enlarge the representable options, especially for the locally-dense cases, significantly reducing the erroneous LUT outputs. In contrast to the conventional CSR method, note that the XORNet compression removes the irregular accessing patterns of compressed weights, which increases the effective memory bandwidth even close to the physical memory speed. However, we still observe irregular memory accessing patterns for extra patches, requiring a considerable amount of hardware overheads to complete the correction step within a few cycles.

2.2 Baseline Memory Interface Architecture with Parallel Decompressors

To develop the sparsity-aware memory interface architecture, for the first time, we evaluated the baseline designs dedicated to the previous compression techniques in terms of the effective memory bandwidth and the required hardware complexity. Fig. 4(a) depicts the baseline hardware architecture for the conventional CSR method, including multiple decompressors to support the high-bandwidth memory (NVIDIA, 2020). Due to the intrinsic weight-level irregularity, each decompressor is designed to accept a different number of survived weights simultaneously, forcing the interface hardware to deploy a distributing network internally. Due to the locally-dense weight distributions in pruned-DNN models (Park et al., 2021), however, the required number of survived weights at each memory-accessing cycle sometimes exceeds the maximum one provided by fully utilizing the peak physical memory bandwidth. In this case, we have to disable some decompressors that cannot receive sufficient survived weights, reducing the number of decompressed dense-format weight matrices, i.e., degrading the effective memory bandwidth seen from the processing engines.

If we adopt the recent XORNet method, as shown in Fig. 4(b), the decompression hardware consists of an XOR-based LUT followed by a patch-correction unit. Note that we now observe the regular accessing patterns of compressed weights, removing the weight-level distributing network in Fig. 4(a). Instead, accessing patches for the correction step

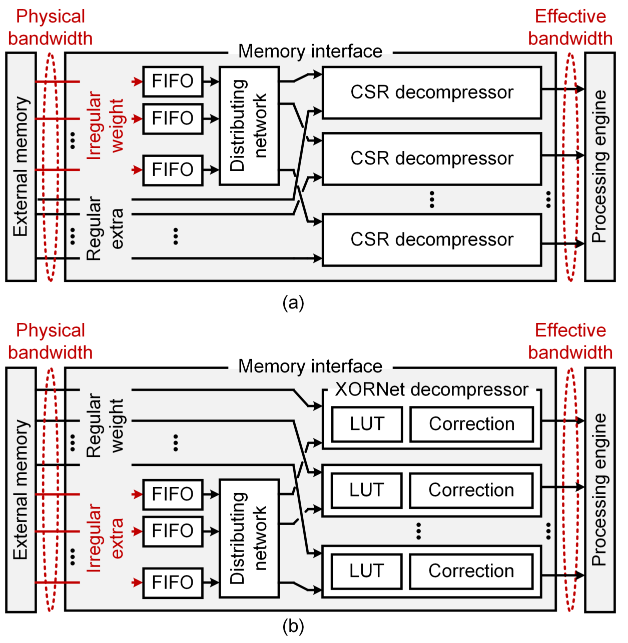

Figure 4. Baseline interface architectures with distributing network for (a) CSR and (b) XORNet methods.

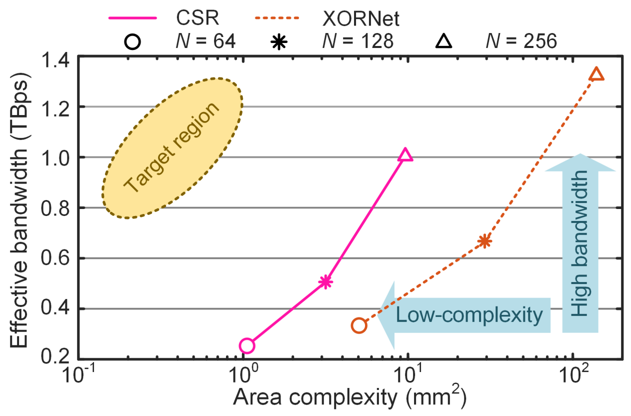

Figure 5. Area-bandwidth trade-offs of previous interface designs.

becomes irregular, adding the patch-level distributing network and transferring patches to the proper decompressors with erroneous outputs. In general, we allocate the physical memory bandwidth for patches by considering the average number of patches to fetch a sufficient number of patches at a time; however, some locally-dense cases sometimes produce much more errors inducing a few memory stalls.

In order to quantitatively investigate the dedicated memory interface unit for supporting the compressed pruned-DNN models, we implemented two interface architectures in Fig. 5, which are equally synthesized in a 28 nm CMOS technology at the operating frequency of 1 GHz. Considering the commercialized GPUs, for fair comparisons, we increase the number of parallel decompressors denoted as $N = 64$, 128, and 256 to test existing compression schemes under the physical memory bandwidths of 240 GBps, 480 GBps, and 960 GBps, respectively, where accessing the extra information is assumed to occupy 1/3 of physical bandwidth. The 0.6-pruned transformer model from (Gale et al., 2019) is used for this case study, clearly showing the limitation of each memory interface architecture as shown in Fig. 5. More precisely, the CSR format severely degrades the effective memory bandwidth as all the compressed weight data are accessed irregularly, resulting in a significant number of memory stalls. On the other hand, the XORNet compression provides higher effective memory bandwidth close to the peak speed as the total size of irregular patches is much smaller than that of irregular weights with CSR format (Kwon et al., 2020). For a single decompressor, however, the bit-level irregular patterns of the XORNet method naturally require much more input in the worst-case scenario compared with the word-level irregular accesses of the CSR format. Therefore, the hardware complexity of distributing network is drastically increased to support the XORNet compression, especially for utilizing massive-parallel decompressors as depicted in Fig. 5. As a result, both model compression methods cannot be the practical solution to deploy the pruned-DNN models when we consider the effective memory bandwidth as well as the cost-efficient implementation at the same time. With the novel algorithm-hardware co-optimization approach, in this work, we target much attractive complexity-bandwidth trade-off results as illustrated in Fig. 5, providing higher effective memory bandwidths with acceptable hardware overheads.

3 Proposed Sparsity-Aware Memory Interface Architecture

3.1 Vertically-Arranged Patches

To correct the erroneous bits in LUT outputs, for adopting the XORNet compression, it is necessary to deliver the required patches to each decompressor correctly. With a case example of patch distribution, Fig. 6 details this path delivery process in the interface architecture, where p_{ij} represents the j-th patch for the i-th vector. Utilizing N parallel decompressors, e.g., $N = 4$ in this case example, note that p_{ij} is then mapped to the $(i \bmod N)$-th decompressor. As shown in the figure, the straightforward XORNet implementation fetches N horizontally-aligned (HA) patches from memory, and the distributing network follows to dynamically construct the proper connections between patches (p_{ij}) and decompressors (D_x). At the first cycle of this example, more precisely, after initializing N two-depth input FIFOs shown in Fig. 4(b), the two patches in the first two positions (p_{00} and p_{01}) should be delivered to the first decompressor, where the last patch position should be connected to the

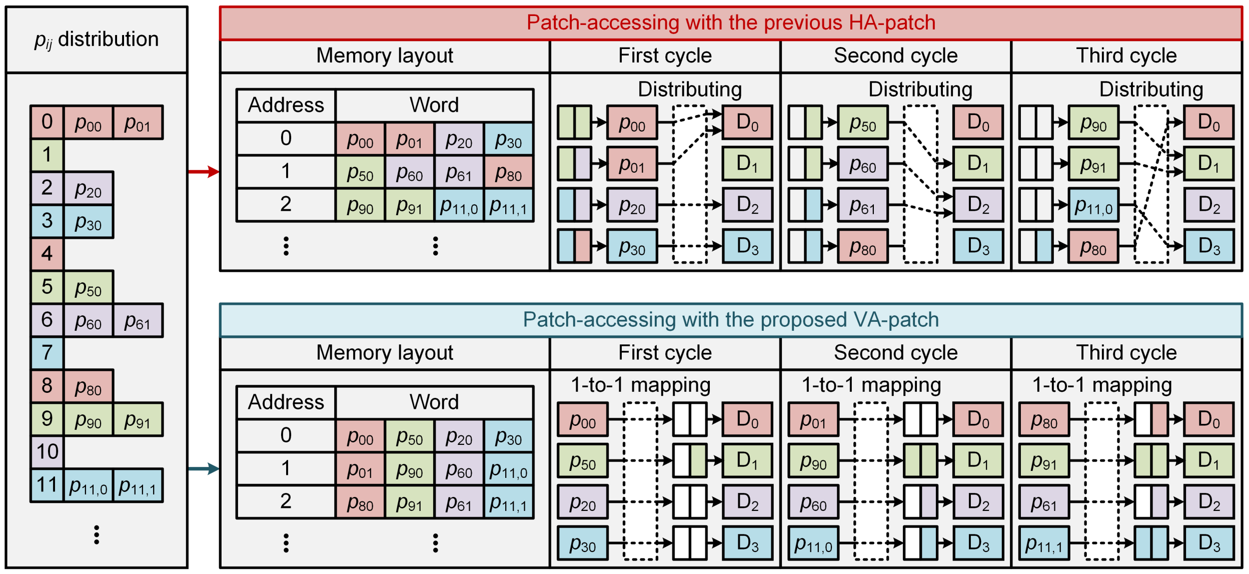

Figure 6. Example of patch-accessing procedures with different memory layouts.

same decompressor for taking p_{80} at the third cycle. In other words, the previous HA-patches can be consumed by any decompressors; therefore, the distributing network with numerous multiplexers is inevitable for handling the patch-level irregularity as depicted in Fig. 6. For the recent XORNet compression with two shift registers (Park et al., 2021), targeting the 0.6-pruned 8-bit quantized transformer from (Gale et al., 2019), Table 1 deeply analyzes the area overheads of the baseline memory interface architecture implemented in a 28 nm CMOS technology. According to the number of decompressors N, note that the complexity of the distributing networks rapidly increases compared with the other modules, actually causing the impractical interface costs to support the recent ultra-high-bandwidth memories.

To relax the hardware complexity without degrading the effective memory bandwidth, in this work, we present a new data storing strategy to keep the generated patches in a vertically-arranged (VA) form. Each vertical patch position is now dedicated to a fixed decompressor, which can allow N one-to-one fixed connections between the fetched patches and decompressors. More specifically, as exemplified in Fig. 6 with four-parallel decompressors, we now have four vertical patch streams, each of which is directly connected to the corresponding decompressor, totally removing the previous high-complexity distributing network. Therefore, fetching N row-wise VA-patches from memory distributes a single patch to each decompressor equally at a time. Note that the previous input FIFOs for collecting the first four HA-patches are also distributed to individual decompressors, each of which is now keeping the patches to be only consumed internally at the same decompressor. As

Table 1. Area overheads of baseline memory interface architecture.

AREA COMPLEXITY (mm^2)	N		
	64	128	256
LUT	0.053	0.106	0.213
CORRECTION	1.689	6.758	27.032
DISTRIBUTING NETWORK	3.311	22.403	111.863
TOTAL	5.053	29.268	139.109

described in Table 2 showing the average number of patches per XORNet$(20, 20/(1-S))$, which is dedicated to the pruned transformer (Gale et al., 2019), feeding one patch per decompressor per memory access is statistically enough to correct XORNet outputs regardless of the pruning ratio of S. If we consider the actual distribution of patch counts per XORNet output shown in Table 2, however, there exist some error cases containing several patches, which may affect the effective memory bandwidth. In addition to introducing the VA-patches, as a result, the proposed memory interface architecture needs to carefully deal with the patch-consuming speed of parallel decompressors, eventually improving the effective memory bandwidth with acceptable hardware costs while supporting the pruned-DNN models.

3.2 Interface Architecture for Patch-Level Imbalance

Fig. 7 shows the proposed memory interface architecture associated with VA-patches, removing the previous distributing network and even adopting the bit-level XORNet compression. To maintain the effective memory bandwidth by adopting the proposed VA-patches, however, we now have

Table 2. Generated patches for compressing pruned transformers.

Sparsity		0.6	0.7	0.8	0.9
Average # of patches		0.26	0.28	0.28	0.38
# of XORNet results	0 patch	986K	759K	502K	244K
	1 patch	175K	126K	74K	39K
	>1 patches	59K	54K	34K	23K

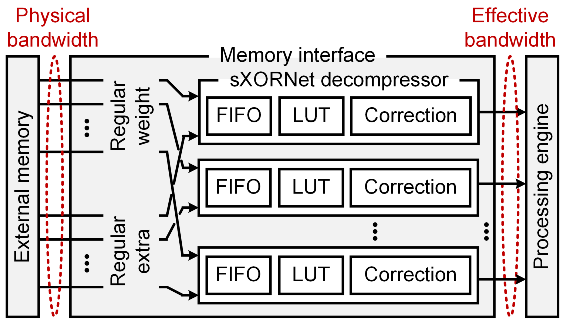

Figure 7. Proposed memory interface based on VA-patches.

to solve the patch-consuming imbalance problem by adjusting the size of the imbalance FIFO of each decompressor. As depicted in Fig. 6, decompressors, in general, consume a different number of patches depending on the corresponding LUT outputs. Deploying multiple decompressors for ultra-high-bandwidth memories, the current N VA-patches fetched together might be consumed at different decompression cycles. If we cannot keep a sufficient number of fetched yet unused patches in a locally-fast decompressor, extra memory cycles should be taken for re-accessing N VA-patches to guarantee the correct decompressed results, potentially reducing the effective memory bandwidth. Therefore, it is important to design a sufficiently-large imbalance FIFO that can temporarily store all the previously-fetched patches before being consumed at the proper correction cycle.

As the locally-slow decompressor should catch up with faster ones by consuming the incoming patches as soon as possible, the corresponding imbalance FIFO always stays in an empty state. Therefore, the maximum amount of patch-level imbalance (I_{max}) can be defined to be the maximum number of patches remaining in the fastest decompressor. As depicted in Fig. 8, to find the optimal FIFO size considering the peak patch-level imbalance, we carefully investigated imbalance changes for processing 0.6-pruned transformer model reported in (Gale et al., 2019). Regardless of N, it clearly shows that the proposed decompressor architecture requires an imbalance FIFO size of at least several hundred to operate the practical pruned-DNN models using the XORNet compression with VA-patches. If we naively set the FIFO size to contain at least I_{max} patches by analyzing the given pruned-DNN model, the patch-level imbalance is perfectly handled. Comparing the hardware costs

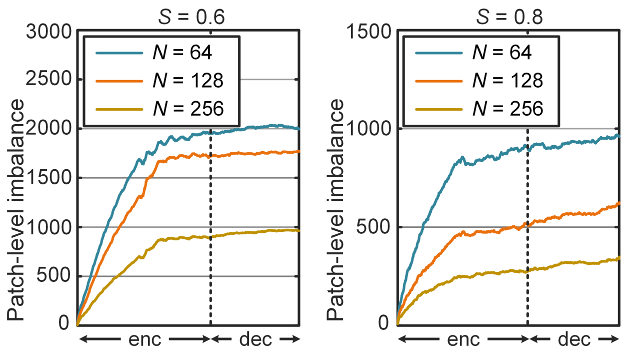

Figure 8. Investigating the peak patch-level imbalances of pruned transformer models.

of a N-to-N distributing network in the previous design and N I_{max} FIFOs added in the proposed work, for the case of 0.6-pruned 8-bit transformer model, our approach utilizing enlarged imbalance FIFOs with VA-patches saves the area costs by 50 times, definitely leading to more pleasurable results for implementing the cost-efficient yet high-speed memory interface architecture.

For the practical applications, we also present the generalized interface architecture by designing the fixed-size imbalance FIFOs, e.g., keeping up to 128, 256, or 512 patches so that the proposed work with VA-patches can decompress arbitrary pruned-DNN models. When I_{max} of the compressed model exceeds the maximum size of imbalance FIFO, for the worst-case decompression scenario, extra cycles are used to relax the current imbalance by re-accessing the missed VA-patches. In contrast to the prior designs that only offer the fixed hardware design due to the irregular memory accesses, for the first time, we can now provide the design-level trade-offs between the hardware complexity and the effective bandwidth for realizing the memory interface architecture. Moreover, the design options become even more cost-efficient if we apply the imbalance-aware data compression described in the following Section.

4 Pruned-DNN Compression using Imbalance-Aware Stacked XORNet

4.1 Analyzing the Patch-Level Imbalance

In this section, we deeply analyze how the patch-level imbalance occurred. Although the XORNet(x, y) is normally designed for support the average sparsity of $\mathbf{v}$ vectors (Kwon et al., 2020), denoted as $S_{\text{ave}} = 1 - x/y$, the local sparsity of each y-bit LUT output vector S_{local} is in general different from S_{ave}. As depicted in Fig. 3(b), the LUT output vector $\mathbf{v}$ may include the errors identified by the additional patches, especially when the S_{local} is much smaller than S_{ave}. To validate this patch-generating condition, for the different local sparsity, we tested the number of patches from

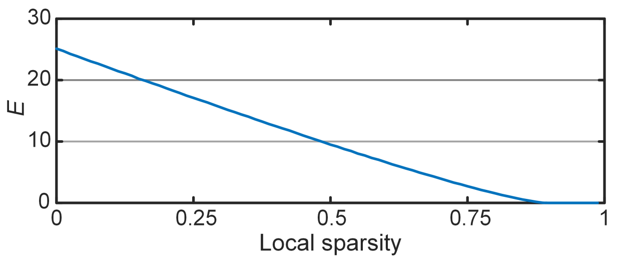

Figure 9. Average number of patches per XORNet(8, 80) output vector according to the local sparsity.

Table 3. Analysis on patches from the encoding part of 0.6-pruned transformer model using XORNet(8, 20) compression.

Layers		Vector group				Total N_P	I_{MAX}
		DD	D	S	SS		
EMB	N_V	12K	292K	403K	143K	113K	702
	E	0.45	0.24	0.09	0.02		
ENC1	N_V	9.8K	53K	55K	38K	42K	781
	E	0.91	0.44	0.18	0.01		
ENC2	N_V	3.2K	48K	82K	23K	22K	875
	E	0.52	0.27	0.09	0.01		
ENC3	N_V	1.5K	44K	94K	16K	16K	890
	E	0.47	0.19	0.06	0.01		
ENC4	N_V	1.2K	44K	98K	14K	14K	899
	E	0.52	0.18	0.06	0.01		
ENC5	N_V	1.0K	43K	99K	14K	14K	906
	E	0.46	0.17	0.06	0.01		
ENC6	N_V	1.2K	43K	99K	14K	14K	903
	E	0.61	0.18	0.06	0.01		

XORNet$(8, 80)$ that is originally designed for compressing vectors of $S_{\text{ave}} = 0.9$, i.e., expecting eight survived bits on average. As depicted in Fig. 9 showing the average number of patches per $\mathbf{v}$, which is denoted as E, it is clear that we observe more errors for generating locally-dense vectors due to the limited number of representable options (Park et al., 2021). Considering the XORNet operations with parallel decompressors, as depicted in Fig. 3, an increased number of patches induces patch buffer imbalance, increasing the overall interface costs with large imbalance buffers not to degrade the effective memory bandwidth.

For the practical case study of the 0.6-pruned transformer model (Gale et al., 2019), we investigated the layer-wise relations between the locally dense $\mathbf{v}$ vectors and the induced patch-level imbalances. More precisely, the XORNet(8, 20) is used to compress the pruned model, where the patches are stored in VA-patch format to make the regular access patterns to 256-parallel decompressors. Table 3 analyzes the patches generated by encoding part of the pruned transformer, i.e., one embedding layer (emb) followed by six encoding layers (enc1 $\sim$ enc6), where all the layers are individually pruned to make the identical layer-level sparsity of 0.6 (Gale et al., 2019). For the sake of simplicity, we define four $\mathbf{v}$ groups depending on the local sparsity; the extremely-dense (DD), dense (D), sparse (S), and extremely-sparse (SS) groups, including vectors whose local sparsity values are $S_{\text{local}} < 0.25$, $0.25 \leq S_{\text{local}} < 0.5$, $0.5 \leq S_{\text{local}} < 0.75$, and $0.75 \leq S_{\text{local}}$, respectively. In the table, in addition, we denote the numbers of vectors and patches as N_{V} and N_{P}, respectively. Considering the average sparsity of this case study, i.e., $S_{\text{ave}} = 0.6$, it is reasonable that group S includes most of the vectors. In terms of the average number of patches per vector (E), the sparse groups (S and SS) definitely contain the patch-free vectors, whereas the vectors in dense groups (D and DD) require much more patches, as we expected. As shown in the table, as a result, the maximum patch-level imbalance $(I_{\max})$ is significantly increased from the first three layers (emb $\sim$ enc2) that have lots of vectors in dense groups with large E values. Therefore, reducing the number of patches, especially for the locally-dense vectors, is definitely effective for relaxing patch-level imbalances.

4.2 Imbalance-Aware Stacked XORNet

In the previous XORNet compression (Kwon et al., 2020), we normally build a single bit-level LUT that generates the fewest patches by checking all the vectors in the pruned-DNN model; therefore, the locally-dense vector may generate much more patches increasing the patch-level imbalance as summarized in Fig. 10(a). To relax the imbalance problem causing the large-sized FIFOs in Fig. 8, in this work, the stacked XORNet (sXORNet) is newly proposed, allowing multiple compression options to reduce the number of patches from the locally-dense vectors. As shown in Fig. 10(b), for the given S_{ave}, we prepare two LUTs denoted as the half-size XORNet(x, y_1) and the full-size XORNet(x, y_2), which are optimized for lower sparsity ratio of $S_{\text{h}} < S_{\text{ave}}$ and higher sparsity ratio of $S_{\text{f}} > S_{\text{ave}}$, respectively. In fact, the LUT of full-size XORNet is constructed by stacking more parts on that of the half-size XORNet. Based on the local sparsity S_{local}, the proposed sXORNet chooses either half-size or full-size LUT for the decompression process as depicted in Fig. 10(b). As we allow the half-size LUT, the compressed data in the memory now includes more $\mathbf{u}$ vectors, each of which additionally includes a LUT-selection bit. By utilizing the full-size LUT dedicated to the local sparsity larger than the average one, moreover, we may slightly increase the number of patches from the locally-sparse vectors, which are weakly related to the patch-level imbalance. Therefore, adopting the proposed sXORNet compression may increase the overall memory footprint to store the pruned-DNN model compared with the previous XORNet method.

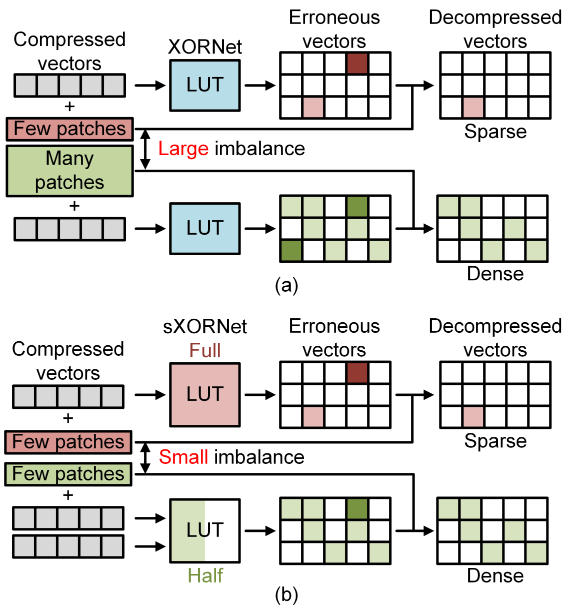

Figure 10. Handling locally dense or sparse vectors with (a) a single LUT in the previous XORNet method, and (b) two dedicated stacked LUTs in our imbalance-aware sXORNet compression.

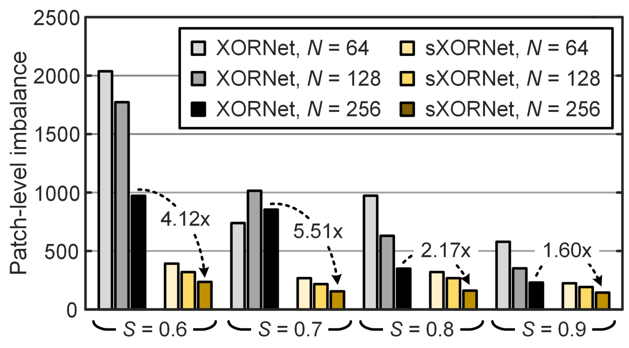

Figure 11. Reducing $I_{\max}$ with the proposed sXORNet method.

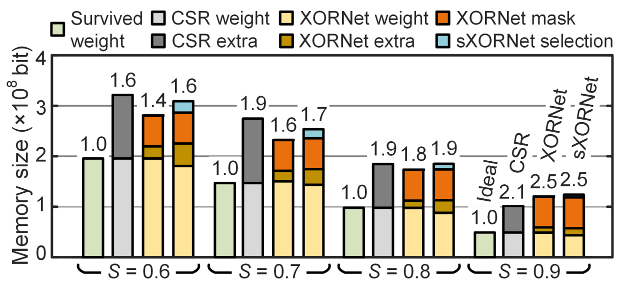

Figure 12. Compression quality to store 8-bit quantized transformer models with different compression schemes.

However, we can now select the dedicated XORNet option to reduce the number of patches generated from the locally-dense vectors, which mainly increases the patch-level imbalance as depicted in Table 3. Therefore, even if we slightly increase the total memory requirements, it is expected to remarkably decrease the value of $I_{\max}$, accordingly relaxing the imbalance buffer costs in Fig. 7. Thanks to the stacked LUT architecture shown in Fig. 10(b), in addition, the extra hardware complexity for realizing the proposed sXORNet compression can be negligible as we only require additional control complexity for providing multiple decompression options depending on the local sparsity. For the transformer models with different pruning ratios, Fig. 11 shows the reduced maximum imbalance $I_{\max}$ by adopting the proposed sXORNet compression. It is clear that our approach leads to a more balanced patch-consuming speed that allows smaller imbalance buffers not to degrade the effective memory bandwidth, e.g., reducing $I_{\max}$ by 76% compared to the previous state-of-the-art method for decompressing 0.6-pruned transformer model with 256 parallel decompressors. As a result, the proposed sXORNet approach drastically relaxes the interface overheads by reducing the maximum patch-level imbalance compared with the previous XORNet solution with a single LUT focusing on the average sparsity.

5 Experimental Results

5.1 Testing Environments

In order to validate the performance improvements from the proposed methods, we designed four different memory interface architectures handling the pruned-DNN models with specialized data compression schemes; 1) the conventional CSR method in Fig. 4(a) (Buluç et al., 2009), 2) the previous XORNet approach with HA-patches in Fig. 4(b) (Kwon et al., 2020), 3) the XORNet method adopting the proposed VA-patches in Fig. 7, and 4) the fully-optimized version based on the proposed sXORNet compression. To evaluate the area complexity, all the interface architectures were designed and synthesized in a 28 nm CMOS technology by allowing the same timing margins for achieving the operating frequency of 1 GHz. For testing the effective memory bandwidths under various pruning ratios, three INT8-quantized pruned-DNN models were used in our case studies, including transformer for machine translation (Vaswani et al., 2017), GPT-2 for natural language processing (Radford et al., 2019), and ResNet-50 for image recognition (He et al., 2016). Note that the pruned transformer models were directly obtained from the previous work[1], whereas the pruned GPT-2 and ResNet-50 models were constructed by magnitude based pruning, without retraining process.

5.2 Compression Quality

The compression quality of each compression technique is defined by the ratio between the ideal memory footprint counting only the survived weights and the actual one stor-

[1]https://github.com/google-research/google-research/tree/master/state_of_sparsity

Table 4. Implementation results of interface architectures based on the proposed VA-patches with different imbalance FIFO sizes.

N	Effective bandwidth (TBps)						Area complexity (mm^2)					
	XORNet			sXORNet			XORNet			sXORNet		
	$B = I_{MAX}$	$B = 512$	$B = 256$	$B = I_{MAX}$	$B = 512$	$B = 256$	$B = I_{MAX}$	$B = 512$	$B = 256$	$B = I_{MAX}$	$B = 512$	$B = 256$
64	0.31	0.22	0.17	0.34	0.34	0.30	1.339	0.401	0.244	0.386	0.428	0.271
128	0.61	0.44	0.34	0.68	0.68	0.63	2.354	0.801	0.489	0.699	0.855	0.542
256	1.22	1.08	0.90	1.36	1.36	1.36	2.783	1.603	0.977	1.061	1.711	1.085

Table 5. Area complexity and effective bandwidth of the conventional and proposed interface architecture for three different models.

N	Effective bandwidth (TBps)					Area complexity (mm^2)			
	Ideal	CSR	XORNet	XORNet	sXORNet	CSR	XORNet	XORNet	sXORNet
			HA-patch	VA-patch ($B = 256$)			HA-patch	VA-patch ($B = 256$)	
Transformer (Vaswani et al., 2017) ($S = 0.6$)									
64	0.60	0.25	0.33	0.17	0.30	1.057	5.053	0.244	0.271
128	1.20	0.51	0.66	0.34	0.63	3.165	29.268	0.489	0.542
256	2.40	1.01	1.32	0.90	1.36	9.687	139.109	0.977	1.085
GPT-2 small (Brown et al., 2020) ($S = 0.6$)									
64	0.60	0.25	0.33	0.09	0.27	1.057	5.053	0.244	0.271
128	1.20	0.50	0.66	0.23	0.58	3.165	29.268	0.489	0.542
256	2.40	0.99	1.32	0.61	1.21	9.687	139.109	0.977	1.085
ResNet-50 (He et al., 2016) ($S = 0.7$)									
64	0.80	0.26	0.41	0.31	0.40	1.227	5.067	0.258	0.271
128	1.60	0.51	0.82	0.62	0.79	3.804	29.295	0.515	0.542
256	3.20	1.02	1.64	1.23	1.57	11.925	139.163	1.031	1.085

ing the pruned-DNN model with the extra information after applying a specific compression method (Kwon et al., 2020). By changing the pruning ratio of the 8-bit transformer model (Vaswani et al., 2017), Fig. 12 compares different compression techniques in terms of the required memory size to store the pruned model. For storing CSR format, as described in Section 2, we need to store the extra indexing information that directly describes the survived weight locations. On the other hand, the previous XORNet compression requires two types of extra data; the patches for the correction step and the bit-wise masking patterns. In the proposed sXORNet method, we additionally store the selection information that allows the on-demand XORNet size selection depending on the local sparsity. However, the compression quality of the proposed work is still comparable with the other approaches, e.g., increasing the storage size by only 0.3% and 6.8% for storing 0.8-pruned model compared with CSR and XORNet methods, respectively, as we can reduce N_V by utilizing larger LUT for locally-sparse vectors. Note that such overheads of compression quality can be totally acceptable if we consider the hardware-level performance enhancements from the proposed optimization schemes.

5.3 Analyzing Hardware-Level Performances

To estimate the hardware-level performances of the proposed interface architecture, we first observed the effects of the imbalance FIFO. Targeting the 0.6-pruned 8-bit transformer model, Table 4 shows the total area complexity and the effective bandwidth of interface designs adopting the proposed VA-patches, where B denotes the size of imbalance FIFO in Fig. 7. If the FIFO is implemented to support the maximum patch-level imbalance, $I_{\max}$, the interface architecture can achieve the fastest effective bandwidth without causing extra memory-accessing cycles. Due to the reduced numbers of vectors as well as the peak imbalance, for this extreme-case implementation, note that the proposed sXORNet compression even provides faster effective bandwidth than the previous XORNet method. If we design the fixed-size FIFO for the generalized interface, the proposed sXORNet approach further improves the effective bandwidth by remarkably reducing $I_{\max}$, e.g., the 256-depth FIFO is enough to support the similar bandwidth as the $I_{\max}$-depth case in this example. In terms of the hardware complexity, for the same B-sized imbalance buffer, note the

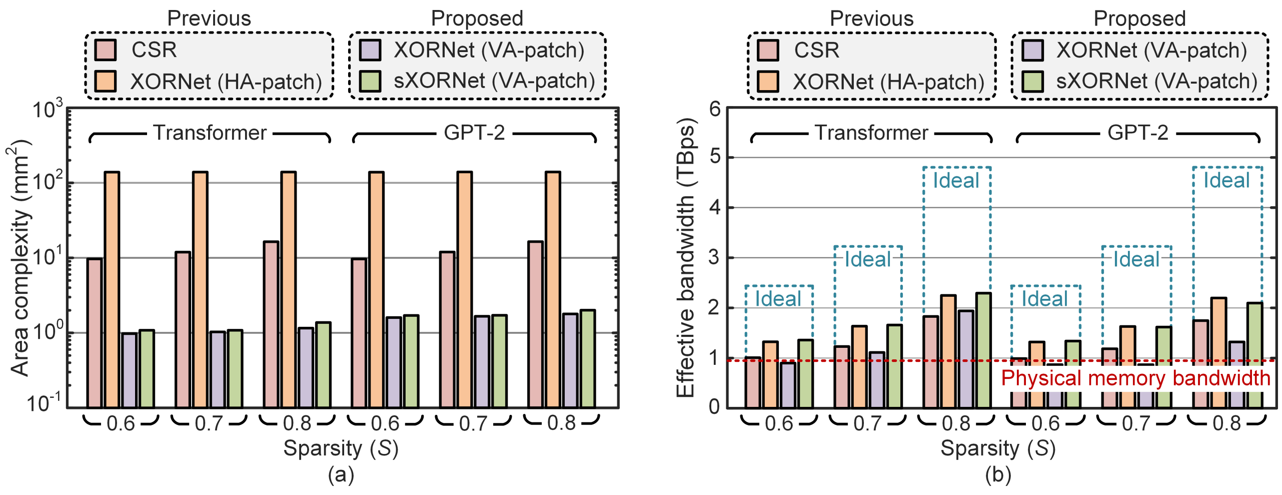

Figure 13. Evaluating interface architectures for various pruned transformers in terms of (a) area complexity and (b) effective bandwidth.

proposed work requires a negligible amount of additional costs by introducing the stacked LUT shown in Fig. 10.

Table 5 compares implementation results of memory interface architectures for different compression techniques to support the pruned-DNN models. Note that the conventional CSR method severely degrades the effective memory bandwidth compared with the ideal peak speed, whereas the previous XORNet approach with HA-patches requires an impractical hardware complexity for increasing the memory bandwidth. Utilizing the fixed imbalance FIFO size $(B = 256)$, the proposed VA-patches definitely lead to a cost-efficient interface design by totally removing the distributing network. Moreover, the proposed sXORNet compression greatly enhances the effective bandwidth regardless of the DNN styles, which is even close to the XORNet approach without memory stalls with HA-patches.

For different transformer-based models, i.e., the original transformer from (Vaswani et al., 2017) and GPT-2 small from (Brown et al., 2020), Fig. 13(a) investigates the interface complexity required by each compression technique, where 256 decompressors are processed in parallel to consider the physical memory bandwidth of 960 GBps. Allowing the regular memory accesses on both the compressed weights and the extra data, as we expected, utilizing VA-patches achieves the most cost-efficient interface hardware by replacing the complicated distributing network with the small-sized imbalance FIFOs. Compared with the state-of-the-art XORNet compression with HA-patches, for example, the area complexity is relaxed by 128 times to support the 0.6-pruned original transformer by introducing the proposed VA-patches. As depicted in Fig. 13(b), in addition, the optimized architecture adopting the proposed sXORNet compression offers comparable effective bandwidths compared to the previous XORNet method with HA-patches. Considering the hardware costs of the distributing network, as a result, the proposed VA-patches show the most attractive effective bandwidth by precisely adjusting patch distributions for locally-dense vectors.

Fig. 14 illustrates the complexity-bandwidth trade-offs by adopting different compression methods. To handle the 0.6-pruned transformer model, the proposed work finally achieves the effective memory bandwidth of 1.36 TBps while requiring the interface-level area complexity of 1.71 mm^2. With the regular memory accessing patterns, the hardware cost for supporting the proposed work is relaxed by 8.9 times and 128.2 times when compared with those for realizing the conventional CSR format (Buluç et al., 2009), and the previous XORNet compression (Kwon et al., 2020; Park et al., 2021), respectively. At the same time, the reduced peak patch-level imbalance from the proposed sXORNet compression allows the fastest effective memory bandwidth, i.e., increasing the memory speed by 34% compared with the CSR method. As a result, the proposed algorithm-hardware co-optimization approach successfully provides the practical memory interface design for accelerating the pruned-DNN models, eventually improving the system-level performance with the recent large-scale networks.

6 Conclusion

In this paper, we have presented the cost-efficient yet high-speed memory interface architecture that can successfully generate sparse dense-format weight matrices from the compressed data. For the first time, we fairly investigate the interface-level overheads to support different compression types, revealing the existing limitations in terms of the memory bandwidth as well as the hardware complexity. Then, the new memory layout with VA-patches is introduced to

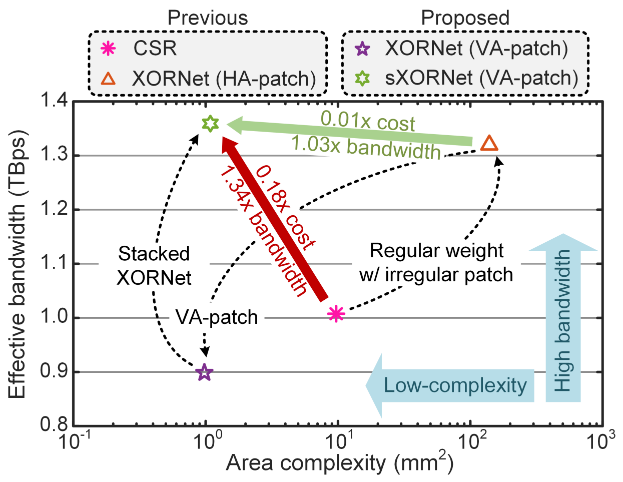

Figure 14. Investigating the complexity-bandwidth trade-offs to support the 0.6-pruned transformer model.

guarantee regular memory-accessing patterns, eliminating the previous distributing network. In addition, the patch-consuming speed per decompressor is evaluated to find the maximum patch-level imbalance, which can be greatly relaxed by adopting the proposed imbalance-aware sXORNet compression. Compared with the recent XORNet compression, experimental results show that the proposed work saves the area complexity by more than 120 times while providing a similar effective bandwidth, providing a sufficient number of weights to the following processing engines for accelerating the pruned-DNN processing.

Acknowledgements

We thank the anonymous reviewers for their feedback in the blind review process. This work was supported in part by the National Research Foundation (NRF) funded by the Korean government (MSIT) under Grant 2020M3H6A1085498 and Grant 2022R1A2C2092521.

References

Ahn, D., Lee, D., Kim, T., and Kim, J.-J. Double viterbi: Weight encoding for high compression ratio and fast on-chip reconstruction for deep neural network. In *International Conference on Learning Representations*, 2019.

Bondarenko, Y., Nagel, M., and Blankevoort, T. Understanding and overcoming the challenges of efficient transformer quantization. *arXiv preprint arXiv:2109.12948*, 2021.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33: 1877–1901, 2020.

Buluç, A., Fineman, J. T., Frigo, M., Gilbert, J. R., and Leiserson, C. E. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the 21th Annual Symposium on Parallelism in Algorithms and Architectures*, pp. 233–244, 2009.

Cai, H., Lin, J., Lin, Y., Liu, Z., Tang, H., Wang, H., Zhu, L., and Han, S. Enable deep learning on mobile devices: Methods, systems, and applications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(3):1–50, 2022.

Capra, M., Bussolino, B., Marchisio, A., Masera, G., Martina, M., and Shafique, M. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access*, 8:225134–225180, 2020.

Cheng, Y., Wang, D., Zhou, P., and Zhang, T. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. PALM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Gale, T., Zaharia, M., Young, C., and Elsen, E. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2020.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Kwon, S. J., Lee, D., Kim, B., Kapoor, P., Park, B., and Wei, G.-Y. Structured compression by weight encryption for unstructured pruning and quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1909–1918, 2020.

Kwon, S. J., Kim, J., Bae, J., Yoo, K. M., Kim, J.-H., Park, B., Kim, B., Ha, J.-W., Sung, N., and Lee, D. Alphatuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models. *arXiv preprint arXiv:2210.03858*, 2022.

Lee, D., Ahn, D., Kim, T., Chuang, P. I., and Kim, J.-J. Viterbi-based pruning for sparse matrix with fixed and high index compression ratio. In *International Conference on Learning Representations*, 2018.

Moon, S., Byun, Y., Park, J., Lee, S., and Lee, Y. Memory-reduced network stacking for edge-level cnn architecture with structured weight pruning. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(4): 735–746, 2019a.

Moon, S., Lee, H., Byun, Y., Park, J., Joe, J., Lee, S., and Lee, Y. FPGA-based sparsity-aware CNN accelerator for noise-resilient edge-level image recognition. In *Proc. Asian Solid-State Circuits Conference*, 2019b.

NVIDIA. NVIDIA ampere GA102 GPU architecture, second-generation RTX whitepaper. 2020.

Park, B. S., Kwon, S. J., Oh, D., Kim, B., and Lee, D. Encoding weights of irregular sparsity for fixed-to-fixed model compression. In *International Conference on Learning Representations*, 2021.

Park, J., Yoon, H., Ahn, D., Choi, J., and Kim, J.-J. OPTIMUS: OPTImized matrix multiplication structure for transformer neural network accelerator. *Proceedings of Machine Learning and Systems*, 2:363–378, 2020.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pp. 8821–8831, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., and Han, S. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2078–2087, 2020.