FNM-Trans: Efficient FPGA-based Transformer Architecture with Full N:M Sparsity

Manting Zhang*, Jialin Cao*, Kejia Shi, Keqing Zhao, Genhao Zhang, Jun Yu, Kun Wang†
School of Microelectronics, Fudan University, Shanghai, China
†kun.wang@ieee.org

ABSTRACT

Transformer models have become popular in various AI applications due to their exceptional performance. However, their impressive performance comes with significant computing and memory costs, hindering efficient deployment of Transformer-based applications. Many solutions focus on leveraging sparsity in weight matrix and attention computation. However, previous studies fail to exploit unified sparse pattern to accelerate all three modules of Transformer (QKV generation, attention computation and FFN). In this paper, we propose *FNM-Trans*, an adaptable and efficient algorithm-hardware co-design aimed at optimizing all three modules of the Transformer by fully harnessing $N:M$ sparsity. At the algorithm level, we fully explore the interplay of dynamic pruning with static pruning under high $N:M$ sparsity. At the hardware level, we develop a dedicated hardware architecture featuring a custom computing engine and a softmax module, tailored to support varying levels of $N:M$ sparsity. Experiment results show that, our algorithm optimizes accuracy by 11.03% under 2:16 attention sparsity and 4:16 weight sparsity, compared to other methods. Additionally, *FNM-Trans* achieves speedups of 27.13× and 21.24× over Intel i9-9900X and NVIDIA RTX 2080 Ti, respectively, and outpaces current FPGA-based Transformers by 1.88× to 36.51×.

KEYWORDS

Algorithm–hardware codesign, Transformer, FPGA

1 INTRODUCTION

Transformer-based networks [19] have achieved great performance in diverse fields, such as BERT [4] in Natural Language Processing (NLP) field and ViT [5] in Computer Vision (CV) field. However, the great performance is attained with heavy memory footprints and computational complexity, which hinders the efficient deployment of Transformer-based models, especially on resource-constrained edge devices.

To tackle this issue, various ASIC and FPGA Transformer accelerators extensively utilize sparsity, focusing on two types: attention sparsity [8, 11, 17] and weight sparsity [2, 6, 7, 14, 15, 18] to accelerate Transformers. Besides considering weight sparsity and attention sparsity separately, there are works that consider them comprehensively. FACT [16] has proposed an algorithm-hardware framework based on eager prediction to fully optimize QKV generation, attention computation, and feedforward network (FFN) of Transformers. However, it mainly exploits sparsity in QKV generation and attention computation, whereas for FFN, it utilizes eager prediction to assign different precision, overlooking the opportunities for sparsity in FFN. Furthermore, the sparsity utilized by FACT is unstructured, which may limit the acceleration performance. Additionally, the eager prediction mechanism may bring additional hardware overhead. Liu *et al.* [10] have also fully utilized weight sparsity and attention sparsity in Transformers. However, it treats these two types of sparsity differently, with unstructured sparsity in weight and local attention sparsity in attention, preventing the reuse of the core computing engine. Based on the above discussions, we are motivated to treat attention and weight sparsity with unified sparsity strategy, in order to fully exploit sparsity in Transformers while maximizing the reuse of the core computing engine.

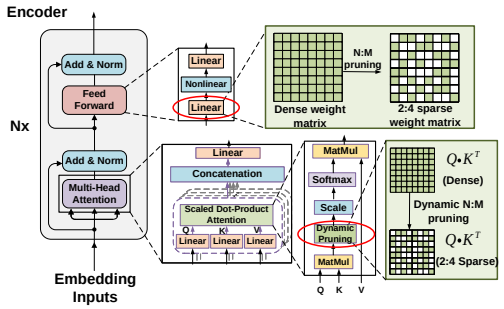

Figure 1: An illustration of *FNM-Trans*. For weight sparsity, we directly apply $N:M$ sparsity to weight matrix. For attention sparsity, we dynamically prune the results of $Q \cdot K^T$.

$N:M$ sparsity, which means there are N non-zero elements in every continuous M elements, is a type of fine-grained structured sparsity and has been adopted in A100 GPU for acceleration. Compared with unstructured sparsity, $N:M$ sparsity is more hardware friendly and has been utilized to accelerate Transformers. Fang *et al.* [6, 7] apply $N:M$ sparsity in weight to accelerate Transformers, but omit the sparsity opportunity in attention. This greatly hinders the acceleration performance of Transformers with long sequence length, because attention has quadratic computational complexity with respect to the sequence length. Chen *et al.* [3] apply $N:M$ attention sparsity based on GPU. However, due to the constraints of hardware support of GPU, they only explore $1:2$ and $2:4$ (50% sparsity) attention sparsity. As demonstrated in Subsection5.2, with our custom pruning algorithm and hardware architecture, we can

*Both authors contributed equally to this research. †Corresponding author

DAC '24, San Francisco, CAUSA,

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656497

accelerate full $N:M$ sparse Transformers under 1:16 (93.75%) attention sparsity and 4:16 (75%) weight sparsity with acceptable accuracy degradation.

In this work, we propose *FNM-Trans*, an efficient FPGA-based Transformer accelerator to comprehensively exploit weight sparsity and attention sparsity with $N:M$ fine-grained structured sparsity. The illustration of our work is in Fig. 1. By adopting a consistent $N:M$ sparsity strategy for both attention sparsity and weight sparsity, we fully leverage our core computing engine. This unified strategy streamlines processing and circumvents the extra hardware overhead incurred by the separate treatment of attention and weight sparsity. The main contributions of this work are as follows:

• To the best of our knowledge, this is the first algorithm-hardware co-optimized framework, leveraging both weight sparsity and attention sparsity of transformer under $N:M$ sparsity on FPGA. We have comprehensively explored the combination of weight and attention sparsity, achieving a concurrent 2:8 sparsity ratio for both weights and attention with minimal loss on accuracy.

• To reduce the additional storage and computational costs of sparse-dense matrix multiplication, we propose a scalable encoding scheme based on binary tree structure. This approach facilitates seamless scaling from low to high sparsity, contributing to the standardization of computations across various sparsity levels.

• We propose a dedicated hardware architecture tailored for diverse $N:M$ sparsity levels, comprising a flexible systolic array (SA) for both sparse and dense operations, a dynamic pruning engine for generating sparse attention matrices and compressed encoding, and a length adaptive softmax module for various length of sparse attention scores. This architecture processes various sparsity levels of weight and attention within a single execution.

• In terms of software experiments, our optimizations have achieved up to 12.26% accuracy improvement under high $N:M$ sparsity. In hardware experiments, *FNM-Trans* has attained up to 19.47× speedup compared to Intel i9-9900X, NVIDIA RTX 2080 Ti, and previous FPGA-based accelerators for Transformers.

2 BACKGROUND AND MOTIVATION

2.1 Transformer Networks

The transformer network structure, shown in Fig. 1, comprises two main components: Multi-Head Attention (MHA) and FFN. The MHA module computes attention scores by multiplying query, key, and value matrices, then combines results using a multi-head mechanism for enhanced representation. The FFN module mainly consists of two linear layers and a nonlinear function in between.

2.2 Sparse Transformer Accelerators

Sparsity is widely used to accelerate transformer models. Previous works usually utilize weight sparsity and attention sparsity to achieve the acceleration of transformer models.

Sparse Attention Acccelerators. A^3 [8] utilizes algorithmic approximation to remove unrelevant computation in attention mechanism. Sanger [11] proposes a dynamic and fine-grained structured pruning technique and corresponding reconfigurable hardware to accelerate attention. SALO [17] accelerates attention for long sequences through hybrid sparse attention mechanisms.

Sparse Weight Accelerators. Song *et al.* [18] utilize hybrid block-structured pruning and pattern pruning strategy to explore weight sparsity in Transformers. Qi *et al.* [15] introduce block-balanced pruning with a novel sparse matrix storage format to accelerate the sparse Transformers. Fang *et al.* [6, 7] utilize $N:M$ sparsity to prune weight.

Full Sparse Accelerators. FACT [16] utilizes eager prediction to utilize full sparsity in transformers. Liu *et al.* [10] also fully utilize weight sparsity and attention sparsity in Transformers.

Aforementioned sparse transformer accelerators mostly either focus on attention-only sparsity or weight-only sparsity. Even for works addressing both, FACT [16] does not explore sparsity in FFN; Liu *et al.* [10] treat attention sparsity and weight sparsity separately, leading to additional hardware overhead. Therefore, we aim to treat attention and weight sparsity with the same structured $N:M$ sparsity strategy, fully exploiting sparsity in Transformers while maximizing core computing engine reuse.

Algorithm 1: Full sparse $N:M$ pruning algorithm

Input: Pre-trained dense model $Model_{pd}$, and its accuracy acc_d, the minimum N_{am} and M_a in attention, the minimum N_{wm} and M_w in weight, accuracy gap threshold acc_t

$N_a \leftarrow M_a, N_w \leftarrow M_w, N_{as} \leftarrow 0, N_{ws} \leftarrow 0$;
$Model_s \leftarrow \emptyset, Model_{cur} \leftarrow Model_{pd}$;
// Stage1: Concurrent iterative pruning
repeat
  $Model_{lb} \leftarrow Model_{cur}$;
  $N_a \leftarrow N_a - 1, N_w \leftarrow N_w - 1$;
  $acc, Model_{cur} \leftarrow \mathbf{FNM}(Model_{lb}, N_a, M_a, N_w, M_w)$;
until $|acc - acc_d| > acc_t$;
$Model_s \leftarrow Model_{lb}, N_{as} \leftarrow N_a + 1, N_{ws} \leftarrow N_w + 1$;
$Model_{cur} \leftarrow Model_s, N_a \leftarrow N_{as}$;
// Stage 2.1: Attention only iterative pruning
repeat
  $Model_{lb} \leftarrow Model_{cur}$;
  $N_a \leftarrow N_a - 1$;
  $acc, Model_{cur} \leftarrow \mathbf{FNM}(Model_{lb}, N_a, M_a, N_{ws}, M_w)$;
until $N_a == N_{am}$;
$Model_{cur} \leftarrow Model_s, N_w \leftarrow N_{ws}$;
// Stage 2.2: Weight only iterative pruning
repeat
  $Model_{lb} \leftarrow Model_{cur}$;
  $N_w \leftarrow N_w - 1$;
  $acc, Model_{cur} \leftarrow \mathbf{FNM}(Model_{lb}, N_{as}, M_a, N_w, M_w)$;
until $N_w == N_{wm}$;

Output: A series of full $N:M$ sparse Transformers with multiple attention and weight sparsity sets

3 PRUNING ALGORITHM

Compared to pruning weight and attention separately, pruning both generally jointly results in greater accuracy degradation. Therefore, we propose a full $N:M$ pruning algorithm to obtain high sparsity with acceptable accuracy degradation. Alg. 1 details our pruning algorithm. The core of our algorithm lies in progressively reducing

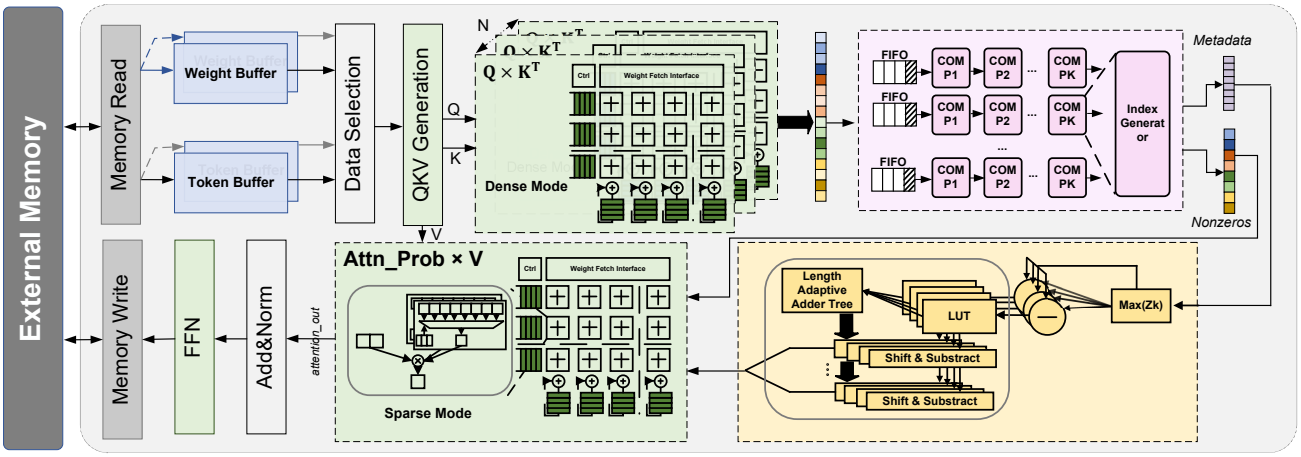

Figure 2: Overall architecture of *FNM-Trans*. The green part represents FNM-SA (Subsection 4.2), the pink part is Dynamic Attention Mechanism (Subsection 4.3), and the green part again denotes Length Adaptive Softmax (Subsection 4.4).

sparsity rather than directly training a full sparse model, preventing abrupt significant damage to the model. Our pruning algorithm encompasses two stages.

Stage 1: Concurrent Iterative Pruning. In stage 1, we conduct concurrent iterative pruning to both weight and attention. **(1)** We start by initiate $N_a = M_a$ and $N_w = M_w$ for attention $N:M$ sparsity and weight $N:M$ sparsity. **(2)** We initiate the last best model $Model_{lb}$, which is to be inherited. **(3)** Then gradually reduce N_a and N_w with the same step (line 5). **(4)** Using the inherited model $Model_{lb}$ and current sparsity parameters ($N_a:M_a$ and $N_w:M_w$ for attention and weight, respectively), we conduct full $N:M$ sparse training via **FNM** function. In the sparse training process, dynamic sparse training [22] is applied for weight sparse training. For attention sparse training, we apply row-wise $N_a:M_a$ pruning to the results of $Q \cdot K^T$ [3]. Note that in $N_w:M_w$ weight pruning, we keep the top N_w elements by absolute values and set the rest to zero, while in $N_a:M_a$ attention pruning, we retain the top N_a elements by actual values, rather than absolute values. **(5)** If $|acc - acc_d| \leq acc_t$, the process goes to **(2)**. Otherwise, **Stage 1** is completed, and the model at the conclusion of **Stage 1** ($Model_{lb}$, not $Model_{cur}$), along with its corresponding sparsity parameters, is saved as the initialization for **Stage 2** (line 8).

Stage 2: Split Iterative Pruning. Stage 2 consists of two independent processes: **Stage 2.1**, attention-only iterative pruning and **Stage 2.2**, weight-only iterative pruning. Take **Stage 2.1** as an example. **(a)** We initiate model and parameters saved from **Stage 1**. **(b)** We assign $Model_{lb}$. **(c)** Then, N_a is gradually reduced while N_w is fixed as N_{ws}. **(d)** Full sparse training is conducted via **FNM** function. **(e)** The process goes to (b) unless $N_a == N_{am}$. **Stage 2.2** is very similar to **Stage 2.1**, except that in **Stage 2.2** N_w is gradually reduced with N_a fixed as N_{as}, and the termination condition is $N_w == N_{wm}$. Following **Stage 1**, **Stage 2** further explores models with higher sparsity.

As Subsection 5.2 demonstrates, model accuracy benefits a lot from our custom pruning algorithm compared with one-shot full $N:M$ pruning. We obtain full $N:M$ sparse Transformers with up to 1:16 (93.75%) attention sparsity and 4:16 (75%) weight sparsity, incurring acceptable accuracy degradation.

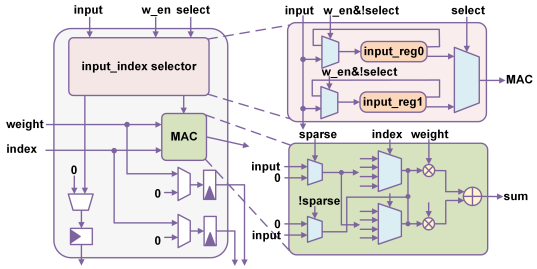

Figure 3: Hierarchical Structure of FNM-SA.

4 HARDWARE ARCHITECTURE

This section presents our hardware architecture tailored for full $N:M$ sparse Transformers. We first introduce the overall architecture of *FNM-Trans* in Subsection 4.1, and then elaborate on the modules specially designed to support dynamic sparsity, including the computational engine in Subsection 4.2, attention design in Subsection 4.3, and the length-adaptive softmax design in Subsection 4.4.

4.1 Overall Architecture

To fully exploit $N:M$ sparsity in accelerating the Transformer model, we design a dedicated hardware architecture (see Fig. 2). The host side initially handles word and positional embedding, transmitting processed data to the FPGA's off-chip DDR memory via a PCIe interface. The FPGA comprises computing, storage, and control components, including dynamic pruning and encoding, length-adaptive softmax, and a sparse computation engine tailored for $N:M$ sparsity. Nonlinear functions are supported by auxiliary processing units. Memory management includes off-chip DDR, on-chip input buffer, and weight buffer. Upon completion of FPGA-side computation, results are sent back to the host.

4.2 FNM-SA

FNM-SA is a specialized computational engine designed for sparse-dense multiplications under various $N:M$ sparsity. Its PE (Processing Element), which is depicted in Fig. 3, is characterized by three main features: compatibility with different levels of $N:M$ sparsity, a continuous weight stream and a flexible modular design.

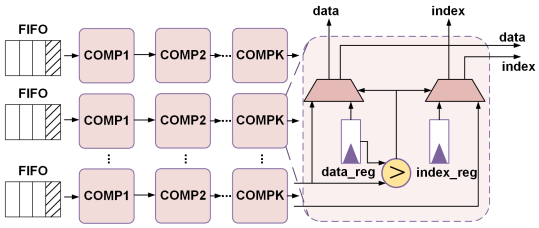

Figure 4: Dynamic Pruning Engine.

Matrix Computations under Various N:M Sparsity Patterns. FNM-SA unifies dense-dense and sparse-dense matrix multiplications across various $N:M$ sparsity levels exceeding 50%, a capability not supported by Ampefre GPU. Our approach overcomes this limitation with dedicated encoding schemes and MAC (Multiply-Accumulate) modules. Sparse patterns are encoded into $2:M$ format, where M is a power of two (detailed in Subsection 4.3). The MAC then selects relevant input data based on the weight index. As a Binary Tree Structure is employed for encoding, lower sparsity encoding is a subset of higher sparsity encoding. Consequently, selectors designed for high sparsity are applicable in low sparsity scenarios, enabling effective management of different sparsity levels across modules. For dense-dense matrix multiplications, the non-zero element selector is bypassed for energy saving.

Supporting Continuous Weight Stream. To maintain uninterrupted weight flow and mitigate loading disruptions from input, we introduce an input selector for each PE in FNM-SA. Since FNM-SA is input stationary, inputs are read from the BRAM into each PE's register file. In conventional SA arrays, during the loading of input, the data flow of weight have to be interrupted. The discontinuous nature significantly hinders the throughput of SA. Input selector, as depicted in Fig 3, includes two sets of buffers: one for external inputs and another for MAC data feeding. The first buffer set stores input validated by the *w_en* signal, and the *select* signal efficiently manages the switch between these sets.

Modular Architecture. In MHA each head requires its own SA, but in the FFN phase, only a single SA is used, leaving others idle. Furthermore, due to increased dimensionality in FFN, more MAC units are required for computation. To fully utilize the idle SAs and meet the computational demands, we combine the separate SAs from the attention phase together for FFN computations. Since FNM-SA is input-stationary, a column of PEs completes a row of elements in the output matrix. By leveraging this characteristic, we can assign different rows of the high-dimensional matrix in the FFN phase to different SAs for processing.

4.3 Dynamic Attention Mechanism

To enable dynamic pruning and encoding, we introduce a flexible compression engine capable of accommodating diverse pruning formats. At the core of this compression design is the dynamic pruning engine. Additionally, we propose the efficient encoding scheme to adapt to the diverse $N:M$ sparsity.

Dynamic Pruning Engine. The primary objective of the dynamic pruning engine is to identify the most significant elements within a given array. However, a straightforward approach would involve sorting the original array, with a time complexity of $O(nlogn)$ and a space complexity of $O(nlog_2n)$. Instead, we design an innovative dynamic pruning engine (see Fig. 4) that achieves a time complexity of $O(n)$ and a space complexity of $O(n)$ for identifying the top elements. The dynamic pruning engine consists of n comparators, where n is the number of elements to be selected. The first comparator fetches an element from external source and compares it with the element stored in its register. The smaller element is then forwarded to the next comparator. This process repeats until the top elements are stored in the registers of these comparators. Additionally, our top-k engine smoothly integrates with the matrix multiplication results, stored in First-In-First-Out (FIFO) buffers. It fetches one element at a time and delivers results once all elements have been extracted from the FIFO.

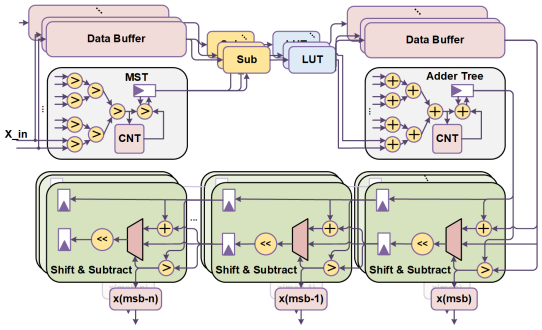

Figure 5: Schematic of the Length Adaptive Softmax Module.

Supporting Various Encoding Formats. To support various level of $N:M$ sparsity, we assess three encoding schemes: bit-mask, index, and pattern encoding. For a 2:8 sparsity ratio, bit-mask encoding requires 8 bits for storage, pattern encoding 5 bits, and index encoding 6 bits. In terms of hardware consumption, bit-mask encoding necessitates two 8-to-1 selectors and additional logic gates for the *enable* signals of each selector, pattern encoding requires one 8-to-2 selector, whereas index encoding only needs two 8-to-1 selectors. Balancing storage and hardware costs, we select index encoding schemes. Furthermore, to support for computations with different $N:M$ sparsity ratios, we adopt a Binary Tree Structure for our encoding scheme. In this structure, nodes at the same leaf depth represent the same sparsity level, with the lowest sparsity near the parent node. Depending on the user's selection, our compression model offers support for a range of compression formats. Additionally, to conserve energy, the second comparator is deactivated when operating in 1:8 and 1:4 selection modes.

4.4 Length Adaptive Softmax Module.

Our softmax module is adaptable to handle varying vector lengths due to dynamic $N:M$ pruning after $Q \cdot K^T$ multiplication. There are two variables that can be easily modified according to requirements in our softmax design. The first one is the length of the input vector, and the second one is the bit width of the output result. As shown in Fig. 5, the length adaptive softmax module consists of five major parts: a partial maximum search unit, a subtraction unit, a space-efficient exponential function unit, a partial sum accumulator, and a scalable divider. The quest for maximum value and summation operations involves binary operations in a tree-like structure, each followed by a counter. The number of branches is determined by the vector length, and a comparison with counting units decides the conclusion of each operation. The exponential function is approximated using a lookup table [1], requiring just

Table 1: Comparison of *FNM-Trans* with SOTA Works and Commercial Products

Platform	CPU	GPU			FPGA					
	i9-9900X	jetson Nano	RTX 2080 Ti	RTX 3090	SOCC'20 [12]	ISLPED'20 [9]	ISQED'21 [14]	STA-Large [7]	*FNM-Trans*	
Chip	Skylake	Tegra X1	TU102	GA102	XCVU13P	XCVU9P	XCU200	XCVU13P	XCU200	
Texhnology	14 nm	20 nm	12 nm	8 nm	16 nm	16 nm	16 nm	16 nm	16nm	
Frequency	3.50 GHz	640 MHz	1.35 GHz	1.70 GHz	200MHz	-	-	200 MHz	200 MHz	
# MAC units	-	-	-	-	4096	5647	3368	4096	4096	
Bit Precision	FP-32	FP-32	FP-32	FP-32	FIX-8	FIX-16	-	FIX-16	FIX-16	
Methods	-	-	-	2:4 group-based pruning	Low-bit quantization	Block-circulant matrix with FFT	Block-based Pruning	$N:M$ group-based pruning	2:8 weight & attention sparsity	2:16 weight & attention sparsity
Test Network	Shallow Transformers									
Latency (ms)	2.17	16.24	1.70	0.46	0.30	2.94	0.32	0.15	0.12	0.08
Batch-1 Throughput (GOP/s)	101.38	13.55	129.41	478.26	733.33	75.34	687.50	1466.67	1833.33	2750
Power (W)	165.00	7.56	250.00	350.00	16.70	22.45	-	26.59	28.23	28.23
Energy Efficiency (GOP/J)	0.61	1.79	0.52	1.37	43.91	3.35	-	55.16	64.94	97.41
MAC Efficiency (GOP/s/unit)	-	-	-	-	0.18	0.01	0.20	0.36	0.45	0.67

one multiplier and one adder. To reduce the latency of division operation, a highly parallel divider is designed with cascaded blocks and pipelines. Each cycle produces a result bit, and in each division unit, subtraction and shifting units replace the division unit to reduce hardware costs.

5 EXPERIMENT RESULTS

5.1 Evaluation Setup

For algorithm evaluation, we select bert-base [4] and MRPC task from GLUE [20] as our baseline model and dataset. Our pruning algorithm is implemented based on Huggingface [21] and Pytorch 1.13.1 [13]. Accuracy is reported for the validation dataset. We finetune our model for 5 epochs with batch size of 24 using AdamW optimizer with learning rate of $2e-5$.

For hardware evaluation, in order to compare fairly with others' work, we test our method on Transformer using WikiText-2 dataset and Shallow Transformer models. For Shallow Transformer, there are two encoder layers and one decoder layer with hidden size $H = 200$, number of heads $A = 4$. We implement *FNM-Trans* on Xilinx Alveo U200 FPGA board via Vivado 2022.2.

5.2 Software Evaluation

We compare our algorithm with one-shot training algorithm under different attention sparsity and weight sparsity combinations. The pruning results are shown in Fig. 6. Compared with one-shot training, our algorithm can achieve significant accuracy improvements at all high sparsity levels. Under $1:16$ (93.75%) & $3:8$ (62.5%) sparsity level, our algorithm achieves up to 12.26% accuracy improvement. At very high sparsity of $1:16$ & $4:16$, our full sparse training algorithm only incurs 3.43% accuracy degradation, while compared to one-shot pruning, it improves by 11.03%. This is advantageous in situations where computational and storage resources are limited, and there is less emphasis on accuracy.

Another interesting observation is that our full sparse training algorithm can even improve accuracy over weight-only iterative pruning. For instance, under the condition of $2:16$ (87.5%) & $4:16$ (75%) full sparsity, our algorithm achieves 83.58% accuracy. However, under $4:16$ weight-only iterative pruning, keeping attention dense and all the other settings the same, we only achieve 81.13% accuracy. That is to say, under the $4:16$ weight sparsity, the model acts even better after we introduce $2:16$ attention sparsity(2.45% accuracy improvement). A probable explanation for this could be that

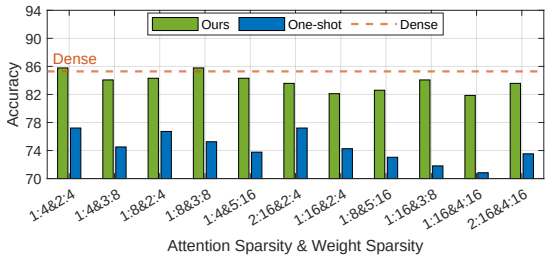

Figure 6: Comparison of pruning results on MRPC dataset between one-shot training and our custom full sparse training under different attention sparsity and weight sparsity combinations. $N_a:M_a$ & $N_l:M_l$ means the full sparse model is under $N_a:M_a$ attention sparsity and $N_l:M_l$ weight sparsity.

attention sparsity adapts to the impact caused by weight sparsity, helping to offset accuracy loss resulting from weight-only sparsity. We will further explore the combined effects of attention sparsity and weight sparsity in future work.

5.3 Hardware Evaluation

In this section, we evaluate hardware advantages of *FNM-Trans* by synthesizing custom hardware.

Resource Consumption. The hardware resource utilization and frequency are obtained from the generated report by synthesizing. We show the seperated resource consumption of *FNM-Trans* as shown in Table 2. *FNM-Trans* consumes 558K LUTs, 1507K FFs, 1581 BRAMs and 4096 DSPs.

Comparison with CPU and GPU. We compare *FNM-Trans* with CPU and GPU and evaluate the performance in terms of latency, throughput, power, energy efficiency, and MAC efficiency. As depicted in Table 1, both methodologies employed by *FNM-Trans* surpass the high-end CPU (i9-9900X), yielding a notable 5.17× and 27.16× improvement in latency. In comparison to GPU platforms, *FNM-Trans* demonstrates enhancements of 3.83×~203× in latency and 29.65×~106.08× in energy efficiency.

Comparison with Other Accelerators. We compare *FNM-Trans* with state-of-the-art sparse Transformer accelerator [14], [7] on FPGA, which employs Block-based pruning and $N:M$ group-based pruning for weight computation, as shown in Table. 1. Differing from [14] and [7] which do not optimize the attention computation and cannot accelerate different sparsity in one execution,

Table 2: Resource Consumption of *FNM-Trans*

	LUT	FF	BRAM	DSP
FNM-SA	438K	1,398K	-	4,096
Dynamic Attention	3.14K	2.33K	8	-
Length Adaptive Softmax	57.36K	69.66K	-	-
Others	59.42K	37.08K	1,573	-
Total	557.92K	1,507.07K	1,581	4,096

FNM-Trans accelerates all three Transformer computation modules through $N:M$ sparsity. Despite extra pruning introduced by dynamic attention, it is concurrent with other matrix operations, avoiding latency impact. *FNM-Trans* achieves a latency of 0.08*ms*, surpassing [14] and [7] by 1.88× and 4×, respectively, and achieves 1.25×~3.35× better MAC Efficiency. Due to the additional cost of the dynamic pruning engine, *FNM-Trans* consumes more power than other approaches.

5.4 System-Level Performance Evaluation

To validate the effectiveness of our optimization strategies with varying sequence lengths , we conduct further investigations. Four distinct groups are specifically set up: 1) Proposed *FNM-Trans* equipped with all optimization strategies, 2) $N:M$ Sparse Transformer with weight sparsity only, 3) $N:M$ Sparse Transformer with attention sparsity only, 4) Dense Transformer. Fig. 7 shows the breakdown of relative cycles of shallow Transformer encoder inference with increasing sequence length. The observations are summarized as follows:

• Experiment on dense data demonstrates that as the sequence length increases, the proportion for QKV generation and FFN decreases, while the proportion of attention-related computations increases. For shorter sequences, leveraging weight sparsity is more crucial. Conversely, for longer sequences, the use of attention sparsity proves to be more effective.

• In the case of $N:M$ Sparse Transformer with weight sparsity only, the proportion of attention operations is significant, growing from 9% (SL=32) to 51% (SL=2048).

• In the case of *FNM-Trans*, the proportion of attention operations is significantly reduced (up to 89.2% at SL=512), demonstrating the superior efficiency of *FNM-Trans* compared to $N:M$ Sparse Transformer with weight sparsity only.

• Thanks to the unification of attention sparsity and weight sparsity through $N:M$ Sparsity, *FNM-Trans* exhibits commendable performance in processing both long and short sequences. Compared to $N:M$ Sparse Transformer with weight sparsity only, there can be up to 62.4% speedup in total execution time.

From this detailed performance analysis, we conclude that *FNM-Trans* provides non-trivial benefits with various sequence lengths through $N:M$ Sparsity.

6 CONCLUSION

We propose an efficient framework for full $N{:}M$ sparse Transformers, enhancing performance across both software and hardware. On the software side, we develop a custom full sparse training algorithm to improve accuracy. On the hardware front, *FNM-Trans* incorporates FNM-SA, a dynamic attention module, and a length-adaptive softmax module, efficiently managing diverse N:M sparsity

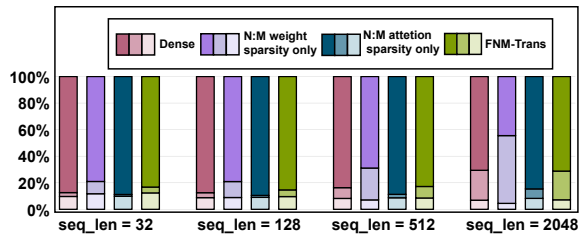

Figure 7: System-Level Performance Evaluation. From deep to shallow representation of QKV generation, attention and FFN.

patterns within a single execution. Demonstrating a significant accuracy improvement of up to 12.26% over one-shot training, our framework outperforms traditional CPUs, GPUs, and FPGA accelerators in key performance indicators, showcasing immense potential for accelerating long-sequence tokens.

ACKNOWLEDGEMENTS

This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFA100-3602, the Shanghai Pujiang Program under Grant 22PJD003 and the Fudan Undergraduate Research Opportunities Program grant (23115).

REFERENCES

[1] Yueyin Bai et al. 2023. LTrans-OPU: A Low-Latency FPGA-Based Overlay Processor for Transformer Networks. In *FPL*. 283–287.
[2] Jialin Cao et al. 2023. PP-Transformer: Enable Efficient Deployment of Transformers Through Pattern Pruning. In *ICCAD*. 1–9.
[3] Zhaodong Chen et al. 2023. Dynamic N: M fine-grained structured sparse attention mechanism. In *PPoPP*. 369–379.
[4] Jacob Devlin et al. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:2020.1810.04805* (2018).
[5] Alexey Dosovitskiy et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:11929* (2020).
[6] Chao Fang et al. 2022. An Efficient Hardware Accelerator for Sparse Transformer Neural Networks. In *ISCAS*. IEEE, 2670–2674.
[7] Chao Fang et al. 2022. An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers. *VLSI* 30, 11 (2022), 1573–1586.
[8] Tae Jun Ham et al. 2020. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *HPCA*. IEEE, 328–341.
[9] Bingbing Li et al. 2020. Ftrans: energy-efficient acceleration of transformers using fpga. In *ISLPED*. 175–180.
[10] Shiwei Liu et al. 2023. 16.2 A 28nm 53.8TOPS/W 8b Sparse Transformer Accelerator with In-Memory Butterfly Zero Skipper for Unstructured-Pruned NN and CIM-Based Local-Attention-Reusable Engine. In *ISSCC*. 250–252.
[11] Liqiang Lu et al. 2021. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO*. 977–991.
[12] Siyuan Lu et al. 2020. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. In *SOCC*. IEEE, 84–89.
[13] Adam Paszke et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS* 32 (2019).
[14] Hongwu Peng et al. 2021. Accelerating transformer-based deep learning models on fpgas using column balanced block pruning. In *ISQED*. IEEE, 142–148.
[15] Panjie Qi et al. 2021. Accommodating transformer onto fpga: Coupling the balanced model compression and fpga-implementation optimization. In *GLSVLSI*.
[16] Yubin Qin et al. 2023. FACT: FFN-Attention Co-optimized Transformer Architecture with Eager Correlation Prediction. In *ISCA*. 1–14.
[17] Guan Shen et al. 2022. SALO: an efficient spatial accelerator enabling hybrid sparse attention mechanisms for long sequences. In *DAC*. 571–576.
[18] Yuhong Song et al. 2021. Dancing along battery: Enabling transformer with run-time reconfigurability on mobile devices. In *DAC*. IEEE, 1003–1008.
[19] Ashish Vaswani et al. 2017. Attention is all you need. *NeurIPS* 30 (2017).
[20] Alex Wang et al. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).
[21] Thomas Wolf et al. 2020. Transformers: State-of-the-art natural language processing. In *EMNLP*. 38–45.
[22] Aojun Zhou et al. 2021. Learning n: m fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010* (2021).