Hardware-Aware Dataflow Accelerator for Sparse Transformers

Junyi Wu

1 Abstract

Transformer models have revolutionized natural language processing by achieving state-of-the-art performance across various tasks. However, the Transformer model's intensive matrix calculation and complex data present a significant challenge for its deployment on resource-constrained devices. Dataflow accelerators, an accelerator architecture, show excellent performance due to their high level of parallelism. Some of them exploit model sparsity to improve inference efficiency. However, existing work often overlooks the workload imbalances between different layers and modules of the transformer. To address this challenge, this research proposal aims to develop a novel dataflow accelerator capable of dynamically allocating hardware resources. Experiments were designed to evaluate and compare the proposed method against baseline platforms comprehensively. Building on previous work, this proposal is expected to achieve higher throughput and lower energy consumption compared to state-of-the-art accelerators, e.g. AccelTran [15].

2 Literature Review

2.1 Algorithmic

Significant efforts have been devoted to studying new algorithmic optimization methods, many focusing on Multi-Head Attention (MHA). Some researchers modified attention structures and sought higher hardware efficiency. [1] introduced grouped-query attention (GQA), which evenly splits query heads into groups, and each group shares a single key and value layer. This is further improved by AsymGQA [3], allowing more flexible grouping and improving model quality.

2.2 Quantization

Researchers have attempted to speed up training and inference by using quantization. This involves reducing the precision of weight and activation and achieving lower computation cost and memory footprint. BinaryBERT [2] compress BERT up to 24× smaller through weight binarization, significantly reducing model size while retaining performance, while [14] focus on compress Transformer-based generative pre-trained language models (PLMs), achieve 14.4× and 13.4× compression rates on GPT-2 and BART, respectively. Unlike previous methods, Dynamic Stashing Quantization (DSQ) [19] prioritizes memory traffic reduction over parameter compression. By aggressively quantizing intermediate results between the forward and backward passes, DRAM traffic is reduced, and the sweet spot for processors is reached.

2.3 Sparsity

Many studies have focused on exploiting sparsity in the Transformer model and avoiding unnecessary computation and memory usage with zeros, leading to faster execution and lower energy consumption. There are two types of sparsity: weight sparsity and activation sparsity. Weight sparsity is zeros in weight matrices of a model; Activation sparsity is zeros in intermediate activation value.

An overview of hardware transformer accelerators exploring sparsity can be seen in [5]. These methods can be broadly categorized based on pattern regularity into unstructured and structural sparsity; each can be coarse-grained (applied to larger units like blocks or heads) or fine-grained

(applied to individual elements like weights). In addition, each method is either static (fixed patterns applied during training or initialization) or dynamic (patterns that adapt during runtime). In particular, [9] introduces Dynamic Sparse Attention (DSA), which leverages dynamic, input-dependent sparsity in Transformer attention mechanisms. [4] propose Sparse Transformer (STA), utilizing N:M fine-grained structured sparsity [22], shows high efficiency and lossless performance.

2.4 Dataflow Accelerator

Among various accelerator architectures, dataflow accelerators have shown significant performance benefits because of their deeply pipelined computation between layers [17] and scalable data parallelism across devices [21]. HASS [20] explored a new sparse dataflow architecture for Deep Neural Networks (DNNs) and improved hardware resource utilization by dynamically adjusting each layer's parallelism. TransPIM [23] focuses on reducing inter-layer data movements by adopting a token-based dataflow and lightweight modifications in the conventional high bandwidth memory (HBM) architecture, improving efficiency significantly. STA [4] exploits N:M structured sparsity [22] on weights for efficient Transformers inference, designed a diverse matrix multiplication engine (DMME) to handle sparse matrix multiplication (MatMul), and a scalable softmax module for faster softmax calculation. [10] highlight that the complexity of Multi-Head Attention (MHA) and Feed-Forward Networks (FFN) is performance bottlenecks. They proposed partitioning the Query and Key matrices into segments to process using a systolic array (SA); this design allows hardware resource sharing between MHA and FFN ResBlock. Similarly, [15] adopts the idea of breaking down large matrix multiplications into smaller tiles to enhance parallelism. They also introduce a new Processing Element (PE) design capable of low-overhead activation pruning and efficient sparse matrix calculation, improving throughput and energy efficiency.

2.5 Research Gap

As pointed out in [20], the overall pipeline performance does not scale proportionally with the total non-zero operations in a sparse dataflow accelerator. Instead, the throughput is limited by the slowest layer. However, pruning methods often introduce imbalances across layers, affecting both parameter counts and computational demands. Even for structured sparsity approaches, variations in matrix sizes and layer types can still cause uneven distributions. This means this imbalance between layers can cause inefficiency in the entire network.

According to my knowledge, no existing work focuses on addressing such imbalance in the Transformer models. The accelerator designed by [10] is highly parallel and makes efficient use of hardware resources. However, this design failed to take advantage of network sparsity. In contrast, STA [4] and AccelTran [15] are optimized for sparse matrix calculation and have high parallelism, while STA is unable to reuse idle hardware efficiently. All three methods above lack the ability to allocate hardware resources across layers based on workload dynamically. To tackle this challenge, HASS [20] proposed dynamically adjusting parallelism according to the sparsity and processing speed of each layer. However, it's designed for convolutional networks and lacks the optimization for Transformer architecture.

The experiment results of HASS [20] have shown significant efficiency improvement, hence proved the feasibility of their algorithm. Additionally, other transformer accelerator designs have introduced various optimization modules. Therefore, I am well-equipped to address the remaining challenges. I plan to integrate the validated approach from HASS and the modular designs from these other works, then further modify modules and the system to ensure compatibility and performance optimization.

3 Methodology

3.1 Research Approach

A Vector Multiplier Design

As mentioned in [10, 16, 7], most of the trainable parameters and the computations are in the multi-head attention (MHA) ResBlock and the position-wise feed-forward network (FFN) ResBlock. To

perform matrix multiplication (MatMuls), we need to design a vector multiplier that meets the following standards:

- **Sparse Matrix Calculation:** Because we are optimizing for sparse networks, vector multiplier should be efficient at handling sparse matrix multiplication.
- **Manageable Size:** The number of vector multipliers per layer will change dynamically. A manageable size is needed to ensure parallelism and flexibility.
- **Hardware Resource Reuse:** To allow hardware resource reuse, the vector multiplier should be able to accelerate both FFN Resblock and MHA Resblock.

The design of the DMME in [4] efficiently handles sparse matrix multiplication, but the use of N layers of $S \times S$ systolic arrays is unnecessarily large. A more practical and streamlined approach would be partitioning matrices and employing multiple $S \times A$ systolic arrays, where S is sequence length and A is an adjustable parameter. [10] shows an SA of size $S \times 64$ is a good design. Through matrix partition and zero padding, SA can handle MatMuls in MHA and FFN ResBlock.

It's worth noticing that vector multiplier size can have different optimum values under different circumstances, so A should be optimized during Design Space Exploration (DSE).

An alternative approach involves modifying the Processing Element (PE) architecture, as detailed in AccelTran's design [15]. In this design, each PE contains multiple Multiply-Accumulate (MAC) lanes that handle tile-wise matrix multiplication in parallel. The design can be altered to support flexible parallelism, make the number of MAC lanes per PE adjustable, and adjust dynamically based on workload requirements.

However, MAC lanes alone do not provide inherent zero-skipping capabilities. In PE design, the DynaTran module induces weight and activation sparsity at runtime, and zero-skipping relies on pre-compute sparsity modules. These modules are particularly designed for simple threshold-based pruning. If alternative pruning methods, such as N:M structured sparsity, are desired, additional modifications would be needed to maintain efficiency.

B Other Modules Design

Apart from Vector Multiplier, some other modules need special designs to improve efficiency. [10] designed highly optimized modules for the two most complicated nonlinear functions, scaled masked-softmax and layer normalization. HASS [20] employed buffering to absorb the instantaneous variance of dynamic processing rates and determine the buffer size following a heuristic approach. With proper adjustments, these modules can be integrated into this accelerator.

C Sparsity Estimation and Initialization

We need to estimate each layer's sparsity to allocate hardware resources efficiently. This can be done by conducting static sparsity analysis of each layer at compile time [20]. Weight sparsity can be directly calculated at compile time, while activation sparsity depends on network input and can only be estimated. Then, we can estimate the computation workload using sparsity, type of layer, and input matrix size and initialize the number of Vector Multipliers in parallel for each layer accordingly.

D Design Space Exploration

The existing method HASS [20] begins DSE from a fully sequential design, then iteratively increases parallelism for the slowest layer and reduces parallelism of non-slowest layers. We can adopt the same algorithm in this design. Parallelism modification can be done by adding or removing extra Vector Multipliers. However, the process could be inefficient for transformer models due to their depth and computational complexity, and the number of iterations needed to reach optimum performance can be large. To enhance efficiency, we can initiate DSE with a predetermined level of parallelism set at compile time, which may reduce the number of iterations needed.

3.2 Experimental Setup

This section presents the details of the experiment setup, including the model, dataset, and baselines considered for comparison.

A Models and Datasets

In terms of model, TinyBERT [6] and shallow Transformer model [8, 12] are good choices. They are all lightweight models, which are good for testing on FPGA and covering translation and language tasks. Hence, they can test the acceleration effect from different aspects.

SST-2 [18] and SQuAD-v2 [13] datasets can be used for testing, both of which are popular benchmarking datasets. SST-2 test model performance on sentiment analysis tasks, while SQuAD-v2 test question-answering task. These are also the datasets used by [15], which makes it convenient for baseline comparison.

B Pruning Method

There are many choices for pruning algorithms, which are detailed in Section A. To start with, we can use AccelTran's [15] pruning method, which involves iterative magnitude-based pruning for both weight and activation. This method is likely to be suitable because AccelTran is also a dataflow accelerator for the transformer model, and this algorithm achieves a balance between simplicity and accuracy.

C Evaluation Baseline

This research aims to improve model efficiency on mobile platforms. We can compare this method with standard mobile platforms, e.g., Raspberry Pi, Apple M1 Chip, and Qualcomm Snapdragon. Other FPGA-base accelerators can also be compared. e.g. AccelTran-Edge [15], STA [4] and [10].

3.3 Data Collection & Analysis Methods

This section presents possible experiments needed to analyze this accelerator.

A Comparison of Pruning Methods

Evaluating the model's performance on the accelerator using different pruning methods is crucial. This evaluation will help determine the optimal pruning method. Model accuracy and throughput of each configuration should be measured.

1. **Weight or Sparsity:** Pruning both weights and activations offers higher speedup and energy efficiency but introduces greater complexity and potential accuracy loss. Pruning only weights simplifies implementation with less accuracy degradation, though it limits performance gains compared to pruning activations as well.
2. **Pruning Algorithms:** Magnitude-based pruning, known for its simplicity, was applied in [15, 20]. N:M fine-grained structured sparsity [22] has been shown by [4, 22] to effectively accelerate networks without sacrificing performance.
3. **Pruning Strategies:** One-shot pruning simplifies deployment by completing the process in a single step. Iterative pruning, though more time-consuming and may have lower net sparsity, often achieves better performance.

B Design Space Exploration

Compare the number of compute and memory stalls for accelerators with varying parameters. These include Vector Multiplier sizes, levels of parallelism increase per iteration, and different buffer sizes. The goal is to find the optimal balance between latency (minimizing stalls) and hardware resources (Vector Multiplier and buffer size).

1. **Vector Multiplier Sizes:** Using a larger vector multiplier can improve parallelism and throughput. However, this can also lead to increased hardware resource consumption and potential underutilization. Hence, the size of the vector multiplier should be determined by the overall sparsity of the network and the dimensions of the input tensor. These factors directly influence the computation workload in the FFN and MHA layers.

2. **Increase of parallelism:** A higher level of parallelism per iteration can lead to faster convergence to an optimal design. However, it may also result in a higher risk of over-provisioning hardware resources, leading to underutilization.

3. **Buffer Sizes:** A larger buffer size can lead to high throughput at the expense of resource usage. We can determine the buffer size following a heuristic approach similar to [11].

C Hardware Utilization

Analyzing each module's utilization and power consumption during inference is crucial for optimizing performance and resource efficiency.

1. **Utilization Analysis:** Tracking the utilization of buffers and modules over time helps determine whether hardware resources are fully utilized. This allows for identifying inefficiencies and potential optimization.

2. **Power Analysis**: Tracking buffers' and modules' power consumption over time is also necessary. Although power optimization is not the primary focus of this research, it can prevent significant power wastage, as the accelerator is designed for mobile platforms.

D Performance Improvements

To validate performance improvements, we must compare this accelerator's performance against baseline platforms. Measurement should include normalized throughput and energy consumption of the same network on different platforms.

E Ablation Analysis

Ablation studies can quantify each module's contribution by removing or modifying specific components and measuring the impact on throughput, energy consumption, and hardware utilization.

1. **Vector Multiplier:** Replace the vector multiplier with a fixed-size systolic array that lacks optimization for sparse matrix calculation.

2. **Resource Allocation:** Disable dynamic resource allocation. Use a static configuration for all layers instead.

3. **Default Configuration:** Start from a fully sequential design without parallelism rather than using the predetermined parallelism level set at compile time.

3.4 Potential Challenges

Transformers are difficult to map onto a single device due to their size, exceeding the hardware resources. While prior work [20] on DNNs addressed this using block-level folding and iterative reconfiguration, this method may not be efficient for the transformer base model due to the larger model size and complexity.

4 Expected Results

This research introduces a dynamic hardware resource allocation strategy to address inefficiencies caused by layer-wise computation imbalances in transformer models. The proposed architecture is expected to optimize transformer performance on edge and mobile platforms, with the following anticipated outcomes:

- Significant reduction in inference time compared to existing baseline platforms.
- Improved energy efficiency through enhanced hardware utilization.

References

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.

[2] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. Binarybert: Pushing the limit of bert quantization, 2021.

[3] Yuang Chen, Cheng Zhang, Xitong Gao, Robert D. Mullins, George A. Constantinides, and Yiren Zhao. Optimised grouped-query attention mechanism for transformers. *ArXiv*, abs/2406.14963, 2024.

[4] Chao Fang, Shouliang Guo, Wei Wu, Jun Lin, Zhongfeng Wang, Ming Kai Hsu, and Lingzhi Liu. An efficient hardware accelerator for sparse transformer neural networks. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2670–2674, 2022.

[5] Kazi Ahmed Asif Fuad and Lizhong Chen. A survey on sparsity exploration in transformer-based accelerators. *Electronics*, 12(10), 2023.

[6] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding, 2020.

[7] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[8] Bingbing Li, Santosh Pandey, Haowen Fang, Yanjun Lyv, Ji Li, Jieyang Chen, Mimi Xie, Lipeng Wan, Hang Liu, and Caiwen Ding. Ftrans: Energy-efficient acceleration of transformers using fpga, 2020.

[9] Liu Liu, Zheng Qu, Zhaodong Chen, Fengbin Tu, Yufei Ding, and Yuan Xie. Dynamic sparse attention for scalable transformer acceleration. *IEEE Transactions on Computers*, 71(12):3165–3178, 2022.

[10] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. In *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*, pages 84–89, 2020.

[11] Alexander Montgomerie-Corcoran, Zhewen Yu, Jianyi Cheng, and Christos-Savvas Bouganis. Pass: Exploiting post-activation sparsity in streaming architectures for cnn acceleration, 2023.

[12] Hongwu Peng, Shaoyi Huang, Tong Geng, Ang Li, Weiwen Jiang, Hang Liu, Shusen Wang, and Caiwen Ding. Accelerating transformer-based deep learning models on fpgas using column balanced block pruning. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 142–148, 2021.

[13] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[14] Chaofan Tao, Lu Hou, Wei Zhang, Lifeng Shang, Xin Jiang, Qun Liu, Ping Luo, and Ngai Wong. Compression of generative pre-trained language models via quantization, 2022.

[15] Shikhar Tuli and Niraj K. Jha. Acceltran: A sparsity-aware accelerator for dynamic inference with transformers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(11):4038–4051, 2023.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[17] Stylianos I. Venieris and Christos-Savvas Bouganis. fpgaconvnet: A framework for mapping convolutional neural networks on fpgas. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47, 2016.

[18] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

[19] Guo Yang, Daniel Lo, Robert Mullins, and Yiren Zhao. Dynamic stashing quantization for efficient transformer training, 2023.

[20] Zhewen Yu, Sudarshan Sreeram, Krish Agrawal, Junyi Wu, Alexander Montgomerie-Corcoran, Cheng Zhang, Jianyi Cheng, Christos-Savvas Bouganis, and Yiren Zhao. Hass: Hardware-aware sparsity search for dataflow dnn accelerator, 2024.

[21] Yaqi Zhang, Nathan Zhang, Tian Zhao, Matt Vilim, Muhammad Shahbaz, and Kunle Olukotun. Sara: Scaling a reconfigurable dataflow accelerator. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1041–1054, 2021.

[22] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch, 2021.

[23] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1071–1085, 2022.