LightMamba: Efficient Mamba Acceleration on FPGA with Quantization and Hardware Co-design

Renjie Wei[12*], Songqiang Xu[5*], Linfeng Zhong[6*], Zebin Yang[12], Qingyu Guo[2],
Yuan Wang[23], Runsheng Wang[234] and Meng Li[123†]
[1]*Institute for Artificial Intelligence &* [2]*School of Integrated Circuits, Peking University, Beijing, China*
[3]*Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China*
[4]*Institute of Electronic Design Automation, Peking University, Wuxi, China*
[5]*School of Software and Microelectronics, Peking University, Beijing, China*
[6]*School of Electronic and Computer Engineering, Peking University, Shenzhen, China*

***Abstract*—State space models (SSMs) like Mamba have recently attracted much attention. Compared to Transformer-based large language models (LLMs), Mamba achieves linear computation complexity with the sequence length and demonstrates superior performance. However, Mamba is hard to accelerate due to the scattered activation outliers and the complex computation dependency, rendering existing LLM accelerators inefficient. In this paper, we propose LightMamba that co-designs the quantization algorithm and FPGA accelerator architecture for efficient Mamba inference. We first propose an FPGA-friendly post-training quantization algorithm that features rotation-assisted quantization and power-of-two SSM quantization to reduce the majority of computation to 4-bit. We further design an FPGA accelerator that partially unrolls the Mamba computation to balance the efficiency and hardware costs. Through computation reordering as well as fine-grained tiling and fusion, the hardware utilization and memory efficiency of the accelerator get drastically improved. We implement LightMamba on Xilinx Versal VCK190 FPGA and achieve 4.65∼6.06× higher energy efficiency over the GPU baseline. When evaluated on Alveo U280 FPGA, LightMamba reaches 93 tokens/s, which is 1.43× that of the GPU baseline.**

***Index Terms*—Mamba, rotation-assisted quantization, FPGA accelerator, computation reordering, fine-grained tiling and fusion**

I. Introduction

State space models (SSMs) like Mamba [1], [2] have recently been proposed and emerged as a promising class of architectures as foundation models. Compared to existing Transformer-based large language models (LLMs) [3]–[6], Mamba only requires linear computational complexity with the increase of input sequence length while demonstrating superior performance on various downstream tasks.

The basic architecture of Mamba [2] is shown in Fig. 1. Each Mamba block mainly consists of two linear projection layers, i.e., input projection and output projection, a 1-dimensional convolution (conv1d) layer, and an SSM layer. The computation of Mamba involves a prefill stage that summarizes the input prompts and an autoregressive decode stage to produce the output tokens. The decode of Mamba only generates and stores a fixed-size hidden state instead of a key-value cache in Transformers that grows linearly with the sequence length. While Mamba has different architectures, we focus on the Mamba2 [2] architecture and all subsequent references to Mamba refer to Mamba2 unless otherwise specified.

This work was supported in part by National Natural Science Foundation of China under Grant 62495102 and Grant 92464104, in part by Beijing Municipal Science and Technology Program under Grant Z241100004224015, and in part by 111 Project under Grant B18001.

*Equal contribution. †Corresponding author.

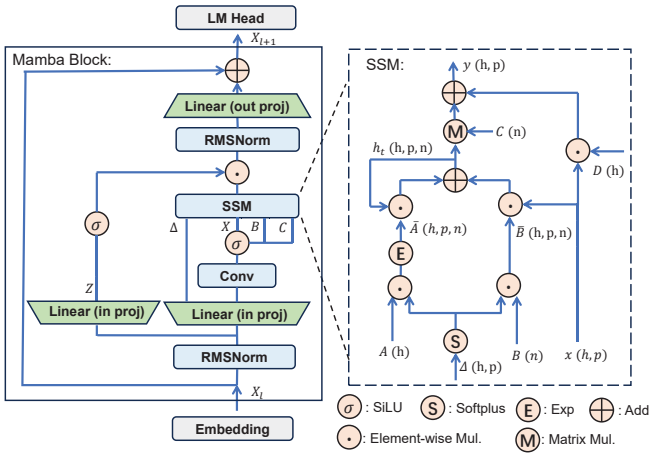


Fig. 1: The model architecture of Mamba2 and the detailed computation graph of the SSM layer.

Though promising, how to accelerate Mamba processing, especially on reconfigurable platforms like FPGA, remains an open question. We observe existing works on FPGA-based LLM acceleration [7], [8] cannot be directly applied due to the following challenges. First, while FPGA-based acceleration benefits from low-bit-precision quantization, Mamba quantization is more challenging than Transformer-based LLMs due to scattered activation outliers. Specifically, for different input tokens, the activation outliers appear in different channels. Naively applying existing quantization algorithms [9]–[11] incurs significant accuracy degradation. Second, an SSM layer involves excessive element-wise computation. While existing works [12]–[14] often ignore the quantization of SSM layers and suffer from a high hardware cost, directly quantizing the SSM layer also introduces significant re-quantization overhead. Third, the SSM layer involves complex computation and data dependency, which prevents unfolding the SSM computation

978-3-9826741-0-0/DATE25/© 2025 EDAA

spatially to improve the acceleration throughput.

To solve these challenges, we propose LightMamba, the first FPGA-based Mamba acceleration framework that co-designs the quantization algorithm and accelerator architecture. We propose a rotation-assisted post-training quantization (PTQ) algorithm for Mamba to remove the scattered outliers, which enables us to quantize the model to 4-bit with minimum accuracy degradation and significantly improve the communication efficiency with off-chip memory. For the SSM layer, we propose a FPGA-friendly power-of-two (PoT) quantization scheme to realize re-quantization with simple shifting for better computation efficiency. For the FPGA accelerator architecture, we design customized rotation modules for the PTQ algorithm and further propose computation reordering as well as fine-grained tiling and fusion to improve the hardware utilization. Our main contributions can be summarized as follows:

- We propose the first PTQ algorithm for the entire Mamba model. Through rotation-assisted quantization, we quantize Mamba to 4-bit with plausible accuracy. The SSM layer is also quantized with FPGA-friendly PoT scheme for better computation efficiency.
- We propose the first FPGA-based Mamba accelerator. The architecture features a customized rotation module co-designed with the proposed quantization algorithm. We also propose computation reordering and fine-grained scheduling to improve the computation throughput and reduce the on-chip memory usage.
- We implement LightMamba on Xilinx Versal VCK190 FPGA achieving 7.21 tokens/s and 4.65∼6.06× higher energy efficiency over the GPU baseline.

II. Related Works

A. LLM Quantization

Quantization maps the weights and activations from high-bit-precision floating point (FP) numbers, e.g., FP16 to low-bit integers, e.g., INT8 or INT4, reducing both memory and computation costs. Existing algorithms mostly leverage PTQ for LLMs and observe the key obstacle comes from the outliers in weights and activations. LLM.int8() [15] uses mix-precision quantization and keeps outliers in high bit-precision, which introduces large computation overhead. SmoothQuant [9] and Outlier Suppression [10], [11] observe the outliers persist in specific channels and thus, calculate the channel-wise scaling or shifting factors to rescale the outliers before quantization. QuaRot [16] and SpinQuant [17] multiply the weight and activation with a rotation matrix to remove outliers. However, these methods are only demonstrated for Transformer-based LLMs. For Mamba, [12] only quantizes the linear layers with round-to-nearest (RTN) method. Mamba-PTQ [13] simply applies SmoothQuant on Mamba. Although they only quantize the linear layers and leave the SSM layer in FP, they still suffer from large accuracy degradation.

B. FPGA-based LLM Accelerators

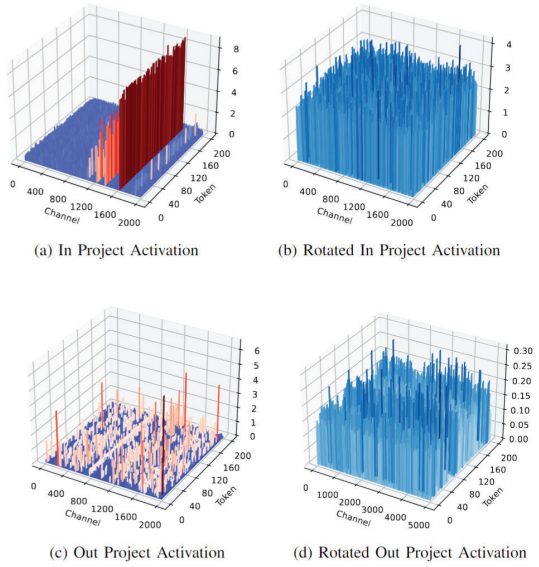
Previous FPGA-based LLM accelerators can be categorized into two types: temporal architecture and spatial architecture.

TABLE I: Qualitative comparison between different paradigms. W4A4 indicates 4-bit weight and 4-bit activation.

	[19]	[7] [8]	Ours
Architecture	Spatial	Temporal	Partial Spat.
Model	Transformer	Transformer	Mamba
Bit Precision	W4A8	W3.5A8 or FP16	W4A4
Latency	Low	High	Low
EM Compatibility	✓	×	✓
MM parallelism	Mid	High	High

TABLE II: 4-bit quantization error of the activation in the out project layer in Mamba2-2.7B with different PTQ methods.

Method	RTN	SQ [9]	OS+ [11]	Ours
Quantization Error	19.5	18.8	309.8	13.1



(a) In Project Activation (b) Rotated In Project Activation

(c) Out Project Activation (d) Rotated Out Project Activation

Fig. 2: Activation distribution in Mamba2-2.7B before and after rotation.

Temporal architecture constructs a processing engine (PE) performing various tasks, especially for matrix multiplication (MM) [7], [8], [18]. This architecture is not well-suited for handling the diverse and complex element-wise multiplications (EM) in Mamba. Spatial architecture customizes PEs for different operations and supports concurrent processing of multiple PEs in pipeline [19]–[22], leading to low latency. However, due to the custom PEs dispersing resources, this design results in lower parallelism for MM. In this paper, we adopt a partially unfolded spatial architecture that unfolds one Mamba block and co-designs the quantization algorithm and architecture to improve hardware utilization and reduce the latency. A comparative analysis of these architectures is presented in Table I.

III. Motivation

Challenge 1: Mamba is difficult to quantize to low bit precision because of the scattered outliers. We find that the activation outliers of the output projection layer exhibit scattered distribu-

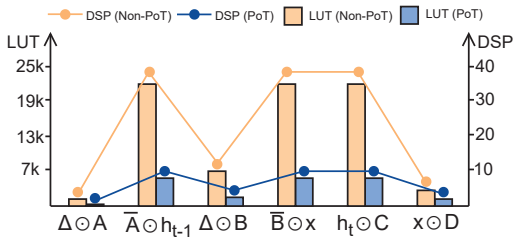
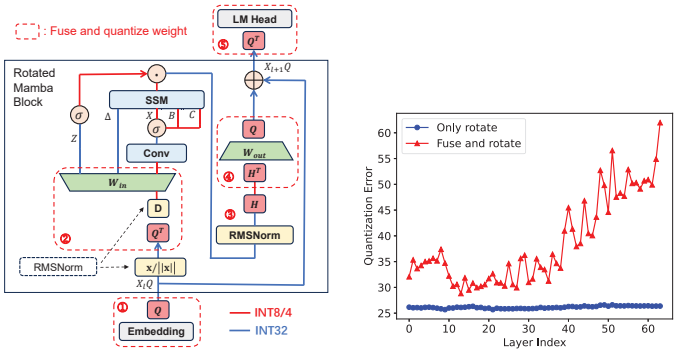


Fig. 3: The hardware cost of different operations in the SSM layer with naive Non-PoT quantization and PoT quantization.



(a) (b)

Fig. 4: (a) The proposed rotation-assisted quantization algorithm. Both Q and H are Hadamard matrices to ensure computation correctness. (b) Quantization error of the output projection weight after only rotation or fusion and rotation.

tion as shown in Fig. 2(c), which renders previous LLM quantization [9]–[11] less effective. This is because in Transformer-based LLMs, outliers persist in fixed channels across different tokens. Hence, it is possible to calculate channel-wise scaling or shifting factors to reduce the magnitude of outlier channels and reduce the quantization error [9]. For Mamba, activation outliers may show up in different channels for different input tokens and may scatter across all channels. As a result, prior art methods, e.g., SmoothQuant [9] and OutlierSuppression+ [10], have comparable or even larger quantization error compared to the baseline RTN quantization as in Table II. Rotation-based quantization algorithms have also been demonstrated for Transformer-based LLMs [16], [17] and do not require outliers to persist in fixed channels. However, as rotation-based algorithms usually require operator fusion and introduce extra computation overhead, it is unclear how it can be applied to Mamba acceleration on FPGAs.

Challenge 2: Existing works [12]–[14] often choose not to quantize the SSM layer in Mamba. The floating-point (FP) computation of SSM layers introduces large hardware cost on FPGA. Moreover, directly quantizing SSM layers to lower bit precision also incurs large hardware cost as shown in Fig. 3. This is because the SSM layer comprises excessive EMs, the re-quantization (transforming output from high bit precision back to low bit precision) overhead of which is significantly large.

Challenge 3: The computation graph of SSM layer is complex, which increases the difficulty of FPGA-based acceleration. On one hand, the inputs of the SSM layer, i.e., X, B, C, Δ, are all generated by the input projection layer. Even though it is possible to unroll the Mamba computation spatially on FPGA, the computation of the input projection and SSM layers are forced to be sequential, leading to only less than 60% hardware utilization. On the other hand, storing the intermediate activations of SSM on chip is memory-consuming, accounting for more than 70% of the total URAM usage.

To address these challenges, we propose LightMamba, an efficient FPGA-based Mamba acceleration framework. Specifically, we propose an FPGA-friendly Mamba quantization algorithm in Sec. IV, featuring a rotation-assisted PTQ algorithm to mitigate the scattered outliers and a PoT-based SSM quantization with minimum re-quantization overhead. We also propose an FPGA-based accelerator in Sec. V, featuring computation reordering to improve the hardware utilization and fine-grained tiling and fusion to reduce the on-chip memory cost.

IV. FPGA-friendly Quantization Algorithm

A. Rotation-assisted Linear Layer Quantization

The rotation-assisted quantization method is first proposed in [16] for Transformer-based LLMs. By multiplying the activation X and weight W with orthogonal matrix Q, i.e., XQQ^TW, the result is identical with XW, while the outliers in X and W are removed. However, it is still unclear whether the rotation method is applicable to Mamba. Therefore, we study the rotation equivalence in Mamba and propose a rotation-assisted method shown in Fig. 4a.

We observe that the activations in the linear layers and SSM layer have large number of outliers, and the outliers of output projection layer exhibit scattered distribution across different channels. Rotation is helpful to remove outliers since it amortizes large outliers with other elements. For the input and output projection layers, we apply rotation and remove the outliers as shown in Fig. 2. It is worth noting that to rotate the activation before the output projection layer we insert an online Hadamard transformation before it in Fig. 4a, which can be efficiently performed by our customized rotation unit in Sec. V. However, we find that SSM cannot be rotated since it does not satisfy the rotation equivalence. Specifically, the original computation in SSM is Eq. 1a. Assuming we can rotate hidden state h_t to remove the outliers, i.e., multiply h_t by Hadamard matrix H, we can derive Eq. 1b and Eq. 1c. However, Eq. 1c cannot derive Eq. 1d because EM does not satisfy matrix associative property. Thus we cannot derive Eq. 1d from Eq. 1a, i.e., SSM does not satisfy the rotation equivalence.

$$h_t = \bar{A} \odot h_{t-1} + \bar{B} \odot X_t \tag{1a}$$
$$h_t H = (\bar{A} \odot h_{t-1} + \bar{B} \odot X_t) H \tag{1b}$$
$$h_t H = (\bar{A} \odot h_{t-1}) H + (\bar{B} \odot X_t) H \tag{1c}$$
$$h_t H = \bar{A} \odot (h_{t-1} H) + \bar{B} \odot (X_t H) \tag{1d}$$

To reduce the computation overhead, we try to fuse rotation with neighboring operations as much as possible. As shown in Fig. 4a, we can fuse the first rotation with the embedding table

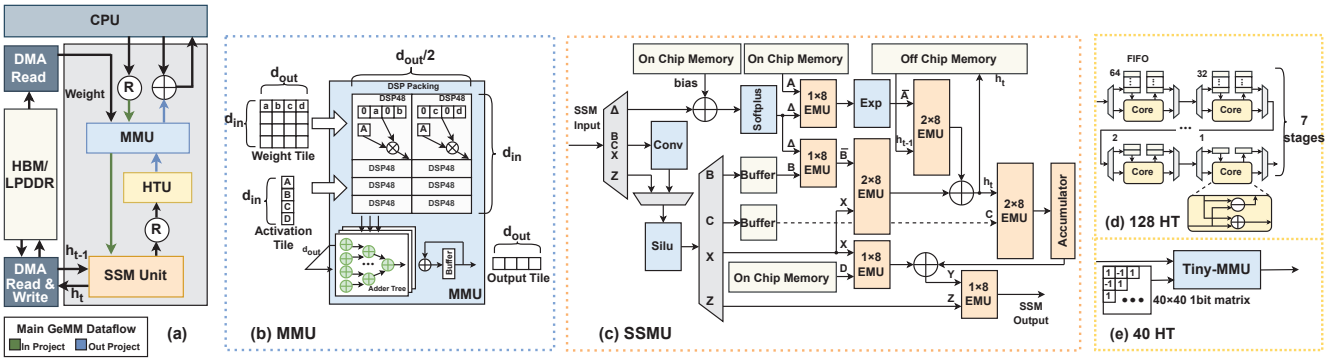


Fig. 5: Diagram of (a) the overall architecture, (b) SSMU, (c) MMU, (d) 128-point HTU, and (e) 40-point HTU.

(i.e., ①), the last rotation with the LM head (i.e., ⑤), as well as the rotations before and after the output projection layers in each Mamba block (i.e., ④). For the rotation next to the first RMSNorm operator (i.e., ②), to ensure the computational invariance, we need to split the scaling factor, i.e., D, of the RMSNorm first, and then, fuse it with the weights of input projection. For the rotation next to the second RMSNorm operator (i.e., ③), we find whether or not to fuse the scaling factor of the RMSNorm to the weight of output projection does not impact the computational invariance, while fusion introduces a larger quantization error as in Fig. 4b. Hence, we choose not to fuse the scaling factor of the second RMSNorm. In our algorithm, only rotation ③ needs to be computed online, which incurs small computation overhead with our customized FPGA module support.

B. FPGA-friendly SSM Quantization

In order to quantize SSM to reduce the heavy hardware cost by FP computations, we leverage INT8 per-group quantization to strike a balance between accuracy and hardware efficiency. However, directly quantizing SSM introduces large re-quantization overhead as shown in Fig. 3, which is because EM has larger re-quantization overhead than MM intrinsically since there is no reduction in EM. To this end, we propose to use PoT quantization for SSM, through which re-quantization can be implemented in bit-shifting rather than multiplication thus reducing the re-quantization overhead significantly.

V. FPGA-based Mamba Acceleration

A. Overall Hardware and Hadamard Transform Unit

The overall architecture of LightMamba is shown in Fig. 5 (a). We design a partially unfolded spatial architecture and unroll the computation of one Mamba block on the FPGA. The model parameters are stored in the off-chip DRAM. LightMamba consists of three main modules: (1) Matrix Multiplication Unit (MMU), which handles the computations of all linear layers, including both input and output projection; (2) SSM Unit (SSMU), which fully unfolds SSM to enable pipelined execution; and (3) Hadamard Transform Unit (HTU), a customized design to support rotation-assisted quantization.

MMU is designed to support input and output projection layers in a time-multiplexed manner. It features a tree-based architecture of multiplier-accumulators (MACs) that receive vectors of d_{in} dimension as inputs. It is also equipped with d_{out} lanes, which performs $d_{in} \times d_{out}$ MACs in one cycle. Altogether, MMU contains $d_{in} \times d_{out}$ MAC units, which are efficiently implemented using $d_{in} \times d_{out}/2$ DSPs, leveraging the DSP packing technique [23], as illustrated in Fig. 5(b).

SSMU features a fine-grained, fully pipelined architecture for computing the SSM layer. As shown in Fig. 5(c), each operator is implemented by a dedicated unit, connected via first-in-first-out buffers (FIFOs). We optimize the parallelism for each operator to ensure a balanced data flow with a minimum FIFO depth. We implement the operators in SSM through Element-wise Multiplication Units (EMUs) which are composed of DSPs. We set different parallelism for different operators.

HTU is dedicated to support the Hadamard transformation in our quantization algorithm. It has two variants: the power-of-two and the non-power-of-two type. For example, in Mamba-2.7B, two types of HTU are required, i.e, 128-point HTU and 40-point HTU as in Fig. 5(d) and Fig. 5(e). The 128-point HTU is based on the Fast Hadamard Transformation (FHT) algorithm [24]. It contains seven stages, each containing a Butterfly Core and two FIFOs. In the first stage, the first 64 elements are pushed into the input FIFO. When the next 64 elements arrive, they are processed in pairs by the Butterfly Core. Outputs are either sent to the next stage or buffered in the output FIFO. Compared to the MM-based Hadamard transform, this design reduces latency by 72% with the same hardware resources. For the small 40-point Hadamard transformation, we directly implement it with a simple MMU and fix one input to the Hadamard matrix with only 1 and -1.

B. Computation Reordering

Due to the distinct computational patterns and the data dependency between SSM and input projection layer, they are forced to execute sequentially as in Fig. 6(a). However, we find that SSM comprises multiple independent heads, and propose a coarse-grained pipeline to improve hardware utilization as in Fig. 6(b). This pipeline design depends on our proposed computation reordering method, which alters the data generation sequence in the input projection layer. Specifically, Δ, B, C are generated first and stored in an on-chip buffer, while X and Z are produced alternatively for computing SSM

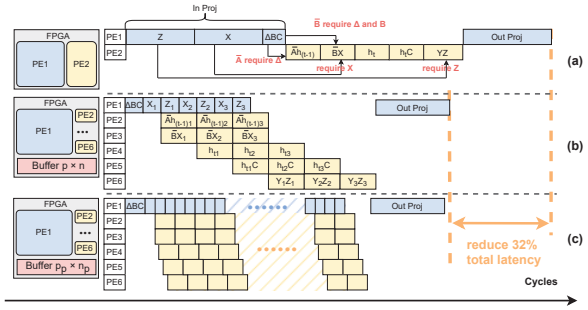


Fig. 6: Pipeline scheme: (a) Naive implementation, (b) Coarse-grained pipeline, (c) Fine-grained pipeline.

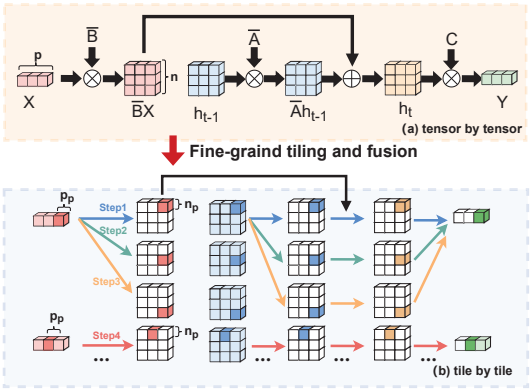


Fig. 7: Fine-grained tiling and fusion.

head-by-head, as shown in Figure 6(b). This reordering allows the SSM computation to begin immediately after Δ, B, C are produced in the MMU. Compared to the traditional sequential implementation, our approach reduces the total computation time of the network by 32% and increases hardware utilization from 58% to 96%, as depicted in Figure 6(b).

C. Fine-grained Tiling and Fusion

Given the current design, we observe the SSMU consumes more than 70% of the on-chip memory as it requires to store all intermediate activations, e.g., $\bar{B}X$, $\bar{A}h_{t-1}$, and h_t, etc, as in Fig.7(a). We propose a fine-grained tiling and fusion strategy. By leveraging operation fusion, we directly feed the output of the previous operator to the next operator, and thus eliminating the on-chip communication and data handling. Additionally, fine-grained tiling is employed to reduce the buffer size. We tile along the head and hidden state dimensions, with a tile size of $n_p \times p_p$ as in Fig. 7(b). The tile-by-tile implementation refines the execution of SSMU to enable the fine-grained pipeline in Fig. 6(c), which reduces the URAM usage of SSMU by 4×. This not only lower the on-chip buffer requirements, but also eliminates the pipeline bubbles for better hardware utilization.

VI. Experiments

A. Experiment Setup

Algorithm We evaluate our proposed quantization algorithm on the Mamba model family [2]. We quantize the entire model

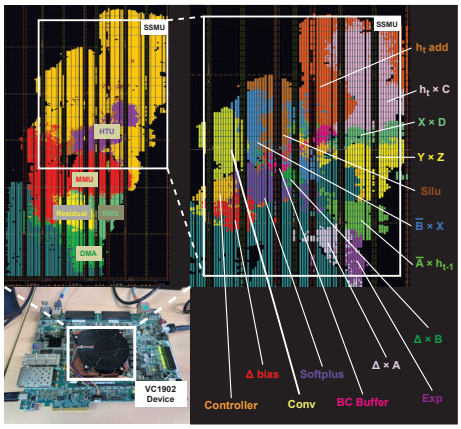


Fig. 8: LightMamba implementation layout on VCK190.

shown in Fig. 4a for hardware efficiency. We use per-channel weight quantization and per-token activation quantization for 8-bit weight and activation quantization (denoted as W8A8). We use per-group weight and activation (group size=128) quantization for 4-bit quantization (denoted as W4A4). We evaluate the perplexity and the zero-shot accuracy on six tasks: LAMBADA [25], HellaSwag [26], PIQA [27], Arc (Easy and Challenge) [28], Winogrande [29], and OpenbookQA [30] using lm-eval-harness [31].

Hardware We use two FPGA platforms for evaluation: Xilinx Versal VCK190 and Alveo U280. We implement LightMamba on VCK190 using Vitis HLS and Vivado Design FLow. Fig. 8 shows the layout of our implementation on VCK190 FPGA. We measure the throughput on-board using the PYNQ framework and power consumption with the Xilinx BEAM tool. For U280 evaluation, we develop a cycle-accurate simulator, which has been verified through HLS emulation using Vitis 2023.2. We choose NVIDIA RTX 2070 and RTX 4090 as our GPU baselines. We use NVIDIA system management interface for power measurements. Table IV shows the hardware parameters of FPGA and GPU, as well as the detailed hardware utilization of LightMamba.

B. Evaluation Result

Algorithm Evaluation We compare our quantization algorithm with the prior art weight-activation PTQ methods SmoothQuant (SQ) [9] and Outlier Suppression+ (OS+) [11] for Transformer-based LLM. Note that these methods can only be applied to linear layers. We re-implement them on Mamba and use 128 random samples from WikiText2 [32] dataset as the calibration data. We evaluate our method LightMamba which quantizes only linear layers, and LightMamba* which quantizes all modules including SSM. As shown in Table III, for W8A8 quantization, LightMamba and LightMamba* have negligible accuracy loss compared to the FP16 model. Although OS+ has better perplexity on the Lambada dataset, it collapses on W4A4. While our methods LightMamba and LightMamba* outperform all other methods on W4A4, improving the perplexity by 1.78 and 1.91 compared to the prior art method SQ, respectively.

TABLE III: Performance comparison of different methods on Mamba2-2.7B. Only LightMamba* quantizes the entire model including SSM while others only quantize linear layers. The **bold** denotes the best and the underlined denotes the second-best.

Method	Bit-precision	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	Winogrande acc ↑	OpenbookQA acc ↑	Average acc ↑
FP16	-	4.10	69.7	66.6	76.4	69.6	36.4	64.0	38.8	60.2
RTN	W8A8	4.26	68.8	66.1	75.8	68.4	36.4	63.6	38.4	59.6
SQ	W8A8	4.28	68.2	66.0	75.9	69.1	37.0	63.4	38.2	59.7
OS+	W8A8	**4.01**	69.9	66.2	76.4	69.5	36.5	63.4	39.0	60.1
LightMamba	W8A8	4.07	69.7	66.5	76.1	69.3	36.9	64.0	38.8	**60.2**
LightMamba*	W8A8	4.03	70.2	66.2	76.1	69.4	36.1	64.6	38.6	**60.2**
RTN	W4A4	17.46	37.7	62.6	70.1	60.1	34.5	57.7	38.2	51.6
SQ	W4A4	8.26	53.4	64.0	73.6	63.7	35.1	59.4	39.0	55.5
OS+	W4A4	> 100	0.0	27.7	54.5	30.8	24.7	48.8	25.6	30.3
LightMamba	W4A4	6.48	57.3	62.7	73.5	65.5	35.3	60.7	37.6	**56.3**
LightMamba*	W4A4	**6.35**	59.6	62.4	74.4	64.7	34.3	59.9	36.4	55.9

TABLE IV: Hardware comparison with GPU.

	LightMamba			GPU Baseline	
Platforms	VCK190	VCK190	U280	RTX 2070	RTX 4090
Frequency	400MHz	400MHz	200MHz	1.62GHz	2.52GHz
Bandwidth	12GB/s	12GB/s	460GB/s	468GB/s	1008GB/s
Precision	W4A4	W8A8	W4A4	FP16	FP16
LUT	107k	111k	297k	-	-
FF	130k	134k	394k	-	-
DSP	228	228	1164	-	-
BRAM	912	914	912	-	-
URAM	61	61	61	-	-
Througput	7.21	3.61	**93**	65	138
Energy Eff.	**2.25**	**1.45**	-	0.371	0.484

Hardware Evaluation We compare the decoding throughput of GPUs, prior art accelerators, and our LightMamba on different output sequence lengths in Fig. 9(a). As prior art accelerators have not supported Mamba, we compare their performance when running Transformer-based LLMs. Since these works did not provide throughput data for long sequence length, we simulated their performance based on the parameters in each paper. On VCK190, LightMamba achieves the practical throughput of 3.61 and 7.21 tokens/s for W8A8 and W4A4, respectively. We also simulate LightMamba on U280 for fair comparison. The throughput of LightMamba achieves **93** tokens/s, which outperforms RTX 2070 by **1.43**× on average. LightMamba achieves more significant acceleration on long sequences as Mamba only records hidden states of a fixed size.

We compare the energy efficiency (Tokens/J) of LightMamba and GPU on different model sizes in Fig. 9(b). LightMamba on VCK190 consistently outperforms GPUs and achieves on average **6.06**× and **4.65**× improvement over RTX 2070 and 4090 GPU, respectively. For small Mamba models, achieves more energy saving as our design reduces the overhead of the SSM layer, which is more costly for small models.

C. Ablation Study

We now conduct the ablation study to show the accuracy and efficiency impact of different techniques in LightMamba. As shown in Fig. 10, through weight and activation quantization, the throughput can be increased from 2.23 to 5.32 tokens/s. With the rotation-assisted quantization algorithm and customized HTU, we boost the accuracy of quantized Mamba by 4.3% with almost the same throughput. Our computation reordering technique improves the hardware utilization and raises

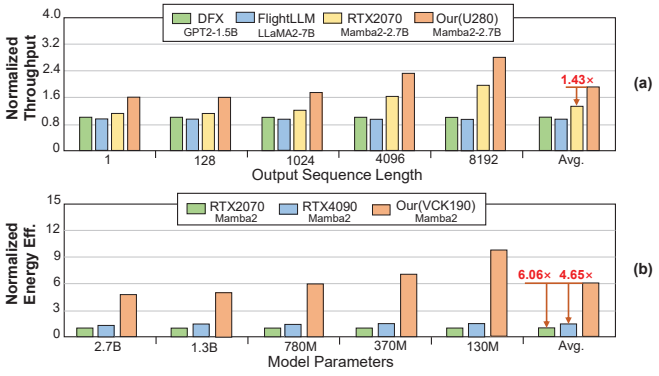


Fig. 9: (a) Throughput with different output sequence length. (b) Energy efficiency with different model sizes.

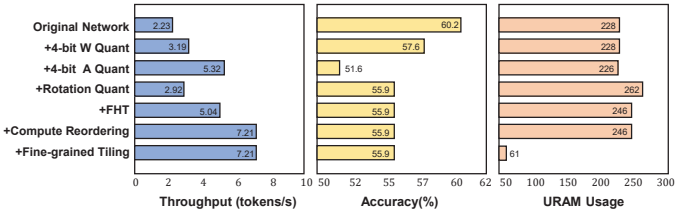


Fig. 10: Impact of different techniques on the computation throughput, accuracy and URAM usage.

the throughput further to 7.21. Finally, through fine-grained tiling and fusion, the on-chip memory consumption is reduced significantly by 4× from 246 to 61.

VII. Conclusion

In this paper, we propose LightMamba, an efficient FPGA-based Mamba acceleration framework featuring quantization and hardware co-design. We point out three challenges of accelerating Mamba on FPGA. To solve them, we propose an FPGA-friendly rotation-assisted PTQ algorithm quantizing Mamba to 4-bit with minimal accuracy degradation. We further propose an FPGA accelerator with computation reordering and fine-grained tiling and fusion. With these methods, LightMamba achieves 7.21 tokens/s on VCK190 FPGA and 4.65∼6.06× higher energy efficiency over the GPU baseline.

REFERENCES

[1] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[2] T. Dao and A. Gu, "Transformers are ssms: Generalized models and efficient algorithms through structured state space duality," *arXiv preprint arXiv:2405.21060*, 2024.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[4] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[5] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg *et al.*, "Sparks of artificial general intelligence: Early experiments with gpt-4," *arXiv preprint arXiv:2303.12712*, 2023.

[6] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[7] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024, pp. 223–234.

[8] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 616–630.

[9] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[10] X. Wei, Y. Zhang, X. Zhang, R. Gong, S. Zhang, Q. Zhang, F. Yu, and X. Liu, "Outlier suppression: Pushing the limit of low-bit transformer language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 402–17 414, 2022.

[11] X. Wei, Y. Zhang, Y. Li, X. Zhang, R. Gong, J. Guo, and X. Liu, "Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling," *arXiv preprint arXiv:2304.09145*, 2023.

[12] S. Li, X. Ning, L. Wang, T. Liu, X. Shi, S. Yan, G. Dai, H. Yang, and Y. Wang, "Evaluating quantized large language models," *arXiv preprint arXiv:2402.18158*, 2024.

[13] A. Pierro and S. Abreu, "Mamba-ptq: Outlier channels in recurrent large language models," *arXiv preprint arXiv:2407.12397*, 2024.

[14] J. Li, S. Huang, J. Xu, J. Liu, L. Ding, N. Xu, and G. Dai, "Marca: Mamba accelerator with reconfigurable architecture," *arXiv preprint arXiv:2409.11440*, 2024.

[15] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm. int8 (): 8-bit matrix multiplication for transformers at scale. corr abs/2208.07339 (2022)," 2022.

[16] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "Quarot: Outlier-free 4-bit inference in rotated llms," *arXiv preprint arXiv:2404.00456*, 2024.

[17] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort, "Spinquant–llm quantization with learned rotations," *arXiv preprint arXiv:2405.16406*, 2024.

[18] Y. Qin, W. Lou, C. Wang, L. Gong, and X. Zhou, "Enhancing long sequence input processing in fpga-based transformer accelerators through attention fusion," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 599–603.

[19] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "Understanding the potential of fpga-based spatial acceleration for large language model inference," *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

[20] Q. Guo, J. Wan, S. Xu, M. Li, and Y. Wang, "Hg-pipe: Vision transformer acceleration with hybrid-grained pipeline," *arXiv preprint arXiv:2407.17879*, 2024.

[21] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "Ftrans: energy-efficient acceleration of transformers using fpga," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 175–180.

[22] H. Fan, T. Chau, S. I. Venieris, R. Lee, A. Kouris, W. Luk, N. D. Lane, and M. S. Abdelfattah, "Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 599–615.

[23] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep learning with int8 optimization on xilinx devices," *White Paper*, 2016.

[24] Fino and Algazi, "Unified matrix treatment of the fast walsh-hadamard transform," *IEEE Transactions on Computers*, vol. 100, no. 11, pp. 1142–1146, 1976.

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[26] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.

[27] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.

[28] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[29] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[30] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? a new dataset for open book question answering," *arXiv preprint arXiv:1809.02789*, 2018.

[31] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "A framework for few-shot language model evaluation," 07 2024. [Online]. Available: https://zenodo.org/records/12608602

[32] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.