2025 62nd ACM/IEEE Design Automation Conference (DAC) | 979-8-3315-0304-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DAC63849.2025.11133171

MHDiff: Memory- and Hardware-Efficient Diffusion Acceleration via Focal Pixel Aware Quantization

Chunyu Qi[1], Xuhang Wang[1], Ruiyang Chen[1], Yuanzheng Yao[1], Naifeng Jing[1], Chen Zhang[1],
Jun Wang[2]*, Zhihui Fu[2], Xiaoyao Liang[1], and Zhuoran Song[1]*
[1]Shanghai Jiao Tong University, Shanghai, China
[2]OPPO Research Institute
[1]{qichunyu, songzhuoran}@sjtu.edu.cn

***Abstract*—Diffusion models have demonstrated superior performance in image generation tasks, thus becoming the mainstream model for generative visual tasks. Diffusion models need to execute multiple timesteps sequentially, resulting in a dramatic increase in workload. Existing accelerators leverage the data similarity between adjacent timesteps and perform mixed-precision differential quantization to accelerate diffusion models. However, merging differential values with raw inputs in each layer of each timestep to ensure computational correctness requires significant memory access for loading raw inputs, which creates a heavy memory burden. Moreover, mixed-precision computations may lead to low hardware utilization if not well designed. Unlike these works, we propose MHDiff, a tailored framework that identifies the focal pixels at the first layer and finetunes them to fit all layers, then represents focal pixels with high-precision while using low-precision for others, thereby accelerating diffusion models while minimizing memory burden. To improve hardware utilization, MHDiff employs a packing module that merges low-precision values into high-precision values to create full high-precision matrices and designs a processing element (PE) array to efficiently process the packed matrices. Extensive experiment results demonstrate that MHDiff can achieve satisfactory performance with negligible quality loss.**

I. Introduction

Currently, diffusion models have demonstrated superior performance in generative vision tasks. In the field of image generation, diffusion models such as Stable Diffusion [1], DALL-E 2 [2], and Midjourney [3] have achieved remarkable results in various tasks (e.g., image generation [4]–[7], image super-resolution [8]–[10], and image inpainting [11], [12]). Due to their powerful generative capabilities, diffusion models are increasingly becoming the mainstream model for generative visual tasks. The fundamental characteristic of diffusion models is the iterative generation process, which requires the entire model to execute multiple timesteps sequentially. While this characteristic helps produce higher-quality images, it comes at the cost of a dramatic increase in workload, hindering their further application.

To eliminate these problems, researchers have designed various accelerators to accelerate the generation process of diffusion models, such as Cambricon-D [13] and HCDiff [14]. Cambricon-D leverages the data similarity between adjacent timesteps in diffusion models by applying outlier-aware differential quantization to accelerate the generation process. As illustrated in Fig. 1(a), at each layer of every timestep, Cambricon-D retrieves the input data of the previous timestep $T-1$ from DRAM (indicated by the purple arrow) and computes the differentials by subtracting it from the current input data. As a result, the differential data can be computed using mixed-precision. Afterwards, the current result is added by the previous result of timestep $T-1$ (indicated by the purple arrow) to obtain the final result of the current timestep T. Next, it writes the resultant data back to DRAM (indicated by the orange arrow), awaiting for the differential data generation during the next timestep $T+1$. In summary, Cambricon-D faces two limitations: 1) It requires two read operations and one write operation to DRAM at each layer of each timestep, which **creates a heavy memory burden**; 2) It uses different processing elements (PEs) to process high-precision and low-precision differentials separately, which **results in low hardware utilization** since the distribution of high- and low-precision differentials is unpredictable and irregular.

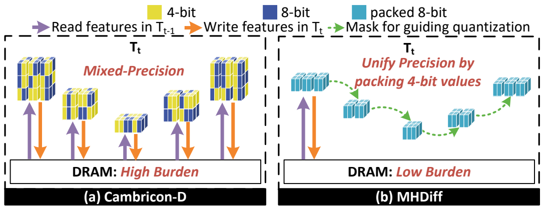

Fig. 1. Comparison of MHDiff and traditional method.

Different from Cambricon-D, our goal is to accelerate diffusion models in a memory-efficient and hardware-efficient manner. In this paper, we observe that the focal pixels to be generated at each timestep play a critical role in image quality, where the focal pixels are defined as the features that change significantly within a timestep. To excavate the acceleration opportunity brought by focal pixels, our study addresses two key challenges: 1) How to identify focal pixels of each timestep in a memory-efficient manner? Given that the focal pixels are consistent across all layers, just like an artist focuses on painting a special area on the canvas for a period of time, we propose a first-predict-then-finetune method. It predicts focal pixels at the beginning of each timestep and then refines them according to the specific operation of each

* Zhuoran Song and Jun Wang are the corresponding authors.

This work is partly supported by the National Natural Science Foundation of China (Grand No. 62202288).

layer, enabling the identified focal pixels to be effectively utilized across all subsequent layers. As shown in Fig. 1(b), compared to Cambricon-D, since we only fetch previous data for prediction at the first layer (indicated by the purple arrow), our approach avoids $n-1$ reads and writes for each timestep given n layers, significantly alleviating the burden on memory. 2) How to accelerate diffusion models based on focal pixels in a hardware-efficient manner? We propose to use high- and low-precision quantization for focal pixels and others respectively, thereby achieving acceleration. Building upon this, we design a specialized packing module that reorganizes the quantized data to pack them into full high-precision matrices. Specifically, based on the principle of avoiding conflicts between high-precision values, the packing module selectively packs adjacent low-precision values into high-precision values as much as possible, thus unifying precision. Therefore, our PE array can process the condensed high-precision matrices at the same speed, resulting in higher hardware utilization.

The main contributions of this paper are listed as follows:

1) On the algorithm level, we propose a MHDiff algorithm, which can identify the focal pixels for each timestep, finetune them according to the specific operation of each layer, and apply high- and low-precision quantization for focal pixels others respectively, thereby achieving acceleration.
2) On the hardware level, we design a MHDiff architecture to support the proposed algorithm. Specifically, MHDiff architecture employs a packing module to pack mixed-precision matrices into full high-precision matrices and use a PE array to process the condensed high-precision data through minor modifications to the PE units.
3) Extensive experiments and in-depth research across various diffusion models consistently validate the advantages of our proposed MHDiff framework, leading to 3.1x, 6.2x, and 38.4x speedup over state-of-the-art diffusion accelerator Cambricon-D, GPU, and CPU while maintaining image quality.

II. Background

A. Diffusion Models

Diffusion models are a new class of machine learning algorithms that have excellent performance in generative vision tasks such as image generation, super-resolution, and image inpainting. Essentially, a diffusion model functions as a denoising model, taking an image composed of Gaussian noise as input and producing a visual image through successive denoising steps. Each denoising step, called a timestep, uses the output from the previous step as input, applying a U-Net model to generate an image with reduced noise. This progressive generation process enables diffusion models to achieve superior image quality compared to previous models. However, this advantage comes at the cost of significantly increased computation and latency, as the model must execute many steps sequentially to produce the final image.

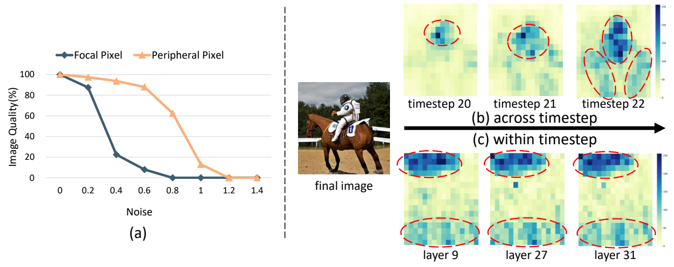

Fig. 2. Explore focal pixels: image quality bearing different noise patterns (a) and the visual focal pixels (marked with red dashed boxes) of different timesteps (b) and different layers (c).

B. Related Work

To address the limitations of diffusion models, researchers have done some work to optimize them from different perspectives, but *they all lack exploration of the relationships between timesteps and the characteristics within each timestep*. On the algorithmic side, DPM-Solver++ [15] introduces a new high-order sampling solver that reduces the number of timesteps required to generate images. Q-Diffusion [16] focuses on model quantization, proposing a timestep-aware post-training quantization method that reduces the activation of diffusion models to 8 bits, thereby accelerating the generation process.

On the hardware side, both Cambricon-D [13] and HCDiff [14] exploit the data similarity between adjacent timesteps in diffusion models by computing the differentials and using outlier-aware quantization to quantize them to a low bitwidth, thus accelerating the inference of diffusion models. Cambricon-D features an outlier-aware PE array to support mixed-precision computations, while HCDiff separates the quantized results into dense normal value matrices and sparse outlier value matrices for separate computation. However, both methods ignore the characteristics within each timestep, thus requiring computing differentials for each layer, which severely increases memory burden. Moreover, they all use different PEs to process high-precision and low-precision differentials separately, which results in low hardware utilization.

III. Motivation

While existing works have achieved notable acceleration, they overlook the existence of focal pixels in diffusion models. To study the impact of focal pixels on image quality, we add different levels of noise to focal pixels and other pixels respectively and measure image quality. As shown in Fig. 2(a), the curve ”Focal Pixel” and the curve ”Peripheral Pixel” indicate that noise is added to the focus pixels and the same number of other pixels, respectively. We can find that focal pixels are more sensitive to noise. Therefore, we propose a pixel-adaptive quantization method to apply high-precision representation for focal pixels and low-precision representation for others, thereby achieving acceleration while ensuring image quality.

Furthermore, we find that the focal pixels in diffusion models differ across timesteps. To validate this, we compare the change levels of pixels in different timesteps of Stable Diffusion and visualize them using heatmaps. As shown in Fig. 2(b), darker colors represent greater changes, while lighter colors represent smaller changes. We can find that the number of focal pixels and their corresponding locations across different timesteps vary greatly. Therefore, we propose a prediction method to identify focal pixels for each timestep before applying pixel-adaptive quantization.

Lastly, we discover focal pixels across different layers within the same timestep exhibit consistency. As shown in Fig. 2(c), the focal regions across different layers within the same timestep align closely, indicating the consistency of these regions. Therefore, we propose a prediction finetuning method that performs prediction only once at the first layer and then finetunes the prediction results to efficiently predict focal pixels in subsequent layers, thus minimizing memory burden.

IV. MHDiff Algorithm

A. Focal Pixel Prediction

To identify the focal pixels at each timestep, we propose the focal pixel prediction operation. Given a $H \times W \times C$ input feature map, it identifies whether a $1 \times C$ vector (pixel P) is a focal pixel based on its change level. Specifically, we compute the L_1 distance between the pixel P_{ij}^t at the current timestep t and the corresponding pixel P_{ij}^{t-1} from $t-1$. As shown in Eqn. 1, where i denotes the row coordinate of the pixel, and j denotes the column coordinate. After calculating the L_1 distance, we compare it with a predefined threshold $Bias$. If the L_1 distance exceeds $Bias$, we think this pixel changes significantly and is a focal pixel; otherwise, we think it is a peripheral pixel. In this way, we divide these pixels into two categories: focal pixels and peripheral pixels, and generate a Bitmask B to record their locations.

$$L_1[i][j] = \sum_{k=1}^{C} abs(P_{ij}^t[k] - P_{ij}^{t-1}[k]) \quad (1)$$

B. Pixel-adaptive Quantization

To accelerate diffusion models based on focal pixels, we propose the pixel-adaptive quantization method, which applies different precisions to different pixels based on the Bitmask from the focal pixel prediction operation. Specifically, we use high-precision quantization for focal pixels as they require detailed generation in current timestep, thus necessitating a large bitwidth to ensure accuracy. In contrast, peripheral pixels exhibit minimal change, allowing us to apply low-precision quantization. Fig. 3 illustrates the process of the pixel-adaptive quantization. We first extract focal and peripheral pixels from input feature map based on the Bitmask, followed by applying uniform high-precision ($int8$) quantization to focal pixels and outlier-aware mixed-precision ($int4 + int8$) quantization to peripheral pixels. Subsequently, we merge the quantized results back into a complete feature map. The input feature map becomes a mixed-precision feature map after quantization, which results in low hardware utilization. Therefore, we design a packing module in Section 5.2 to address this problem.

C. Prediction Finetuning

To identify focal pixels for each layer within a timestep in a memory-efficient manner, we propose the prediction finetuning operation that performs the same calculation process on Bitmasks as on each layer so that Bitmasks can match the focal

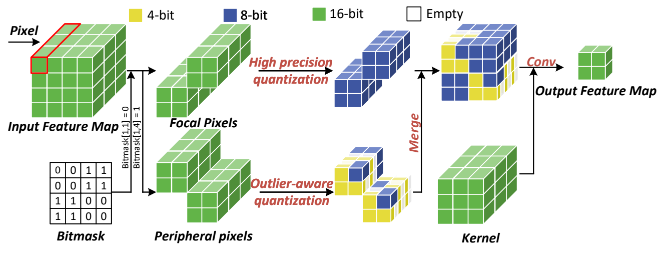

Fig. 3. The process of the pixel-adaptive quantization method.

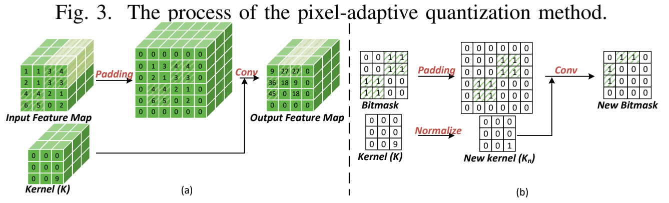

Fig. 4. An example of the prediction finetuning operation.

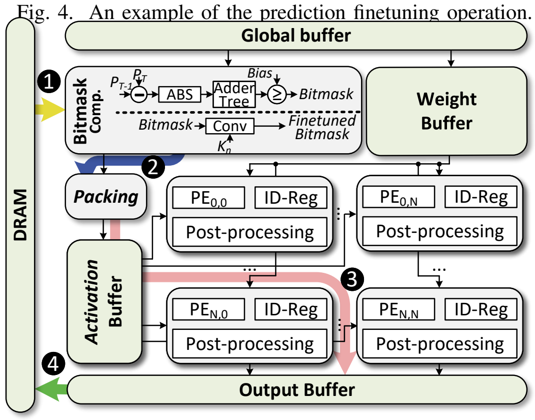

Fig. 5. Overview of the MHDiff architecture.

pixels of each layer. The necessity of the prediction finetuning operation is that the window-based sliding processing of each layer will cause a slight shift in the positions of focal pixels, which makes the Bitmask obtained in the first layer no longer valid in subsequent layers.

Specifically, as depicted in Fig. 4(a), we use a $3 \times 3 \times 3$ convolution kernel K and a $4 \times 4 \times 3$ input feature map as an example. For simplicity, we only show the calculation on one channel. We can find that since the weights are concentrated in the bottom right corner, when K is used to perform convolution on the input feature map, the focal pixels (the shaded area in Fig. 4(a)) will be shifted to the upper left corner. Therefore, current Bitmask is no longer valid for the next layer. To solve this problem, as shown in Fig. 4(b), we finetune this Bitmask according to the calculation process of convolution operation. We first normalize K to generate a new kernel K_n. Then we use K_n to perform convolution on the original Bitmask to obtain a new one, where the distribution of "1" perfectly matches the distribution of focal pixels in the output feature map. Through this method, we achieve finetuning Bitmasks, allowing Bitmasks to effectively guide the quantization for subsequent layers with minimal overhead.

V. MHDiff Architecture

A. Overview

Fig. 5 presents an overview of the MHDiff architecture, which consists of a Bitmask computation module, a packing

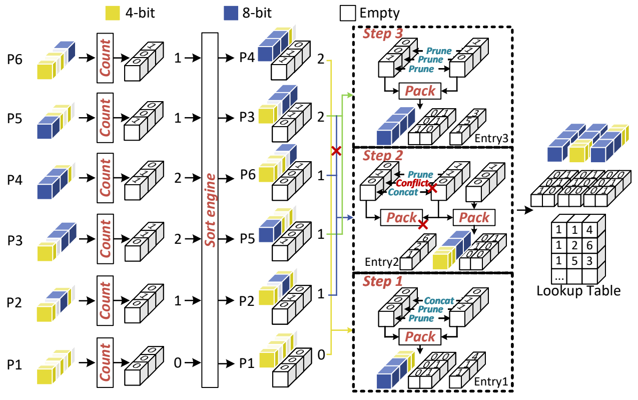

Fig. 6. The process of packing.

module, a PE array, and four buffers. The Bitmask computation module performs focal pixel prediction and prediction finetuning operations. The packing module packs mixed-precision feature maps into high-precision feature maps to facilitate efficient processing of the PE array. The PE array executes tensor multiplication operations. The on-chip buffers store activations, weights, outputs, and intermediate data generated by these modules, respectively.

The dataflow of the MHDiff architecture is also depicted in Fig. 5. In step 1, the Bitmask computation module first determines whether it is currently at the first layer of a timestep. If so, it reads the feature map of the previous timestep from DRAM and performs the focal pixel prediction operation to generate a Bitmask for the current timestep. Otherwise, it performs the prediction finetuning operation on the existing Bitmask to generate a new one for current layer. In step 2, the packing module extracts peripheral pixels from feature maps based on the Bitmask. Then they are packed into high-precision pixels. In step 3, the PE array reads the packed feature maps composed of focal pixels and packed peripheral pixels from activation buffer and the weight vectors from weight buffer for computation and writes the results to the output buffer. Finally, in step 4, the results in the output buffer is requantized and written back to DRAM.

B. Packing Module

To support mixed-precision ($int4 + int8$) computation, the PE units in the PE array need to be able to perform int4 or int8 calculations. However, PE units that perform $int4$ computation will complete faster and need to wait for that perform $int8$ computation, which results in low hardware utilization due to frequent synchronization. Therefore, we propose to pack two $int4$ values to form a $int8$ value, resulting in PE units running in the same speed. As a result, we are able to avoid synchronization, enabling higher hardware utilization.

Since only the peripheral pixels are mixed-precision, we need to pack these pixels to condense into full high-precision pixels. As the positions of $int8$ values in the quantized peripheral pixels are unpredictable, there are three possible cases during packing: (1) packing two $int4$ values, (2) packing a $int4$ value with a $int8$ value, and (3) packing two $int8$ values. For case 1, we can directly pack them into a $int8$ value that is called $int4+int4$ format. For case 2, considering

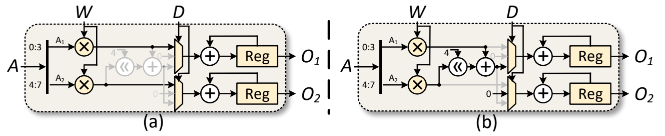

Fig. 7. The structure of PE unit.

that $int4$ values have little impact on model accuracy, we can directly discard the $int4$ value and only save the $int8$ value, which is called $int8 + 0$ format. For case 3, since $int8$ values are critical to model accuracy, we can't discard any of them. Therefore, we need to avoid this case. To this end, we design a packing module based on the greedy algorithm that attempts to avoid two $int8$ values conflicts.

The packing module consists of multiple count units, a sort engine, and a merge unit, which are responsible for recording the number of $int8$ values and their positions, sorting peripheral pixels, and merging them, respectively. The dataflow of the packing module is shown in Fig. 6. Upon receiving peripheral pixels, each count unit will count the number of $int8$ values and generate an identifier vector to record their positions. Then the sort engine will sort these pixels based on their number of $int8$ values. Afterwards, the merge unit will pack the pixel with the most $int8$ values and that with the least $int8$ values. In the example of Fig. 6, the packing module needs to pack 6 pixels ($P1-P6$). In step 1, it tries to pack $P4$ with $P1$. To avoid packing two $int8$ values, it first performs a logical AND operation on their identifier vectors and finds the result contains no "1", which means that these pixels don't have $int8$ values at the same position, so they can be packed together. It then packs these two pixels and their identifier vectors, and creates an entry to record the positions of these two pixels before packing. In step 2, the packing module attempts to pack $P3$ with $P2$, but finds they have two $int8$ values at position 2. Therefore, they cannot be packed together. It then attempts to pack $P2$ with $P6$. Finally, these two pixels are successfully packed together. In step 3, the packing module packs the remaining two pixels ($P5$ and $P3$), whose packing process is the same as step 1. In this way, we successfully pack the mixed-precision peripheral pixels into high-precision pixels and use the packed identifier vectors to record the type of each packed high-precision value. At the same time, we form a lookup table with the generated entries to facilitate the PE array to write results back to the correct buffer addresses.

C. PE Unit

In this section, we design a new PE unit to process the packed high-precision ($int8$) values. The specific structure of this PE unit is shown in Fig. 7. To support the $int4 + int4$ format, as shown in Fig. 7(a), the PE unit reads an activation ($int8$) and a weight ($int16$) from ports A and W, respectively. It then splits the high and low parts of the activation into two $int4$ values, A_1 and A_2. Then A_1 and A_2 are multiplied with the weight, and the results are accumulated into two registers. To support the $int8+0$ format, as shown in Fig. 7(b), after A_1 and A_2 are multiplied with the weight, the result of A_2 is left-shifted by 4 bits and then added to the result of A_1 to complete high-precision calculation. This result is then accumulated into one of the registers. In this way, the PE unit efficiently supports the packed high-precision values.

Since this PE unit needs to support two different data formats, a key issue is how it can identify the format of the activation. To solve this problem, we add an additional port D, which reads a 2-bit identifier generated by the packing module. If the identifier is “00”, it indicates the activation is formed by packing two $int4$ values. In this case, the PE unit will be configured as shown in Fig. 7(a). If the identifier is “10” or “01”, it indicates the activation is originally a $int8$ value. In this case, the PE unit will be configured as shown in Fig. 7(b) and the identifier will control the multiplexer to write the result into the upper or lower register.

D. PE array

Building upon such PE units, we design a PE array to perform computations on the packed high-precision feature maps. Since the packing module disrupts the locations of pixels in the original feature map, it introduces two challenges. The first challenge is how to find the corresponding weights for each feature. The second challenge is how to write results back to the correct buffer addresses after computation.

For the first challenge, although the packing module disrupts the locations of pixels along the H (height) and W (width) dimensions, the locations along the C (input channel) dimension remain unaffected. Therefore, we propose to prioritize convolution calculations in the C dimension rather than in the H and W dimensions. Based on this, we design a PE array where PE units share a broadcast weight vector ($1 \times C$) but receive different activation vectors ($1 \times C$) for dot-product computations.

To address the second challenge, we equip each PE unit with a post-processing module that uses the lookup table generated by packing module to correctly write the results back to the output buffer. The lookup table we established is shown in Fig. 6. The i_{th} row of the table stores the information about the i_{th} pixel in the packed feature map, with three values indicating whether the vector is a packed vector and the positions of the two packed vectors in the original feature map. For example, in Fig. 6, the first row of the lookup table stores the information about the first pixel in the packed feature map. After a PE unit completes the calculation of this pixel, its post-processing module will read the first value of this row and find the value is “1”, indicating that this pixel is a packed pixel. Then, according to the last two values, the post-processing module will read the two results from the O_1 and O_2 ports of the PE unit and write them to address “1” and “4” in the output buffer, respectively.

E. Bitmask Computation Module

The Bitmask computation module consists of multiple L_1 distance computation units and a convolution unit, which are responsible for performing the focal pixel prediction operation and the prediction finetuning operation. The L_1 distance computation unit reads a pixel at the current timestep and the corresponding pixel at the previous timestep. The two pixels undergo element-wise subtraction, followed by the absolute function, resulting in an absolute value vector. This vector is then passed into an adder tree for accumulation. Finally, the output from the adder tree is compared with a preset threshold to generate an element of the Bitmask. When the prediction finetuning operation is to be executed, the Bitmask computation module will perform convolution on the Bitmask, and generate the new Bitmask to guide the next layer.

TABLE I
AREA AND POWER BREAKDOWN OF MHDIFF.

Modules	Parameters	Area (mm^2)	Power(mW)
PE Array	32×32 PEs	0.474	164.5
Packing Module	128 comparators 128 adders	0.045	16.9
Bitmask Comp.	1×9 MACs 128 adders	0.031	10.4
On-chip Buffers	360KB Global buffer 132KB Activation buffer 48KB Weight buffer 132KB Output buffer	0.778	50.7
Total	UMC $28nm$: Area=1.328 mm^2, Power=242.5 mW		

TABLE II
PRECISIONS BEFORE AND AFTER MHDIFF.

Model	FP16	Int8	MHDiff
Stable Diffusion 512	60%	59%	59%
Guided Diffusion 128	65%	64%	64%
Guided Diffusion 512	67%	66%	66%
Latent Diffusion-bed	67%	67%	66%
Latent Diffusion-church	63%	62%	62%

VI. EVALUATION

A. Evaluation Methodology

Software implementation To verify the effectiveness of the MHDiff algorithm, we select Stable Diffusion [1], Latent Diffusion [4], and Guided Diffusion [17] as benchmarks. Stable Diffusion is assessed on the Conceptual Captions dataset (at 512×512 resolution) [18]. Latent Diffusion is tested on LSUN datasets (bedroom and church-outdoor, both at 256x256 resolution) [19]. Guided Diffusion is tested on ImageNet datasets (at resolutions of 128x128 and 512x512) [20].

Hardware implementation We implement the proposed MHDiff architecture in Verilog and synthesize it by Synopsys Design Compiler to get the area and power consumption under 28nm technology, with a design frequency of 500MHz. For SRAM-based on-chip buffers, we use CACTI 7 [21] to model their area and power consumption. We then rescale the area and power data to 28nm technology using DeepScaleTool [22]. Table I provides the detailed design parameters, area, and power breakdown for the MHDiff.

Platforms for comparison We compare MHDiff with widely used hardware platforms, including a server CPU (Intel Xeon Gold 6226R) and a GPU (Nvidia A100). We also compare a state-of-the-art Diffusion accelerator Cambricon-D [13] on the same area budget.

B. Experiment Result

1) Precision: We measure the image quality of each workload using precision, which is a widely used evaluation metric [13], [17]. As illustrated in Table II, compared to the

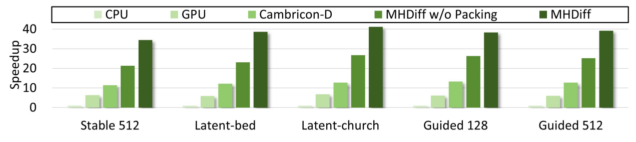

Fig. 8. Speedup of MHDiff.

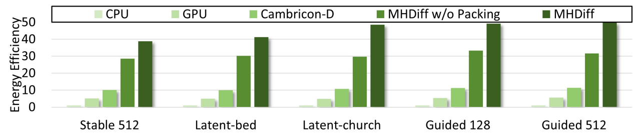

Fig. 9. Energy efficiency of MHDiff.

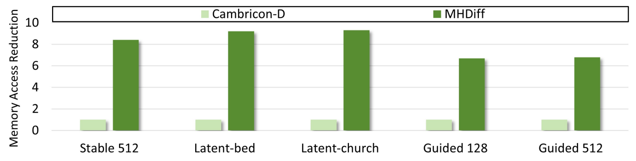

Fig. 10. Memory traffic of MHDiff compared with Cambricon-D.

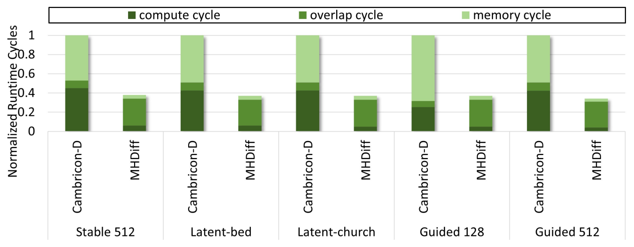

Fig. 11. Runtime breakdown of MHDiff compared with Cambricon-D.

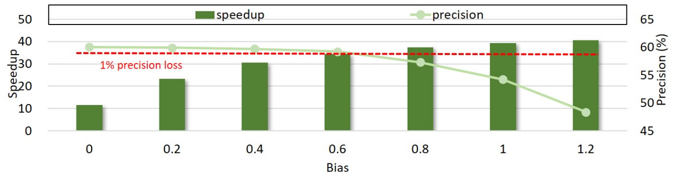

Fig. 12. Exploration on predefined threshold Bias.

baseline and $int8$ quantization, the precision of the MHDiff algorithm drops by less than 1% on average, which indicates that the MHDiff algorithm can maintain image quality well. Moreover, the MHDiff algorithm creates 83% $int4$ values and 17% $int8$ values on average, which greatly reduces computation compared to $int8$ quantization.

2) Speedup: Fig. 8 shows the speedup of the MHDiff over CPU, GPU, and Cambricon-D. MHDiff outperforms CPU, GPU, and Cambricon-D by 38.4×, 6.2×, and 3.1×, respectively. Compared to CPU and GPU, the speedup mainly comes from the proposed pixel-adaptive quantization algorithm that discards redundancy in diffusion models. Compared to Cambricon-D, the speedup originates from two aspects. First, Cambricon-D's high memory burden increases memory access latency, which drags down its acceleration gains. As we can see in Fig. 10, compared to Cambricon-D, we reduce memory traffic by 8.1x, which proves that MHDiff reduces the memory burden very well. Second, Cambricon-D has low hardware utilization. For example, as shown in Fig. 11, the PE array in Cambricon-D is idle nearly 50% of the time, while the idle time of the PE array in MHDiff is close to 0%. Therefore, MHDiff can make full use of its computing resources and achieve better acceleration.

In order to discuss the acceleration gained by the packing module, we implement a MHDiff accelerator without the packing module (labeled as MHDiff w/o Packing). As we can see in Fig. 8, MHDiff outperforms MHDiff w/o Packing by 1.6x. This is mainly because the packing module unifies precision, thus eliminating synchronization overhead and improving hardware utilization.

3) Energy Efficiency: As shown in Fig. 9, the energy efficiency of MHDiff is 45.2×, 8.3×, and 4.4× over CPU, GPU, and Cambricon-D. Compared to CPU and GPU, MHDiff uses fewer bits to represent data and perform calculations, which reduces the energy consumed by the computing units and memory. Compared to Cambricon-D, energy savings can be attributed to the following factors: 1) since MHDiff reduces memory accesses, the energy consumption caused by memory is reduced; 2) the packing module improves hardware utilization, allowing MHDiff to save more computation energy.

4) Runtime Breakdown: Fig. 11 shows the runtime breakdown of MHDiff compared to the Cambricon-D, where the compute cycle, memory cycle, and overlap cycle refer to the computation time, memory access time, and the time they can run in parallel. We observe that MHDiff has less computation time than Cambricon-D because MHDiff has higher hardware utilization. Moreover, we find that the overlap cycle of MHDiff accounts for a large proportion, while that of Cambricon-D accounts for a small proportion. This is because Cambricon-D's heavy memory burden makes its PE array often idle due to lack of input data, resulting in serial execution of computation and memory access. In contrast, our memory traffic is relatively light, allowing computation and memory access to be performed in parallel. This is the main reason why the runtime of MHDiff can be significantly reduced.

5) Design Space Exploration: The goal of the MHDiff algorithm is to achieve a balance between optimizing efficiency and maintaining image quality by identifying focal pixels, in which the threshold $Bias$ plays an important role. Specifically, a larger $Bias$ means that fewer pixels are identified as focus pixels, which results in higher speedup but lower image quality. To explore the impact of $Bias$, we vary $Bias$ from 0 to 1.2 and see the precision and speedup of Stable Diffusion model on Conceptual Captions dataset. As illustrated in Fig. 12, when $Bias$ increases from 0 to 0.6, fewer pixels are identified as focus pixels, leading to higher speedup. But when we keep increasing $Bias$, the precision drops severely. As a result, we set $Bias$ as 0.6 to ensure both precision and performance.

VII. Conclusion

This paper proposes MHDiff, a framework for accelerating diffusion models. On the algorithm side, MHDiff identifies focal pixels for each timestep and applies different precision representations for different pixels, thereby achieving acceleration. On the architecture side, MHDiff transforms mixed-precision matrices into high-precision matrices and designs a specialized PE array through minor modifications to PE units. Experiments show that MHDiff can achieve satisfactory performance while maintaining image quality.

References

[1] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel *et al.*, "Scaling rectified flow transformers for high-resolution image synthesis, march 2024," *URL http://arxiv.org/abs/2403.03206*.

[2] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, vol. 1, no. 2, p. 3, 2022.

[3] W. A., "Midjourney," https://www.midjourney.com/home, accessed:11.08.2024.

[4] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[5] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans, "Cascaded diffusion models for high fidelity image generation," *Journal of Machine Learning Research*, vol. 23, no. 47, pp. 1–33, 2022.

[6] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," *arXiv preprint arXiv:2112.10741*, 2021.

[7] D. Epstein, A. Jabri, B. Poole, A. Efros, and A. Holynski, "Diffusion self-guidance for controllable image generation," *Advances in Neural Information Processing Systems*, vol. 36, pp. 16 222–16 239, 2023.

[8] Z. Yue, J. Wang, and C. C. Loy, "Resshift: Efficient diffusion model for image super-resolution by residual shifting," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[9] H. Li, Y. Yang, M. Chang, S. Chen, H. Feng, Z. Xu, Q. Li, and Y. Chen, "Srdiff: Single image super-resolution with diffusion probabilistic models," *Neurocomputing*, vol. 479, pp. 47–59, 2022.

[10] J. Wang, Z. Yue, S. Zhou, K. C. Chan, and C. C. Loy, "Exploiting diffusion prior for real-world image super-resolution," *International Journal of Computer Vision*, pp. 1–21, 2024.

[11] C. Corneanu, R. Gadde, and A. M. Martinez, "Latentpaint: Image inpainting in latent space with diffusion models," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 4334–4343.

[12] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 461–11 471.

[13] W. Kong, Y. Hao, Q. Guo, Y. Zhao, X. Song, X. Li, M. Zou, Z. Du, R. Zhang, C. Liu *et al.*, "Cambricon-d: Full-network differential acceleration for diffusion models," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 903–914.

[14] R. Guo, L. Wang, X. Chen, H. Sun, Z. Yue, Y. Qin, H. Han, Y. Wang, F. Tu, S. Wei *et al.*, "20.2 a 28nm 74.34 tflops/w bf16 heterogenous cim-based accelerator exploiting denoising-similarity for diffusion models," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 362–364.

[15] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5775–5787, 2022.

[16] X. Li, Y. Liu, L. Lian, H. Yang, Z. Dong, D. Kang, S. Zhang, and K. Keutzer, "Q-diffusion: Quantizing diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 535–17 545.

[17] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

[18] P. Sharma, N. Ding, S. Goodman, and R. Soricut, "Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 2556–2565.

[19] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[21] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[22] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.