Mortar: Morphing the Bit Level Sparsity for General Purpose Deep Learning Acceleration

Yunhung Gao
School of Electronics Engineering and Computer Science
Peking University
Beijing, China

Hongyan Li
State Key Lab of Computer Architecture, Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
Beijing, China

Kevin Zhang
School of Electronics Engineering and Computer Science
Peking University
Beijing, China

Xueru Yu
Shanghai Integrated Circuits R&D Center Co. Ltd
Shanghai, China

Hang Lu
State Key Lab of Computer Architecture, Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
Beijing, China

ABSTRACT

Vanilla Deep Neural Networks (DNN) after training are represented with native floating-point 32 (fp32) weights. We observe that the bit-level sparsity of these weights is very abundant in the mantissa and can be directly exploited to speed up model inference. In this paper, we propose *Mortar*, an off-line/on-line collaborated approach for fp32 DNN acceleration, which includes two parts: first, an off-line bit sparsification algorithm to construct the target formulation by "mantissa morphing", which maintains higher model accuracy while increasing bit-level sparsity; second, the associating hardware accelerator architecture to speed up the on-line fp32 inference through manipulating the enlarged bit sparsity. We highlight the following results by evaluating various deep learning tasks, including image classification, object detection, video understanding, video & image super-resolution, etc.: We (1) increase bit-level sparsity up to 1.28~2.51x with only a negligible -0.09~0.23% accuracy loss, (2) maintain on average 3.55% higher model accuracy while increasing more bit-level sparsity than the baseline, (3)and our hardware accelerator outperforms up to 4.8x over the baseline, with an area of 0.031 mm^2 and power of 68.58 mW.

CCS CONCEPTS

•Computer systems organization •Neural networks

KEYWORDS

deep learning accelerator, neural networks, bit-level sparsity

ACM Reference format:
Yunhung Gao, Hongyan Li, Kevin Zhang, Xueru Yu and Hang Lu. 2022. Mortar: Morphing the Bit Level Sparsity for General Purpose Deep Learning Acceleration. In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23). ACM,* Tokyo, Japan*, 6 pages.* https://doi.org/10.1145/3566097.3567868

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.
ASPDAC '23, January 16–19, 2023, Tokyo, Japan
© 2023 Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-9783-4/23/01.
https://doi.org/10.1145/3566097.3567868

1 Introduction

Since the prosperity of deep learning from 2012, a variety of deep neural networks (DNNs) are deployed on the cloud to provide special and important services, ranging from video understanding to recommender systems. For instance, cloud service providers such as Google and Amazon rely on efficient deep learning to provide precise recommendations to their customers. The features of these deployed DNNs are usually vanilla models represented in floating-point 32 (fp32 hereafter) precision, trained using high-performance GPUs.

The widely adopted, vanilla fp32 DNN usually exhibits satisfiable performance with high native model accuracy. The developer also need not worry about accuracy loss introduced by model optimization methods including pruning and quantization, especially when facing corner cases that do not exist in the training dataset. However, fp32 DNNs are also unfavorable for their slow speed compared with lower precisions such as fp16 or int8. The ideal case is that the developer could, on one hand, acquire the vanilla accuracy without worrying about corner cases, and on the other hand, obtain the fast inference that is on par with low-precision models.

From an architectural perspective, existing DNN accelerators barely specialize their architectures for the fp32 precision by ignoring special "features" of fp32 operands. For example, the general-purpose accelerators TPU [1], KunLun [2], Enflame DTU [3], and MLU290 [4] employ the most conventional fp32 multiply-and-accumulation (MAC) as the fundamental micro-operation for computing convolutions and matrix multiplications. The tedious floating-point arithmetic inevitably drags down the inference speed, despite the improvement brought by frequency boosts or technology nodes.

While from our observations, there are exactly certain special features that can be utilized to accelerate the fp32 arithmetic. As will be shown in Section II, fp32 demonstrates abundant "bit-level sparsity", especially after "exponent alignment". The shifted position of the mantissa must be padded with zeros for addition and multiplication, which provides a unique opportunity for manipulating the newly generated sparsity during the calculation. Moreover, another side-effect of exponent matching is that the essential bits ('1' bits) will be shifted to the rear according to IEEE 754 standard, and the rear position is often less significant but large in the proportion of essential bits. This

Author 1, 2 and 3 contributed equally. Corresponding author is Hang Lu, email: luhang@ict.ac.cn.

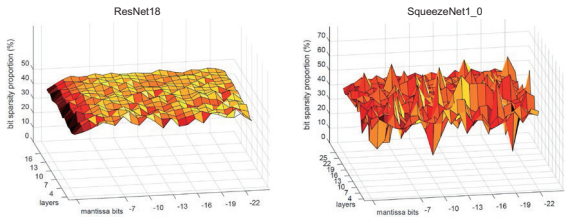


Figure 1: Bit-level sparsity before exponent alignment.

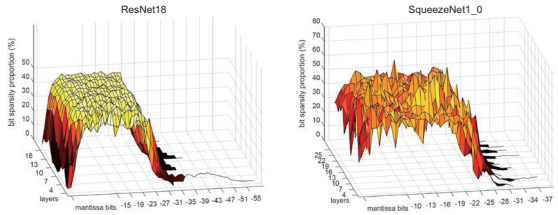


Figure 2: Bit-level sparsity after exponent alignment.

phenomenon also necessitates optimization for the abundant less-important bit 1s.

Therefore, in this paper, we propose a novel DNN acceleration methodology, termed *Mortar*, for faster and higher-performing fp32 inference. *Mortar* is a collaborative methodology that contains two parts: off-line morphing and on-line acceleration. The off-line morphing serves to re-organize the fp32 mantissa by reducing the less important bit 1s, complementing the accuracy loss, and shortening the valid length used in on-line acceleration. The on-line acceleration section involves the practical hardware accelerator with specialized micro-operations and bit-level computing architectures to concretely enforce the updated DNN model after off-line morphing.

Mortar is a cost-effective method, as the software operation, i.e., the off-line morphing, only manipulates the target mantissa using a low-complexity optimization algorithm that neither is time-consuming nor requires powerful training facilities. The time spent is also adequate depending on the model size. Furthermore, the proposed on-line acceleration is a low-cost hardware architecture using combinatorial circuits to fulfill fp32 MACs. Generally speaking, this paper makes the following contributions:

(1) **We propose *Mortar*, a novel on/off-line collaborative approach for general-purpose deep learning acceleration.** Based on the two key observations, our method targets and manipulates the abundant bit-level sparsity in the fp32 mantissa as well as trivial bit ones to form a more hardware-friendly DNN. The actual inference is implemented on-line on the proposed Mortar accelerator.

(2) We thoroughly evaluate *Mortar* and compare it with several state-of-the-art baselines. The following results are highlighted:

Accuracy & Sparsity Ratio: *Mortar* achieves 1.28x~2.51x sparsity improvement with negligible model accuracy loss of -0.09~0.23%. Compared with baseline BitX [5], Mortar can achieve an average of 3.55% higher accuracy while improving 1.05x bit-sparsity than BitX.

Accelerator Performance: Comparing *Mortar* Accelerator's performance with other state-of-art accelerators, we achieve 4.607x and 6.032x performance improvement over Pragmatic [6] as the baseline.

2 Background and Motivation

2.1 Sparsity Parallelism

Targeting the bit-level sparsity for DNN acceleration is not a new idea. Many schemes in the literature have directly leveraged in-situ zero-bit skipping mechanisms to avoid ineffectual computations [6] [7], or special encoding methods to create more bit sparsity headroom [8] [9]. However, these approaches mostly focus on the fixed-point or integer operands, which means their associating accelerators are restricted to only DNN inference and not general training purposes. Very few works try to accelerate MACs by targeting the bit sparsity in the fp32 operands. A standard fp32 operand includes the signed bit (1 bit), exponent (8 bits), and mantissa (23 bits). Because the mantissa is 23-bit long, its sparsity is more abundant, and it exhibits special features that could be leveraged for general-purposed acceleration.

Figure 1 illustrates our first observation, analyzing the sparsity distribution in vanilla DNN weights. The X-axis denotes the mantissa bit positions from 1 ~ 23 (the first hidden bit 1 is not accounted [6]), Y-axis denotes sequential layers of the model, and Z-axis shows the bit sparsity proportion, calculated by the percent of '1' bits over the total number of binary bits. The figure reveals a uniform behavior in all tested DNNs, that is, the sparsity distribution is even (~50%) throughout the bit positions. This is reasonable as each bit has a 50% possibility to be 0 or 1. We call this phenomenon "sparsity parallelism". The problem with this is that the less significant bit positions have a similar bit sparsity but only play a trivial role in the final product. It motivates us to reorganize the mantissa bits to align the sparsity proportions with each bit's significance in the product. Hence, the less important bit positions can expose more bit-sparsity for zero skipping. *Mortar* leverages this characteristic in off-line morphing to enlarge the sparsity in the rear positions under tight accuracy requirements.

2.2 Sparsity Irregularity

As the particular feature of the fp32 arithmetic, exponent matching aims to align two mantissas with the same bit significance in the same position. On top of the previous observation of the static and uniform sparsity, our second observation is that the exponent matching will generate more sparsity due to the shifting operation making the sparsity distribution highly irregular. As shown in Figure 2, the X-axis now has a longer bit length to contain the shifted bits. The rear bit position extends beyond 23 and attains even the 58th-bit position, i.e. 2^{-58}. The whole distribution exhibits an "arch" shape, with the front essential-bit proportion dropping from 50% to less than 10% on average. Most bit 1s concentrate in positions 10~28, and we call this phenomenon "sparsity irregularity".

This observation implicates a potential opportunity to compute fewer bit 1s while still maintaining target accuracy. Since the rear bits are tiny in value, we can safely migrate several rear bit 1s (setting rear bit 1s to 0s) to compensate for the front zero bit that is turned to one, because the sum of several rear bit 1s can approximate the value of a front bit 1. Therefore, if all layers in a DNN are reorganized in this manner, the accelerator only needs to target the front essential bits to effectively reduce the computational overheads of the fp32 DNN. However, achieving this objective is complicated, which entails an accuracy constraint and the associated hardware design. In the next section, we will elaborate on how *Mortar* is designed for these purposes.

3 Methodology

3.1 General Concept

Without loss of generality, a floating-point operand following IEEE-754 [10] standards consists of three parts: signed bit (S), mantissa (M), and exponent (E). Employing the float-32 format (fp32), the mantissa is 23 bits long, the exponent occupies 8 bits, and the sign is one bit long. The floating-point operand *fp* can be expressed as $fp = (-1)^s 1.m \times 2^{e-127}$, in which *e* is the actual position of the binary point plus 127.

We consider a series of fp32 MACs in computing the partial sum of convolutions, and the expression [11] can be transformed from:

$$\sum_{i=0}^{N-1} A_i \times W_i = \sum_{i=0}^{N-1} (-1)^{S_{W_i}} A_i \times M_{W_i} \times 2^{E_{W_i}}]$$

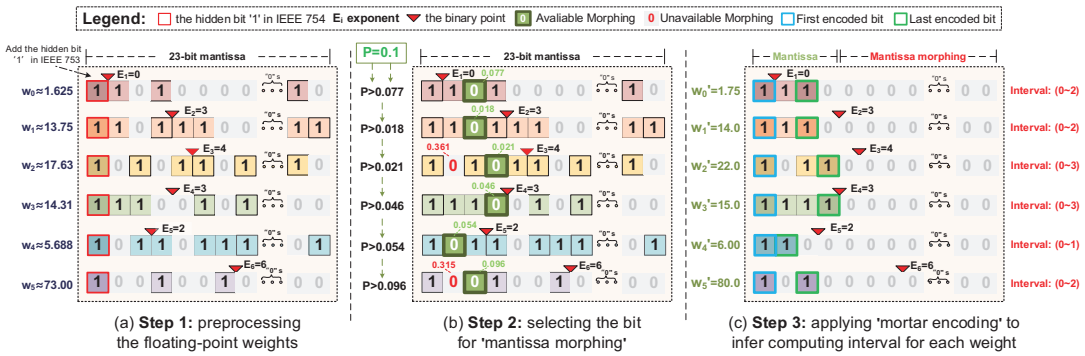


Figure 3: The off-line procedure of Mortar. Bit matrix for pre-processing is shown in (a). (b) demonstrates the process of selecting the bits to apply 'mantissa morphing' with P=0.1. And Mortar encoding, as shown in (c) to infer the data interval for each weight.

Algorithm 1: Mantissa Morphing

Input: Original fp32 weight, W_i

Output: New weight after mantissa morphing, W_i',

1: Interpret the n-bit exponent $E = [e_1, \ldots, e_n]$ and mantissa
2: $M = [m_1, \ldots m_n]$, the actual position of E is determined.
3: Set the value for parameter 'P' in Precision function
4: foreach column j in W
5: if $W_j = 1$ and $W_{j-1} = 0$:
6: $W'_{j-1} = 1$;
7: foreach column k in W $[j : c]$
8: $W'_k = 0$;
9: if $(Precision(W, W', P))$ # precision judge
10: Return $W', j + 1$;
11: else $W' = W$;
12: continue

*Loop 7 can be parallelized for speedups

$$= \sum_{i=0}^{N-1} \sum_{b=E_i-E_{max}}^{E_i-E_{max}-23} [\,(-1)^{S_{W_i} \oplus S_{A_i}} \cdot (M_{A_i} \times M_{W_i}^{b})] \times 2^{E_{max}+b}] \quad (1)$$

Therefore, we infer that floating-point MACs are equivalent to a series of bit-level operations on the corresponding mantissa, which means the floating-point partial sum can be converted into bit-level operations, with sparsity considered. Moreover, based on our observations, the bit level sparsity of the DNN model's weights is plentiful in the mantissa, which can be directly leveraged to accelerate model inference.

The state-of-the-art techniques for decreasing the computing cost of data size in DNN tasks include pruning and compression. Based on bit-level manipulation, a faster and more efficient DNN computation can be obtained by reducing the number of bits to be computed. However, reducing too many bits causes a decrease in model accuracy, so there are two conflicting optimization objectives for pruning and compression, which respectively are the maximization of accuracy and the minimization of computation. We notice the conflict between the two goals: pruning reduces the number of MACs operations by increasing the bit-sparsity of weights at the price of model accuracy. Especially when the number of eliminated bits exceeds a certain threshold, the model accuracy begins to decline as shown in BitX . To tackle these problems, our technique delays the trade-off inflection point in order to retain the model's accuracy to the greatest extent while significantly increasing the sparsity of the weights.

3.2 Mortar

We propose *Mortar*, a software and hardware co-design accelerating DNN inference for general purposes. First, *Mortar* uses "mantissa morphing," a special bit sparsification approach based on bit-level operations, to maximize the model's bit-level sparsity and DNN acceleration for software-level optimization. Figure 4 provides a straightforward illustration of the concept of mantissa morphing:

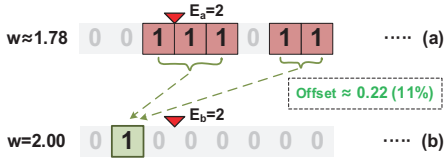


Figure 4: An example of 'Mantissa morphing'.

The initial weight is shown in Figure 4(a). Through analyzing the bit significance, we infer that the overall value represented by multiple valid bits of figure (a) is very similar to the single valid bit located in figure (b). This leads us to design an algorithm locating the most significant '1' bit that can be optimized, i.e., a '1' bit preceded by a '0'. We turn the preceding '0' into a '1' bit and clear all succeeding bits.

Intuitively, the original weight is only transformed to the new weight if their difference falls within an acceptable range. Consequently, deciding which bit to compensate into a '1' is essential to mantissa morphing. A precision algorithm is introduced to establish the error range P to control the difference between the morphing weight $W'$and the initial weight W. Below is the precise formula for *Precision*:

$$Precision = \left(\left| \frac{W_i' - W_i}{W_i} \right| = \left| \frac{\varepsilon_i}{W_i} \right| < P \right) \; ? \; 1 : 0 \quad (2)$$

where P is a hyperparameter weighing the tradeoff between sparsity and accuracy. The precision function is called on each optimizing '1' bit, and if the left-hand side is smaller than P, then we apply mantissa morphing at this position, replacing W with W'. However, when the error is greater than P, the compensation effect exceeds a suitable range, and the search for the next valid bit is necessary. Not only does this indicator avoid over and under-compensation, but it also enables the flexibility of adjusting the tradeoff between the two objectives of accuracy and pruning: if P is extremely large, the morphing conditions are looser and the algorithm will delete more bit 1 s, increasing sparsity at the cost of model accuracy. On the other hand, a lower P will be more restrictive when selecting morphing bits, preserving more information for accuracy.

At the off-line level, the trained fp32 weights are first processed through mantissa morphing, which establishes a hardware-friendly approach to fully use the numerous insignificant bits while preserving the original accuracy. The detailed technique of Mortar's off−line algorithm is elaborated below.

(1) Pre-processing

Consider the example of the mantissa of six fp32 weights in Figure 3(a) We obtain a bit matrix displaying the binary mantissa stored in memory. The example shows 23 bit-width mantissa with the leftmost bit having the largest significance and the rightmost having the smallest significance that corresponds to values 2^{-1} to 2^{-23}. Each row

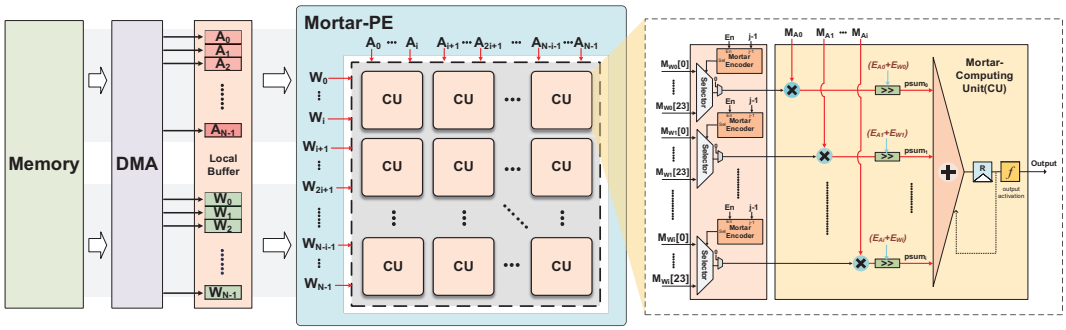


Figure 5: Overview of Mortar-overall architecture and the microarchitecture of Mortar Computing Unit (CU).

represents a particular weight and is marked with a different color. According to IEEE 754, a hidden '1' is inserted into the mantissa's leftmost bit. The triangle mark represents the true relevance of weight by the value of the exponent.

(2) Mantissa morphing

Figure 3(b) describes the core operation for mantissa morphing, and the pseudo-code is provided in Algorithm 1. The initial weights are adjusted off-line, and the parameter P is the threshold for morphing. For each weight W_i, the conditioned search begins from the most significant bit and progresses to the least important bit, and *Mortar* finds a j such that $W_{i,j}$ is valued '1' (line 4~5). Then, the preceding $W_{i,j-1}$ bit is converted to '1' and all subsequent bits to '0', which we declare as the new weight W_i' (line 6~8). Finally, W_i, W_i', and P are input into the Precision function. In Figure 3(b), '0' bits with green backgrounds represent positions satisfying the morphing requirements. Finally, the new W_i is returned along with the morphing bit's location. Contrarily, 0 bits in red are positions failing the precision function test and where the weight is preserved until the next suitable bit. (Line 9~12)

(3) Mortar encoding

Figure 3(c) shows the mortar encoding process. Due to the mechanism of the mortar algorithm, we can automatically spot the specific location $j-1$ of the last valid bit in each weight, followed by continuous '0' bits. Since the first valid bit is always the hidden '1' of each weight, the computing interval of each weight can be readily obtained from these data. This interval determines the specific computation range for the associating on-line accelerator design, substantially reducing the computational cycles and avoiding invalid operations to non-trivially improve the computation's efficiency.

3.3 Benefits of Off-line Mantissa Morphing

The previous examples highlight two key features of our technique:

(1) **The sparsity of weights' bits has been significantly improved:** The bit-level sparsity of Figure 4(a) is five times that of Figure 4(b) with a negligible error of merely 0.22 (11%). Due to the aforementioned property that MACs can be converted into bit-level operations; our technique is a cost-effective tradeoff to increase computation speed through data sparsification while retaining model precision.

(2) **The irregularity of the weights' bits has significantly improved:** The initial weight in Figure 4(a) reveals that the '1' bits may be irregularly distributed (especially the important last bit), necessitating implementations of zero-skipping procedures for hardware accelerators that creates design overheads. However, our algorithm eliminates the need for such zero-skipping techniques by obtaining the morphing position as the final bit and clearing subsequent bits. Therefore, regardless of how subsequent bits are distributed, they are all set to '0', consequently resulting in fewer computation cycles.

To conclude, the main difference between mantissa morphing and conventional pruning is the employment of the technique bit compensation. Rather than only pruning the bits with lower significance, a '0' bit with a higher significance is changed to a '1' bit, minimizing the model's accuracy degradation.

4 Mortar Accelerator

After the proposed off-line 'mantissa morphing' algorithm, we then design an accompanying hardware accelerator for the on-line speed up of fp-32 inference. The overall architecture of the Mortar accelerator is shown in Figure 5, and the area and energy breakdown are given inFigure 7 For the memory systems, the memory access is through DMA, and the local buffer stores the data fetched from the Memory. Mortar PE, which comprises of an array of Mortar CU, receives the input of activations $A_0 \sim A_{N-1}$ and weights $W_0 \sim W_{N-1}$.

In the **microarchitecture of CU**, we use a serial architecture to perform a $ib \times ib$ Multiply-Accumulate (MAC) operation per cycle, where i indicates the i-th input of A and W.The mantissa bits of activation $M_{A0\sim i}$ are serially inputted into the CU along with weight mantissa bits $M_{W0\sim i}$. Each selector receiving the weight mantissa is controlled by the 'mortar encoding' signal to select only the valid data interval for each weight; the unselected status will output 0 automatically. With the advantages of serially computing M_{Wm} and M_{Am} $(m = 0,1 \cdot \cdot i)$, we can utilize a simple combinational logic to output the multiplication value rather than a multiplier in each cycle, reducing the overhead of hardware design. The shifter then shifts the output value of each cycle with the corresponding bit significance to ensure the correctness of the result. For floating point numbers, the shifting operation is performed by the exponent accumulation in A, not introducing much overhead. The corresponding shift for MACs is $E_{Wm} + E_{Am}$, representing the multiplication of activation and weight bits in the cycle. Lastly, the adder tree performs the final partial-sum $Psum_m$ accumulation.

5 Evaluation and Discussion

5.1 Benchmark and Framework

The deep learning models and the pre-trained parameters on ImageNet [12] dataset is directly obtained from PyTorch [13]. The benchmark models are shown in Table 1 to demonstrate the ability of *Mortar* for general purpose DNN task acceleration. We choose the tasks from different domains including Image Classification, Object Detection, Video Understanding, Video & Image Super Resolution, and style transfer. The benchmarks cover "large" models with the parameter size such as 88.79M (ResNext101 [14]), 86.61M (ViT [15]), as well as "small" models with the parameter size of 2.58M (D3DNet [16]). To further demonstrate the generalization capability of *Mortar*, YoloV3 [17] is trained on the COCO [18] dataset. Our experiments extensively evaluate *Mortar*'s effect on model accuracy and sparsity increase, and we compare the performance with SOTA hardware pruning accelerator

Table 1: Benchmark DNNs and their original specs for evaluating Mortar's Performance.

Models	Domain	Type	Dataset	Metric	GFLOPS	Weights (M)	Orig. Accuracy
DenseNet161	Image Classification	2D Convolution	ILSVRC2012	Top-1 %	15.64	28.68	75.28
ResNext101	Image Classification	2D Convolution	ILSVRC2012	Top-1 %	33.02	88.79	78.24
ResNet18	Image Classification	2D Convolution	ILSVRC2012	Top-1	3.64	11.69	67.28
YoloV3	Object Detection	2D Convolution	COCO	mAP	25.42	61.95	52.73
FCOS	Object Detection	Feature Pyramid	COCO	mAP	80.14	32.02	0.382
ViT	Video Understanding	Transformers	ILSVRC2012	Top_1(%)	29.42	86.61	83.89
D3DNet	Video Super Resolution	3D Deformable	Viemo-90k	PSNR	408.82	2.58	36.05
				SSIM			0.94
LapSRN	Image Super Resolution	2D De-Convolution	SET14	/	736.73	0.87	See Figure 8(a)
CartoonGAN	Style Transfer	GAN	Flickr	/	108.98	11.69	See Figure 8(b)

Table 2: Model accuracy and sparsity change after applying mantissa morphing at P=0.1.

Models	Baseline		Mortar	
	Accuracy / Sparsity			
DenseNet161	75.28/1x		75.37/1.58x	
ResNext101	78.24/1x		78.26/1.81x	
ResNet18	67.28/1x		67.22/2.09x	
YoloV3	52.73/1x		52.50/1.28x	
FCOS	0.382/1x		0.378/1.38x	
ViT	83.89/1x		83.66/2.51x	
D3DNet	36.05	1x	36.05	2.28x
	0.94		0.94	
Avg. loss/sparsity	0.00/1x		**-0.06/1.85x**	

Table 3: Accuracy/Sparsity comparison with BitX.

Models	Original	BitX	Mortar
DenseNet161	75.28/1x	74.79/1.61x	**75.37/1.58x**
ResNext101	78.24/1x	73.00/1.74x	**78.26/1.81x**
ResNet18	67.28/1x	62.52/1.90x	**67.22/2.09x**
Avg. loss / sparsity	0.00/1x	-3.50/1.75x	**+0.05/1.83x**

BitX. We selected discrete values for hyperparameter *P* in our design space exploration, which attempt to explore the tradeoff in the algorithm.

5.2 Accuracy and Sparsity

Table 2 shows the accuracy and sparsity changes for different types of datasets that apply mantissa morphing using the threshold 'P = 0.1'. We empirically find this value of *P* to be a well-balanced tradeoff for most models. Result shows that in general cases, the sparsity improvement of a model can reach 2x with a neglectable accuracy degradation.

Additionally, we compared *Mortar* with BitX, a SOTA DNN accelerator using hardware pruning to increase bit-level sparsity for inference acceleration. We selected several models (DenseNet161 [19], ResNext101 [20], ResNet18) for the image classification task, and we focus on the difference in accuracy losses between *Mortar* and BitX at similar sparsity improvement levels. As shown in Table 3, *Mortar* greatly outperforms SOTA approach in accuracy while maintaining a slightly improved sparsity. Especially on ResNext101 and ResNet18, the difference in model accuracy is close to 5%, which is a non-trivial improvement for the classification task.

Discussion: As shown in our experiments, the sparsity of the model can be substantially improved by our algorithm with a negligible accuracy loss. However, despite the parameter *P* set at 0.1 having strong generality, we still find that there is no accuracy degradation on D3DNet. On YoloV3, the sparsity improvement is not as large when P=0.1. Above datum indicates that *P* needs to be finely adjusted for certain model, and the detail discussion is illustrated later.

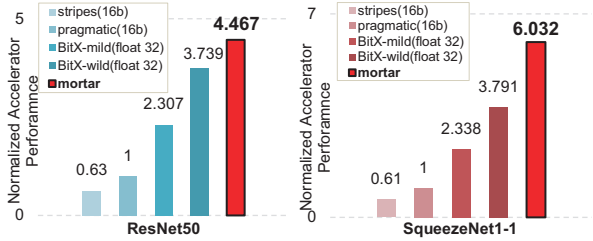


Figure 6: Speedup comparison of mortar and other SOTA accelerators in (a) 4.467x over the baseline in ResNet50, (b) 6.032x over the baseline in SqueezeNet1_1.

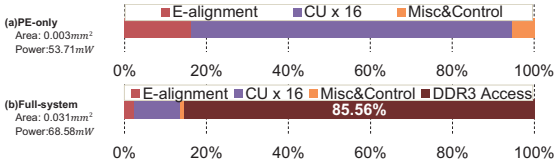


Figure 7: Mortar' Area & Energy Breakdown for PE-only and full system.

Although BitX greatly improves the sparsity of model weights, it falls short in not considering the difference of each weight in the model, where its fixed-position pruning leads to over/under-pruning. *Mortar*, on the other hand, accounts for the different sparsity proportions of each weight in mantissa morphing, compressing weights according to bit significance and utilizing a bitwise compensation to implement fine-grained compression that preserves high accuracy. Thus, we conclude that *Mortar* addresses sparsification more precisely, saturating bit-level sparsity and greatly extending the cost-benefit tradeoff between accuracy and compression.

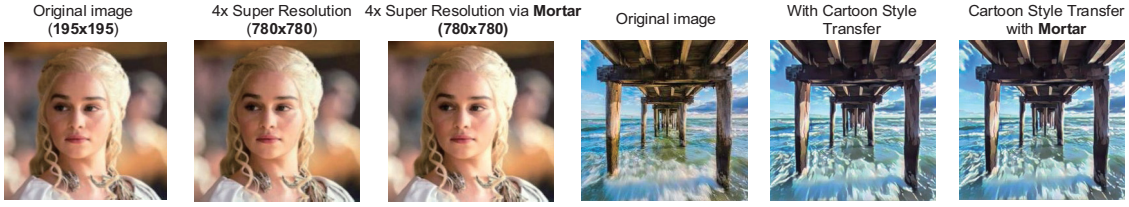
5.3 Accelerator Speedup and Energy-use Analysis

We concretely analyze the speedup by focusing on the performance of *Mortar* hardware accelerator compared to other SOTA accelerators. In Figure 6, we selected SOTA accelerator Pragmatic [8] as the baseline and ResNet50 and SqueezeNet1_1 [21] as inference models. Mortar's performance on ResNet50 is 4.467x that of the baseline. On SqueezeNet1_1, *Mortar* outperforms the baseline by 6.032x, surpassing all other methods including BitX . Moreover, we evaluated the power and physical area of *Mortar* in Figure 7.

Discussion: The experimental results illustrate that *Mortar* outperforms BitX Accelerator substantially in acceleration and energy efficiency. Mortar's encoding mechanism greatly reduces cycles that do not contribute significantly to the model performance by only focusing on the encoded intervals. Other compared methods rely primarily on zero-skipping techniques to determine the computational interval. *Mortar* eliminates such design overheads and accelerates performance as the process is effortlessly obtained with the off-line algorithm.

Table 4: Design space exploration of key design parameter 'P' on various models and P=0.1 is the turning-point for most models.

Model	Baseline	P=0.0001	P=0.0005	P=0.001	P= 0.005	P=0.01	P=0.05	P=0.1	P=0.3	P=0.5
DenseNet161	75.28	75.28	75.28	75.28	75.28	75.27	75.29	75.31	75.37	75.06
ResNext101	78.24	78.24	78.24	78.25	78.25	78.27	78.29	78.26	77.93	77.41
ResNet18	67.28	67.29	67.29	67.29	67.29	67.28	67.28	67.22	67.13	66.92
YoloV3	52.73	52.75	52.75	52.75	52.73	52.73	52.69	52.50	51.72	51.38
FCOS	0.382	0.382	0.382	0.382	0.382	0.382	0.382	0.378	0.318	0.258
ViT	83.89	83.89	83.89	83.88	83.88	83.88	83.76	83.66	83.33	82.91
D3DNet	36.05	36.05	36.05	36.05	36.05	36.05	36.05	36.05	36.02	35.97
	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94



(a) visual comparison for LapSRN (b) visual comparison for CartoonGAN

Figure 8: visual demonstrations of (a) 4x super resolution inference via Mortar, (b) cartoon style transfer via Mortar.

5.4 Design Space Exploration

In the previous section, we analyzed the model sensitivity towards P =0.1 on accuracy and sparsity. Experimental results empirically prove that this value outperforms BitX. However, P = 0.1 is not the optimal tradeoff turning point for all models; hence, as shown in Table 4, the design space exploration is conducted to find the optimal value through testing accuracy under different P. We find that for most models, setting $0.0001 < P < 0.05$ maintains equal accuracy with the baseline, and even improvements in certain cases. However, most model accuracy starts to decrease when $P \geq 0.1$, and when $P \geq 0.5$ the accuracy decreases significantly.

Discussion: Our exploration indicates that P = 0.1 is a relatively general threshold applicable for most models. For certain model such as D3DNet, the turning point have not been reached even when P reaches 0.5; thus, we can increase the pruning effort to further improve sparsity. This also demonstrates that we can adjust the value of P appropriately for each model to align the data closer to the inflection point when accelerating different models.

5.6 Visual Comparison

To qualitatively analyze *Mortar*, we apply our approach on multiple image processing tasks to visually display Mortar's effect on image outputs. In Figure 8, we apply *Mortar* on both 4x Super Resolution with LapSRN [22] and CartoonGAN [23], showing results for both original and enhanced models.

Discussion: Our results show that Mortar's effect on the original model is not only quantitatively minimal, but also qualitatively imperceptible to the end-user. *Mortar* maintains a high-level quality of its outputs.

6 Conclusion

In this paper, we propose a novel off-line/on-line collaborated approach for general purpose deep learning acceleration — the software optimization "mantissa morphing," and the associating hardware accelerator design "*Mortar* accelerator". *Mortar* leverages the concept of bit compensation when optimizing bit-level operation, significantly increasing the bit-level sparsity for accelerating while maintaining high inference accuracy with models based on fp32. Moreover, the performance of the algorithm has strong generalization capabilities in different model tasks and datasets, concurrently outperforming existing hardware pruning accelerators. To further cater for specific needs, the parameter "P" can be adjusted automatically to search for optimal accuracy-speed tradeoff. We hope this work will encourage future accelerator designs to become more efficient and more all-purposed.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 62172387, in part by the Youth Innovation Promotion Association CAS under Grant 2021098.

REFERENCES

[1] N. P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor ProcessinUnit," in ISCA, 2017.
[2] J. Ouyang et al., "3.3 Kunlun: A 14nm High-Performance AI Processor for Diversified Workloads," in ISSCC, 2021.
[3] E. Technology. "Enflame DTU," https://www.servethehome.com/enflame-dtu-1-0-ai-compute-chip-at-hot-chips-33/.
[4] Cambricon. "CambriconMLU290," https://www.cambricon.com/index.php?m=content&c=index&a=lists&catid=340.
[5] H. Li et al., "BitX: Empower Versatile Inference with Hardware Runtime Pruning," in ICPP, 2021.
[6] J. Albericio et al., "Bit-Pragmatic Deep Neural Network Computing," in MICRO, 2017.
[7] H. Lu et al., "Tetris: Re-architecting Convolutional Neural Network Computation for Machine Learning Accelerators," in ICCAD, 2018.
[8] F. Tu et al., "A 28nm 29.2TFLOPS/W BF16 and 36.5TOPS/W INT8 Reconfigurable Digital CIM Processor with Unified FP/INT Pipeline and Bitwise In-Memory Booth Multiplication for Cloud Deep Learning Acceleration," in ISSCC, 2022.
[9] S. Sharify et al., "Laconic deep learning inference acceleration," in ISCA, 2019.
[10] IEEE. "IEEE Standard for Floating-Point Arithmetic (754-2019)," https://standards.ieee.org/standard/754-2019.html.
[11] H. Lu et al., "Distilling Bit-level Sparsity Parallelism for General Purpose Deep Learning Acceleration," in MICRO, 2021.
[12] J. Deng et al., "ImageNet: A large-scale hierarchical image database," in CVPR, 2009.
[13] Facebook. "Pytorch," https://pytorch.org/.
[14] S. Xie et al., "Aggregated Residual Transformations for Deep Neural Networks," in CVPR, 2017.
[15] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in ICLR, 2020.
[16] X. Ying et al., "Deformable 3D Convolution for Video Super-Resolution," arXiv:2004.02803, 2020.
[17] J. Redmon, and A. Farhadi, "YOLOv3: An Incremental Improvement," in CVPR, 2018.
[18] T.-Y. Lin et al., "Microsoft COCO: Common Objects in Context," in ECCV, 2014.
[19] G. Huang et al., "Densely Connected Convolutional Networks," in CVPR, 2017.
[20] K. He et al., "Deep Residual Learning for Image Recognition," in CVPR, 2016.
[21] F. Iandola et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," arXiv:1602.07360, 2010.
[22] W.-S. Lai et al., "Deep Laplacian Pyramid Networks for Fast and Accurate Super-Resolution," in CVPR, 2017.
[23] Y. Chen et al., "CartoonGAN: Generative Adversarial Networks for Photo Cartoonization," in CVPR, 2018.