OPAL : Outlier-Preserved Microscaling Quantization Accelerator for Generative Large Language Models

Jahyun Koo*
DGIST
jhkoo@dgist.ac.kr

Dahoon Park*
Korea University
manyteacher93@korea.ac.kr

Sangwoo Jung
DGIST
jsangwoo123@dgist.ac.kr

Jaeha Kung†
Korea University
jhkung@korea.ac.kr

ABSTRACT

To overcome the burden on the memory size and bandwidth due to ever-increasing size of large language models (LLMs), aggressive weight quantization has been recently studied, while lacking research on quantizing activations. In this paper, we present a hardware-software co-design method that results in an energy-efficient LLM accelerator, named OPAL, for generation tasks. First of all, a novel activation quantization method that leverages the microscaling data format while preserving several outliers per sub-tensor block (e.g., four out of 128 elements) is proposed. Second, on top of preserving outliers, mixed precision is utilized that sets 5-bit for inputs to sensitive layers in the decoder block of an LLM, while keeping inputs to less sensitive layers to 3-bit. Finally, we present the OPAL hardware architecture that consists of FP units for handling outliers and vectorized INT multipliers for dominant non-outlier related operations. In addition, OPAL uses log2-based approximation on softmax operations that only requires shift and subtraction to maximize power efficiency. As a result, we are able to improve the energy efficiency by 1.6~2.2×, and reduce the area by 2.4~3.1× with negligible accuracy loss, i.e., <1 perplexity increase.

ACM Reference Format:
Jahyun Koo, Dahoon Park, Sangwoo Jung, and Jaeha Kung. 2024. OPAL : Outlier-Preserved Microscaling Quantization Accelerator for Generative Large Language Models. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (DAC'24)*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Large-scale language models (LLMs) have been a game changer that work well in various language-related tasks, including translation [15] and text-to-image diffusion [12], to name a few. However, due to the massive model size, running a generation task using LLMs is economically expensive and consumes substantial energy even at servicing inference. For instance, to run the Llama2-70B model [14], it needs to store 140GB of memory using FP16, and it requires at least 140 GFLOPs to generate only one token. In addition, it takes at least two A100 80GB GPUs to perform inference that costs more than $200K. As the number of parameters in an LLM keeps growing, e.g., GPT3-175B, service costs are expected to increase dramatically. To mitigate this challenge in deploying LLMs, various compression techniques on LLMs have been actively studied to fit the model into single-GPU memory capacity [2, 4, 5].

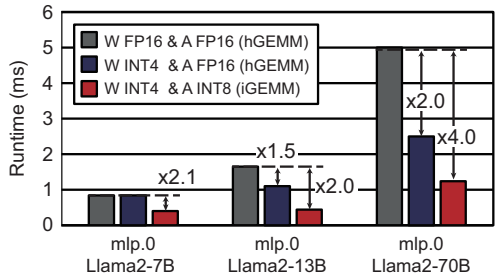

Figure 1: The comparison of single-batch latency in running the Llama2's first FFN layer (*mlp.0*) at various sizes and bit-widths using CUTLASS [10]. The 'hGEMM' uses FP16 computing units while 'iGEMM' uses INT8 computing units.

Among various compression techniques, quantization has been prevailed upon others due to its effectiveness in both performance boost and memory reduction. Especially, due to high training time of LLMs, post-training quantization (PTQ) has become the de facto standard in quantizing LLMs [1, 2, 5, 16]. The common observation from the prior work on quantizing LLMs is that some input channels in activations consistently produce large values, i.e., *outliers*, allocating few bits to non-outliers resulting in high quantization errors. Therefore, several studies have identified those activation outliers during the *weight-only quantization* [2, 5]. Quantizing weights is extremely effective since generation tasks are memory-bounded due to small arithmetic intensity (FLOPs/Byte). However, when aggressive weight quantization is applied, e.g., 4-bit, the workload moves toward the compute-bounded region necessitating the need for activation quantization as well (**motivation 1**). Fig. 1 shows that quantizing weights of (memory-bounded) Llama-13B and Llama-70B models to INT4, while computing with FP16 units (hGEMM in CUTLASS), reduces the runtime by 1.5× and 2.0×, respectively. This is mainly due to the higher core utilization according to the increased arithmetic intensity thanks to the quantization.

There is another research direction trying to quantize both activations and weights [1, 16]. The prior work successfully quantized both tensors to INT8 with little accuracy loss (but tested a model with up to 13B parameters). With 8-bit activations, we now can utilize INT8 computing units within a GPU pushing the computational roof higher (iGEMM in CUTLASS). We tested Llama2 models at various sizes while quantizing the weights to INT4 and activations to INT8. As a result, it was possible to achieve 2.0~4.0× speed-up

*J. Koo and D. Park are equally contributed authors.
†J. Kung is the corresponding author (jhkung@korea.ac.kr).

DAC'24, June 23–27, 2024, San Francisco, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

for all tested models thanks to the higher number of available computing units. However, quantizing activations on the fly requires min/max extraction and dividers to account for scale factors, which incur significant hardware overhead (**motivation 2**).

With these motivations in mind, we present a hardware accelerator for LLM-based generation tasks, dubbed OPAL (**O**utlier-**P**reserved microscaling quantization **A**ccelerator for **L**LMs), which is based on a novel algorithm-hardware co-design method. The OPAL utilizes a microscaling data format (MX format [11]) that allows quantizing activations to 3/5-bit or 4/7-bit with the support of preserving several outliers within a pre-defined block. The microscaling data format allows us to replace dividers to simple shifters to reflect scale factors, which is more suitable for dynamic quantization. In addition, OPAL hardware is equipped with lightweight softmax units and reconfigurable PEs that improve the energy efficiency by 2.41/3.18× compared to the bfloat16 baseline. Our main contributions can be summarized as follows:

- ***Shift-based Dynamic Quantizer***: We propose an *outlier-preserved microscaling integer format* (MX-OPAL) to implement shift-based dynamic quantizers. Since activations in LLMs have outliers, naïve MXINT8 leads to significant accuracy degradation. Thus, in this work, we keep a small number of outliers, i.e., four largest absolute values per block, in bfloat16 while quantizing non-outliers in 3/5-bit or 4/7-bit.
- ***Hardware-friendly Softmax Approximation***: We present log2-based approximation on softmax functions in an LLM to perform '$softmax(Q{\cdot}K^T){\cdot}V$' with only shift-and-subtractions. The approximated softmax only increases the perplexity by <0.4 PPL on WikText-2 while saving 1.56× power consumption of conventional softmax.
- ***Hardware Accelerator for LLM-based Generation***: Finally, we present an energy-efficient hardware accelerator for generative LLM, named OPAL. The OPAL enjoys power efficiency of shift-based quantizers and softmax units that are tested on recent LLMs. The OPAL improves the energy efficiency by 1.6×/2.2× under PPL increase of 0.33/0.62 compared to the state-of-the-art weight-only quantized models.

2 BACKGROUND

2.1 Quantization for LLMs

Weight-only quantization is the most popular approach to compress LLMs, since the weights occupy most of the memory space in LLM-based generation tasks. OPTQ (also known as GPTQ [2]) looks at Hessian matrices to find sensitive input channels and apply quantization in order of sensitivity to achieve 3-/4-bit weights. On top of OPTQ, OWQ [5] applies mixed-precision quantization that keeps weights in FP16 at highly sensitive input channels, while others in 3-/4-bit. As a result, OWQ shows that 3.01-bit weights are enough to provide comparable LLM accuracy to the 4-bit OPTQ model (i.e., 25% memory reduction). However, since the prior work on *weight-only quantization* performs matrix multiplication (matmul) in FP16, the computing energy is not reduced compared to the original FP16 model.

Quantizing both inputs and weights to INT8 [1, 16] reduces the inference energy by using more power-efficient INT8 units and fetching 2× less data from DRAM. ZeroQuant [16] is a fine-grained compression method utilizing group-wise weight quantization and

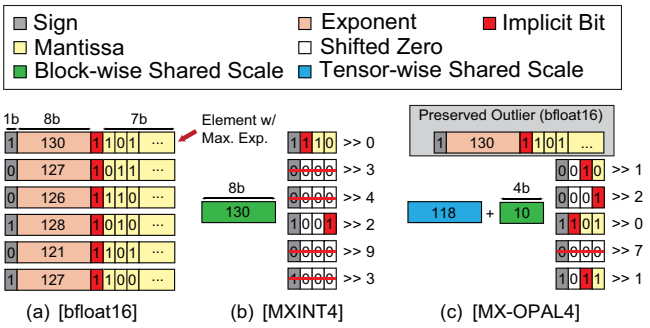

Figure 2: Various data formats: (a) bfloat16, (b) original MXINT4 [11], and (c) the proposed outlier-preserved MXINT4 format (i.e., MX-OPAL4).

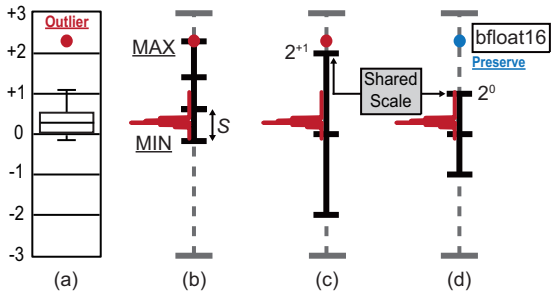

Figure 3: Comparison between different data formats on quantizing the original data of 128 elements extracted from the 2nd decoder block in Llama2-7B. (a) Original data, (b) 2-bit MinMax, (c) MXINT2, and (d) MX-OPAL2.

token-level activation quantization. Unlike the weights, activations are dynamically quantized, i.e., extracting min/max range per token, to minimize quantization error since activations are input-specific. In LLM.int8() [1], vector-wise normalization and outlier decomposition are used to maintain the LLM accuracy while quantizing most of the values to INT8 for both tensors. However, dynamic quantization consumes too much power since floating-point (FP) divisions are required to convert output values to integer values with no accuracy loss.

2.2 Microscaling Data Format

Recently, microscaling (MX) data formats [11] have been presented showing their effectiveness on both inference and training of various deep learning benchmarks. The MX format packs 'k' data into a block and shares one exponent per block (i.e., *shared scale*; 2^s), and normalizes each element in the block with the shared scale. Each element can be represented by either a low bit-width floating-point (e.g., MXFP6) or integer number (e.g., MXINT8). The integer version of an MX format (MXINT) is also known as the block floating point [6]. Converting FP numbers to MXINT has two benefits: (i) **hardware efficiency**—reduced bit-width and INT-based arithmetic maximize computing and storage efficiency, and (ii) **simplified conversion**—unlike the conventional quantizer, which requires FP divisions by the scale factor, MXINT can scale each element by simple shift operations (suited at dynamic quantization).

Fig. 2 shows an example of converting bfloat16 values to MXINT4. First, we find the shared scale which is the maximum exponent

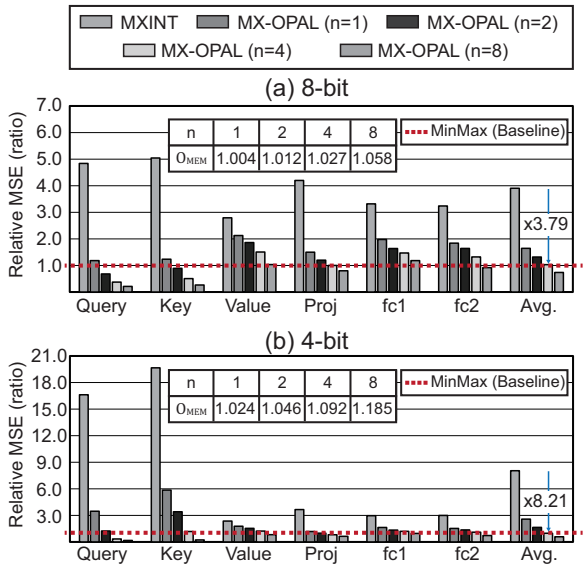

Figure 4: Comparison of the impact of preserving varying number of outliers (n) on the quantization noise with MX formats at the 20th decoder block in Llama2-7B. The block size k is set to 128, and (a) 'sign + mantissa bits' = 8 (b = 8), (b) 'sign + mantissa bits' = 4 (b = 4).

within a block (in this example, k = 6). For instance, the maximum exponent is 130 in Fig. 2(a), which becomes the shared scale. Then, mantissa bits are shifted right by the difference between the shared scale and its own exponent (Fig. 2(b)). Due to the shift operation, there could be underflow that results in the quantization error (crossed out by the red line). Thus, it could be problematic if there is few significantly large outliers in the block, which makes other non-outliers zero. As observed by the prior work and our analysis, there are such outliers in activations for LLMs.

3 PROPOSED MX-OPAL DATA FORMAT

3.1 Outlier-Preserved Microscaling Data Format

To address this challenge, we propose an outlier-preserved microscaling format, named MX-OPAL, that maintains the LLM accuracy by keeping significantly large outliers in bfloat16. Note that we apply the MX-OPAL format for activations while using OWQ [5] for weights. They are a good match since OWQ also maintains the weights in FP format at channels where activation outliers occur, while the remaining weights are stored in INT3 (or INT4). An example of converting bfloat16 to 4-bit MX-OPAL is shown in Fig. 2(c). In MX-OPAL, we preserve the top-n outliers where n = 1 in the example. After preserving the outlier(s), we extract the $(n{+}1)^{\text{th}}$ highest exponent value as a shared scale. In the proposed MX-OPAL format, we set a tensor-wise global shared scale (118 in Fig. 2(c)) and store a 4-bit block-wise offset for every k elements (10 in Fig. 2(c)) to minimize the data size of shared scales. Compared to MXINT4, we can reduce the number of underflows by using the proposed MX-OPAL4 as shown in Fig. 2(c).

Fig. 3 compares the quality of three different quantizers, i.e., 2-bit MinMax [16], MXINT2 [11], and MX-OPAL2, on actual data extracted from the input to the projection layer (*self_attn.o_proj*) in the 2nd decoder block of Llama2-7B. We extracted 128 elements from 4,096 elements in the input vector that makes a block (thus, 32 blocks are formed). The data distribution of the first 128 elements is provided in Fig. 3(a). There is an outlier that is away from the other data. The MinMax quantizer sets the outlier as the maximum value and finds the other end as the minimum. Then, it divides each value by the scale factor $S = (Max - Min)/(2^b - 1)$, where '$b$' is the bit-width, and rounds the result to the nearest neighbor. The MXINT format finds the outlier and sets its exponent as the shared scale. In Fig. 3(c), the quantization level right below the outlier becomes this shared scale. After setting the shared scale, which is the power of 2, the elements are quantized by the given 'sign+mantissa' bits (2-bit in this example). A large portion of elements then falls in the same quantization bin, resulting in a large quantization error. With the proposed MX-OPAL format, we keep the outlier in bfloat16 and finds the second largest exponent within the block (Fig. 3(d)). Effectively, the shared scale moves closer to the mean of the distribution and the step size becomes smaller (lowering the quantization error), since the smaller exponent has been selected as the scale factor.

3.2 Impact of Preserving Activation Outliers

Preserving several outliers allows us to push majority non-outliers to a low bit-width, e.g., 3-bit or 5-bit, while the outliers are kept in 16-bit and require bfloat16 computation in MX-OPAL. Thus, we have to make sure the memory overhead of storing those outliers is not critical. The memory overhead (O_{MEM}) of the MX-OPAL format relative to the original MXINT (or MinMax) format can be formulated as follows:

$$O_{MEM} = \frac{MEM_{OPAL}}{MEM_{MXINT}} = \frac{(k-n)\cdot b + 16\cdot n + 4}{k\cdot b + 8}, \qquad (1)$$

where k is # of elements in a block, n is # of preserved outliers, b is the bit-width of non-outliers. The constant 4 is added in the numerator since we use 4-bit to store one block-wise shared scale per block. According to Eq. (1), the memory overhead of storing tensors in MX-OPAL format becomes negligible when the block size 'k' is large enough. For instance, only 2.7% of additional memory space is needed when k = 128, n = 4, and b = 8.

In Fig. 4, we examined the impact of preserving outliers by computing the mean squared error (MSE) between the original bfloat16 inputs and their MX-OPAL quantized values at various layer outputs within the 20^{th} decoder block. To show the relative quantization error compared to the conventional MinMax quantization, we normalized MSEs of MX-OPAL at varying numbers of outliers (i.e., n = 1, 2, 4 and 8) to the MSE of the MinMax quantizer. In addition, the relative MSEs of MXINT formats are also provided to show the effectiveness of preserving several outliers in MX-OPAL (3.79× and 8.21× lower error on average than MXINT for b = 8 and b = 4, respectively). The quantization error becomes similar to the baseline (i.e., MinMax), or sometimes below the bar, when four outliers among 128 elements are preserved. Thus, throughout the paper, we set the number of preserved outliers 'n' to four. When n = 4, the memory overhead compared to the MinMax or MXINT format becomes 2.7% and 9.2% when b = 8 and b = 4, respectively.

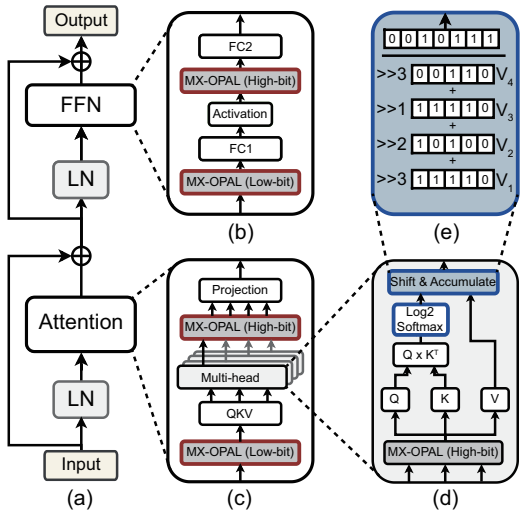

Figure 5: Overview of OPAL computation flow: (a) one decoder block, (b) a feed forward network (i.e., two FC layers), (c) an attention layer, (d) a multi-head attention layer, and (e) shift-based '$Attn \cdot V$' owing to log2-based softmax.

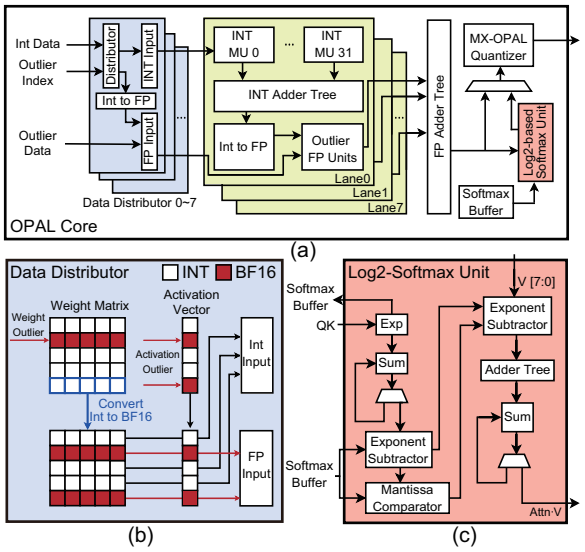

Figure 6: Microarchitecture of OPAL: (a) the main core of OPAL, (b) the data distributor for outliers and non-outliers, and (c) the proposed log2-based softmax unit.

4 HARDWARE ARCHITECTURE OF OPAL

4.1 OPAL Computation Flow

Fig. 5 shows the OPAL computation flow that is implemented as a hardware accelerator described in Section 4.3. Fig. 5(a) shows one decoder block which is repeated 32/80 times to form Llama2-7B/70B. There are two main computing layers which require high number of parameters: a feedforward network (FFN in Fig. 5(b)), and an attention layer (Fig. 5(c)). Since activations are normalized by layer normalization (LN), the distribution of activations is limited to a specific range. Thus, further reduced bit-width can be used, e.g., MX-OPAL3, to quantize activations after the LN layer. Activations at other layers are kept in higher bit-width, e.g., MX-OPAL5, to maintain the accuracy at a similar level to 16-bit activations. In the attention layer (Fig. 5(c-d)), multiple matrix-vector multiplications (MxV) are performed to compute Q, K, V, and the attention map ($Attn$). Thanks to the proposed log2-based softmax unit, we simply perform shift-and-accumulate to compute the output Z (Fig. 5(e)).

4.2 Proposed Log_2-based Softmax Unit

In LLMs, softmax is one of the most hardware-unfriendly operations, that requires floating-point dividers, thus several algorithms have been proposed to approximate the softmax function [7, 13]. Unfortunately, the prior work require fine-tuning to compensate for the approximation error which is not suitable for LLMs. Therefore, we modified the log2-based softmax method [8], an approach for PTQ, on optimizing LLM inference. The output of the log2-based softmax in the multi-head attention layer is defined as:

$$Attn_Q = clip(-\lceil log_2(softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}))\rfloor, 0, 2^b - 1), \quad (2)$$

where Q and K are query and key, d_k is the embedding dimension, and b is the bit-width. The $log_2(softmax(\cdot))$ always outputs a negative value since softmax result lies within (0, 1). Eq. (2) can be considered as the log2 quantization of the attention map ($Attn = softmax(Q \cdot K^T/\sqrt{d_k})$), since it can be computed by 2^{-Attn_Q}. With such log2 quantization, we can express a wider range of values in the attention map, and compute the ouput '$Z = Attn \cdot V$' with simple shift-and-accumulate operations. However, the previous work [8] requires FP multipliers, FP dividers, and log2 unit to compute $log_2(softmax(\cdot))$. In OPAL, we simplify the hardware of log2-based softmax by modifying the computation as follows:

$$\begin{aligned} \lceil log_2(\frac{e^{x_i}}{\Sigma e^{x_i}})\rfloor &= \lceil log_2(e^{x_i}) - log_2(\sum e^{x_i})\rfloor \\ &= \lceil log_2(2^{E_i} \cdot 1.M_i) - log_2(2^{E_\Sigma} \cdot 1.M_\Sigma)\rfloor \\ &= (E_i - E_\Sigma) + \lceil log_2(1.M_i) - log_2(1.M_\Sigma)\rfloor \\ &= (E_i - E_\Sigma) + Sign(M_i - M_\Sigma) \circ 1_{|M_i - M_\Sigma| \geq 0.5}, \end{aligned} \quad (3)$$

where E_i is the exponent of e^{x_i}, and $1.M_i$ is '1 + mantissa' of e^{x_i}, E_Σ is the exponent of $\sum e^{x_i}$, and $1.M_\Sigma$ is '1 + mantissa' of $\sum e^{x_i}$. Since $1.M$ always exists between 1 and 2, $log_2(1.M)$ is a value between 0 and 1. As a result, we can compute the complex '$log_2(softmax(\cdot))$' function using INT subtractors for $(E_i - E_\Sigma)$ and $(M_i - M_\Sigma)$, and comparators.

4.3 OPAL Microarchitecture

4.3.1 Data Distributor. The microarchitecture of the main core in OPAL is illustrated in Fig. 6(a). Each core consists of eight data distributors, eight MxV compute lanes, an FP adder tree, a log2-based softmax unit, and an MX-OPAL quantizer. To each core, 128 activations, either in low-bit INT (3 or 4-bit) or high-bit INT (5 or 7-bit) with one 4-bit shared scale, are provided as an input which includes four bfloat16 (BF16) outliers. The data distributor attached in front of each compute lane routes non-outliers to INT multiply units (`INT MUs`) and outliers to `FP units`. Since there exist more outliers in activations than weights (~3% vs. ~0.3%), we convert some channels in the weight matrix that align with the activation outliers to BF16 from INT3/4 (highlighted in a blue box in Fig. 6(b)). Most

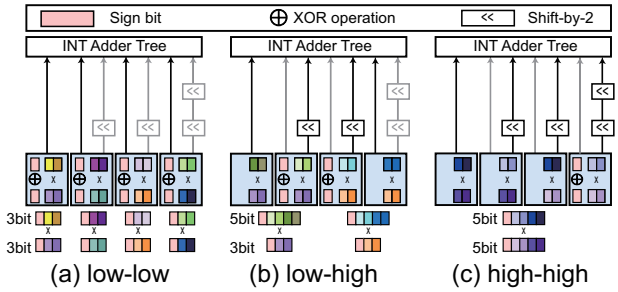

Figure 7: Operation modes of an `INT MU`: (a) `low-low` mode, (b) `low-high` mode, and (c) `high-high` mode.

Table 1: Perplexity (↓; the lower, the better) of Llama2 and OPT tested on WikiText-2 at various quantization schemes

Model	Llama2-7B	Llama2-13B	OPT-6.7B	OPT-13B
bfloat16 (BF16) (Baseline)	5.472	4.884	10.854	10.132
W4A16 (OWQ [5])	6.031	5.113	10.949	10.275
W4A7 (MinMax [16])	6.693	5.267	10.903	10.179
W4A7 (MX-OPAL)	6.431	5.256	10.894	10.191
W4A4/7 (MinMax [16])	6.546	5.472	12.071	11.586
W4A4/7 (MX-OPAL)	6.492	5.302	10.977	10.312
W3A16 (OWQ [5])	6.684	5.653	11.183	11.741
W3A3/5 (MinMax [16])	32.747	10.835	28.742	95.791
W3A3/5 (MX-OPAL)	7.400	6.168	14.343	16.225

activations and weights are directed to `INT MUs` saving significant amount of power in OPAL.

4.3.2 Compute Lane. Each compute lane consists of 32 `INT MUs`, one of which contains four INT multipliers. Each lane also has four `FP Units` to handle outliers. Each `INT MU` can be reconfigured to operate at either 3- or 5-bit mode, or it can be designed to support 4- and 7-bit modes for more accurate LLM inference (Section 5.1) at a higher hardware cost. In this work, all weights except outliers are stored in INT3 or INT4 (i.e., low-bit INT) based on OWQ [5] method. Thus, the `INT MU` in OPAL supports three modes, i.e., '`low-low`', '`low-high`', and '`high-high`' modes (Fig. 7). The `low-low` mode is set to perform MxV between non-outlier weights and low-bit activations (i.e., after LN layer). The `low-high` mode is used for MxV between non-outlier weights and high-bit activations. The `high-high` mode is used to perform MxV between high-bit activations, such as $Q \cdot K^T$. Our `INT MUs` can flexibly support three modes, with the `low-low` mode providing 4× throughput over the `high-high` mode. The `INT MU` outputs are accumulated to a single output through an INT Adder Tree, and the result is converted to BF16 by incorporating the shared scale via an Int-to-FP unit. Then, it is accumulated with BF16 results from outlier FP units.

4.3.3 Softmax Unit and MX-OPAL Quantizer. After MxV computations, the final output is generated by passing the results from eight lanes through the FP Adder Tree. Then, outputs are selectively moved (for $Q \cdot K^T$) to log2-based softmax unit to transform them to attention map (Fig. 6(c)). By using our log2-based softmax, we can cut down 32.3% of the area and 35.7% of the power compared to the conventional softmax unit. The 128 output values in BF16 are then grouped together and converted to the MX-OPAL format

Table 2: Performance of Llama2 tested on language modeling with WikiText-2 (`Wiki`) and C4 (`C4`) (perplexity ↓), and on zero-shot question answering tasks with ARC_Challenge (`ARC`) and Physical Interaction: Question Answering (`PIQA`) (accuracy ↑)

Task	Wiki ↓	C4 ↓	ARC ↑	PIQA ↑
Llama2-7B OWQ W4A16	6.031	7.744	37.97	76.55
Llama2-7B MX-OPAL W4A4/7	6.492	8.084	37.80	75.24
Llama2-7B OWQ W3A16	6.684	8.645	37.46	75.63
Llama2-7B MX-OPAL W3A3/5	7.400	9.588	35.49	73.29
Llama2-13B OWQ W4A16	5.113	7.051	42.75	77.74
Llama2-13B MX-OPAL W4A4/7	5.302	7.313	42.41	77.69
Llama2-13B OWQ W3A16	5.653	7.710	39.33	76.88
Llama2-13B MX-OPAL W3A3/5	6.168	8.445	38.73	75.03
Llama2-70B OWQ W4A16	3.496	5.837	46.75	81.01
Llama2-70B MX-OPAL W4A4/7	3.596	5.933	46.50	80.96
Llama2-70B OWQ W3A16	3.997	6.227	47.61	80.46
Llama2-70B MX-OPAL W3A3/5	4.348	6.573	45.81	79.11

Table 3: Area and power breakdowns of one OPAL core

OPAL Core (`W4A4/7`)	Area (μm^2)	Power (mW)
Compute Lanes	670,126.34 (72.11%)	229.65 (68.38%)
Data distributors	139,713.48 (15.03%)	63.20 (18.82%)
Log2-based Softmax Unit	76,330.92 (8.21%)	27.62 (8.22%)
MX-OPAL Quantizer	34,670.88 (3.73%)	14.11 (4.20%)
FP Adder Tree	8,470.80 (0.91%)	1.28 (0.38%)
Total	**929,312.41**	**335.85**

at the shift-based MX-OPAL quantizer to minimize the size of intermediate data. Then, the output of the quantizer is connected to either LN hardware or SRAM that is outside the OPAL core.

5 EXPERIMENTAL RESULTS

5.1 Accuracy Analysis of MX-OPAL

We modified block floating point (BFP) implementation in QPyTorch [3] to evaluate our MX-OPAL format on inferencing LLMs at various language tasks. We have quantized activations prior to all MxV operations, as shown in Fig. 5. Quantizing to MX-OPAL on activations along with low-bit OWQ on weights allows mostly integer operations, reducing the latency of the LLM inference. Only channels where outliers happen at either activations or weights are handled by BF16 units. Due to the high storage requirement of weights in LLM (making LLM memory-bounded), we perform aggressive quantization using OWQ [5] to quantize weights to 4-bit (or 3-bit) with only 0.25% (or 0.33%) of BF16 outliers.

In Table 1, we compared the performance of Llama2 and OPT models on language modeling (WikiText-2) at various quantization schemes. Activations after layernorm (LN) are pushed to a low bit-width (i.e., 4-bit or 3-bit) to maximize the efficiency, since the normalized distribution becomes more robust to quantization noise. Accordingly, we evaluate the activation quantization of `A4/7` and `A3/5` (i.e., low-bit/high-bit in Fig. 5). All weights are quantized to the low-bit for all cases, i.e., 4-bit (`W4`) or 3-bit (`W3`). With `W4A4/7`, MX-OPAL merely increases the perplexity (PPL) by 0.435 on average, while MinMax increases PPL by 1.083, when compared to the BF16 baseline. The performance gap between the proposed MX-OPAL (0.616↑ on Llama2) and the conventional MinMax quantization (3.822↑ on Llama2) becomes much larger at `W3A3/5`. Compared

to OWQ, i.e., W4A16 (or W3A16), MX-OPAL increases PPL by 0.325 (or 0.616) on Llama2 and 0.0325 (or 3.822) on OPT, respectively. Note that OPT performs worse even at BF16 and is more prone to aggressive quantization than Llama2 due to architectural difference.

Thus, we evaluated MX-OPAL in more detail using Llama2 on various language tasks with WikiText-2, C4, ARC_Challenge, and PIQA datasets. Since the main focus of MX-OPAL is in activation quantization, we compared all benchmarks to OWQ which uses BF16 for activations, while keeping the weights in the same bit-width. When using W4A4/7, MX-OPAL only increases PPL by 0.241 (on Wiki and C4) and loses accuracy by 0.36% (on ARC and PIQA), on average. With more aggressive quantization, i.e., W3A3/5, MX-OPAL increases PPL by 0.601 (on Wiki and C4) and loses accuracy by 1.65% (on ARC and PIQA), on average.

5.2 Hardware Efficiency of OPAL

To evaluate the proposed OPAL hardware in detail, we implemented RTL of all building blocks shown in Fig. 6 excepts SRAMs. Then, we used Synopsys Design Compiler to estimate power consumption and area based on 65nm CMOS technology. To estimate the energy consumption of on-chip memory, including the leakage power, CACTI [9] was used. We assume an OPAL hardware with a 512KB global buffer for weights and activations, and a 2KB softmax buffer within the OPAL core. The eight lanes in each OPAL core are capable of computing $32 \times 8 = 256$ MACs in the high-high mode. The number of operations ramps up to 512 and 1,024 in the low-high and low-low modes, respectively. Table 3 reports the area and power breakdowns of one OPAL core that uses a W4A4/7 format. As expected, most of the power and area (72% and 68%, respectively) is consumed by eight compute lanes.

To evaluate the area and energy efficiency of the OPAL hardware, we have set two baselines, i.e., an BF16-based and an OWQ-based accelerator. Since the inference of an LLM-based generation task is latency critical, a large portion of on-chip area is occupied by the global buffer. The main challenge in deploying a large on-chip buffer lies not only in the large memory footprint but also in the high leakage power. Fig. 8 compares the energy consumption of generating one token with Llama2-70B and the area of two OPAL hardware variants (W3A3/5 and W4A4/7) to two baselines. The latency of generating one token was 1.98 sec for Llama2-70B on OPAL. The OWQ saves 32.5% of the energy consumption on average by shrinking down the weight buffer while keeping intermediate activation data in BF16 (+ all computations are done in BF16 units). By utilizing the proposed MX-OPAL format and its dedicated hardware, we can save the energy consumption by 53.5%/68.6% with W3A3/5 and 38.6%/58.6% with W4A4/7 on average compared to OWQ/BF16, respectively. The OPAL hardware saves the energy by reducing the required size of on-chip buffer (effectively, reducing the leakage energy), and by simplifying the design of the main core.

6 CONCLUSION

In this paper, we proposed OPAL, an energy-efficient hardware accelerator for generative LLMs. To achieve high energy efficiency, we focused on quantizing activations to low bit-width INT format with a shared scale, i.e., MX-OPAL format, while preserving several outliers for every 128 elements. By using the proposed MX-OPAL format, 96.9% of computations are done in INT multipliers significantly saving the computing energy. In addition, we

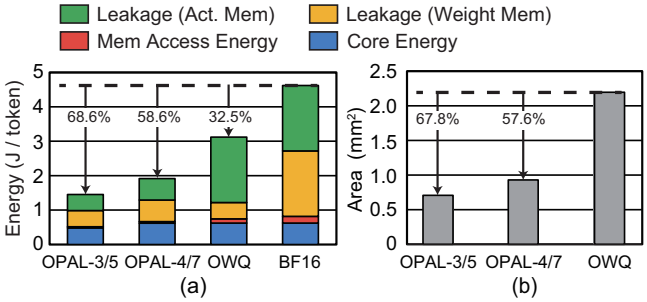

Figure 8: Comparison on (a) the energy consumption of generating one token with Llama2-70B and (b) the area between OPAL and other baselines (OWQ [5] and BF16).

could reduce the required size of on-chip activation buffers owing to the lower activation bit-width significantly saving the leakage energy in the global buffer. Overall, we were able to reduce the total energy of LLM inference on generation tasks by up to 46.5% using the W3A3/5 MX-OPAL format compared to the weight-only quantization method (OWQ), with negligible accuracy loss.

ACKNOWLEDGMENT

This work was partially supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant (RS-2023-00229849), and the National Research Foundation of Korea (NRF) grant (NRF-2023R1A2C2006290) funded by the Korean government (MSIT).

REFERENCES

[1] T. Dettmers et al. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35. 30318–30332.
[2] E. Frantar et al. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323.
[3] N. Ho et al. 2022. Qtorch+: Next Generation Arithmetic for Pytorch Machine Learning. In *Conference on Next Generation Arithmetic (CoNGA)*. Springer, 31–49.
[4] W. Kwon et al. 2022. A Fast Post-Training Pruning Framework for Transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*. 24101–24116.
[5] C. Lee et al. 2024. OWQ: Outlier-Aware Weight Quantization for Efficient Fine-Tuning and Inference of Large Language Models. In *Proc. The Association for the Advancement of Artificial Intelligence (AAAI)*, Vol. 38. 13355–13364.
[6] S. Lee et al. 2023. DBPS: Dynamic Block Size and Precision Scaling for Efficient DNN Training Supported by RISC-V ISA Extensions. In *Proc. ACM/IEEE Design Automation Conference (DAC)*. 1–6.
[7] Z. Li et al. 2023. I-ViT: Integer-only Quantization for Efficient Vision Transformer Inference. In *Proc. IEEE/CVF International Conference on Computer Vision (CVPR)*. 17065–17075.
[8] Y. Lin et al. 2022. FQ-ViT: Post-Training Quantization for Fully Quantized Vision Transformer. In *Proc. International Joint Conference on Artificial Intelligence, IJCAI*. 1173–1179.
[9] N. Muralimanohar et al. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.
[10] NVIDIA. 2019. *CUTLASS: CUDA Templates for Linear Algebra Subroutines and Solvers.* https://nvidia.github.io/cutlass/
[11] B. D. Rouhani et al. 2023. Microscaling Data Formats for Deep Learning. arXiv:2310.10537.
[12] C. Saharia et al. 2022. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35. 36479–36494.
[13] J. R. Stevens et al. 2021. Softermax: Hardware/Software Co-design of an Efficient Softmax for Transformers. In *Proc. ACM/IEEE Design Automation Conference (DAC)*. IEEE, 469–474.
[14] H. Touvron et al. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971.
[15] A. Vaswani et al. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 30. 1–11.
[16] Z. Yao et al. 2022. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35. 27168–27183.