PacQ: A SIMT Microarchitecture for Efficient Dataflow in Hyper-asymmetric GEMMs

Ruokai Yin
Yale University
ruokai.yin@yale.edu

Yuhang Li
Yale University
yuhang.li@yale.edu

Priyadarshini Panda
Yale University
priya.panda@yale.edu

***Abstract*—Weight-only quantization has been widely explored in large language models (LLMs) to reduce memory storage and data loading overhead. During deployment on single-instruction-multiple-threads (SIMT) architectures, weights are stored in low-precision integer (INT) format, while activations remain in full-precision floating-point (FP) format to preserve inference accuracy. Although memory footprint and data loading requirements for weight matrices are reduced, computation performance gains remain limited due to the need to convert weights back to FP format through unpacking and dequantization before GEMM operations. In this work, we investigate methods to accelerate GEMM operations involving packed low-precision INT weights and high-precision FP activations, defining this as the hyper-asymmetric GEMM problem. Our approach co-optimizes tile-level packing and dataflow strategies for INT weight matrices. We further design a specialized FP-INT multiplier unit tailored to our packing and dataflow strategies, enabling parallel processing of multiple INT weights. Finally, we integrate the packing, dataflow, and multiplier unit into PacQ, a SIMT microarchitecture designed to efficiently accelerate hyper-asymmetric GEMMs. We show that PacQ can achieve up to $1.99\times$ speedup and 81.4% reduction in EDP compared to weight-only quantized LLM workloads running on conventional SIMT baselines.**

I. Introduction

Large language models (LLMs) have demonstrated exceptional performance across a wide range of complex natural language processing tasks [19], [23]. However, their substantial parameter counts introduce significant deployment challenges, including increased memory footprint and computational costs. Quantization has emerged as a popular solution to improve the deployment efficiency of LLMs, particularly on edge devices.

Conventional deep neural network (DNN) quantization reduces both computation and memory traffic by compressing weights and activations into low-precision integers [5], [8]. However, quantizing activations in LLMs is particularly challenging due to their dynamic range and the occurrence of salient values [10], [21]. As a result, many recent studies have focused on weight-only quantization, where weights are compressed to very low-precision integers (e.g., $\leq$INT4), while activations are preserved in high-precision floating-point formats, typically FP16 [2], [9], [10], [20], [21].

While weight quantization significantly reduces memory traffic for LLMs (e.g., Llama2-70B [19] requires 131.6 GB with FP16 weights but only 35.8 GB with INT4 weight-only quantization), it does not lower computational costs. This limitation stems from the current deployment workflow on single-instruction multiple-thread (SIMT) hardware, widely used in commercial ML accelerators such as NVIDIA GPUs. As shown in Figure 1(a), the inference deployment flow of weight-only quantized LLMs on SIMT hardware faces three major challenges: (1) ***Inefficient memory hierarchy utilization:*** Although low-precision integer (INT) weights are stored and fetched from off-chip DRAM in packed format, they are unpacked and dequantized into high-precision floating-point (FP) format upon loading into the L1 cache, losing memory benefits at and above the L1 cache. (2) ***Overhead of unpacking and dequantization:*** Unpacking and dequantizing weights introduce significant latency and computational overhead [10]. (3) ***Lack of computational savings:*** GEMM computations are performed in FP after dequantization, forfeiting the computational efficiency of low-precision integer weights.

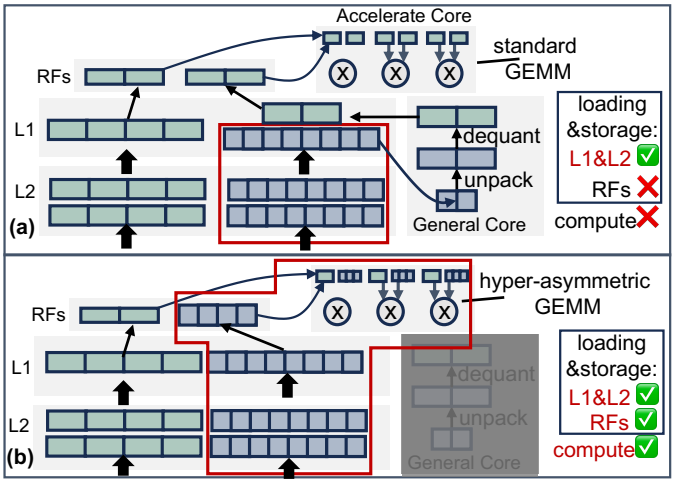

Fig. 1. (a) Standard inference flow on SIMT architecture for weight-only quantized LLMs. Green denotes high-precision floating-point activations, and blue denotes low-precision integer weights. (b) Our proposed flow leverages the hyper-asymmetric GEMM to achieve the benefits of reduced loading, storage, and compute costs from low-precision integer weights throughout the entire GEMM computation stack.

Despite these limitations, existing weight-only quantization techniques often achieve inference speedup on SIMT architectures in memory-bound scenarios, such as single-batch text generation. However, real-world LLM serving systems predominantly adopt multi-batch processing to achieve better hardware utilization [22]. Multi-batch processing, in contrast, is typically compute-bound, with its performance constrained by the three limitations outlined earlier.

To address the limitations in deploying weight-only quan-

tized models on SIMT architectures, we propose to retain INT weights in packed format throughout the entire GEMM computation stack. We define this problem as **hyper-asymmetric GEMM**, where the operand precisions are highly unbalanced (asymmetric), as illustrated in Figure 1(b). In this work, we explore the design space for accelerating hyper-asymmetric GEMM on SIMT-based hardware. Our key contributions are:
(1) We pinpoint the critical role of INT weight packing direction, a factor often overlooked in prior work, for optimizing hyper-asymmetric GEMM. Specifically, we analyze the inefficiencies of packing weights along the input-feature (k) dimension, a common approach in existing quantized LLM deployment frameworks [1], [11]. To address this, we propose an alternative packing strategy and dataflow optimized for hyper-asymmetric GEMM. Our strategy has 54.3% reduction of register file accesses compared to baseline approaches.
(2) We propose a parallel FP-INT multiplier optimized for our packing and dataflow strategy. By identifying constant patterns in the FP representation of INT weight values, our design efficiently processes four FP16×INT4 or eight FP16×INT2 multiplications in parallel within a single cycle, reusing ∼73% of hardware resources from standard FP16 multipliers. This design achieves up to 6.8× throughput/watt improvement compared to conventional designs.
(3) We encapsulate the proposed packing, dataflow, and parallel multiplier design into PacQ, a SIMT microarchitecture tailored for hyper-asymmetric GEMM. Compared to NVIDIA GPUs as a baseline, PacQ achieves up to 81.4% EDP reduction for selected LLM inference workloads.

II. Background

SIMT Architecture. The single-instruction multiple-thread (SIMT) architectures are widely used in commercial ML accelerators, such as NVIDIA GPUs, to achieve high computational, memory, and control parallelism via sets of consecutively indexed threads (warps). These architectures tile large ML workloads (e.g., GEMMs) into smaller blocks mapped to individual threads, improving bandwidth efficiency and data reuse [18]. NVIDIA's Volta GPUs introduced the first streaming multiprocessors (SMs) with dedicated ML acceleration capabilities, including tensor cores (TCs), specialized for GEMM acceleration [14], [16].

In Volta, multiple SMs are interconnected via an on-chip network to a last-level cache backed up by HBM. Each SM contains sub-cores sharing an L1 cache, with sub-cores further divided into general-purpose cores and tensor cores. General-purpose cores handle operations like unpacking and dequantization, while tensor cores are optimized for GEMM operations. Operand data is stored in register files for efficient reuse before being processed by tensor cores. Without loss of generality, our study examines the dataflow and microarchitecture of Volta's tensor cores as the baseline.

Floating Point Number. In Figure 2, we show the standard IEEE 754 format for FP16. In this work, we focus on the normalized format (i.e., the hidden bit for the mantissa is always 1). During the floating point multiplication, the hidden bit needs to be considered for the integer multiplication between the mantissas, as shown on the right side of Figure 2.

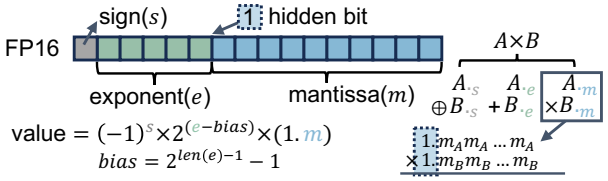

Fig. 2. IEEE 754 standard FP16 format.

Related Work. Previous studies [4], [17] on accelerating mixed-precision GEMM for quantized DNNs have primarily targeted workloads with low operand precision imbalance (difference $\leq 4\times$). In contrast, our work focuses on hyper-asymmetric GEMM, characterized by a high operand precision imbalance (difference$\geq 4\times$).

Recent work, FIGNA [6], introduced an FP-INT unit to eliminate unpacking and dequantization overhead during the deployment of weight-only quantized LLMs. However, their approach emphasizes co-designing the FP-INT multiplier with a block floating-point format. In contrast, our work integrates the FP-INT multiplier design with packing and dataflow strategies for processing packed INT weights. Furthermore, our design improves throughput by enabling parallel multiplication of FP activations with multiple INT weights, a capability not explored in FIGNA.

Other methods for accelerating mixed-precision GEMM in weight-only quantized LLMs employ lookup table (LUT) techniques, either by designing new LUT-friendly hardware [12] or focusing on single-batch text generation on GPUs [15]. In contrast, our work enhances general SIMT-based architectures to support multi-batch inference, a scenario more common in real-world LLM deployments. Additionally, approaches targeting efficient dequantization kernels [10], [24] address complementary challenges and are orthogonal to our work.

III. Packing flow for Hyper-asymmetric GEMM

Basic GEMM Dataflow. For illustration, we describe the Volta GPU's execution of the FP16 tensor core MMA instruction *mma.sync.m16n16k16* for *W16A16*, where both weights and activations are in FP16 precision, as shown in Figure 3. In this example, each matrix has a size of 16×16. As shown in Figure 3(a), matrices A, B, and C are collaboratively fetched by a warp consisting of 32 threads. The workload is then evenly distributed among four *octets* [16] (groups of 8 threads), as illustrated in Figure 3(b). As depicted by Figure 3(c), each *octet* fetches tiles using a weight-stationary dataflow, represented by the left three-nested-for-loop, and computations within each tile follow an output-stationary dataflow, represented by the right three-nested-for-loop. Finally, Figure 3(d) illustrates how octet-level workloads are mapped onto the hardware (i.e., tensor cores in Volta GPUs). A 4×4 tile of B is fetched from the register files into a buffer shared by all eight threads within the *octet*. Meanwhile, 2×4 tiles of A are loaded into separate buffers, each shared by four threads. Two four-element dot-product units (DP-4) are responsible to compute the 4×4 tile of C.

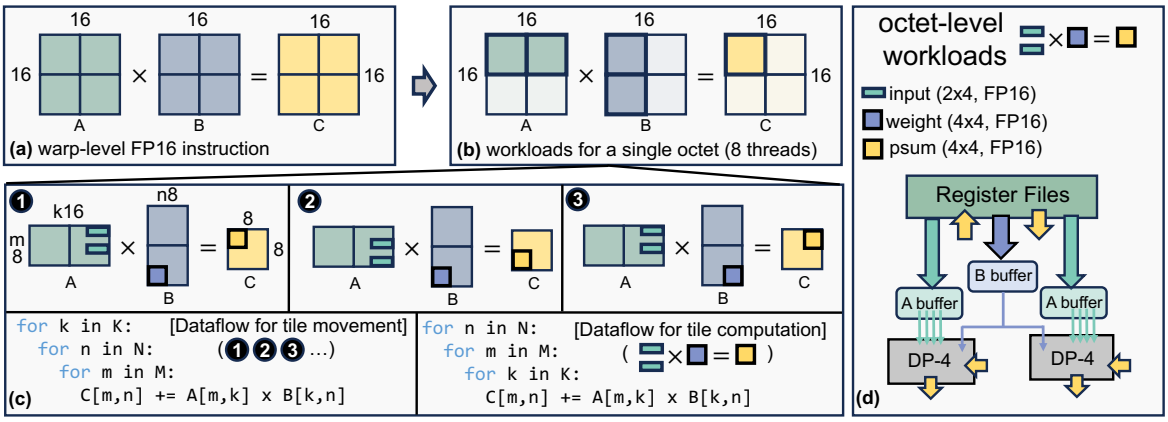

Fig. 3. (a) & (b): The tile mapping of the warp-level (a set of 32 threads) *mma.sync.m16n16k16* instruction to different octets (a subset of 8 threads). Here, A represents the input matrix with a shape of $[m, k]$, B represents the weight matrix with a shape of $[k, n]$, and C represents the output (partial-sum) matrix with a shape of $[m, n]$. (c) An example of a step-by-step breakdown of the octet-level data movement and computation flow. (d) The mapping of octet-level workloads onto the baseline tensor processing units. DP-4 stands for four-element dot-product units that performs 4×4 inner-product.

Hyper-asymmetric GEMM Packing and Dataflow. We define hyper-asymmetric GEMM as a GEMM operation where the operands A and B have a precision difference of at least $4\times$ (e.g., *W4A16* or *W2A16*). While largely overlooked in prior dequantization-based GEMM methods, the dimension along which the INT weights are packed becomes a critical factor in hyper-asymmetric GEMM. Formally, we define the packing of weight matrix B using the format $P(B_x)_y$, where x represents the number of elements of B packed along the y dimension. For instance, in *W4A16*, if the weights (B) are packed along the k dimension into the INT16 datatype, the packing can be formally described as $P(B_4)_k$. Once fetched to the L1 cache, the INT16 is unpacked to 4 INT4 weights and dequantized to FP16 for being executed in the standard FP16 GEMM.

Although there is limited research on the packing direction of INT weight matrices, nearly all prior weight-only LLM quantization frameworks [1], [3], [11] choose to pack weights along the input feature dimension (k-dim). While the packing dimension is unimportant in dequantization-based GEMM, as packed INT weights are unpacked at the L1 cache regardless of the packing direction, this is not the case for hyper-asymmetric GEMM. In hyper-asymmetric GEMM, INT operands are fetched in their packed form, creating hyper-asymmetry in the amount of data staged in the registers and tensor cores. An improper packing dimension combined with this hyper-asymmetry leads to significant performance degradation.

Specifically, when weights are packed along the k-dim, each packed INT weight fetch requires multiple activation fetch instructions because the k-dim must align between the two operands during GEMM computation. Figure 4(a) provides an example: for processing INT weights packed in the $P(B_4)_k$ format, four distinct fetch instructions for A must be issued to align the operands. Furthermore, packing INT weights (B) along the k-dim results in poor data reuse of activations (A). As illustrated in Figure 4(b), new data from A along the k-dim must be continuously fetched, leading to eviction of buffers inside the tensor cores and preventing data reuse of A. This causes excessive data access from large register files,

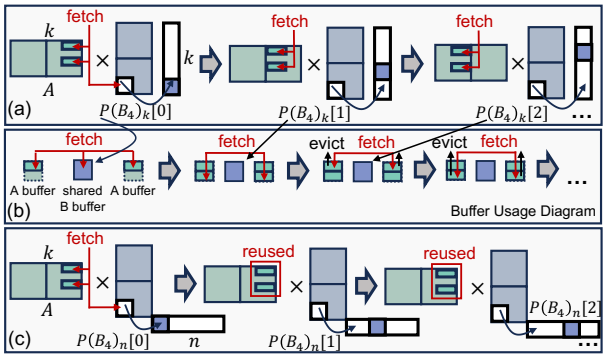

Fig. 4. (a) Illustration of multiple fetch instructions required by the hyper-asymmetric GEMM when packing weights along the k dimension. (b) Illustration of the poor data reuse of A in k dimension packing. (c) Illustration of the improved data reuse and reduction in fetch instructions achieved by packing along the n dimension. The tile size is identical to Figure. 3.

which is highly power-intensive. When the operand precision asymmetry is significant, this issue can even escalate beyond the register file level to the L1 cache.

We argue that INT weights should instead be packed along the output feature (n) dimension. This approach avoids additional internal loops of fetch instructions compared to processing INT weights packed along the k-dim. As shown in Figure 4(c), once the activations (A) and packed weights ($P(B_4)_n$) are fetched following the standard flow, no extra fetch instructions are required. Furthermore, our proposed packing strategy ensures no eviction of A during the processing of packed B, enabling full reuse of activations across the packed weights. Moreover, instead of using the weight stationary dataflow for tile movement as in Figure 3(c), we use output stationary for both tile movement and tile computation. This packing and dataflow combination strikes a good balance between the data reuse and the partial sum traffic.

IV. Proposed Parallel FP-INT Multiplier

As discussed previously, our proposed packing and data flow reuses activations across all packed INT weights within a single data fetch instruction. In the standard approach,

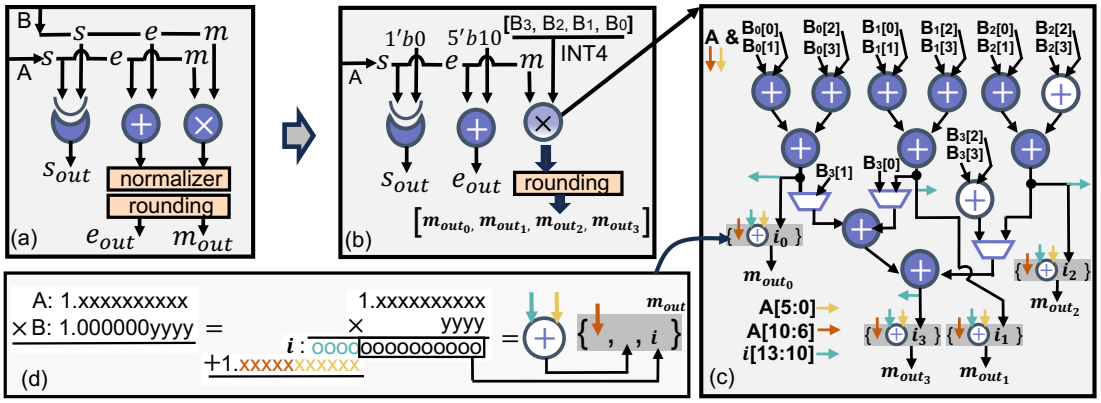

Fig. 5. (a) Standard FP multiplier design. s stands for sign, e represents exponent, and m denotes the mantissa. (b) Overview of our proposed parallel FP-INT-16 multiplier. In the provided example, our design can generate four output mantissas in one cycle. (c) Detailed diagram of the modifications made to the original 11-bit integer multiplier. The elements in purple are part of the original multiplier, while the elements in white represent additional units. All bits of B's mantissas perform a logical AND operation with shifted A's mantissas before being fed into the adders. (d) Explanation of how the final output mantissa (shaded gray) is assembled from different value sources. {} denotes the append operation.

these data are dequantized and processed sequentially by the pipelined FP multiplier. However, we identify an opportunity to perform the multiplication between a single FP activation and multiple INT weights in parallel. To exploit this, we propose a specialized parallel FP-INT multiplier.

Observation from FP representation of INT values. When representing an INT $x \in [1024, 2048)$ in FP16 format, certain patterns emerge during the conversion process. First, the FP16 representation of x consistently has an exponent value of 11001 (corresponding to $2^{25-15} = 1024$ in decimal). Second, the mantissa of x is always in the form of $10'b0|(x - 1024)$. Based on these patterns, we make further observations on representing low-precision INT weights in FP16 format. Without loss of generality, we use INT4 as an example, which can be easily extended to INT2 [7].

Assume $B \in [-8, 7)$ is a signed INT4 weight scalar. To simplify arithmetic operations, we first transform B to an unsigned representation by adding 8, resulting in $B+8 \in [0, 15)$. Since B is in INT4 format, we ensure that $B + 8 + 1024$ $(B + 1032)$ lies within the range $[1024, 2048)$. Leveraging the observed patterns, we conclude: ❶ The FP16 representation of $B + 1032$ always has an exponent value of 11001. ❷ The mantissa of $B + 1032$ is consistently in the form $000000yyyy$, where $yyyy$ is the 4-bit integer representation of $B + 8$.

Proposed Design. Leveraging ❶ and ❷, we design a parallel FP-INT multiplier that maximally reuses the hardware resources of the original FP multiplier, as shown in Figure 5(a). Our design enables the multiplication of one FP16 input with four INT4 values or eight INT2 values in parallel, producing all outputs in a single cycle. For illustration, we focus on INT4.

Since B is transformed into an unsigned representation, the signs of all four outputs depend only on the sign of A. Thus, we compute the common output sign s_{out} by XOR-ing A's sign bit with 0. Using ❶, we derive the shared exponent e_{out} for all outputs by adding A's 5-bit exponent to 11001, as shown in Figure 5(b). Figure 5(c) depicts the details of our proposed parallel FP-INT multiplier. The original FP16 integer

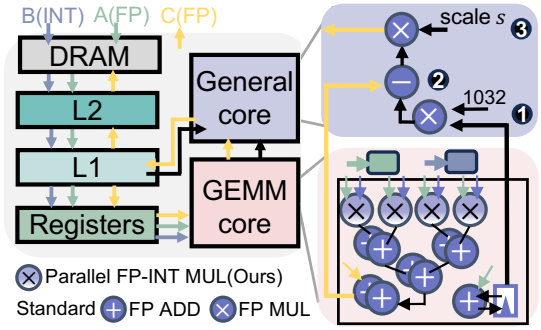

Fig. 6. Overview of the PacQ architecture. B represents the INT weight vectors, A represents the FP activation vectors, and C represents the outputs or partial sums in FP16. The steps illustrated in the general core reflect only the required instructions in sequence. PacQ does not require any hardware modifications to the general core.

multiplier requires 10 parallel adders (16-bit) to compute one 11-bit×11-bit multiplications (colored purple). Utilizing ❷, we break the original four 11-bit×11-bit multiplications into four 11-bit×4-bit multiplication. By introducing two additional 16-bit parallel adders (colored white), our design simultaneously performs all four 11-bit×4-bit multiplications.

Finally, as shown in Figure 5(d), we ensure the hidden bit of B's mantissa is accounted for by performing a 6-bit addition between the top-4 MSBs of the intermediate results i (colored turquoise, from 11-bit×4-bit) and the 6 LSBs of A (colored yellow). We then concatenate the top-5 MSBs of A (colored brown) with the addition results and i to obtain the four final mantissa products m_{out}. These results are passed to the rounding units and truncated to 10 bits (excluding the hidden bit). Since both A and B have normalized mantissa and B's mantissa values are constrained to a 4-bit unsigned integer, normalization to the outputs is unnecessary.

Overall Architecture (PacQ). We encapsulate our proposed packing and dataflow strategies along with the parallel FP-INT multiplier units into PacQ: a SIMT microarchitecture for accelerating hyper-asymmetric GEMMs. The overall architecture is depicted in Figure 6. Our proposed microarchitecture retains

most components from the Volta GPU baseline, requiring hardware modifications only in the GEMM acceleration core (i.e., tensor core). We introduce the following three key changes to the original tensor core:

First, we replace all original FP16 multipliers with our proposed parallel FP-INT multipliers. **Second**, we duplicate the original adder trees in the DP-4 by twice, enabling the accumulation of the inner product of 16 values in 2 cycles for INT4 weights (or 32 values in 4 cycles for INT2 weights). **Third**, we equip the GEMM core with small accumulators to store processed A values. Recall that before feeding the packed INT weights into the parallel FP-INT multipliers, we transform all weight values B into $B + 1032$. To retrieve the original weight values, we subtract 1032 later in the computation. By leveraging the small accumulators, we fuse this subtraction directly into the inner product calculation:

$$\sum_{0}^{k}(A_k(B_k - 1032)) = \sum_{0}^{k} A_k B_k - 1032 \times \sum_{0}^{k} A_k, \quad (1)$$

where k denotes the dimension of the inner product. The accumulators are responsible for generating the results of $\sum_0^k A_k$. As shown in the general core of Figure 6, the accumulated results are multiplied by 1032 in ❶, then subtracted from the inner product results ($\sum_0^k A_k B_k$) in ❷. Finally, the group quantization scales s are applied in ❸ to scale the outputs back to their expected range.

V. Experimental Results

Experimental Setup. For the packing and dataflow evaluation, we developed a custom simulator in Python to monitor memory access patterns and record execution cycles for PacQ and the baselines. Key components of PacQ and our baselines were implemented in RTL and synthesized using Synopsys Design Compiler at 400MHz with 32nm technology. We utilized CACTI7.0 [13] to model on-chip SRAM and register files for getting memory statistics. Configuration details for PacQ and the baseline architectures are summarized in Table I. Unless otherwise specified, in the rest of the section, INT2 and INT4 refer to the precision of weights, while activations and partial sums are always represented in FP16 format. For hardware unit comparisons, 'baseline' always refers to the standard baseline designs as denoted in Table. I. $P(B_{4(8)})_k$ denotes the baseline of hyper-asymmetric GEMM flow that packs 4 INT4 (8 INT2) weights in one INT16 along k-dim.

Proposed Packing and Data Flow. In Figure. 7(a), we present the number of register file accesses for PacQ compared to the hyper-asymmetric GEMM with INT4 (INT2) weights packed along k. The results demonstrate that with improved data reuse, our approach achieves up to a 54.3% reduction in register file accesses. Furthermore, in Figure. 7(b), we show the normalized speedup of PacQ compared to the hyper-asymmetric GEMM packing along k. Thanks to our parallel processing strategy, we achieve an average speedup of $1.99\times$.

We want to confirm here that PacQ ***does not require any quantization algorithm modifications*** to achieve hardware efficiency, as there is no approximation in our design.

TABLE I
Configuration of the PacQ and Baselines.

INT11 MUL (baseline)	10 INT16 adders
Parallel INT11 MUL	12 INT16 adders, 4 INT6 adders
FP16 MUL (baseline)	1 INT11 MUL, 1 INT5 adder 1 normalization unit, 1 rounding unit
Parallel FP-INT-16 MUL	1 parallel INT11 MUL, 1 INT5 adder 1 normalization unit, 4 rounding units
FP-16 DP-4 (baseline)	4 FP16 MUL, 4 FP16 adders
Parallel FP-INT-16 DP-4	4 parallel FP-INT-16 MUL, 8 FP16 adders
Tensor Core	4 Parallel FP-INT-16 DP-4 baseline: 4 FP16 DP-4 2×3072-bits buffer, 256KB register files [14]
Streaming Multiprocessor	8 tensor cores, 96 KB shared L1 cache

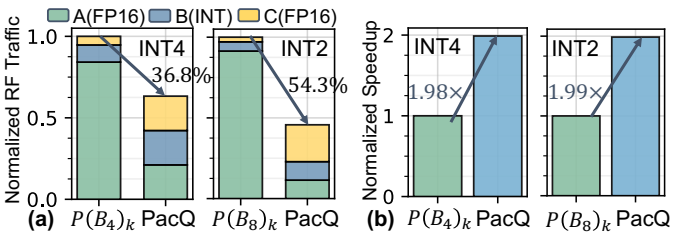

Fig. 7. (a) Comparison of the normalized number of register file accesses. (b) Comparison of normalized speedup between PacQ and hyper-asymmetric GEMM with INT weights packed along k. The workload is *m16n16k16*. $P(B_{4(8)})_k$ represents packing 4 INT4 weights (or 8 INT2 weights) into one INT16 data along the k dimension.

However, we observe that a small adjustment to the existing PTQ algorithm can further enhance the efficiency of PacQ. Specifically, by spanning the quantization group across both dimensions ($[n, k]$) instead of solely on the input feature dimension (k), we can reduce the number of fetches of the quantization scale s for PacQ in the general core, as illustrated in Figure 6-❸. We adapt the standard round-to-nearest (RTN) based PTQ algorithm to define quantization groups across both n and k dimensions within the existing framework [11]. The perplexity results for Llama2-7B [19] are shown in Table II. The results demonstrate that with these quantization group-dimension modifications, we achieve iso-perplexity with standard RTN 4-bit weight-only quantized models.

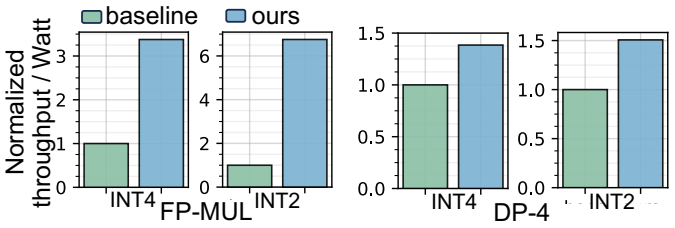

Fig. 8. Normalized performance comparison between the baseline FP16 designs with our parallel FP-INT-16 designs of running INT4 and INT2 weights. For DP-4, we consider the workload of *m2n4k4*.

Parallel FP-INT Multiplier. We first evaluate the throughput / Watt to measure the overall performance of our proposed parallel FP-INT multiplier. Figure 8 compares the normalized performance of our design against the original FP16 multiplier and the FP16 DP-4 unit. The throughput of the original FP16 multiplier is 1, whereas it is 4 (8) for our parallel design with

TABLE II
PERPLEXITY OF RTN-BASED PTQ LLAMA2 MODELS ON WIKITEXT-2 AND C4. W4A16-G128 REFERS TO 4-BIT WEIGHT-ONLY QUANTIZATION WITH A 128-GROUP SIZE ALONG k-DIM. G[32,4] INDICATES A 128-GROUP SIZE DISTRIBUTED AS 32 GROUPS ALONG k AND 4 GROUPS ALONG n.

Model Llama2-7B	W16A16 fp16 baseline	W4A16 g128	W4A16 g[32,4]	W4A16 g256	W4A16 g[64,4]
wikitext-2	5.47	5.73	5.72	5.75	5.77
C4	7.26	7.58	7.59	7.64	7.66

INT4 (INT2) weights. The original FP16 DP-4 unit requires 11 cycles to generate 8 FP16 outputs, while our parallel design takes 19 (35) cycles to generate 32 (64) FP16 outputs for INT4 (INT2) weights. Compared to the original FP16 multiplier, our design achieves 3.38× better performance for INT4 weights and 6.75× better performance for INT2.

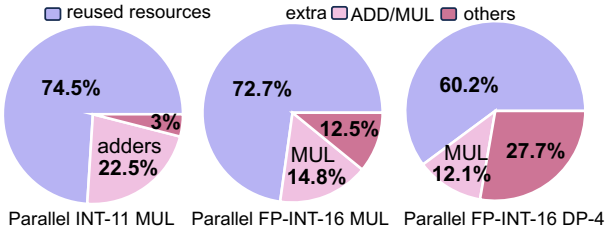

Fig. 9. Power breakdown of our proposed units. MUL stands for multiplier.

We further show the power breakdown of our proposed units in Figure 9. Our design objective is to maximize the reuse of hardware resources from the standard design. As shown by the purple-colored portions in the figure, we successfully reuse nearly 75% of the original INT-11 multiplier resources, which serve as the fundamental building blocks in our parallel FP-INT multiplier. For the DP-4 unit, we achieve approximately 60% hardware resource reuse. The observed decrease in the resource reuse ratio stems from the duplication of additional units; for instance, we needed to double the FP16 adder trees in the DP-4 design. Despite this, our design maintains an average hardware resource reuse ratio of 69%, which significantly minimizes resource duplication and enhances overall efficiency.

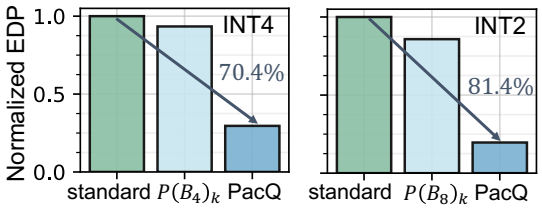

Fig. 10. Normalized EDP of PacQ vs. baselines. 'Standard' refers to the standard dequantization-based W16A16 GEMM. $P(B_{4(8)})_k$ denotes the hyper-asymmetric GEMM with INT4 (INT2) weights packed along k.

Other Results. In Figure. 10, we present the overall energy-delay-product (EDP) comparison of PacQ against Volta-like SIMT baselines. Two baseline cases are evaluated: the standard dequantization-based GEMM flow and the hyper-asymmetric GEMM flow with weights packed along the k dimension. Results demonstrate that PacQ achieves up to an 81.4% reduction in EDP for the workload $m16n4096k4096$, which represents a FFN layer in Llama2-7B with 16 batches.

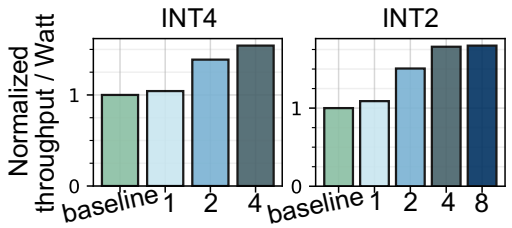

Fig. 11. Ablation study on the adder tree duplication. The number indicates the level of duplication of the adder trees (with a total of 4 FP16 adders in the baseline). For example, a duplication level of 4 would indicate 16 FP16 adders in our parallel FP-INT-16 DP-4 design.

In Figure. 11, we investigate the impact of duplicating FP16 adder trees in our parallel FP-INT DP-4 unit on overall performance (throughput / Watt) for the *m16n16k16* workload. We observe that while increasing the level of duplication continues to improve performance, a duplication factor of 2 provides the most significant gain. It achieves a 1.33× (1.38×) improvement over a duplication factor of 1 for INT4 (INT2). In comparison, a duplication factor of 4 results in only a 1.11× (1.18×) improvement over duplication factor 2 for INT4 (INT2).

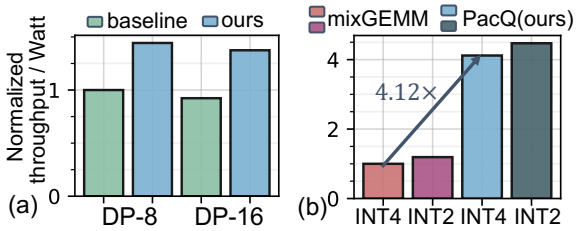

Fig. 12. (a) Comparison of different sizes of the DP unit. (b) Comparison of PacQ with Mix-GEMM [17]. Both (a) and (b) use the metric of throughput per watt. The workloads for both comparisons are *m16n16k16*.

In Figure. 12(a), we further study the effect of the size of the DP unit (DP-8 and DP-16). The results show that PacQ's performance improvements are orthogonal to the size of the DP units. In Figure. 12(b), we compare the performance of PacQ with prior work, Mix-GEMM [17], which also aims to improve the throughput of mixed-precision GEMM workloads through binary segmentation. With FP16 activation, PacQ achieves a 4.12× performance improvement for INT4 weights and a 3.75× improvement for INT2. This performance advantage is attributed to the fact that the binary segmentation technique performs poorly for hyper-asymmetric GEMM.

VI. CONCLUSION

In this work, we present PacQ, a novel SIMT microarchitecture designed to efficiently handle hyper-asymmetric GEMMs with mixed-precision weights (INT4 and INT2) and activations (FP16). By introducing a new packing and dataflow strategy, coupled with our parallel FP-INT multiplier, PacQ achieves significant improvements in hardware efficiency and throughput. Through extensive experimental evaluation, we demonstrate that PacQ delivers up to 54.3% reduction in

register file accesses, a $1.99\times$ speedup in throughput, and up to 81.4% reduction in energy-delay product (EDP) compared to conventional SIMT baselines. We believe our work can provide the community with insights into the design space exploration for hyper-asymmetric GEMM acceleration.

VII. Acknowledgement

This work was supported in part by CoCoSys, a JUMP2.0 center sponsored by DARPA and SRC, the National Science Foundation (CAREER Award, Grant #2312366, Grant #2318152), the DARPA Young Faculty Award and the DoE MMICC center SEA-CROGS (Award #DE-SC0023198).

References

[1] A. contributors, "Autogptq," https://github.com/AutoGPTQ, 2024.

[2] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[3] ——, "Optq: Accurate quantization for generative pre-trained transformers," in *The Eleventh International Conference on Learning Representations*, 2022.

[4] D. Gope, J. Beu, and M. Mattina, "High throughput matrix-matrix multiplication between asymmetric bit-width operands," *arXiv preprint arXiv:2008.00638*, 2020.

[5] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[6] J. Jang *et al.*, "Figna: Integer unit-based accelerator design for fp-int gemm preserving numerical accuracy," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 760–773.

[7] Y. J. Kim, R. Henry, R. Fahim, and H. H. Awadalla, "Who says elephants can't run: Bringing large scale moe models into cloud scale production," *arXiv preprint arXiv:2211.10017*, 2022.

[8] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "Brecq: Pushing the limit of post-training quantization by block reconstruction," *arXiv preprint arXiv:2102.05426*, 2021.

[9] Y. Li and P. Panda, "Tesseraq: Ultra low-bit llm post-training quantization with block reconstruction," *arXiv preprint arXiv:2410.19103*, 2024.

[10] J. Lin *et al.*, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024.

[11] llmc contributors, "llmc: Towards accurate and efficient llm compression," https://github.com/ModelTC/llmc, 2024.

[12] Z. Mo *et al.*, "Lut tensor core: Lookup table enables efficient low-bit llm inference acceleration," *arXiv preprint arXiv:2408.06003*, 2024.

[13] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, 2009.

[14] NVIDIA Corporation, "NVIDIA TESLA V100 GPU ARCHITECTURE," http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, June 2017.

[15] G. Park, B. Park, M. Kim, S. Lee, J. Kim, B. Kwon, S. J. Kwon, B. Kim, Y. Lee, and D. Lee, "Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models," *arXiv preprint arXiv:2206.09557*, 2022.

[16] M. A. Raihan *et al.*, "Modeling deep learning accelerator enabled gpus," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 79–92.

[17] E. Reggiani *et al.*, "Mix-gemm: An efficient hw-sw architecture for mixed-precision quantized deep neural networks inference on edge devices," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 1085–1098.

[18] G. Tan *et al.*, "Fast implementation of dgemm on fermi gpu," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–11.

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

[20] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, "Bitnet: Scaling 1-bit transformers for large language models," *arXiv preprint arXiv:2310.11453*, 2023.

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

[22] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[23] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[24] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate llm serving," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 196–209, 2024.