Received: 13 March 2024 | Revised: 26 July 2024 | Accepted: 16 August 2024
DOI: 10.4218/etrij.2024-0111

SPECIAL ISSUE

ETRI Journal WILEY

PF-GEMV: Utilization maximizing architecture in fast matrix–vector multiplication for GPT-2 inference

Hyeji Kim | **Yeongmin Lee** | **Chun-Gi Lyuh**

Hyperscale AI SoC Research Section, Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea

Correspondence
Hyeji Kim, Hyperscale AI SoC Research Section, Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea.
Email: hyejikim@etri.re.kr

Funding information
Next-generation Intelligence Semiconductor Foundation grant funded by the Korea government (MSIT) (no. 2020-0-01308, Intelligent Mobile Processor based on Deep-Learning Micro Core Array).

Abstract
Owing to the widespread advancement of transformer-based artificial neural networks, artificial intelligence (AI) processors are now required to perform matrix–vector multiplication in addition to the conventional matrix–matrix multiplication. However, current AI processor architectures are optimized for general matrix–matrix multiplications (GEMMs), which causes significant throughput degradation when processing general matrix–vector multiplications (GEMVs). In this study, we proposed a port-folding GEMV (PF-GEMV) scheme employing multiformat and low-precision techniques while reusing an outer product-based processor optimized for conventional GEMM operations. This approach achieves 93.7% utilization in GEMV operations with an 8-bit format on an 8 × 8 processor, thus resulting in a 7.5 × increase in throughput compared with that of the original scheme. Furthermore, when applied to the matrix operation of the GPT-2 large model, an increase in speed by 7 × is achieved in single-batch inferences.

KEYWORDS
low-precision, matrix–vector multiplication, multiformat, utilization

1 | INTRODUCTION

As recent artificial neural networks diversify and transformer-based language models [1] become extensively used, the use of matrix–matrix and matrix–vector multiplication operations increases. [2–5] The existing artificial intelligence (AI) accelerators are optimized for the most computationally complex operations, namely, general matrix–matrix multiplications (GEMMs). [6–10] Utilizing these architectures for general matrix–vector multiplication (GEMV) operations significantly decreases the throughput compared with that of GEMM operations, thus reducing resource utilization.

This performance degradation arises owing to memory bandwidth limitations. In other words, to accelerate GEMV and increase accelerator utilization, increasing the amount of data transferred per second from memory for vector inputs is necessary. However, AI accelerators are fundamentally designed such that the array and buffer sizes are specified to maximize the memory bandwidth supported by the system within an allowed area. Therefore, if only the memory bandwidth is increased to enhance the throughput of GEMV, then unnecessary complexity may arise in the overall system because the memory bandwidth is not fully utilized for GEMM operations.

In this study, we proposed a GEMM–GEMV integrated tensor processing architecture and methodology that minimizes additional hardware usage by leveraging the conventional outer-product-based processor [8] optimized for GEMMs. The proposed approach enhances the operational speed of GEMVs by applying multiformat and low-precision techniques to fully utilize the memory bandwidth and increase the operational utilization of the processor for GEMVs. The main contributions of this study are as follows:

- We propose an architecture and operating methodology for the PF-GEMV scheme combined with the multiformat low-precision technique to enhance the throughput of GEMV operations.
- We quantify the performance of the proposed PF-GEMV scheme in terms of utilization maximization and speedup for various array sizes and data precisions.
- We confirm the improved computational speed by applying the PF-GEMV scheme to GPT-2 inferences.
- We propose a hybrid approach that integrates GEMM and PF-GEMV for multibatch GPT-2 inference to further enhance the computational speed.

The following section explains the operation of the outer product-based processor and discusses the overhead caused by GEMV in GPT-2 inference. Subsequently, Section 3 describes the proposed PF-GEMV architecture and operational methodology, and Section 4 presents the numerical confirmation of speed enhancement when applying the proposed PF-GEMV to GPT-2 inference operations.

2 | PRELIMINARY

In this section, we describe the basic structure of the outer product-based processor [8] and its utilization in the GEMM and the GEMV operations. Additionally, we briefly discussed the types of GEMV operations occurring in the inference computations of GPT-2 [4] and the performance degradation arising from processing using conventional outer-product-based processors.

2.1 | Outer product-based tensor processor

The outer product-based tensor processor [8] is an architecture characterized by a property based on which the input data bandwidth increases by a factor of N, which cause the number of processing elements (PEs) within the internal processor to increase by N^2, thus increasing the throughput for GEMM operations. This architecture typically adheres to the structure illustrated in Figure 1. The tensor processor receives instructions, control

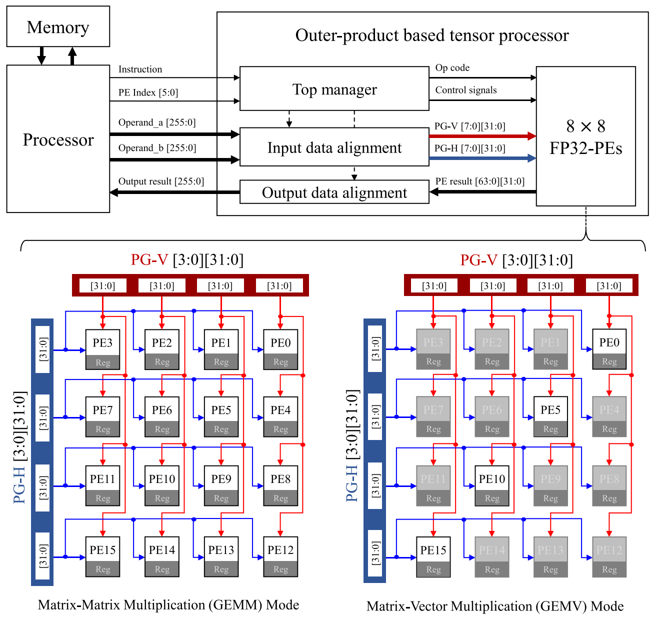

FIGURE 1 Conventional outer-product based tensor processor [8] specified with deterministic bitwidth using 8 × 8 FP32-PEs. Matrix operations show an example using half of the input operands, with connections based on relationships illustrated. White PEs are activated for computation and gray PEs are deactivated. PE, processing element.

signals, and two operands from an external main processor and outputs the PE results with the same bitwidth as the operand. Because the output port cannot accommodate the results of all PEs at once, it outputs the permitted results from the specified PE index in a multicycle manner. The operands are internally segregated into two port groups, PG-H and PG-V, which are directly input to the PEs. Each data port broadcasts signals to all PEs aligned horizontally or vertically. The accumulated information for the multiply and accumulate (MAC) operation is managed using a register within the PE.

GEMM operation

When performing GEMM operations with matrix A of size $[M_A \times K]$ and matrix B of size $[K \times M_B]$, matrix A is first transposed into matrix A^T with a size of $[K \times M_A]$. To adapt to processing within an $N \times N$ processor, segments of A_p^T and B_p with sizes of $[K \times N]$ are prepared based on the M_A and M_B dimensions positioned horizontally, to match the N in terms of array size. In every cycle, N data from each segmented matrix are input into the processor, thus resulting in N^2 operations being performed per cycle. The MAC operation in the PE repeats for K cycles and ultimately outputs a result of size $[N \times N]$.

GEMV operation

In cases where the input A or B are vectors instead of matrices, the port group for the vectors receives only one valid data point per cycle instead of N data points, and the processor can perform N operations per cycle. This architecture results in significantly reduced memory bandwidth occupancy and decreased operational utilization owing to element-wise inputs. Consequently, the throughput of GEMV operations decreases to $1/N$ compared with that of GEMM operations. In this study, we proposed an extended architecture for enhancing PE utilization in GEMV operations, thus ultimately alleviating throughput degradation while maintaining the fundamental outer product structure to sustain the throughput of GEMM operations.

2.2 | GEMV overhead in GPT-2 inferences

GPT-2 [4] is one of the prominent models extensively used for language modeling, structured in a stacked form of multiple block layers based on the scale of the GPT-2 model. Each block layer consists of a sequence of self-attention modules and feed-forward networks. Internally, the self-attention module comprises matrix operations, such as the generation of query, key, and value matrices, multihead attention (MHA) operations, and the projection operations are composed of fully connected layers, which are sequentially connected.

Depending on whether GPT-2 is performing inference or training operations, the behavior of internal matrix operations varies. During training, all matrix operations can be implemented using GEMM. However, during inference, as shown in Table 1, when operating in single-batch scenario cases, all matrix operations operate as GEMVs, whereas in multibatch scenario cases, only the MHA operations operate as GEMVs, while the rest function as GEMMs.

TABLE 1 GEMV overhead for block layer of GPT-S [11] inference model.

	Batch = 1	**Batch = 8**	
Complexity	**MHA + others**	**MHA**	**Others**
Operations	100	18%	82%
Cycles	100%	**64%**	**36%**
Function type	GEMV	GEMV	GEMM

Note: The word iteration was set to the maximum value of 1024. The computation cycle was calculated for an 8×8 PE.

Abbreviations: GEMV, general matrix–vector multiplication; MHA, multihead attention.

Table 1 presents the computational percentages and operation cycles for inference-specific matrix operations in a single block layer of the GPT-2 small (GPT-S) [11], divided into the MHA and the remaining part. Herein, the word iteration is set to the maximum value of 1024 and represents the case in which the last word of most sentences is generated. The operation cycle is calculated based on the throughput of GEMV and GEMM, set at 8 and 64, respectively, using an outer product-based 8×8 PEs.

In a single-batch scenario, all matrix operations are implemented using the GEMV function, which may lead to inefficient operation in the conventional neural processing architectures optimized for GEMM operations. Furthermore, in a multibatch scenario, although the number of operations performed by the GEMV function accounts for only 18% of the total, owing to the throughput degradation of GEMV operations in actual hardware operation, it occupies 64% of the operation cycles. Ultimately, the degradation in operation speed is primarily caused by GEMV operations.

3 | PF-GEMV: PE UTILIZATION MAXIMIZING ARCHITECTURE

This section introduces a method for improving the operational efficiency of GEMV without altering the conventional outer-product-based architecture or compromising the overall computational speed of GEMM. For the conventional GEMV shown in Figure 1, matrix data are assigned to PG-H and a scalar datum of the vector is utilized among the data transmitted to PG-V per cycle. Consequently, whereas N^2 PEs are fully utilized in the $N \times N$ array architecture for GEMM, the array utilization rate for GEMV decreases to $1/N$.

In this study, we proposed a GEMV scheme, namely, port-folding (PF), to enhance operational efficiency by extending the matrix data to the port for vector data while integrating low-precision formats. The interface and basic operational structure between the outer-product-based tensor processor and main processor adhere to the conventional architecture shown in Figure 1. This paper describes a method to improve the existing PE structure to enhance the GEMV throughput.

3.1 | Multilevel low-precision transmission

The proposed PF-GEMV enhances PE utilization by allocating matrix data to both input operand ports. As illustrated in the conventional structure in Figure 1, the existing GEMV has low PE utilization. The PF-GEMV technique improves throughput by utilizing the otherwise unused PEs without requiring additional PE implementations. Additionally, by reducing the precision of only the matrix data by the folding level L, the PF scheme enables the packing and transmission of more data to the PEs within a single cycle. This approach maximizes the utilization of an $N \times N$ PE structure by a factor of L $(2 - 1/N)$. The maximum value of folding level L is limited to $N/2$. For the vector input, a scalar datum is input at every cycle and broadcast to all valid PEs to be allocated as a common input along with the extended matrix data.

For instance, when using a 32-bit PE arithmetic unit with a 16-bit matrix format, L is 2 and two 16-bit elements are packed into a single 32-bit port, such as H0[1:0][15:0], and then internally distributed to two separate 32-bit PEs (see Figure 2B). The vector element is transmitted, while its original 32-bit precision is maintained. When an 8-bit input format is applied, that is, $L = 4$, four 8-bit matrix elements are packed into a single 32-bit port, such as H0 [3:0] [7:0], and then internally distributed to four separate 32-bit PEs (see Figure 2C). As L increases to 2 and 4, the PE utilization improves by factors of $(4 - 2/N)$ and $(8 - 4/N)$, respectively. When L is 1, no reduction in precision is observed; therefore, the utilization is improved by a factor of $(2 - 1/N)$ (see Figure 2A).

The proposed PF-GEMV scheme is applicable to both integer and floating-point (FP) PE arithmetic units, with the low-precision format determined by the data type supported by the PE. Before initiating the GEMV operation, the matrix data are converted into a low-precision format (i.e., quantization). Although reducing the precision of the matrix data can affect the accuracy of neural networks, performance degradation can be prevented by utilizing existing data-calibration techniques for integers [12, 13] and exponent scaling for FPs [14, 15] during data quantization phase. Recent studies pertaining to quantization indicate that for transformer-based language models such as Llama [16], which perform better than GPT-2, the weight matrices can be quantized to 4-bit integer precision [13] and 6-bit FP precision [14] while maintaining an inference performance comparable to that of the baseline model.

3.2 | Allocation of PEs

The outer product-based array structure has a directional flow, in which one input data port is broadcast to the internal PEs, either horizontally or vertically, as illustrated in Figure 1. To select the activated PEs for PF-GEMV, the operational conditions are set as follows.

- Each internal PE only accepts two external inputs, which are limited to the vector and matrix data.
- The data ports constituting the horizontal and vertical operands are uniquely allocated to the PEs forming the array structure and only the allocated PEs are activated to perform GEMV operations.
- When L data elements are integrated into a single port, where each of the L elements is distributed to the corresponding L internal PEs in the respective direction for computation.

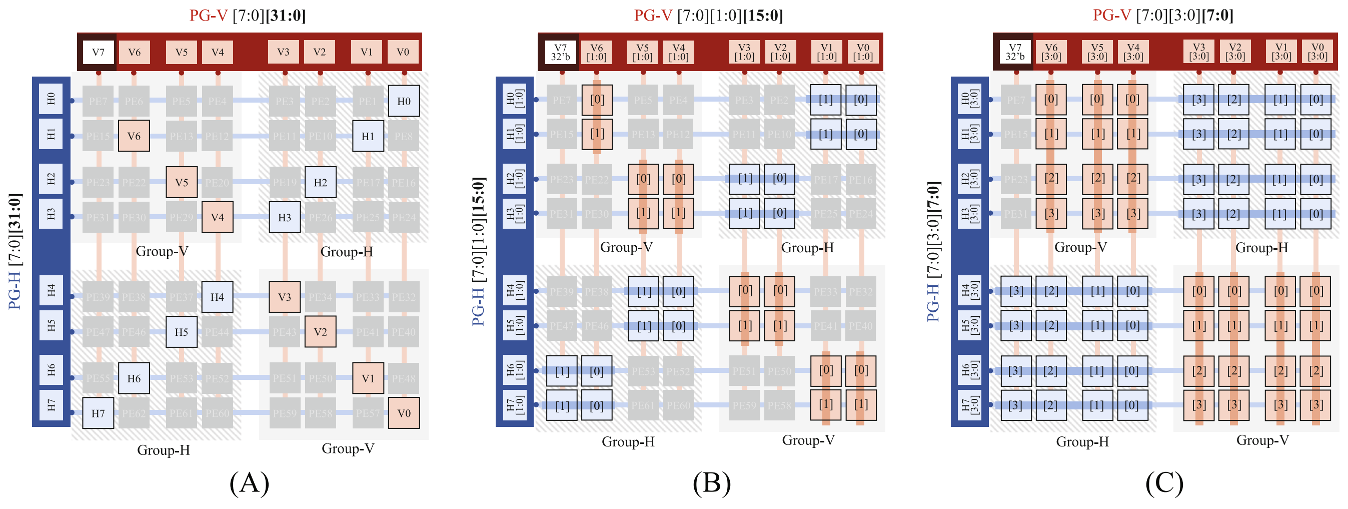

FIGURE 2 Examples of PE allocation and data distribution for multilevel port-folding schemes using 8 × 8 32-bit PEs. Bit-configuration of PG-V and PG-H is defined with [dimension of PE structure, N][folding-level, L][reduced bitwidth of matrix]. Overall structure and connections are briefly illustrated. Detailed connections between the input port and PEs adhere to the PE structure shown in Figure 1. (A) Level 1 PF-GEMV ($L = 1$), (B) Level 2 PF-GEMV ($L = 2$), and (C) Level 4 PF-GEMV ($L = 4$). PE, processing element; PF-GEMV, port-folding GEMV.

- For the N PEs connected to a single port in either the horizontal or vertical direction, L PEs are assigned from that port, whereas the remaining PEs can be assigned individually from the ports in the orthogonal direction.

For instance, in the case in which a 16-bit data format is used for the matrices (see Figure 2B), the H0 input port is uniquely assigned to two of the eight internal PEs in the horizontal direction. Similarly, the V0 input port is uniquely assigned to two (among eight) internal PEs in the vertical direction. From the third row of PEs (PE16 to PE23), four PEs are activated, two of which are affected by the H2 input, one by the V4 input, and one by the V5 input.

3.3 | Workflow

This section describes the utilization of the PF scheme to perform the multiplication operation of the $[M \times K]$ matrix and $[K \times 1]$ vector, as well as the method of outputting the computed results stored in the PE registers.

Input segmentation

The $[M \times K]$ matrix is segmented into partial matrices of size $[L(2N-1) \times K]$, where L and N represent the folding level and the size of the array structure, respectively. Each partial matrix operation is multiplied by the $[K \times 1]$ vector using the PEs. To ensure compatibility, the matrix size M is adjusted to be a multiple of $L(2N-1)$, with any remaining area padded with zeros. Additionally, in neural network inference, the matrix for GEMV operation typically corresponds to the weight parameters of a pretrained network. Therefore, the pretrained weights can be prequantized to a low-precision format that matches the desired folding level before the operation commences.

Data distribution

As illustrated in Figure 2, the PE structure is classified into four regions comprising two Group-H and two Group-V, which manage the horizontal and vertical inputs, respectively. In the conventional multiplication of matrices A and B (GEMM), matrix A is assigned to PG-H, which connects to the Op-A of PEs within Group-H and Group-V, whereas matrix B is assigned to PG-V and connects to the Op-B of the PEs (see Figure 3A). Therefore, in conventional GEMV operations, to reuse the existing

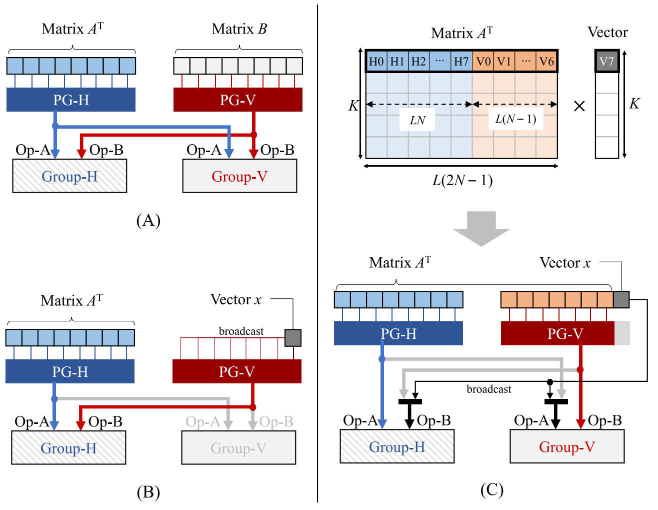

FIGURE 3 Input-data distribution for conventional and proposed operations. Matrix illustration for PF-GEMV. (C) Example of 8×8 PE structure ($N = 8$) at folding level L. Each input port, such as H0, contains L matrix elements: (A) conventional distribution for GEMM, (B) conventional distribution for GEMV, and (C) distribution for PF-GEMV. PF-GEMV, port-folding GEMV.

port connections, the scalar of the vector connected to PG-V is broadcast using only some of the PEs in Group-H (see Figure 3B).

The proposed PF-GEMV maintains the existing port connections for GEMM operations while utilizing PG-V for the matrix input and adding a multiplexer to distribute the scalar of the vector to both Group-H and Group-V (see Figure 3C). The matrix input is connected to Group-H and Group-V in only one direction, whereas the vector, that is, the *V7* port in PG-V, is directed toward Group-H and Group-V through separate ports and broadcasts internally. Compared with the conventional data distribution, the data distribution for the PF-GEMV scheme includes a multiplexer for vector- and data-masking logic directed toward specific PEs.

Processing

As shown in Figure 4A, in the case of level 4 folding, each port contains four consecutive data elements, with each data element being distributed to different PEs. The MAC operations inside the PE maintain the original input, accumulation, and output precision of 32 bits, regardless of the input-data precision. Therefore, when level 4 folding is applied to a 32-bit PE, as illustrated in Figure 4B, the input 8-bit matrix data are reconverted to 32-bit data within the PE using a format converter (CVT). Similarly, if the 16-bit input matrix data (in the case of level 2 folding) are used with a 32-bit PE, then they are reconverted to their original precision by the CVT. The vector data are consistently input with the same precision as the PE, regardless of the folding level.

The internal PE configuration determines the input ports to be assigned to the matrix and vector based on the group of the current PE. The PE in the Group-H region assigns 8-bit matrix data from PG-H to the Op-A input and 32-bit vector data to the Op-B input. Conversely, the PE in the Group-V region assigns 16-bit matrix data from PG-V to the Op-B input and 32-bit vector data to the Op-A input.

The total number of operation cycles for the segmented partial GEMV is determined by the intermediate dimension K of the original matrix and vector. Based on the outer product-based PE structure, the results of the operations are individually stored in the accumulation registers (which depend on the internal PEs). During each cycle, the MAC operation receives the stored results from the previous cycle as inputs for addition. Therefore, once K cycles of operations are completed, the final result can be obtained by outputting the PE registers.

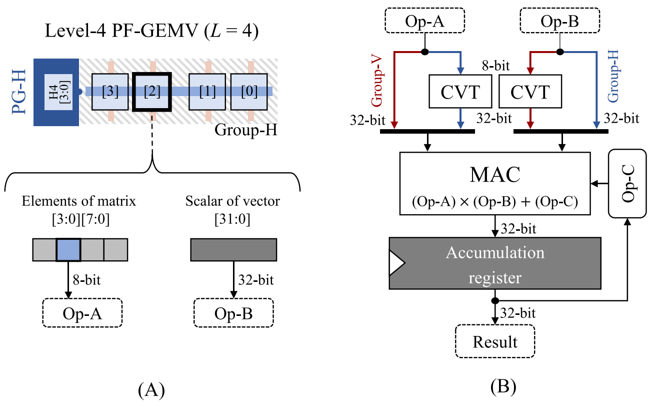

FIGURE 4 Example of data masking and assignment with 8 × 8 32-bit PEs in level 4 PF-GEMV: (A) masking for multilevel PF-GEMV and (B) MAC operation in 32bit-PE. PE, processing element; PF-GEMV, port-folding GEMV.

Output results

The overall structure of the tensor processor is shown in Figure 1; the bitwidth of the output data port is the same as that of a single input operand port. Therefore, the register values of all the PE structures are not output simultaneously; instead, only the amount of data permitted by the output bitwidth is partially output over multiple cycles. Figure 5 illustrates the amount and sequence of output data over multiple cycles for both the conventional and proposed GEMV.

In the conventional GEMV, the output values are stored in PEs positioned along a unidirectional diagonal, thus allowing data to be output in a single cycle (see Figure 5A). By contrast, the proposed PF-GEMV generates $L(2 - 1/N)$ times more values within the same number of operation cycles as the conventional GEMV. Consequently, depending on the folding level, data blocks of up to eight packed elements are output over two, four, and eight cycles (see Figure 5B).

To accommodate this, the ST-D (store for diagonal PEs) instruction used for data output in the conventional architecture must be extended to support bidirectional diagonals. Additionally, new instructions need to be implemented: ST-DB (store for diagonal block PEs) to support blocked diagonal store for level 2 folding and ST-LB (store for linear block PEs) to support blocked linear store for level 4 folding.

Figure 6 illustrates the total effective cycles for the GEMV operation of multiplying an $[M \times K]$ matrix by a $[K \times 1]$ vector, specifically when M is $8N$. In the

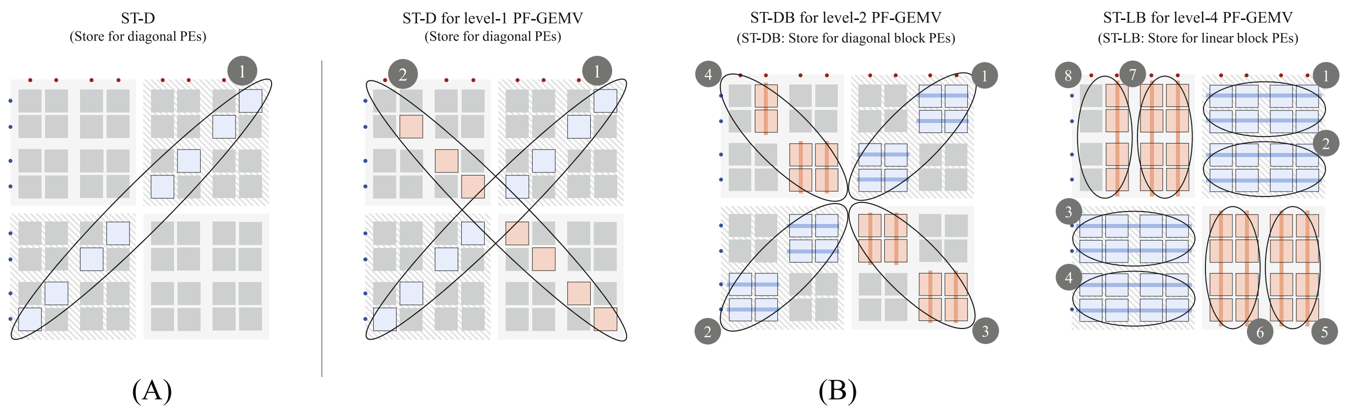
ST-D
(Store for diagonal PEs)
(A)
ST-D for level-1 PF-GEMV
(Store for diagonal PEs)
ST-DB for level-2 PF-GEMV
(ST-DB: Store for diagonal block PEs)
ST-LB for level-4 PF-GEMV
(ST-LB: Store for linear block PEs)
(B)

FIGURE 5 Amount and sequence of output data for both conventional and proposed PF-GEMV results with 8 × 8 PE structure. Based on the overall structure shown in Figure 1, the amount of data that can be output per cycle is equal to the PE structure dimension N: (A) conventional GEMV and (B) proposed PF-GEMV for levels 1, 2, and 4. PF-GEMV, port-folding GEMV.

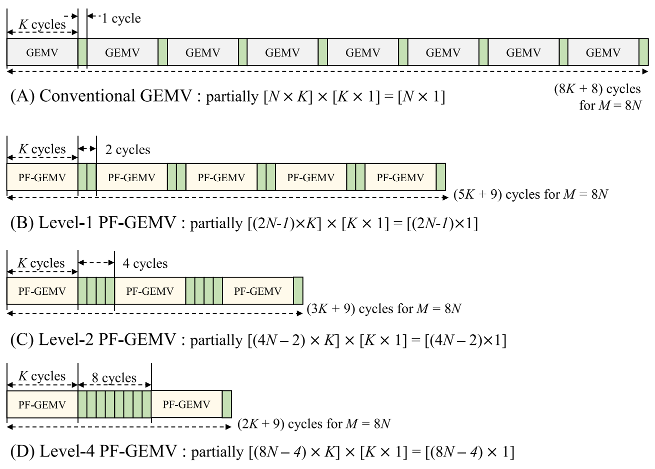
K cycles
1 cycle
GEMV
(A) Conventional GEMV : partially [N × K] × [K × 1] = [N × 1]
(8K + 8) cycles for M = 8N
K cycles
2 cycles
PF-GEMV
(5K + 9) cycles for M = 8N
(B) Level-1 PF-GEMV : partially [(2N-1)×K] × [K × 1] = [(2N-1)×1]
K cycles
4 cycles
PF-GEMV
(3K + 9) cycles for M = 8N
(C) Level-2 PF-GEMV : partially [(4N – 2) × K] × [K × 1] = [(4N – 2)×1]
K cycles
8 cycles
PF-GEMV
(2K + 9) cycles for M = 8N
(D) Level-4 PF-GEMV : partially [(8N – 4) × K] × [K × 1] = [(8N – 4) × 1]

FIGURE 6 Total effective cycles including GEMV and data output operations in the multiplication of $(M \times K)$ matrix and $(K \times 1)$ vector. Indicated cycles are calculated for $M = 8N$. GEMV, general matrix–vector multiplication.

conventional case, partial GEMV operations are repeated eight times, with each requiring K cycles and each repetition followed by a one-cycle output command, thus resulting in a total of $(8K + 8)$ cycles. The proposed level 1 PF-GEMV performs four repetitions of the partial PF-GEMV operation over K cycles each, followed by additional operations on the residual matrix $[4 \times K]$. This results in five PF-GEMV operations and nine data outputs, which require $(5K + 9)$ cycles. Similarly, for level 2 and level 4 PF-GEMV, including operations on the residual matrix $[4 \times K]$, the total numbers of cycles required are $(3K + 9)$ and $(2K + 9)$, respectively. Thus, as the folding level increases, the total GEMV operation time decreases.

4 | EXPERIMENTAL RESULTS

In this section, we verify the improvement in computational speed for GPT-2 [4] inference using the proposed PF-GEMV schemes. Experiments were conducted based on three types of GPT-2 models: small (S) [11], medium (M) [17], and large (L) [18], as shown in Table 2. The batch size was varied while considering the word iteration at a maximum value of 1024 to represent the highest computational complexity.

4.1 | PE utilization

The baseline architecture for calculating the computation cycles is an outer product-based 8×8 processor, where each PE supports 32-bit format MAC operations and accepts data in 32-, 16-, and 8-bit formats. The 32-bit format can be considered the FP 32-bit format, 16-bit as the BF16 format [19], and 8-bit as the hybrid-FP8 [15] format. The operation time is represented in terms of the calculated operation cycles, where the cycles denote the total number of operations for GEMM or GEMV divided by the throughput of the processor, as shown in Table 3.

TABLE 2 Dimensions for three types of GPT-2 models.

Configurations	GPT2-S [11]	GPT2-M [17]	GPT2-L [18]
Embedding size	768	1024	1280
Heads	12	16	20
Layers	12	24	36
Maximum words	1024	1024	1024
Depth	64	64	64

TABLE 3 Architectural features of conventional 8×8 array and proposed GEMV schemes.

		Proposed PF-GEMV		
8 × 8 array architecture	**Original GEMV**	**Level 1**	**Level 2**	**Level 4**
Matrix precision	32-bit	32-bit	16-bit	8-bit
GEMV throughput (operations/cycle)	8	15	30	60
PE utilization	12.5%	23.4%	46.9%	93.7%

Note: The GEMM throughput is equal to 64 in all approaches.
Abbreviations: GEMV, general matrix–vector multiplication; PE, processing element.

The proposed level 1 PF scheme extends the ports based on a 32-bit format, whereas level 2 folding involves packing two 16-bit data points per port, and level 4 entails four 8-bit data points per port. The throughput of the proposed levels 1, 2, and 4 PF-GEMV schemes increased by 1.88, 3.75, and 7.5 times, respectively, compared with the conventional GEMV method. The throughput of the GEMM operations remained consistent at 64 across all architectures without performance degradation.

The GEMV utilization rate for the 8×8 array architecture shown in Figure 2 increased up to 93.7% by employing a level 4 folding scheme with minimal 8-bit precision. As shown in Table 4, in a 4×4 array configuration based on a 32-bit PE and 32-bit data port, the utilization of a 16-bit data format with a level 2 folding scheme enables the availability of 14 among 16 PEs, thus resulting in a maximum utilization rate of 87.5%. Similarly, in a 16×16 array configuration based on a 16-bit

TABLE 4 Throughput and utilization performance of original and proposed PF-GEMV schemes by array size.

	4 × 4 PE	8 × 8 PE	16 × 16 PE
GEMV scheme	Throughput / Utilization		
Original	4 / 25.0%	8 / 12.5%	16 / 6.2%
PF-GEMV level 1	7 / 43.7%	15 / 23.4%	31 / 12.1%
PF-GEMV level 2	14 / 87.5%	30 / 46.9%	62 / 24.2%
PF-GEMV level 4	- / -	60 / 93.7%	124 / 48.4%
PE precision	32-bit	32-bit	16-bit
GEMM throughput	16	64	256

Abbreviations: GEMV, general matrix–vector multiplication; PE, processing element.

PE and 16-bit data port, adopting a 4-bit data format with a level 4 folding scheme allows for the availability of 124 among 256 PEs, thus resulting in a maximum utilization rate of 48.4%.

Maximizing utilization

Based on recent studies, when the capability of conducting GPT-2 assessments and training under the hybrid-FP8 format [20] is considered, employing an 8 × 8 array structure (where the 32-bit data format is used for GEMM and 8-bit data format is utilized for GEMV) achieves the highest computational efficiency and operational utilization in GPT-2 operations.

Maximizing speed increases

To enhance the processing speed of GPT-2 computations, using the 4-bit data format [21] can realize high-speed processing in a 16 × 16 array structure. Under 16-bit GEMM operations, 256 data points can be processed per cycle, whereas 4-bit matrix data are employed in GEMV operations, thus resulting in a utilization rate of 48.4%, which significantly enhances the processing speed.

4.2 | GEMV Acceleration in GPT-2

As described in Section 2.2, GPT-2 inference operations behave differently depending on the batch size. When the batch size is 1, the matrix operations in the self-attention and feedforward networks are executed using GEMV. For batch sizes larger than 1, only the MHA operations in the self-attention module are composed of GEMV, whereas the remaining matrix operations use GEMM.

In this section, we demonstrate that the proposed PF scheme with an 8 × 8 processor can effectively accelerate GEMV operations in GPT-2 inferences. Furthermore, we show that, depending on the batch size, replacing or mixing GEMM with the proposed GEMV scheme is advantageous in terms of speed.

4.2.1 | Single-batch inferences

We analyzed the reduction in GEMV operation time for three types of GPT-2 models under a batch size of 1. Figure 7 illustrates the reduction in the number of operation cycles when the proposed PF schemes are applied to three types of GPT-2 models: GPT-S [11], GPT-M [17], and GPT-L [18]. Compared with the original 32-bit scheme, in the GPT-S model case, the levels 1, 2, and 4 schemes reduced the number of operation cycles by 45%, 72%, and 85%, respectively. This translates to speed improvements of 1.83, 3.58, and 6.83 ×, respectively. When the 8-bit, level 4 scheme was applied to the three GPT-2 models above, the performance improvements increased with the model scale, and the speed increased by 6.83, 6.87, and 7.02 ×, respectively.

4.2.2 | Multibatch inferences

We calculated the number of operation cycles for the GPT-L model [18] as a function of the batch size. We aimed to verify the decrease in the number of GEMV operations for a single layer of GPT-2 block operation. Additionally, we introduced a hybrid approach that separates operations to enable the utilization of both GEMM and GEMV for a specific batch size. Ultimately, we confirmed that the level 4 folding technique with an 8-bit format exhibited a linear performance improvement as the batch size increased.

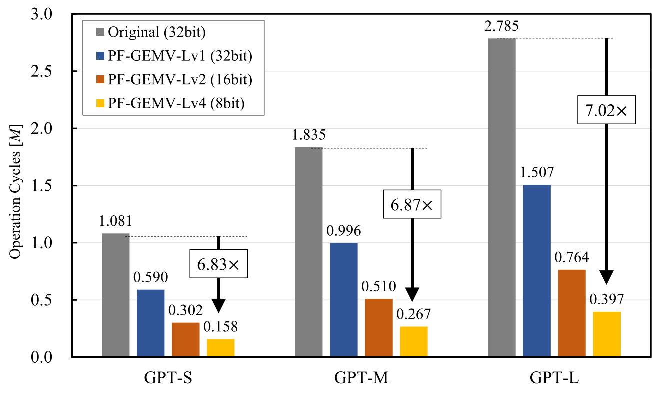

FIGURE 7 Acceleration of GPT-2 single-batch inference.

Decreasing GEMV cycles for GPT-2 inferences

The operations within the MHA module are processed as GEMV operations regardless of the batch size, while the remaining operations can be handled as GEMM, when the batch size is greater than one. Figure 8 depicts a graph illustrating the cycle counts for GEMM and GEMV operations as batch size increases. When the batch size was equal to the array size (e.g., eight), using the original GEMV scheme resulted in GEMV operations that constituted 51.6% of the total operation (including GEMM). By applying the proposed levels 1, 2, and 4 PF-GEMV schemes, the GEMV percentages decreased to 38.3%, 25.7%, and 17.2%, respectively.

Conditional hybrid computing

For a specific batch size, the hybrid approach combining GEMM and PF-GEMV operations resulted in shorter computation times compared with individual approaches. The hybrid approach is an operational method aimed at reducing unnecessary computations caused by zero padding during GEMM. Zero padding occurs when the matrix size is not a multiple of the array size *N*. To prevent unnecessary computations resulting from zero padding, we combined GEMM and PF-GEMV. This hybrid approach, as illustrated in Figure 9, considers the remainder area of the input matrix *B* divided by the array size

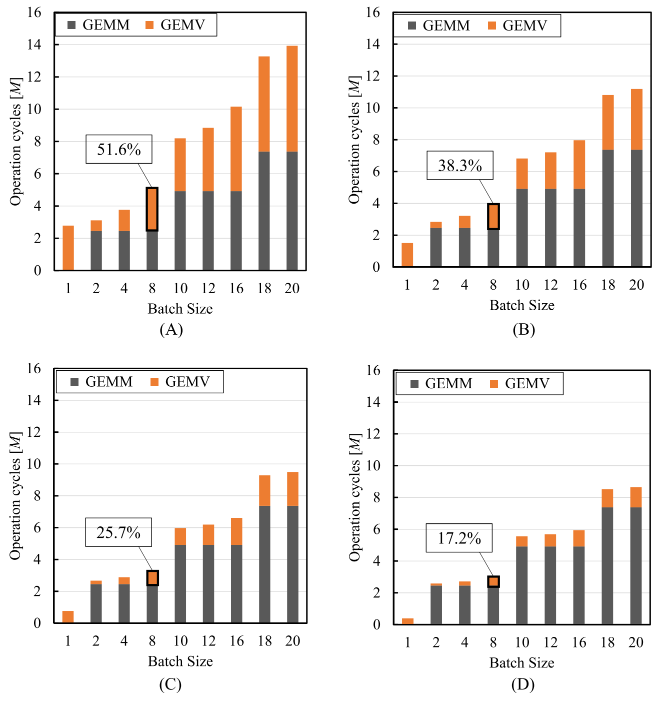

FIGURE 8 Decreasing GEMV operation cycles of GPT2-L inference for multibatch embedding without using a hybrid approach: (A) original (32 bit), (B) PF-GEMV-Lv1 (32 bit), (C) PF-GEMV-Lv2 (16 bit), and (D) PF-GEMV-Lv4 (8 bit). PF, port-folding GEMV.

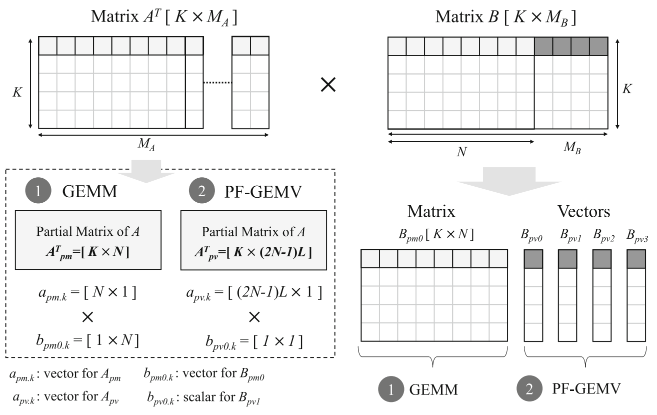
Matrix A^T [$K \times M_A$]
Matrix B [$K \times M_B$]
K
M_A
N
M_B
1 GEMM
2 PF-GEMV
Partial Matrix of A A^T_{pm}=[$K \times N$]
Partial Matrix of A A^T_{pv}=[$K \times (2N\text{-}1)L$]
$a_{pm.k}$ = [$N \times 1$] × $b_{pm0.k}$ = [$1 \times N$]
$a_{pv.k}$ = [$(2N\text{-}1)L \times 1$] × $b_{pv0.k}$ = [1×1]
$a_{pm.k}$: vector for A_{pm}
$a_{pv.k}$: vector for A_{pv}
$b_{pm0.k}$: vector for B_{pm0}
$b_{pv0.k}$: scalar for B_{pv1}
Matrix B_{pm0} [$K \times N$]
Vectors B_{pv0} B_{pv1} B_{pv2} B_{pv3}
1 GEMM
2 PF-GEMV

FIGURE 9 Hybrid computation involving GEMM and PF-GEMV. GEMM, general matrix–matrix multiplication; PF-GEMV, port-folding GEMV.

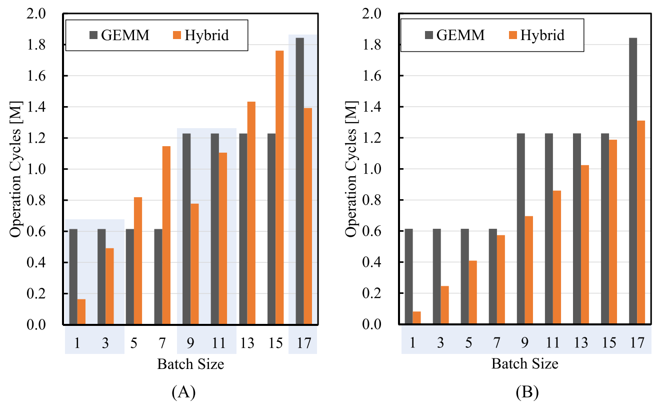

FIGURE 10 Comparison of computing cycles for cases involving GEMM, GEMV, and hybrid approach. The target function is QKV generation in GPT2-L: (A) QKV generation with PF-GEMV-Lv2 (16 bit) and (B) QKV gene ration with PF-GEMV-Lv4 (8 bit).

N as the input vector for PF-GEMV and considers another matrix *A* as the input matrix for PF-GEMV. In this case, zero padding is only applied to the areas corresponding to the input matrix of PF-GEMV, whereas the areas regarded as vectors are processed immediately without padding.

In the context of GPT-2 block operations, we investigated the performance differences when GEMM and the hybrid approach were used for a selected operation that utilized GEMM in multibatch scenarios. Figure 10 presents a performance comparison of the operations associated with generating Q, K, and V matrices in the self-attention module as a function of batch size.

For the 16-bit format, level 2 PF-GEMV demonstrated throughput of 46.9%, as shown in Table 4. This implies that level 2 PF-GEMV can achieve higher speeds than GEMM when the partial matrix size is less than half the array size. Therefore, for an 8 × 8 array, when the batch size has a remainder ranging from 1 to 3 after its divisions by 8, the hybrid approach performs faster processing than GEMM. Meanwhile, in the remaining cases, GEMM computations are faster.

For the 8-bit format, as shown in Table 4, level 4 PF-GEMV demonstrates a throughput of 93.7%. This indicates that level 4 PF-GEMV can perform faster processing than GEMM when the partial matrix size is smaller than the array size. Consequently, when the batch size divided by 8 has a remainder (ranging from 1 to 7), the hybrid approach performs faster processing than GEMM.

In levels 2 and 4 folding cases, when the batch size was less than 8, GEMM operations vanished in the hybrid computation, thus indicating that the hybrid approach operated exclusively under PF-GEMV. Additionally, when the batch size was a multiple of 8, the hybrid approach operated exclusively under GEMM. Therefore, in the 8 × 8 array structure, applying 8-bit, level 4 PF-GEMV affords a higher computational speed for all batch sizes than the original GEMM-only approach.

Alleviating performance flatten

Figure 11 shows a graph comparing the computation cycles of the original and three proposed PF-GEMV schemes for a single layer of the GPT-L model [18]. The performance of the proposed levels 2 and 4 PF-GEMV was achieved when the hybrid approach that separates GEMM and PF-GEMV operations based on the three types of matrix operations for a block layer was applied, namely, the QKV generation of the self-attention module, projection operations, and feedforward network operations. The level 1 PF-GEMV graph represents the outcomes obtained when only PF-GEMV was applied to MHA operations, whereas the other operations were computed using GEMM without using the hybrid approach.

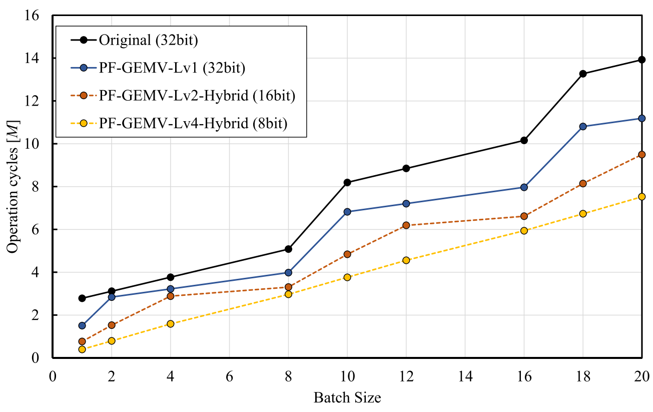

FIGURE 11 Overall performance for block layer of GPT2-L.

Overall, the proposed PF-GEMV scheme effectively reduced the number of computation cycles. In particular, employing the level 4 PF-GEMV scheme with an 8-bit format for an 8 × 8 processor increased the GEMV throughput to match the GEMM throughput, as shown in Table 4. This enhanced the utilization for GEMV operations and structurally alleviated the memory bound, thus addressing the performance-flattening issue associated with batch-size variations.

Performance flattening occurred during GEMM operations when the input matrix size was not a multiple of the array size, thus resulting in zero padding for input matrices *A* and *B*. This resulted in inefficient operations within the padded regions. However, when combined with level 4 PF-GEMV, which exhibited a throughput similar to GEMM on an 8 × 8 array, the computation cycles decreased linearly as the matrix size changed.

5 | CONCLUSION

This paper discussed the architecture and methodology required to improve the utilization and throughput of GEMV operations without degrading GEMM performance in an outer product-based processor. Applying the proposed level 4 PF-GEMV to an 8 × 8 processor enhanced array utilization by up to 93.7%. By utilizing the proposed structure, GEMV operations achieved a similar computational performance to GEMM operations. Consequently, the performance flattening caused by matrix zero padding in GEMM functions was addressed using PF-GEMV functions in parallel. In the case of inference operations using the GPT-2 large model, the proposed level 4 PF-GEMV achieved a 7 × higher computational speed in single-batch operations, which was approximately 3.9 × and 2.4 × higher under batch sizes of 2 and 4, respectively.

In future studies, we shall focus on implementing the proposed PF-GEMV structure in a neural processor with a memory system. This change will allow the measurement of the comprehensive computation time consumed during the execution of GEMM and GEMV kernels, as well as the verification of area increase owing to PF-GEMV.

ORCID

Hyeji Kim https://orcid.org/0000-0002-8318-1868
Yeongmin Lee https://orcid.org/0000-0002-8165-4502

REFERENCES

1. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, Adv. Neural Inf. Process. Syst. **30** (2017), 261–272.

2. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, and S. Gelly, *An image is worth 16 × 16 words: transformers for image recognition at scale*, arXiv preprint, 2020. https://doi.org/10.48550/arXiv.2010.11929
3. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, *Swin transformer: hierarchical vision transformer using shifted windows*, (Proceedings of the IEEE/CVF Int. Conf. on Computer Vision, Montreal, Canada), 2021, pp. pp. 10012–10022.
4. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, *Language models are unsupervised multitask learners*, OpenAI blog **1** (2019), no. 8, 9.
5. A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, *Zero-shot text-to-image generation*, (Int. Conf. Mach. Learn., Virtual Only), 2021, pp. 8821–8831.
6. Y. C. P. Cho, J. Chung, J. Yang, C.-G. Lyuh, H. Kim, C. Kim, J. Ham, M. Choi, K. Shin, J. Han, and Y. Kwon, *AB9: A neural processor for inference acceleration*, ETRI J. **42** (2020), no. 4, 491–504.
7. J. Chung, H. Kim, K. Shin, C.-G. Lyuh, Y. C. P. Cho, J. Han, Y. Kwon, Y.-H. Gong, and S. W. Chung, *A layer-wise frequency scaling for a neural processing unit*, ETRI J. **44** (2022), no. 5, 849–858.
8. W. Jeon, Y. C. P. Cho, H. M. Kim, H. Kim, J. Chung, J. Kim, M. Lee, C.-G. Lyuh, J. Han, and Y. Kwon, *M3FPU: multiformat matrix multiplication FPU architectures for neural network computations*, (IEEE 4th Int. Conf. Artif. Intell. Circuits Syst., Incheon, Rep. of Korea), 2022, pp. 150–153.
9. N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, and C. Young, *TPU v4: an optically reconfigurable supercomputer for machine learning with hardware support for embeddings*, (Proc. 50th Annual Int. Symp. Comput Architecture, Orlando, FL, USA), 2023, pp. 1–14.
10. H. Kim, C.-G. Lyuh, and Y. Kwon, *Automated optimization for memory-efficient high-performance deep neural network accelerators*, ETRI J. **42** (2020), no. 4, 505–517.
11. H. Face, GPT-2, 2024. Available from: https://huggingface.co/openai-community/gpt2 [last accessed March 2024].
12. E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, *GPTQ: accurate post-training quantization for generative pre-trained transformers*, arXiv preprint, 2022. https://doi.org/10.48550/arXiv.2210.17323
13. J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, *AWQ: activation-aware weight quantization for on-device LLM compression and acceleration*, Proc. Mach. Learn. Syst. **6** (2024), 87–100.
14. B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, and S. Dusan, *Microscaling data formats for deep learning*, arXiv preprint, 2023. https://doi.org/10.48550/arXiv.2310.10537
15. X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan, *Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks*, Adv. Neural Inf. Process. Syst. **32** (2019), 4900–4909.
16. H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, and A. Rodriguez, *LLAMA: open and efficient foundation language models*, arXiv preprint, 2023. https://doi.org/10.48550/arXiv.2302.13971
17. H. Face, GPT-2 medium, 2024. Available from: https://huggingface.co/openai-community/gpt2-medium [last accessed March 2024].
18. H. Face, GPT-2 large, 2024. Available from: https://huggingface.co/openai-community/gpt2-large [last accessed March 2024].
19. D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, and J. Yang, *A study of BFLOAT16 for deep learning training*, arXiv preprint, 2019. https://doi.org/10.48550/arXiv.1905.12322
20. P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, and N. Mellempudi, FP8 formats for deep learning, arXiv preprint, 2022. https://doi.org/10.48550/arXiv.2209.05433
21. X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, *Ultra-low precision 4-bit training of deep neural networks*, Adv. Neural Inf. Process. Syst. **33** (2020), 1796–1807.

AUTHOR BIOGRAPHIES

Hyeji Kim received her B.S. and M.S. degrees in Electrical Engineering from Chungnam National University, Daejeon, Republic of Korea, in 2013 and 2015, respectively, and her Ph.D. degree in Electrical Engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 2020. Since 2020, she has been with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, where she is a senior researcher at the AI SoC Research Division. Her current research interests include artificial intelligence, algorithm and hardware co-design for neural network training, AI processors, and AI frameworks.

Yeongmin Lee received his B.S. degree in Electrical Engineering from Inha University, Incheon, Republic of Korea, in 2012, and his M.S. and Ph.D. degrees in Electrical Engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2015 and 2020, respectively. From 2020 to 2022, he was a staff engineer at System LSI, Samsung Electronics, where he was involved in the development of algorithms for image processing and computer vision for mobile devices. Since 2022, he has worked as a senior research engineer at the Electronics and

Telecommunications Research Institute. His research interests include system-on-chip design, artificial intelligence, neural processing, 3D graphics, and computer vision.

Chun-Gi Lyuh received his BS degree in Computer Engineering from Kyungpook National University, Daegu, Republic of Korea, in 1998. He received his M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea, in 2000 and 2004, respectively. He joined the Electronics and Tele-communications Research Institute (ETRI), Daejeon, Republic of Korea, in 2004, and is currently a principal member of the research staff. His current research interests include mobile deep-learning processors, high-performance CPU designs for HPC, and SoC architecture optimization for artificial neural networks based on transformers.

How to cite this article: H. Kim, Y. Lee, and C.-G. Lyuh, *PF-GEMV: Utilization maximizing architecture in fast matrix–vector multiplication for GPT-2 inference*, ETRI Journal **46** (2024), 817–828, DOI 10.4218/etrij.2024-0111