LV: Latency-Versatile Floating-Point Engine for High-Performance Deep Neural Networks

Yun-Chen Lo, Yu-Chih Tsai, and Ren-Shuo Liu

***Abstract*—Computing latency is an important system metric for Deep Neural Networks (DNNs) accelerators. To reduce latency, this work proposes LV, a latency-versatile floating-point engine (FP-PE), which contains the following key contributions: 1) an approximate bit-versatile multiplier-and-accumulate (BV-MAC) unit with early shifter and 2) an on-demand fixed-point-to-floating-point conversion (FXP2FP) unit. The extensive experimental results show that LV outperforms baseline FP-PE and redundancy-aware FP-PE by up to 2.12× and 1.3× speedup using TSMC 40-nm technology, achieving comparable accuracy on the ImageNet classification tasks.**

***Index Terms*—Approximate computation, floating point, latency-versatile architecture.**

I. Introduction

Multiply-accumulate (MAC) computing speed is an important specification of high-performance Deep Neural Network (DNN) accelerators. Previous work shows that speeding up fixed-point MAC by exploiting its latency variations is viable [5], [6]. They dynamically select one out of 28 clock phases based on the operands of MAC. In addition to fixed-point MAC, previous papers have also focused on speeding up floating-point (FP) MAC in DNN accelerators. However, to our knowledge, a technique that exploits the latency variations of FP MAC is not available yet.

More specifically, the existing techniques speed up FP MAC either by exploiting sparsity (i.e., ineffectual computation) or customizing FP formats. The former skip ineffectual computations, e.g., zeros (zero-aware architectures) and products that are much smaller (i.e., out of bound, OB) compared with the partial sum under accumulation (OB-aware architectures) [1]. The latter revisits and customize FP according to AI characteristics to speed up FP MAC. For instance, Brain FP (BF16) [7] is an AI-oriented FP configuration (i.e., 1-bit sign, 7-bit mantissa, 8-bit exponent). MSFP [3] and BSFP [8] propose FP formats that can efficiently represent FP vectors.

This work proposes **LV**, latency-versatile processing element (PE) architecture that exposes and exploits the latency variations of FP MAC. Specifically, we propose two following methods to make latency variations exploitable: 1) We restructure the FP-PE and let the exponent difference serve as an indicator for the activated bitwidth of the multiplier and adder. Further, we reduce the clock period for low-bit MAC operations to exploit latency variation. 2) We track the exponent of partial sum and only activate the time-consuming fixed-point-to-floating (FXP2FP) module when the exponent changes. Therefore, we do not need FXP2FP in most cycles, and thus these cycles can utilize faster clock frequency. In sum, the key contributions are as below:

- We profile the latencies of three state-of-the-art FP-PEs (i.e., FP32, BF16, FP16).
- We are the first to propose a methodology that exploits latency variation for DNN floating-point processing element (FP-PE).
- We propose an approximate bitwidth-versatile multiplier and accumulator, enabling us to utilize exponent difference for selecting computing latency.
- We propose an on-demand FXP2FP module to enhance the speedup.
- The experimental results demonstrate that our proposed LV FP-PE outperforms the performance of baseline FP-PE and redundancy-aware FP-PE by up to 2.12× and 1.3× respectively while achieving similar accuracy ($\leq$0.1% top-1 accuracy drop).

II. Backgrounds and Prior Works

A. Floating Point and Processing Element

Floating point numbers are important for their capability to support both training and inference. An IEEE-754 compliant floating point number contains sign (s), mantissa (m), exponent (e), and bias[1], which represents the following value:

$$(-1)^s \times 1.m \times 2^{e-bias} \qquad (1)$$

To efficiently compute floating-point weights and activations of DNNs, several works have designed processing element to accelerate Multiply-and-Accumulate (MAC). As shown in Fig. 1(a) and (b), a general processing element [4] performs a floating-point MAC, which can be easily integrated with embedded CPUs. As discussed before, some prior efforts have proposed to exploit sparsity and out-of-bound computations [1]. Fig. 1(c) shows typical redundancy-aware FP-PE, which adds

Manuscript received 9 May 2023; accepted 29 May 2023. Date of publication 25 August 2023; date of current version 27 September 2023. This work was supported in part by NSTC under Grants 111-2823-8-007-001 and 112-2218-E-007-012, in part by MOE under Grant 112QR001N4, in part by Synopsys Fellowship, and in part by NSTC Fellowship. *(Corresponding author: Ren-Shuo Liu.)*

The authors are with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu 300044, Taiwan (e-mail: yunchen.lo@gapp.nthu.edu.tw; williemiku0504@gmail.com; renshuo@ee.nthu.edu.tw).

Digital Object Identifier 10.1109/LCA.2023.3287096

[1]In the following context, we use M to represent $(-1)^s \times 1.m$ and E to represent $e - bias$ for brevity.

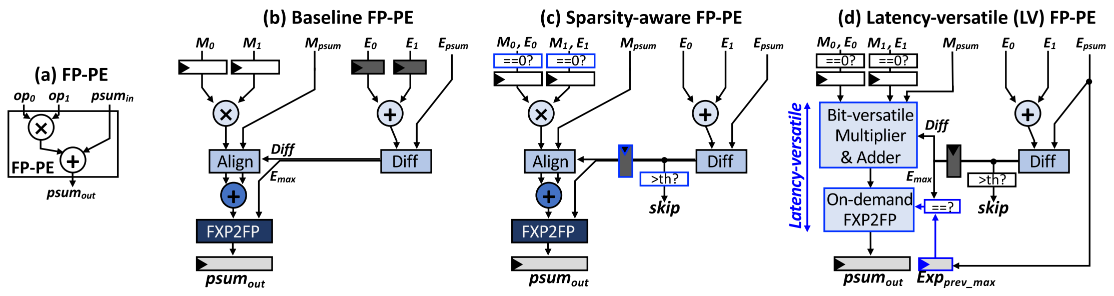

Fig. 1. (a) Operations of floating point processing element (FP-PE). (b) Baseline FP-PE. (c) Redundancy-aware FP-PE [1] that exploits zero and out-of-bound computation. (d) The proposed latency-versatile (LV) FP-PE.

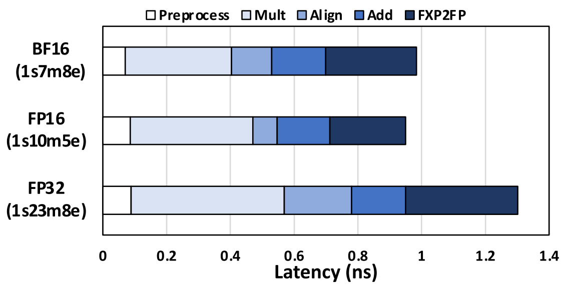

Fig. 2. Latency breakdown of baseline FP-PE with FP32, FP16, and BF16 operands on TSMC 40-nm node. All PEs utilize FP32 accumulator.

zero detection on the input slots of multiplication and inserts threshold circuit to identify out-of-bound MACs. This class of PEs first checks whether the operands require subsequent MAC and only activates PEs when needed.

B. *Latency-Variation Architecture*

Since this work focuses on exploiting latency variation for DNNs, we discuss prior work in this direction. DynOR [2] utilizes LUT to memorize the latency for different operations of ALU. However, the PEs of the DNN accelerator only need to support MAC, eliminating the possibility of exploiting inter-operation latency variation (e.g., multiplication is slower than bitwise-AND). Jia et al. exploit latency variation to build DNN accelerator [6]. They have demonstrated that variable-latency circuitry is realizable using a silicon prototype. Specifically, they utilize the number of bit transitions and corresponding significance to predict latency. However, their latency detection method is inefficient for FP numbers because of the design of hidden-one mantissa, which increases the bit-flip possibility on MSB. In comparison, this work is the first to exploit the characteristic of floating-point MAC for creating exploitable latency variation.

III. Latency Breakdown Analysis

Fig. 2 shows the latency breakdown of baseline FP-PE on TSMC 40-nm node. We implement and synthesize general FP-PE (Fig. 1(b)) to characterize the module-wise latency breakdown. The key takeaways are: 1) The most time-consuming modules are multiplier and FXP2FP units. Thus, we prioritize optimizing the dynamic timing variation for these two units. 2) The multiplier latency is proportional to mantissa precision. 3) The adder latency is relatively consistent across different floating point numbers because all designs adopt FP32 accumulator [4].

IV. Latency-Versatile (LV) FP-PE

Fig. 1(d) presents the latency-versatile FP-PE, where we highlight the key modifications, i.e., bit-versatile MAC (BV-MAC) and on-demand FXP2FP. In the following paragraph, we first summarize key intuition of the proposals and then unveil their details.

Intuition for BV-MAC: In general, low-bit MAC is faster than high-bit MAC because a low-bit multiplier needs fewer terms of partial products. The proposed bit-versatile MAC (BV-MAC) exploits this phenomenon by activating only part of the input bitwidth for the multiplier and adder to create latency variation.

Furthermore, we judiciously use the exponent difference between operands and partial sum to select the activated bitwidth for MAC engine. If the exponent difference is more significant, the multiplier requires a lower effective bitwidth and is faster. Similarly, the exponent difference can determine the activated bitwidth and latency for the adder. Please note that exponent difference is already required in FP-PE. Thus generating it does not require additional area overheads.

Intuition for on-demand FXP2FP: The intuition for on-demand FXP2FP is relatively straightforward. We observe that the exponent of the partial sum does not frequently vary during accumulation, and hence current architecture that always performs FXP2FP is yet optimized. In response, we propose equipping floating-point PE with fixed-point-domain and floating-point-domain registers. Therefore, we can primarily accumulate on the fixed-point-domain register and seldom accumulate on the floating-point-domain register to create the latency variation for the FXP2FP unit.

A. *Bit-Versatile Multiplier and Adder*

We first present the mathematical derivation of bit-versatile multiplier and then use an example to illustrate the idea. We assume that the exponent of psum is larger than the exponent of multiplication product, which is the condition that our proposal targets. The typical processing element performs multiplication between two operands (op0 and op1) and then accumulates the product into partial sum (psum), which can be described using the following equations:

$$psum_{out} = (op0 \times op1) + psum_{in} \tag{2}$$

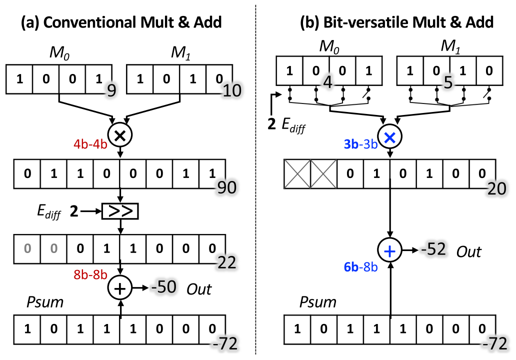

Fig. 3. (a) Conventional bit-fixed MAC and subsequent alignment unit. (b) Bit-versatile multiplier with bitwidth selection unit.

$$= (M_0 \times M_1) \times 2^{E_0+E_1} + M_{psum} \times 2^{E_{psum}} \quad (3)$$

$$= (M_{psum} + M_0 \times M_1 \times 2^{(E_0+E_1-E_{psum})}) \times 2^{E_{psum}} \quad (4)$$

$$= (M_{psum} \underbrace{+}_{fixed} ((M_0 \underbrace{\times}_{fixed} M_1) \gg E_{diff})) \gg E_{psum} \quad (5)$$

Typically, the circuit is designed to follow the PEMDAS rule to compute (4). The challenges of exploiting latency variation on a conventional circuit, i.e., (5), are twofold: 1) the highlighted multiplier and adder of mantissa require large precision, e.g., 24b-to-24b for FP32, and there is no existing method to identify minimal bitwidth for computation dynamically. 2) Floating point numbers adopt a hidden-one design for the mantissa. Thus the operands for the multiplier are typically fully-utilized and contain bit-flip on MSB.

In response, we propose re-ordering the equation and utilizing exponent difference to select minimally activated input precision. In other words, we move the alignment from the output side of the multiplier to its input side. The corresponding equation becomes the following:

$$(M_{psum} \underbrace{+}_{versatile} (M_0 \gg E_{diff0}) \underbrace{\times}_{versatile} (M_1 \gg E_{diff1})) \gg E_{psum} \quad (6)$$

Here, we evenly right-shift M_0 and M_1 ($E_{diff0} + E_{diff1} = E_{diff}$). Since the right shift is performed first, the highlighted multiplier and adder can now easily determine the required bitwidth to activate, hence making it easy for the circuit to exploit latency variations.

Fig. 3 shows an example and compares (a) conventional MAC and (b) bit-versatile MAC. The example multiplies 9 with 10, right-shifts the product with 2, and then accumulates the aligned product with the existing partial sum -72 to generate an output value of -50. The conventional circuit generates the same output value while adopting the largest input bitwidth for MAC, hence becoming hard to exploit latency variation.

Our proposed bit-versatile MAC moves the alignment earlier, which right-shifts the operand before performing multiplication and accumulation. Therefore, the multiplier and adder only require smaller bitwidth, shortening the computation latency. In the given example, if the exponent difference is 2, we only need a 3b-3b multiplier and a 6b-8b adder while generating a similar result. Further, the generated partial sum is -52, similar to the original -50.

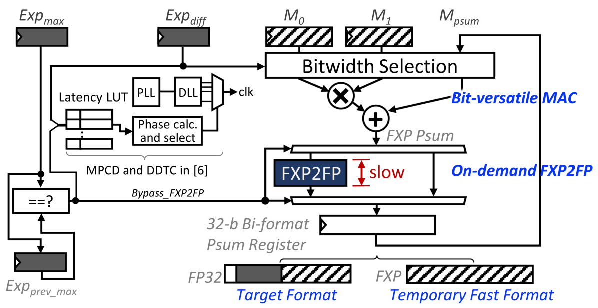

Fig. 4. Latency-versatile (LV) PE, which integrates bit-versatile MAC, on-demand FXP2FP, and latency selection (MPCD and DDTC [6]).

B. On-Demand FXP2FP

FXP2FP is slow since it requires normalization and other units to fully utilize the mantissa field. These modules are time-consuming because of their tree-based structure (e.g., leading one detector in normalization). Two interesting observations are: 1) During the computations, the FXP2FP circuit takes in largest exponent ($exp_{\max}$) to adjust the exponent. 2) The exponent of the partial sum is mostly stable, hence consistently performing FXP2FP is not required. In response, Fig. 4 shows the proposed on-demand FXP2FP, which mostly accumulates partial sum using the non-normalized format and selectively activates the FXP2FP unit to accumulate in FP format. To support the proposed technique, we add a register to store the psum exponent and utilize it to check if the partial sum changes significantly in two subsequent cycles. Specifically, if the exponent of partial sum changes, we perform FXP2FP and store partial sum using FP format. If the exponent of the partial sum is the same, we bypass the FXP2FP to store it in FXP format, which reduces the latency.

C. Multi-Phase Clocking and Asynchronous Interface

We assume the peripheral clocking circuitry to be similar to the multi-phase clock distribution (MPCD) and data detection and timing control (DDTC) proposed in [6]. Specifically, they employ one root clock (i.e., a PLL, a DLL, and a multi-phase clock distribution bus) and multiple phase-selection circuits to form multiple asynchronous clock domains. By switching among multiple clock phases per cycle according to the latency variations of FP MAC this work identified and exposed, as tight a clock period as possible is achieved. LV is asynchronous architecture because of clock phase switching. FIFOs and toggle-based handshaking are standard approaches to handle asynchronous clock domain crossing. Asynchronous multi-PE is also feasible [6].

V. Evaluation

A. Experimental Setup

We choose four representative DNNs, i.e., ResNet-18, MobileNet-v1, MobileNet-v2, and ShuffleNet-v2. All networks are for ImageNet classification tasks. We quantize these networks to target FP configurations from pretrained checkpoints.

TABLE I
AREA, POWER, AND LATENCY BREAKDOWN OF FP32 AND FP16 LV PEs ON TSMC 40 NM

	Component	Area (μm^2)	%	Power (mW)	%	Latency (ns)
FP32	Pre-process	624	11	0.180	8	0.2
	Multiplier	2946	52	1.116	48	0.45~0.78
	Shifter	631	11	0.406	17	
	Adder	780	14	0.402	17	
	On-demand FXP2FP	672	12	0.235	10	0 or 0.34
	Total	5653	100	2.339	100	0.99~1.32
FP16	Pre-process	449	9	0.107	5	0.13
	Multiplier	2157	42	0.791	40	0.45~0.61
	Shifter	827	16	0.311	16	
	Adder	751	15	0.312	16	
	On-demand FXP2FP	945	18	0.462	23	0 or 0.24
	Total	5129	100	1.983	100	0.58~0.98

The results do not include MPCD and DDTC.

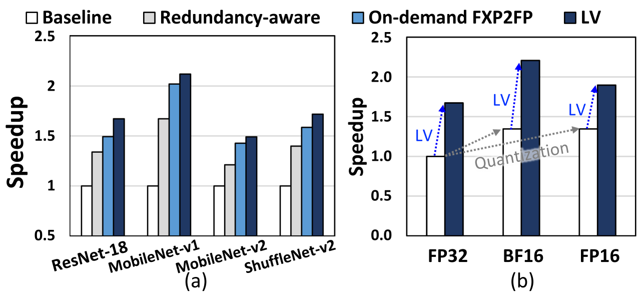

Fig. 5. (a) LV's Speedup on four FP32 newtorks. (b) Speedup comparisons between LV and quantized FPs, normalized to the FP32 baseline.

We do not apply any special compression and quantization techniques. We implement the designs using Verilog. We synthesize all of the designs using Synopsys Design Compiler with TSMC 40-nm technology. We perform static timing analysis (STA) to determine the latency selection condition. We extend PyTorch to create a simulator for performance estimation.

B. Area, Power, and Latency Breakdown

Table I shows the area and latency breakdown of the FP32 and FP16 LV FP-PEs on TSMC 40-nm node. For FP32 FP, the BV-MAC module is the most area-consuming module, which occupies 4357 μm^2. Due to the proposed bit-versatile MAC and on-demand FXP2FP, the completion time for MAC ranges from 0.45$\sim$0.78 ns. FP16 reduces the latency and area cost due to its compact format. The area overheads for bit-versatile MAC and on-demand FXP2FP are manageable, which only requires an extra exponent register and a multiplexer.

C. Speedup Analysis

Fig. 5(a) compares the performance of baseline, redundancy-aware baseline, bit-versatile MAC, and bit-versatile MAC+on-demand FXP2FP (LV). In brief, the bit-versatile MAC improves the performance by up to 1.7×, compared to baseline. Furthermore, the complete LV improves the performance by up to 2.12× and 1.3×, compared to baseline and redundancy-aware baseline. Fig. 5(b) further shows that LV can significantly speed up quantized FPs, too, e.g., from 1.35 to 2.21× (BF16).

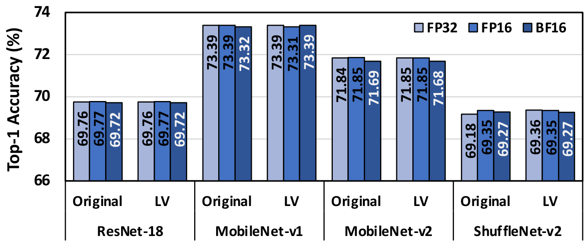

Fig. 6. Top-1 accuracy comparison on four networks.

D. Accuracy Analysis

Fig. 6 shows the accuracy of adopting the original PEMDAS computation order and the one using a bit-versatile MAC engine (LV). The inference accuracy is comparable to the original design ($\leq$0.1% top-1 accuracy), and is even higher than the baseline under some scenarios. The bit-versatile MAC is a good approximation method that leads to negligible accuracy drop and helps exploit variation.

VI. CONCLUSION

This article proposes **LV**, a latency-versatile floating-point engine (FP-PE), which contains the following key contributions: 1) an approximate bit-versatile multiplier-and-accumulate (BV-MAC) unit with an early shifter. 2) a on-demand fixed-point-to-floating-point conversion (FXP2FP) unit. The extensive experimental results show that LV outperforms baseline FP-PE and redundancy-aware FP-PE by up to 2.12× and 1.3× speedup using TSMC 40-nm technology, achieving comparable accuracy.

ACKNOWLEDGMENTS

We thank the reviewers for their constructive comments. We thank NCHC and TSRI for server services.

REFERENCES

[1] O. M. Awad et al., "FPRaker: A processing element for accelerating neural network training," in *Proc. IEEE/ACM 54th Annu. Int. Symp. Microarchitecture*, 2021, pp. 857–869.
[2] J. Constantin, A. Bonetti, A. Teman, C. Müller, L. Schmid, and A. Burg, "DynOR: A 32-bit microprocessor in 28 nm FD-SOI with cycle-by-cycle dynamic clock adjustment," in *Proc. 42nd Eur. Solid-State Circuits Conf.*, 2016, pp. 261–264.
[3] B. Darvish Rouhani et al., "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 10271–10281.
[4] G. Jeong et al., "RASA: Efficient register-aware systolic array matrix engine for CPU," in *Proc. IEEE/ACM 58th Des. Automat. Conf.*, 2021, pp. 253–258.
[5] T. Jia, R. Joseph, and J. Gu, "An adaptive clock management scheme exploiting instruction-based dynamic timing slack for a general-purpose graphics processor unit with deep pipeline and out-of-order execution," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2019, pp. 318–320.
[6] T. Jia, Y. Ju, and J. Gu, "A compute-adaptive elastic clock-chain technique with dynamic timing enhancement for 2D PE-Array-Based accelerators," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2020, pp. 482–484.
[7] D. D. Kalamkar et al., "A study of BFLOAT16 for deep learning training," 2019, *arXiv: 1905.12322*.
[8] Y.-C. Lo, T.-K. Lee, and R.-S. Liu, "Block and subword-scaling floating-point (BSFP) : An efficient non-uniform quantization for low precision inference," in *Proc. Int. Conf. Learn. Representations*, 2023.