An FMA Unit Based on Error-Free Transformation

Xingyu Lu†, YiYu Tan†, Toshiyuki Imamura §
† Faculty of Science and Engineering, Iwate University, Japan
§ RIKEN Center for Computational Science, Japan

(i) Introduction

The rapid advancement of deep learning and high-performance computing has driven widespread adoption of low-precision floating-point formats in modern hardware accelerators. Brain Floating Point (BF16), with its 8-bit exponent and 7-bit mantissa, has become popular in AI accelerators due to its hardware efficiency and dynamic range comparable to FP32. However, many applications in scientific computing and neural network training still require FP32-level precision to ensure numerical stability and accuracy. This study develops an FMA unit based on BF16 operations through decomposing an FP32 datum into multiple BF16 data and maintaining the accuracy of FP32.

(ii) Methods

❖ **Data decomposition**. An FP32 datum is decomposed into four BF16 data through bit truncation and extension. The decomposition is similar as the error free splitting [1].

Algorithm : Splitting of a scalar $x \in \mathbb{F}_{FP32}$ into 4 BF16 data

($\mathbf{a}_{split}[j]$ are BF16 data, $c[j]$ are scale factors. Lines 5-7 compute each split data)

1: **function** ($\mathbf{a}_{split}[0:3]$, $c[0:3]$) = SplitFP32toBF16(x)
2: $x_{cur} = x$ *// Current value to be splited*
3: **for** $j = 0$ to 3 **do**
4: $c[j] = 7j$ *// Store the displacement value. Scale: 0, 7, 14, 21*
5: $\mathbf{a}_{split}[j] = \mathrm{trunc}_{BF16}(x_{cur})$ *// Truncate to BF16*
6: $x_{tmp} = \mathrm{fl}_{FP32}(x_{cur} - \mathrm{ext}_{FP32}(\mathbf{a}_{split}[j]))$ *// Extract the lower 16 bits of the mantissa via subtraction.*
7: $x_{cur} = x_{tmp} \times 2^7$ *// Add the exponent by + 7 to prevent underflow.*
8: **end for**
9: **return** ($\mathbf{a}_{split}[0:3]$, $c[0:3]$)
10: **end function**

$\mathrm{trunc}_{BF16}(x)$: Convert FP32 to BF16 by truncating the mantissa with 7 bits.

$\mathrm{ext}_{FP32}(a)$: Extend BF16 to FP32 by padding the low 16 bits with 0

$\mathrm{fl}_{FP32}(\cdot)$: Floating-point operation with FP32

Reconstruction : $x = \sum_{j=0}^{3} \mathbf{a}_{split}[j] \times 2^{-c[j]} = a_0 + a_1 \cdot 2^{-7} + a_2 \cdot 2^{-14} + a_3 \cdot 2^{-21}$

Figure 1: Data Decomposition

❖ **FMA architecture** (Figure 2). Two FP32 data are decomposed into A0,A1,A2,A3 and B0,B1,B2,B3, respectively. Therefore：

$$A \times B = (A0 + A1 + A2 + A3) \times (B0 + B1 + B2 + B3)$$

$$= \sum_{i=0}^{3} \sum_{j=0}^{3} (A_i \times B_j)$$

❖ Ten terms with an exponent larger than -23 are computed using a customized BF16 multiplier, in which two inputs are BF16 data and the product is FP32 to avoid rounding error.

❖ The products along the diagonal have the same scale, and they are first summed and then scaled. The scaled summations are added with C through adders with FP32 to get the result.

(iii) Evaluation Results

❖ **Simulation results.** Table 1 presents the computation results of several representative cases. The FP32 and BF16 Direct are direct computing results using the FMA with FP32 and BF16, respectively. As shown in Table 1, the proposed method can obtain the same results as the FP32, and the accuracy is much higher than the BF16.

❖ **Hardware resource utilization.** The FMA unit is implemented using an Intel Stratix 10 FPGA (1SG280HU1F50E2VG). The FPGA development environment is Quartus Prime Pro 21.2. The customized BF16 multiplier and FP32 adder are all implemented by the IP cores provided by the Quartus Prime Pro.

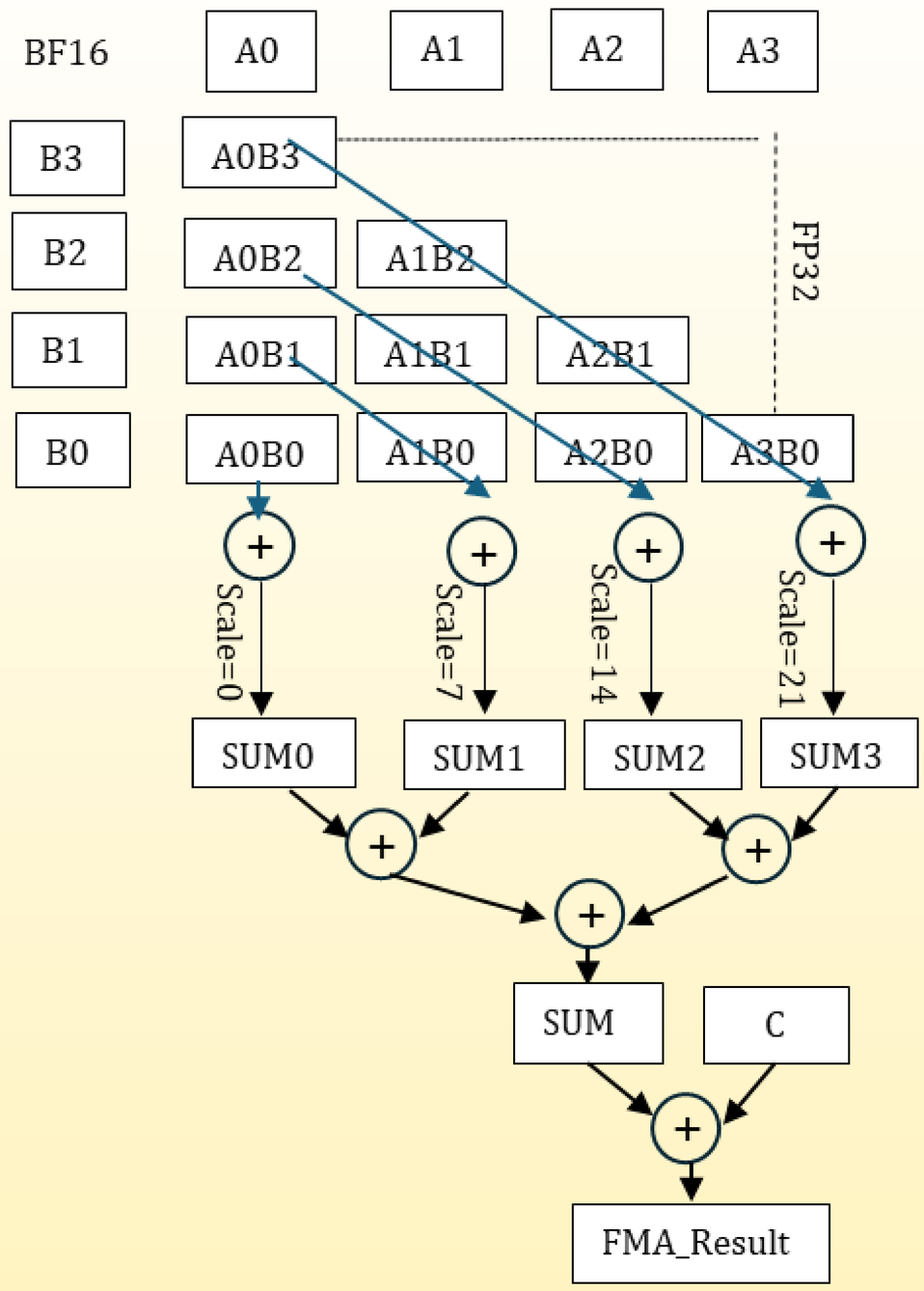

Figure 2: FMA Architecture

➢ Table 1: Computation Results

Expression	FP32	BF16 Direct	Proposed method
1.001×0.999+0.999001	1.999000	1.992188	1.999000
1.001953125^2+1.00390625	2.007816	2.000000	2.007816
1.0625^2+0.12890625	1.257812	1.250000	1.257812
100×200+500	20500.00	20352.00	20500.00
π×e	8.539721	8.437500	8.539721
0.1^2+0.99	1.000000	0.996094	1.000000

➢ Table 2: Hardware Resource Utilization

Logic (ALMs)	Registers	RAM blocks	DSP blocks	Clock frequency
735	1474	1	15	390 MHz

(iv) Conclusions

With the decomposition of an FP32 datum into four BF16 data, an FMA with customized BF16 multipliers and FP32 adders, which are implemented by the native DSP blocks inside the FPGA to high performance, can get similar accuracy to the FP32. In future work, the FMA unit will be applied to develop a matrix multiplier.

Acknowledgements

Thanks for Intel's donation of the software tools through University Program. This work was partly supported by the JSPS KAKENHI Grant Number JP25K03126.

Reference:

[1] K. Ozaki, T. Ogita, S. Oishi, M. Rummp, 2011. Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. Numerical Algorithms, 59(1), 95–118.