Accelerating AI on RISC-V Optimizing BFloat16 for Improved Efficiency

Ruhma Rizwan[1], Maah Noor[1], Zeshan Rehman[1], Umer Imran[1]

[1]10xEngineers, Lahore, Pakistan

June 3, 2024

Abstract

This paper presents an opensource[1] design of a possible implementation for a low-cost, modular, BFloat16 (BF16) FPU micro-architecture which targets RISC-V based embedded cores. BF16 is recommended in deep learning since it covers the same value range as the 32-bit floating-point format (FP32), ensuring models train and run effectively without extra parameter adjustments and allows simple conversions to and from Fp32. We use the CV32E40P core with our BF16 module that contains custom instructions for multiply-accumulate, conversion, and min-max operations. Our design relaxes certain features of the IEEE Floating-Point Standard to realize a cost-effective hardware implementation. Particularly, it applies zero-flushing to subnormal operands, uses just one rounding mode i.e. RNE (Round to nearest, tie to even), does not propagate input Nan's, and applies zero-skipping for sparse networks. Using these optimizations, we have achieved almost 35% reduction in silicon area compared to an IEEE compliant FP32 implementation with minimal impact on computational accuracy.

Introduction

BFLOAT16 is a 16-bit format optimized for machine learning, with a wider dynamic range than FP16, making it better suited for certain neural network computations. It aligns with the IEEE single-precision format by maintaining the same exponent width while reducing the fraction bits from 23 to 7. BF16's fewer mantissa bits mean its multipliers are approximately half the size of those for FP16 and eight times smaller than FP32 multiplier. Due to smaller data sizes, it has optimized on-chip memory usage and improved memory bandwidth. This results in reduced computational resources, faster processing, lower power use, and compatibility with 32-bit floating-point environments. Neural networks are more sensitive to the size of the exponent than the mantissa and can maintain accuracy with lower-precision values like FP16 or BF16. The same exponent size as FP32 ensures models train and run effectively without extra costly parameter adjustments and network redesign.[1]

The resilience of Neural Networks to numerical inaccuracies has led to varied adherence to the IEEE-754 standards. This results in different implementations across various platforms like Google, ARM, Intel, and RISC-V for BF16 arithmetic. RISC-V extends BF16 vector support to include subnormal numbers and all IEEE-defined rounding modes, accommodating a broad spectrum of computational demands. Meanwhile, ARM's dot product and multiply accumulate instructions enhance matrix multiplication efficiency with numeric simplifications such as a single rounding mode and flushing subnormals to zero. Intel's deep learning boost leverages BF16 without denormal support and exception handling. Google's implementation of BF16 in Cloud TPUs focuses on hardware efficiency, flushing denormals and using one rounding mode only. This diversity in BF16 implementation reflects the balance between efficiency and precision needed in high-performance computing.

[1] `https://github.com/10x-Engineers/cv32e40p`

Methodology

This paper proposes custom Bfloat16 instructions for a simplified implementation of the IEEE-Floating point standard, suitable for low-precision tolerant applications like Neural Networks. This design implementation is currently integrated with CV32E40P core, previously known as RI5CY. It is a high-performance, low-power 32-bit RISC-V core developed as part of the open-source CORE-V family of processors, aimed specifically at embedded system applications. This integration introduces custom instructions for multiply-accumulate (FMACC) operations, conversion, and minmax functions, with the FMACC module supporting addition, subtraction, and both fused-multiply-add and subtract operations.

The simplifications for the proposed relaxed implementation are as follows:

- Subnormal Inputs are flushed to zero
- Only one rounding mode i.e. RNE (Round to nearest, tie to even) is used
- Only one type of Nan is ever returned i.e. "default Nan" and zero-values are skipped

I. Subnormal Flush-to-zero

BF16's 8-bit exponent offers sufficient dynamic range for neural network computations even without subnormals, and this simplification does not impact NN operation [2]. In contrast, FP16 with only 5 exponent bits requires scaling operations to avoid overflow and error accumulation. Therefore, this simplifies the costly FPU logic needed to handle subnormals in multipliers and adders. This simplification saves area of our BFloat16 design by 21-22% as compared to an IEEE compliant implementation that handles subnormals.

II. Single Rounding Mode (RNE)

The IEEE-754 standard includes multiple rounding modes, but we limit ourselves to a single one, i.e., round-nearest even. It is preferred in deep learning for its balanced handling of ties and error reduction in neural network computations. This approach minimizes rounding bias, improving accuracy and fairness, and shrinks our implementation's area by 13-14% as compared to full IEEE compliant implementation.

III. Propagating Nans and Zero-skipping

We achieved marginal reductions in area by limiting NaN propagation to a single default value. Furthermore, acknowledging that a substantial portion of operations within low-precision networks involves zero values [3], we implement zero-skipping to bypass computations on these values leading to an increase in speed that becomes more pronounced in deeper network layers, which tend to exhibit increased sparsity.

Experimental Evaluation

The BF16 design implementation has been synthesized at 1GHz using TSMC 65nm GP Standard Cell Libraries with the Genus Synthesis tool under nominal conditions (1V, 25°C) consuming 4.5W total power and at 500MHz under worst-case conditions (0.9V, 125°C) consuming 1.8mW. The total area of the BF16 optimized FPU is 2.38kGE, with the FMACC occupying 2.14kGE. The simplifications resulted in up to 35% area savings relative to a fully IEEE compliant implementation.

Polybench Benchmark To evaluate the proposed BF16 implementation, we utilized the PolyBench benchmark suite, a collection of floating-point-intensive programs. This analysis focused on measuring the percentage mean relative error of BF16 compared to FP32 which served as the reference standard as can be seen in Figure 1. Each benchmark's FPU output was compared to an IEEE compliant FP32 design.

We observe that the mean relative error using BF16 compared to FP32 is low, mostly below 0.5% except 'cholesky' application with a higher error of 2.7% due to it's varying magnitudes causing accuracy loss with reduced FP precision.

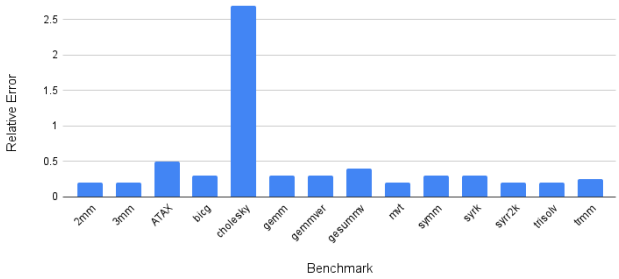

Figure 1: *Polybench mean relative error comparison of our BF16 module against fully compliant IEEE standard FP32 as reference.*

Table 1: *NN models accuracy with different FP formats.*

Training/Inference	Resnet	Inception
FP32(T)/BF16(I)	0.7496	0.7754
BF16(T)/BF16(I)	0.7376	0.7243
FP32(T)/FP32(I)	0.7495	0.7761

Neural Network Models Utilizing a Bfloat16 software model with RNE rounding and flush-to-zero for subnormals, we ran ResNet and Inception on CIFAR-10. The comparison in Table 1 of NN model accuracies across training and inference precisions (FP32/BF16 and BF16/BF16) against traditional FP32/FP32 precision demonstrates negligible impact on performance, affirming neural networks' resilience to reduced precision computing, rounding mode, and subnormal flushing.

Conclusion The paper proposes a power and area efficient BF16 implementation using single rounding mode, subnormal flushing, and other optimizations. Our analysis shows that these simplifications preserve accuracy across Polybench applications and neural network (NN) models. This demonstrates the module's capability to enhance embedded NN applications efficiently without significant performance trade-offs.

References

[1] Dhiraj Kalamkar et al. "A study of BFLOAT16 for deep learning training". In: *arXiv preprint arXiv:1905.12322* (2019).

[2] Neil Burgess et al. "Bfloat16 Processing for Neural Networks". In: *2019 IEEE 26th Symposium on Computer Arithmetic*. 2019. DOI: 10.1109/ARITH.2019.00022.

[3] Ganesh Venkatesh, Eriko Nurvitadhi, and Debbie Marr. "Accelerating deep convolutional networks using low-precision and sparsity". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 2861–2865.