DLFloat: A 16-b Floating Point format designed for Deep Learning Training and Inference

Ankur Agrawal, Silvia M. Mueller¹, Bruce Fleischer, Jungwook Choi, Xiao Sun, Naigang Wang and Kailash Gopalakrishnan

IBM TJ Watson Research Center; ¹IBM Systems Group
Background

• Deep Learning has shown remarkable success in tasks such as image and speech recognition, machine translation etc.

• Training deep neural networks requires **100s of ExaOps** of computations
  • Typically performed on a cluster of CPUs or GPUs

• Strong trend towards building specialized ASICs for Deep Learning inference and training
  • Reduced precision computation exploits the resiliency of these algorithms to reduce power consumption and bandwidth requirements
Reduced Precision key to IBM’s AI acceleration

• We showcased our 1.5 Tflop/s deep learning accelerator engine at VLSI’18, consisting of a 2D array of FP16 FPU
s
• We also announced successful training of Deep networks using hybrid FP8-FP16 computation
• Both these breakthroughs rely on an optimized FP16 format designed for Deep Learning – **DLFloat**
Outline

• Introduction
• DLFloat details
• Neural network training experiments
• Hardware design
• Conclusions
Proposed 16-b floating point format: DLFloat

\[ X = -1^s \times 2^{e-b} \times (1 + \frac{m}{512}) \]

Features:
- Exponent bias (b) = -31
- No sub-normal numbers to simplify FPU logic
- Unsigned zero
- Last binade isn’t reserved for NaNs and infinity
Merged Nan-Infinity

• Observation: if one of the input operands to an FMA instruction is NaN or Infinity, the result is always NaN or infinity.

• We merge NaN and infinity into one symbol
  • Encountering Nan-infinity implies “something went wrong” and exception flag is raised

• Nan-infinity is unsigned (sign-bit is a don’t care)
**DLFloat Format and Instructions**

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Fraction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>0000000000</td>
<td>0</td>
</tr>
<tr>
<td>000000</td>
<td>!= 0000000000</td>
<td>$2^{-31} \times 1.f$</td>
</tr>
<tr>
<td>000001 ... 111110</td>
<td>*</td>
<td>$2^e \times 1.f$</td>
</tr>
<tr>
<td>111111</td>
<td>!= 111111111</td>
<td>$2^{32} \times 1.f$</td>
</tr>
<tr>
<td>111111</td>
<td>111111111</td>
<td>Nan-infinity</td>
</tr>
</tbody>
</table>

- **FP16 FMA Instruction:** $R = C + A \times B$
  - All operands are DLFloat16
  - Result is DLFloat16 with Round-nearest-up rounding-mode

- **FP8 FMA instruction:** $R = C + A \times B$
  - $R, C : DLFloat16$
  - $A, B : DLFloat8$ (8-bit floating point)
Comparison with other FP16 formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Exp bits</th>
<th>Frac bits</th>
<th>Total bit-width</th>
<th>Smallest representable number</th>
<th>Largest representable number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFloat16</td>
<td>8</td>
<td>7</td>
<td>16</td>
<td>$2^{-133}$</td>
<td>$2^{128}$-ulp</td>
</tr>
<tr>
<td>IEEE-half</td>
<td>5</td>
<td>10</td>
<td>16</td>
<td>$2^{-24}$</td>
<td>$2^{16}$-ulp</td>
</tr>
<tr>
<td>DLFloat (proposed)</td>
<td>6</td>
<td>9</td>
<td>16</td>
<td>$2^{(-31)}+ulp$</td>
<td>$2^{33}$-2ulp</td>
</tr>
</tbody>
</table>

- BFloat16 and IEEE-half FPUs employ a mixed-precision FMA instruction (16-b multiplication, 32-b addition) to prevent accumulation errors
  - Limited logic savings

- IEEE-half employs APEX technique in DL training to automatically find a suitable scaling factor to prevent overflows and underflows
  - Software overhead
Back-propagation with DLFloat16 engine

• All matrix operations are performed using DLFloat16 FMA instruction

• Only weight updates are performed using 32-b summation

• 2 copies of weights maintained, all other quantities stored only in DLFloat16 format

$Q(.) = \text{round nearest-up quantization}$
Results – comparison with Baseline (IEEE-32)

- Trained network indistinguishable from baseline
- In our experiments, we did not need to adjust network hyper-parameters to obtain good convergence
  - Allows application development to be decoupled from compute precision in hardware
Comparison with other FP16 formats

• In all experiments, inner-product accumulation done in 16-bits

• IEEE half training does not converge unless APEX technique is applied

• BFloat16 training converges with slight degradation in QoR

• DLFloat16 trained network indistinguishable from baseline

Long Short-term Memory (LSTM) network trained on Penn Tree Bank dataset for text generation
**BFloat16 vs DLFloat16 – a closer look**

- With only 7 fraction bits, BFloat16 is likely to introduce accumulation errors when performing large inner products
  - commonly encountered in language processing tasks

- We chose a popular language translation network, Transformer, and kept the precision of all layers at FP32 except the last layer that requires an inner product length of 42720

- Persistent performance gap if accumulation is performed in 16-bit precision
DLFloat accumulation enables FP8-training

- GEMM mult. : FP8
- GEMM accum. : FP16
- Weight update : FP16

- Hybrid FP8-FP16 has 2x bandwidth efficiency and 2x power efficiency over regular FP16, with no loss of accuracy over a variety of benchmark networks

(N. Wang et al., NeurIPS’18)
FP8 training with BFloat vs DLFloat accumulation

- FP8 FMA instruction: \( R = C + A \times B \)
  - \( R, C : \text{DLFloat16} \)
  - \( A, B : \text{DLFloat8} \) (8-bit floating point)
  - 8b multiplication, 16b accumulation

- FP8 format is kept constant, FP16 format is DLFloat and BFloat

- DLFloat comes much closer to baseline than BFloat, thus is a better choice for accumulation format
  - Gap can be reduced by keeping last layer training in FP16, as is the case in previous slide

Long Short-term Memory (LSTM) network trained on Penn Tree Bank dataset for text generation
Accumulation length = 10000
Using DLFFloat in an AI Training and Inference ASIC

- Throughput = 1.5 TFLOPs
- Density = 0.17 TFLOPs/mm²
- DLFFloat FPUs are 20x smaller than IBM 64b FPUs

B.Fleischer et al., “A Scalable Multi-TeraOPS Deep Learning Processor Core for AI Training and Inference” Symposium VLSI 2018
FMA block diagram

• True 16-b pipeline with R, A, B, C in DLFloat format

• 10-bit multiplier
  • 6 radix-4 booth terms
  • 3 stages of 3:2 CSAs

• 34-bit adder
  • Simpler than 22-bit adder + 12-bit incrementor
  • Designed as 32-bit adder with carry-in

• LZA over entire 34 bits

• Eliminating subnormals simplifies FPU logic
• Also eliminated special logic for signs, NaNs, Infinities
Round nearest up rounding mode

• Table shows the rounding decision (1 = increment, 0 = truncate)

• For Round-nearest up, sticky information need not be preserved
  → simplifies normalize, rounder

<table>
<thead>
<tr>
<th>LSB</th>
<th>Guard</th>
<th>Sticky</th>
<th>RN-Up</th>
<th>RN-down</th>
<th>RN-even</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
FMA block diagram

DLFloat16 FPU is 20X smaller compared to IBM double-precision FPUs!

Area breakdown very different from typical single- and double-precision FPUs!
Conclusions

• Demonstrated a 16-bit floating point format optimized for Deep Learning applications
  • Lower overheads compared to IEEE-half precision FP and BFloat16

• Balanced exponent and mantissa width selection for best range vs resolution trade-off
  • allows straightforward substitution when FP16 FMA is employed
  • enables hybrid FP8-FP16 FMA-based training algorithms

• Demonstrated ASIC core comprising of 512 DLFloat16-FPUs
  • Reduced precision compute enables dense, power-efficient engine
  • Excluding some IEEE-754 features results in a lean FPU implementation
Thank you!

For more information on AI work at IBM Research, please go to http://www.research.ibm.com/artificial-intelligence/hardware
Backup
Training is sensitive to quantization in the last layer. If the last layer is converted to FP16, training performance improves.
FP8 training procedure

Figure 2: A diagram showing the precision settings for (a) three GEMM functions during forward and backward passes, and (b) three AXPY operations during a standard SGD weight update process.

AXPY results are stochastically rounded to FP16