Approximate Computing for DNN
Olivier Sentieys

To cite this version:
Olivier Sentieys. Approximate Computing for DNN: Improving performance and energy efficiency of deep-learning hardware accelerators with controlled arithmetic approximations. CSW 2021 - HiPEAC Computing Systems Week, Oct 2021, Lyon, France. hal-03494932

HAL Id: hal-03494932
https://hal.science/hal-03494932
Submitted on 20 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Approximate Deep Learning Accelerators

Improving performance and energy efficiency of deep-learning hardware accelerators with controlled arithmetic approximations

Olivier Sentieys
Univ. Rennes, Inria
Energy Cost in a Processor/SoC

- 64-bit FPU: 20pJ/op
- 32-bit addition: 0.05pJ
- 16-bit multiply: 0.25pJ

Wire energy
- 240fJ/bit/mm per ↓↑
- 32 bits: 40pJ/word/mm
- 8 bits: 10pJ/word/mm

Memory/Register-File
- Depends on word-length

[Adapted from Dally, IPDPS’11]

Energy strongly depends on data representation and size
Complexity Issues of Deep NNs

• Deep (Convolutional) Neural Networks
Approximate Computing

• Many applications are **error resilient**
  – media processing, data mining, machine learning, web search, etc.

• AxC performs **approximations** to reduce **energy** and increase execution speed while keeping **accuracy in acceptable limits**
  – Relaxing the need for fully precise operations
  – *Number representations and word-length*

• Design-time/run-time
• Different levels
Resilience of ANN?

Aoccdnig to a rscheearch at Cmabrigde Univertisy, it deosn’t mtttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteer be at the rghit pclae. And we spnet hlaf our lfie larenng how to splel wrods. Amzanig, no! [O. Temam, ISCA10]

- Our biological neurons are tolerant to computing errors and noisy inputs
- **Quantization** of parameters and computations provides **benefits in throughput, energy, storage**
Even Worse for Training...

- Carbon footprint of DNN training

Analyzing the carbon footprint of current natural-language processing models shows an alarming trend: *training one huge model for machine translation emits the same amount of CO2 as five cars in their lifetimes (fuel included)*

- Many more operations than inference
- More pressure on memory access
- Much more difficult to accelerate

Need for a Significant Reduction of the Carbon Footprint of Neural Network Training Hardware

[Strubell et al., ACL 2019]
This rest of this talk is about

• Approximations in DNNs

• Reducing the **numerical precision** of arithmetic operations is a general way to increase performance and energy efficiency in computing
  – How does this apply to DNN?
  – Can we design low-precision accelerators for inference and **training**?
Number Representations

• **Floating-Point (FIP)**

\[ x = (-1)^s \times m \times 2^{e-127} \]

- \( s \): sign, \( m \): mantissa, \( e \): exponent

- Easy to use
- High dynamic range
- IEEE 754

<table>
<thead>
<tr>
<th>Format</th>
<th>e</th>
<th>m</th>
<th>bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Precision</td>
<td>8</td>
<td>23</td>
<td>127</td>
</tr>
<tr>
<td>Double Precision</td>
<td>11</td>
<td>52</td>
<td>1023</td>
</tr>
</tbody>
</table>

- **Fixed-Point (FxP)**

\[ x = p \times K \]

- \( p \): integer, \( K=2^{-n} \): fixed scale factor
- Integer arithmetic
- Efficient operators
- Speed, power, cost
- Hard to use...

\[ x = s.(-2)^m + \sum_{i=-n}^{m-1} b_i.2^i \]

- \( s \): sign, \( m \): magnitude, \( n \): fractional

<table>
<thead>
<tr>
<th>2^{m-1}</th>
<th>2^1</th>
<th>2^0</th>
<th>2^{-1}</th>
<th>2^{-n}</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>( b_{m-1} )</td>
<td>( b_{m-2} )</td>
<td>( b_1 )</td>
<td>( b_0 )</td>
</tr>
</tbody>
</table>

Integer part: \( m \) bits  Fractional part: \( n \) bits
Number Representations

- Energy, delay, and area vary a lot between numeric formats and word-length

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy (pJ)</th>
<th>Area (µm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b Add</td>
<td>0.03</td>
<td>36</td>
</tr>
<tr>
<td>16b Add</td>
<td>0.05</td>
<td>67</td>
</tr>
<tr>
<td>32b Add</td>
<td>0.1</td>
<td>137</td>
</tr>
<tr>
<td>16b FB Add</td>
<td>0.4</td>
<td>1360</td>
</tr>
<tr>
<td>32b FB Add</td>
<td>0.9</td>
<td>4184</td>
</tr>
<tr>
<td>8b Mult</td>
<td>0.2</td>
<td>282</td>
</tr>
<tr>
<td>32b Mult</td>
<td>3.1</td>
<td>3495</td>
</tr>
<tr>
<td>16b FB Mult</td>
<td>1.1</td>
<td>1640</td>
</tr>
<tr>
<td>32b FB Mult</td>
<td>3.7</td>
<td>7700</td>
</tr>
<tr>
<td>32b SRAM Read</td>
<td>5</td>
<td>N/A</td>
</tr>
<tr>
<td>32b DRAM Read</td>
<td>640</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Addition | Multiplication
---|---
8-bit integer | 0.03pJ / 36µm² | 0.2pJ / 282µm²
32-bit float | 0.9pJ / 4184µm² | 3.7pJ / 7700µm²

Energy numbers are from Mark Horowitz "Computing’s Energy problem (and what we can do about it)". ISSCC 2014
Area numbers are from synthesized result using Design compiler under TSMC 45nm tech node. FP units used DesignWare Library.
Floating-Point Arithmetic

- Floating-point hardware is doing the job for you!

- FIP operators are therefore more complex

[J.-M. Muller et al., Handbook of Floating-point arithmetic, Springer, 2009]
What can be customized?

- Of course precision
  - Exponent (E) and Mantissa (M) bit-width
  - $e$ and $m$ both impact accuracy
- Play with exponent **bias**
- **Sub-normal** numbers or not?
- 0, $\infty$, NaN?
- **Rounding** modes
  - to nearest, truncation, to 0/$\infty$
- Inexact integer operators
LP-Floating-Point Multiplication

• Example: 7 bits, (2,5)

\[ s_x e_x \]

\[ s_y e_y \]

\[ s_z = s_x \ XOR s_y \]

\[ e_z = e_x + e_y + (m_x \ AND \ m_y) \]

\[ s_z \]

\[ 1.m_x = \{1.0; 1.5\} \]

\[ 1.m_y = \{1.0; 1.5\} \]

\[ 1.m_z = \{1.0; 1.5; 2.25\} \]

\[ 1.m_z = \{1.0; 1.5; 1.5 \text{ or } 1.0\} \]

\[ m_z = m_x \ OR/XOR m_y \]

• 5-bit adder and 3 gates!
FxP vs. FlP: Adders

- **FxP**ₙ
  - *N*-bit Fixed-Point
- **FIT**ₙ(ₑ)
  - *N*-bit Float
  - Exponent *ₑ* bits

- FxP adders are always smaller, faster, less energy

28nm FDSOI technology, Catapult (HLS), Design Compiler, PrimeTime
FxP vs. FlP: Multipliers

- $\text{FxP}_N$
  - Fixed-Point
  - $N$ bits

- $\text{FlP}_N(E)$
  - Floating-Point
  - $N$ bits
  - Exponent $E$ bits

- FlP multipliers are smaller, faster, but consume more energy

28nm FDSOI technology, Catapult (HLS), Design Compiler, PrimeTime
Custom Floating-Point

- Difference in cost/energy between float/fixed is smaller for low-precision operators
  - e.g., 8-bit float is still effective for K-means clustering [SiPS’17]

- Slower increase of errors for floating-point

Approximate K-Means Clustering

Reference: double

Floating-Point: ct_float₈
5-bit exponent
3-bit mantissa
Custom Floating-Point

- **ct_float**: a Custom Floating-Point C++ Library
  - Synthesizable (with HLS) library
  - Templated C++ class
    - `ct_float<e,m,r>`
      - Exponent width `e` (int)
      - Mantissa width `m` (int)
      - Rounding method `r`
      - Bias `b`
  
- Many possible design points
  - latency constraints, rounding modes, etc.

https://gitlab.inria.fr/sentieys/ctfloat
Approximate DNNs

- **Float**
  - half-precision
  - Bfloat16
- **Fixed-point**
  - INT8
- **Block floating-point**
- **BNN/TNN**
Approximate DNNs: Low-Precision

- Not only \textbf{Weights}, but also \textbf{Activations}, Per-Layer Quantization, etc.

4-bit activations and 10-bit weights keeps accuracy near (98.4%) 32-bit float reference.
What is still difficult: learning

• Learning: gradient descent and backpropagation

\[ w_{ij}^t = w_{ij}^{t-1} - \alpha \frac{\partial \ell}{\partial w_{ij}^{t-1}} \]

• This is very expensive to compute, even in HW
  • Approximating and accelerating learning is much more difficult
Mixed-Precision Training

2. Make an FP16 copy and forward/backward propagate in FP16

1. Keep weights in FP32

3. Do weight update in FP32

[NVIDIA, Mixed precision training, 2018]
Low-Precision Training of DNNs

VGG16 training with Cifar-10
Accuracy and Hw Aware Exploration

Approximate Operators Library (AOL) and Accuracy Analytical Model Databases (AAMD) and Performance Models (PM): Area, Power, Execution Time, etc..

Training & Test Databases

Data Conditioning

Modeling → Learning → Optimization → Test → Code Generator

Accuracy Aware Optimizer (AAO)

AAMD PM

AOL

C++

Optimized Code

HLS
Open-Cl

Back-end

AxCNN HW Accelerator

N2D2
Accuracy and Hw Aware Exploration

- Optimization process
  - Determine the number format and word-length for each data
  - Constrained by quality degradation

\[
\begin{align*}
\min (C(w, t)) & \quad \text{s.t.} \\
\lambda(w, t) & \leq \lambda_{obj} \\
\max (\lambda(w, t)) & \quad \text{s.t.} \\
C(w, t) & \leq C_{\text{max}}
\end{align*}
\]
Conclusions

• Most applications tolerate imprecision
• Playing with precision is an effective way to save energy consumption
  – Number representations, low-precision
  – Not only computation, but also memory and transfers
  – Run-time accuracy adaptation would increase energy efficiency even further

• Low-Precision Training and Inference
Open Issues

• Exploring number representations and word-length is a difficult problem for large applications
  – Mainly limited by simulation time to evaluate accuracy
  – Automatizing the choice between (or combining) float and fixed is a challenge
    • Towards an automatic optimizing compiler framework
  – Domain-specific knowledge is a key
• Evaluating cost is also an important (and less studied) issue
  – e.g., #weights alone is not a good metric
  – e.g., unstructured pruning reduces performance
  – Hardware-aware pruning/quantization requires a good cot model