DC x EurIPS Event

Event Context

- **Conference:** EurIPS Copenaghen 2025
- **Date: 4 December**
- **Date:** December 2 - December 7
- **Topic:** FP4 Training/Inference
- **Goal:** Lay down the ground for the following presentation that will be on Quartet

"Yeah I think we can just talk about the Blackwell generation, about the decision of hardware supported chips for FP4 and what that means. Then we can talk about differences with AMD v TPUs, finally show some benchmarks around inference and then hand over to training. The general crowd wont be super ML chip people for example like san fran, so like giving them something basic will really help with the second presentation."

Ideas for slides:

The goal of the slides is what we would want to learn about quantization and the interplay between hardware and software.

- Brief intro to Verda.
- Why is quantization so important for AI workloads?
    - Matmul speed for 16 bit 8 bit and 4 bit.
- Different ways to quantize
    - QAT (deepseek did fp8 training) versus PTQ (fine tune an bf16) also GPTQ post training.
- FP vs Int
    - Higher density around 0 vs equally spaced.
- Brief intro for FP
    - Sign + Exponent + Mantissa
    - NVFP4
    - E2M1 vs E1M2 and also scaling 16 x16 blocks
- Rack versus single GPU 925 GB/s All_reduce on NVL72 vs 380 G/Bs.
    - So communication improvements.also.
- Can we train nanochat speedrun on NVL72. (dont worry too much)?

Story for the Slides:

The idea is to tell the story of Low Precision AI first and then move into the details for NVFP4 and Blackwell architecture.

We will start with a timeline starting from the FP32 format (everyone should be familiar with), then move to half-precision (fp16). Mention the advantages of using bf16 over regular fp16,

and how for both it's needed a mixed precision setting to do the training.
After that mention the fp8 introduced for training by DeepSeek
Then move to NVFP4.

- https://developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era/#:~:text=The%20introduction%20of%20NVIDIA%20NVFP4,lower%20costs%20per%20token%20generated.
- https://resources.nvidia.com/en-us-gpu-resources/datasheet?lx=CPwSfP&ncid=no-ncid&search=blackwell
- https://wccftech.com/nvidia-blackwell-ultra-gb300-gpu-fastest-ai-chip-dual-reticle-gpu-over-20k-cores-288-gb-hbm3e/
- https://huggingface.co/blog/RakshitAralimatti/learn-ai-with-me
- https://www.emergentmind.com/topics/nvfp4-quantization
- https://www.emergentmind.com/papers/2509.25149
- https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf
- https://developer.nvidia.com/blog/nvfp4-trains-with-precision-of-16-bit-and-speed-and-efficiency-of-4-bit/?ncid=no-ncid
- https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/

Resources

Quartet Paper

- Paper
- Code
- GPU MODE video
- Nvidia paper on pre training

Background

MX Formats

NVFP4/MX formats / https://arxiv.org/pdf/2310.10537 /
https://fpga.org/category/microscaling-mx-formats/
Microscaling (MX) → data formats that comnbine a per-block scaling factor with narrow floating point and integer types for individual elements.

MX has to balance:

1. HW efficiency
    a. Maximize compute and storage efficiency via reduced bit-width
2. Model accuracy

    a. Minimize the gap in the quality of the results obtained with the baselines (larger data formats such as FP16 or FP32)

3. User friction
    a. Make sure there is a seamless integration within existing frameworks and workloads

2 Microscaling

A basic unit of data in an MX format represents a vector of k numbers and consists of a single *shared scale* X and k scalar *elements* $\{P_i\}_{i=1}^{k}$ (see Figure 1). This unit of data is called an MX block and is defined by the combination of *block size* k, scale data format, and element data format. The two data formats are independent of one another, and all k elements share the same element data format. The layout of an MX block is not prescribed — an implementation may store X contiguously with or separately from the elements.

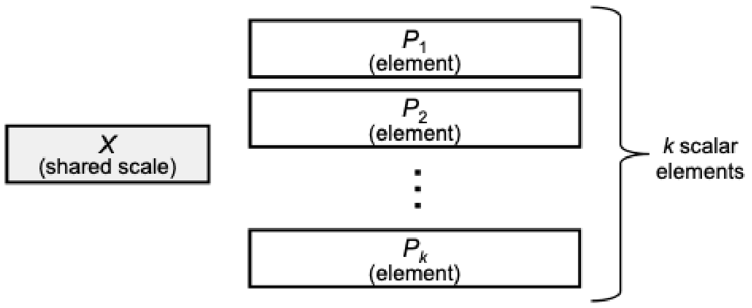

Figure 1: A single block in a Microscaling data format. The block encodes a vector of k numbers, each with value XP_i.

Let $\{v_i\}_{i=1}^{k}$ be the k real numbers represented in an MX block. The value of each number can be inferred as follows:

- If $X = \text{NaN}$, then $v_i = \text{NaN}$ for all i
- If $|XP_i| > Vmax_{Float32}$ then v_i is implementation-defined
- Otherwise, $v_i = XP_i$

where $Vmax_{Float32}$ refers to the largest representable magnitude in IEEE Float32.

The following image is the visualization of the Dot Product of Two MX-compliant Format Vectors introduced in the [OCP MX Specification] Sec. 6.1 and Sec. 6.2:

We have the two MX vectors

$$A = \left\{ X^{(A)},\ \left[P_i^{(A)} \right]_{i=1}^{k} \right\} \text{ and } \quad B = \left\{ X^{(B)},\ \left[P_i^{(B)} \right]_{i=1}^{k} \right\}$$

And we want to compute the dot product:

$$C = \text{Dot}(A, B) = X^{(A)} \cdot X^{(B)} \sum_{i=1}^{k} \left(p_i^{(A)} \cdot p_i^{(B)} \right)$$

C is the output of the dotproduct of two vectors that can be generalised to matrices or arbitrary tensors as:

$$C = \mathrm{DotGeneral}(A, B) = \sum_{j=1}^{n} \mathrm{Dot}(A_j, B_j)$$

The computation diagram for the FWD and BWD pass for a MatMul will be the following:

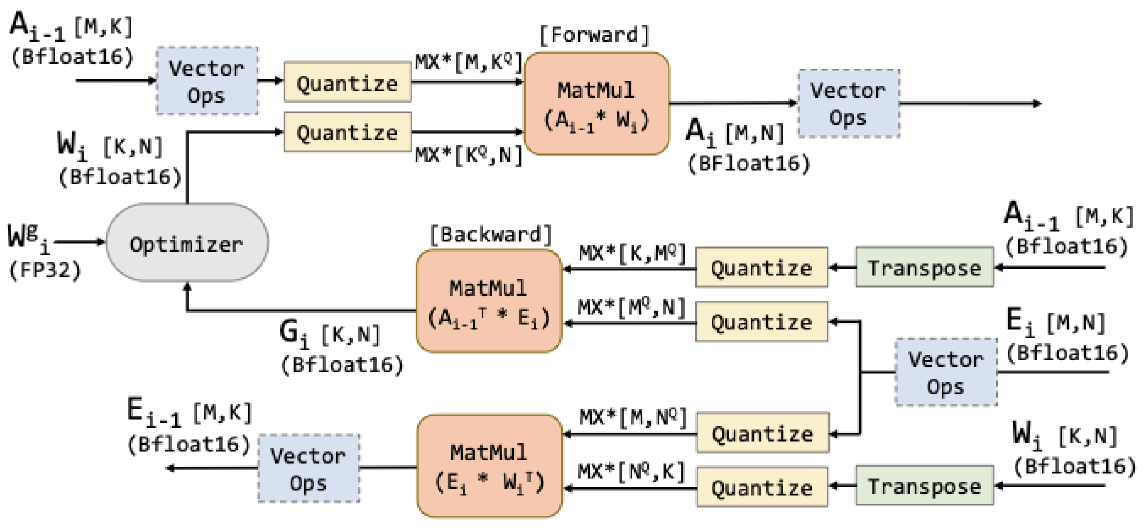

Figure 2: Compute flow with MX formats (denoted as MX*). In the diagram, MatMul includes any dot product operation such as matmul, linear, and convolution. Vector Ops include non-dot product operations like activations, normalization, Softmax, and residual add.

The FWD is the upper part of the figure we get A_{i-1} of shape [M, K] and the weights W_i of shape [K, N]

6.3 Conversion from Vector of Scalar Elements to MX-compliant Format

A mechanism **must** be provided for converting a (k)-length vector $V = [V_i]_{i=1}^{k}$ of scalar elements to an MX-compliant format $\{X, [P_i]_{i=1}^{k}\}$ by producing the block scale X and the elements P_i.
The following semantics should be minimally supported:

---

1. Compute the block scale $X = \dfrac{\max\limits_{V_i \in V} |V_i|_{\text{pow2}}}{X_{\text{max pow2}}}$

Where:

- $\max\limits_{V_i \in V} |V_i|_{\text{pow2}}$ is the **largest power-of-two** $\leq \max|V_i|$
- $X_{\text{max pow2}}$ is the **largest representable power-of-two** in the element data type.

---

2. Compute and quantize each element P_i

$$P_i = \text{Quantize}\left(\frac{V_i}{X}\right)$$

Quantization rules:

- Use the element data type’s representable range.
- Values exceeding the max normal representable number are clamped to the max normal (sign preserved).
- Quantization must support the rounding mode:

$$roundTiesToEven$$

Other rounding modes *may* be supported.

Blackwell Arch

Blackwell Technical Brief: https://resources.nvidia.com/en-us-blackwell-architecture

https://developer.nvidia.com/blo`g/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/

Quantization

- GPU MODE quantization