MI300X FP8/OCP FP8/MXFP Open Standard Design and Considerations to SGLang

Henry HAI X
11/02/2024

AMD
together we advance_

Scope & new ML data types with builtin Matrix Core support in HW

Not just the storage type

- Newer HW offers Matrix Core support for these new formats: **FP8**, MXFP6/4
- They will be used as primary data types in ML stack, and framework like float16 in Apex/torch.amp

Not just for Wo (weight only)

- Older HW due to lack of Matrix Core support, computation will be on higher precision, they can be used as storage type, as:
  $Storage\ Type\ (lower\ precision) \rightarrow higher\ precision\ (Matrix\ Core) \rightarrow GEMM/V\ Compute\ \rightarrow Storage\ Type\ (lower\ precision)$
- As newer HW has built in their support in Matrix Core support, utilizing faster native computation with them is preferred, as:
  $Tensor\ Type\ (higher\ precision) \rightarrow lower\ precision\ (Matrix\ Core) \rightarrow GEMM/V\ Compute\ \rightarrow Tensor\ Type\ (higher\ precision)$
- Matrix Core in HW is symmetric in design most cases, so not only weight but also activation (Xs) need to be quantized above
  $\boldsymbol{annotation}: \rightarrow dequant/upcast; \rightarrow quant/downcast$

Focus type of the talk: FP8 E4M3 (Inference), other examples

- INT8 – similar TFLOPS in HW as FP8, but inferior to FP8 in dynamic range and worse linear resolution (float: dense@near-0).
- INT4 – storage type still (used for **wo**)

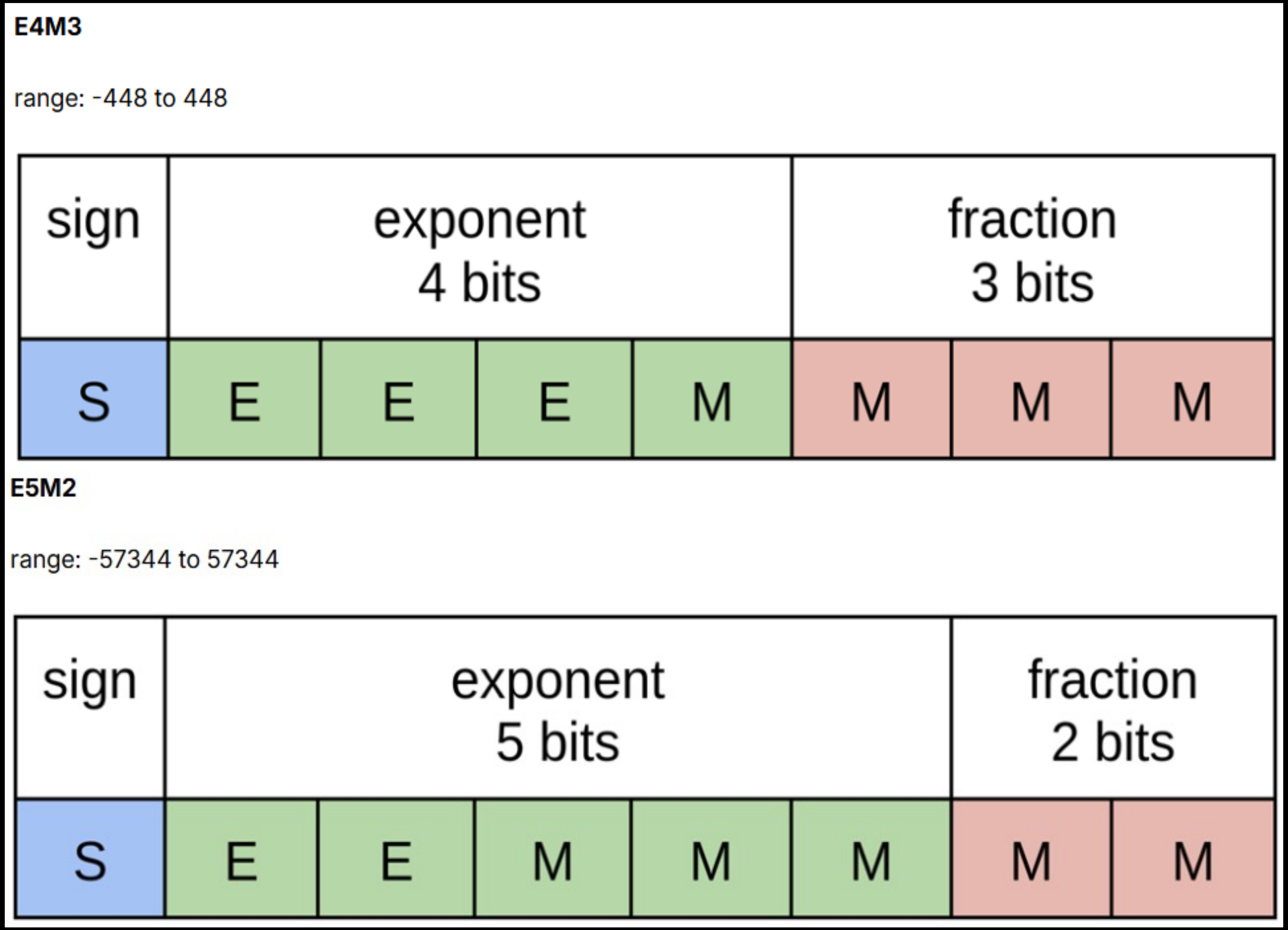
Scale or not, dynamic or not

- Scale always: to adapt different numeric domain, compress or expand to lower precision, reverse to higher precision domain
- Dynamic scale: if scaling factor compute isn't slow; smaller granularity, e.g. per block (MXFP). Use (typical): loss sensitive, training
- Static scale: if scaling factor compute is costly; larger granularity, e.g. per tensor. Computed during PTQ, training. Use: inference

Types defined (LLVM/torch/triton/ONNX[https://onnx.ai/onnx/technical/float8.html])

- LLVM/MLIR: Float8E4M3FNType/ Float8E5M2Type, Float8E4M3FNUZType/Float8E5M2FNUZType
- Torch: torch.float8_e4m3fn/torch.float8_e5m2, torch.float8_e4m3fnuz/torch.float8_e5m2fnuz
- Triton: float8e4nv(fp8e4nv)/float8e5(fp8e5), float8e4b8(fp8e4b8)/float8e5b16(fp8e5b16)

Formats and Use cases (other than storage type)



In Fwd. (Inference) for Weight/Activation

- Inference: weight can be pre-quantized, activation to be quantized at real time

***Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks**, Xiao Sun, Jungwook Choi, et. al. NeurIPS 2019*

Note – newer HW supports Matrix FMA Instructions with either e4m3 or e5m2 as operands, e.g. MI300X:
V_MFMA_F32_{*}_BF8_BF8
V_MFMA_F32_{*}_BF8_FP8
V_MFMA_F32_{*}_FP8_BF8
V_MFMA_F32_{*}_FP8_FP8

In Bwd. (training) for Gradient

- E5M2 only used for training – gradient is more range, but less precision sensitive.
- E5M2 used as storage type – not the mention here, but scaling solution still preferred to preserve sub-normal (denormal) numbers from higher precision

MI300X ISA: https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/instruction-set-architectures/amd-instinct-mi300-cdna3-instruction-set-architecture.pdf

Different HW & Numeric

1-4-3 (e4m3) byte codes OCP/Nvidia vs. MI300X

	Sign	Exponent				Mant			Nvidia(IEEE Mode Bias = 7)		AMD(NANOO Mode Bias = 8)	
									Val	Val in Decimal	Val	Val in Decimal
Positive Values	0	0	0	0	0	0	0	0	+0	+0	0	0
	0	0	0	0	0	0	0	1	0.001x2^-6	0.001953125	0.001x2^-7	0.000976563
	0	0	0	0	0	0	1	0	0.010x2^-6	0.00390625	0.010x2^-7	0.001953125
	0	....	....	....	....	....	....	....	....	....	....	....
	0	1	1	1	0	1	1	0	1.110x2^7	224	1.110x2^6	112
	0	1	1	1	0	1	1	1	1.111x2^7	240	1.111x2^6	120
	0	1	1	1	1	0	0	0	1.000x2^8	256	1.000x2^7	128
	0	1	1	1	1	0	0	1	1.001x2^8	288	1.001x2^7	144
	0	1	1	1	1	0	1	0	1.010x2^8	320	1.010x2^7	160
	0	1	1	1	1	0	1	1	1.011x2^8	352	1.011x2^7	176
	0	1	1	1	1	1	0	0	1.100x2^8	384	1.100x2^7	192
	0	1	1	1	1	1	0	1	1.101x2^8	416	1.101x2^7	208
	0	1	1	1	1	1	1	0	1.110x2^8	448	1.110x2^7	224
	0	1	1	1	1	1	1	1	NAN	NAN	1.111x2^7	240
Negative Values	1	0	0	0	0	0	0	0	-0	-0	INF/NAN	INF/NAN
	1	0	0	0	0	0	0	1	0.001x2^-6	0.001953125	0.001x2^-7	0.000976563
	1	0	0	0	0	0	1	0	0.010x2^-6	0.00390625	0.010x2^-7	0.001953125
	1	....	....	....	....	....	....	....	....	....	....	....
	1	1	1	1	0	1	1	0	1.110x2^7	-224	1.110x2^6	112
	1	1	1	1	0	1	1	1	1.111x2^7	-240	1.111x2^6	-120
	1	1	1	1	1	0	0	0	1.000x2^8	256	1.000x2^7	128
	1	1	1	1	1	0	0	1	1.001x2^8	288	1.001x2^7	144
	1	1	1	1	1	0	1	0	1.010x2^8	320	1.010x2^7	160
	1	1	1	1	1	0	1	1	1.011x2^8	352	1.011x2^7	176
	1	1	1	1	1	1	0	0	1.100x2^8	384	1.100x2^7	192
	1	1	1	1	1	1	0	1	1.101x2^8	416	1.101x2^7	208
	1	1	1	1	1	1	1	0	1.110x2^8	448	1.110x2^7	224
	1	1	1	1	1	1	1	1	NAN	NAN	1.111x2^7	240

AMD
together we advance_

Interop Solutions (Inference)

paste a few pseudo code

Assumption for Interoperability

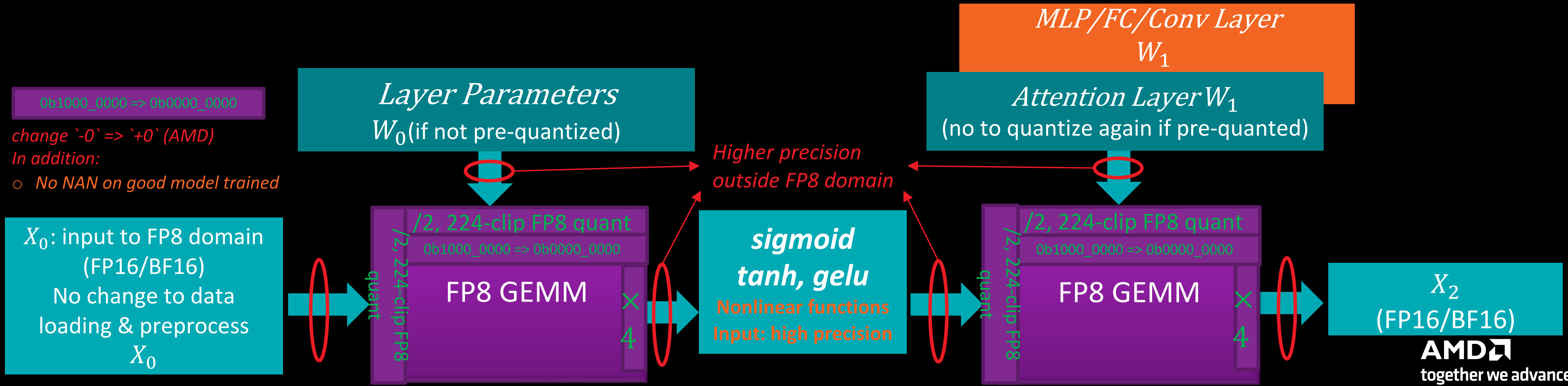
- Network weight is either pre-quantized (PTQ) or QATed into OCP/NV FP8 E4M3 format, and associated scaling factors are pre-computed based on this format in case static scaling
- No NaN/Inf in trained and/or quantized weights to inference

Solution

- Not to translate numbers between 2 different FP8 numericals
- But to adjust scaling factors by 2, and handle -0 (green box)
- HWs run in native FP8 format built in MFMA for performance

									Nvidia(IEEE Mode Bias = 7)		AMD(NANOO Mode Bias = 8)	
									Val	Val in Decimal	Val	Val in Decimal
	Sign	Exponent				Mant						
Positive Values	0	0	0	0	0	0	0	0	+0	+0	0	0
	0	0	0	0	0	0	0	1	0.001x2^-6	0.001953125	0.001x2^-7	0.000976563
	0	0	0	0	0	0	1	0	0.010x2^-6	0.00390625	0.010x2^-7	0.001953125
	0	....	....	....	....	....	....	....	.....	.....	.....	.....
	0	1	1	1	0	1	1	0	1.110x2^7	224	1.110x2^6	112
	0	1	1	1	0	1	1	1	1.111x2^7	240	1.111x2^6	120
	0	1	1	1	1	0	0	0	1.000x2^8	256	1.000x2^7	128
	0	1	1	1	1	0	0	1	1.001x2^8	288	1.001x2^7	144
	0	1	1	1	1	0	1	0	1.010x2^8	320	1.010x2^7	160
	0	1	1	1	1	0	1	1	1.011x2^8	352	1.011x2^7	176
	0	1	1	1	1	1	0	0	1.100x2^8	384	1.100x2^7	192
	0	1	1	1	1	1	0	1	1.101x2^8	416	1.101x2^7	208
	0	1	1	1	1	1	1	0	1.110x2^8	448	1.110x2^7	224
	0	1	1	1	1	1	1	1	NAN	NAN	1.111x2^7	240
Negative Values	1	0	0	0	0	0	0	0	-0	-0	INF/NAN	INF/NAN
	1	0	0	0	0	0	0	1	0.001x2^-6	0.001953125	0.001x2^-7	0.000976563
	1	0	0	0	0	0	1	0	0.010x2^-6	0.00390625	0.010x2^-7	0.001953125
	1	....	....	....	....	....	....	....	.....	.....	.....	.....
	1	1	1	1	0	1	1	0	1.110x2^7	-224	1.110x2^6	112
	1	1	1	1	0	1	1	1	1.111x2^7	-240	1.111x2^6	-120
	1	1	1	1	1	0	0	0	1.000x2^8	256	1.000x2^7	128
	1	1	1	1	1	0	0	1	1.001x2^8	288	1.001x2^7	144
	1	1	1	1	1	0	1	0	1.010x2^8	320	1.010x2^7	160
	1	1	1	1	1	0	1	1	1.011x2^8	352	1.011x2^7	176
	1	1	1	1	1	1	0	0	1.100x2^8	384	1.100x2^7	192
	1	1	1	1	1	1	0	1	1.101x2^8	416	1.101x2^7	208
	1	1	1	1	1	1	1	0	1.110x2^8	448	1.110x2^7	224
	1	1	1	1	1	1	1	1	NAN	NAN	1.111x2^7	240

(Cap to 224.0)



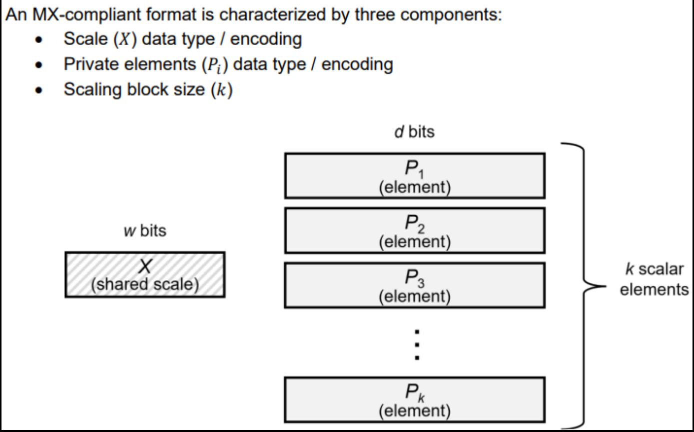
AMD
together we advance_

MXFP6/4 (add a few open domain graphs, msft BFP)

The open proposal and reference

- https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf
- https://arxiv.org/abs/2310.10537
- https://github.com/microsoft/microxcaling
- Backward to Block Floating Point, MSFP: https://www.microsoft.com/en-us/research/blog/a-microsoft-custom-data-type-for-efficient-inference/

Highlight



Format Name	Element Data Type	Element Bits (d)	Scaling Block Size (k)	Scale Data Type	Scale Bits (w)
MXFP8	FP8 (E5M2)	8	32	E8M0	8
	FP8 (E4M3)				
MXFP6	FP6 (E3M2)	6	32	E8M0	8
	FP6 (E2M3)				
MXFP4	FP4 (E2M1)	4	32	E8M0	8
MXINT8	INT8	8	32	E8M0	8

Design points

- New HW to build in support at MFMA level: Blackwell, MI325
- Shared scaling at block level, size of 32 (or multiple)
- Mostly to address training first, will be used for inference
- Other than stationary weights, no need to do pre-quant with pre-compute scaling factors
- Scaling factor (shared scale) is 8-bit E8M0 (not float)
- ML framework level (torch, etc.), key is storage/tensor design for scaling factor grouping, associating and faster update

AMD
together we advance_

Quantizers and Toolkit

Nvidia AMMO (TensorRT Model Optimizer)

- https://github.com/NVIDIA/TensorRT-Model-Optimizer
- https://nvidia.github.io/TensorRT-Model-Optimizer/

AMD Quantizer

- https://quark.docs.amd.com/latest/
- Support all data types (AMMO compatible), including upcoming MXFP, and most algorithms

Open Source

- https://github.com/neuralmagic/AutoFP8
- Many … …

Optimizations

For best performance – collapse quant in prior non-linear elementwise kernel

GEMM kernel write out higher precision most cases, **lower precision** only occasionally (e.g. QKV projections)

- Simply have lower precision GEMM write out low precision output isn't helpful unless memory bound (MFMA/ACC in high precision): $COST_{quant} + COST_{dequant} > MemBW\_SAVING_{low\_precision}$ very likely.
- This is because non-linear op comes after GEMM most cases, and non-linear op works best in higher precision (except special case)
- But if GEMM's output is bound to storage (KV cache) vs. non-linear op, have it writing out low-precision will help, especially if low-precision in memory data is to be used without dequant next (e.g. low-prec. GEMM).

GEMM kernel to consume multiple scaling factors per Matrix (new requirement), and design choices

Approach and Goal to support newer HW and SW design

Support newer low-precision capable HW functions –

- Accelerated conversion
- Accelerated compute

Design adjacent storage interface and extensible API accordingly

- For **scaling factors** management and uses, propose **multi-tiered** solution (in the order of $first \rightarrow last$):
  $\rightarrow default = 1.0\ for\ all$ (1.0 be used if no next two)
  $\rightarrow overload\ to\ value\ read\ from\ checkpoint\ files's\ predefined\ keys\ if\ ANY\ (e.g.\ .kv_{scale}, .k_{scale}, .v_{scale}, etc.)$
  $\rightarrow overload\ to\ value\ read\ from\ scaling\ factor\ files\ (same\ hierachy\ as\ model\ file, with\ .scale\ keys\ only) -$
  $for\ easy\ loading\ and\ update\ from\ recalibration\ (e.g.\ finetune\ over\ custom\ dataset)$
- Provide extraction scripts w.r.t. all major quantizers output, and update scripts from recalibrations (checkpoint no change)

Support major quantizer toolkits, promote open research and standard (OCP)

Make newer algorithm easier to integrate, and new features easy to add

- torch.ao, etc.

Foster newer ideas, but not enforce (optional approach)

- Not to make default to not widely used or not widely advantageous approaches
- New implementation can be set optional at completion