CoNGA

Nov 14th, 2026

OAK RIDGE National Laboratory

Nov 14th, 2026, Chicago @ CoNGA

SQUID DGEMM

Symmetric block Quantized DGEMM for HPC using GPU Tensor cores — one low-precision GEMM with block scaling on H100 and MI300A.

Narasinga Rao Miniskar,

Valero-Lara Pedro, Piyush Sao,
Keita Teranishi, Jeffrey S Vetter, Sudip Seal

U.S. DEPARTMENT OF ENERGY

ORNL IS MANAGED BY UT-BATTELLE LLC
FOR THE US DEPARTMENT OF ENERGY



About Speaker: Narasinga Rao Miniskar

Hardware–software co-design across the computing stack

PH.D. · EMBEDDED SYSTEMS

imec · KU Leuven

Doctoral research at IMEC and KU Leuven, Belgium

INDUSTRY RESEARCH · SAMSUNG (8.5 years)

Samsung R&D India · SAIT Korea

- Samsung Reconfigurable Processor
- Samsung Neural Processor

NOW (7+ years) OAK RIDGE National Laboratory

Group Lead, Architectures and Performance Group (APG)

APG Group Research

Cross-cutting research · HW/SW co-design

Hardware — Co-design — Software

High-level programming models and Algorithms
Julia and Python for productive, portable HPC code, Math algos

Intelligent Heterogeneous Runtime · IRIS
Scheduling across CPUs, GPUs, FPGAs, QPUs

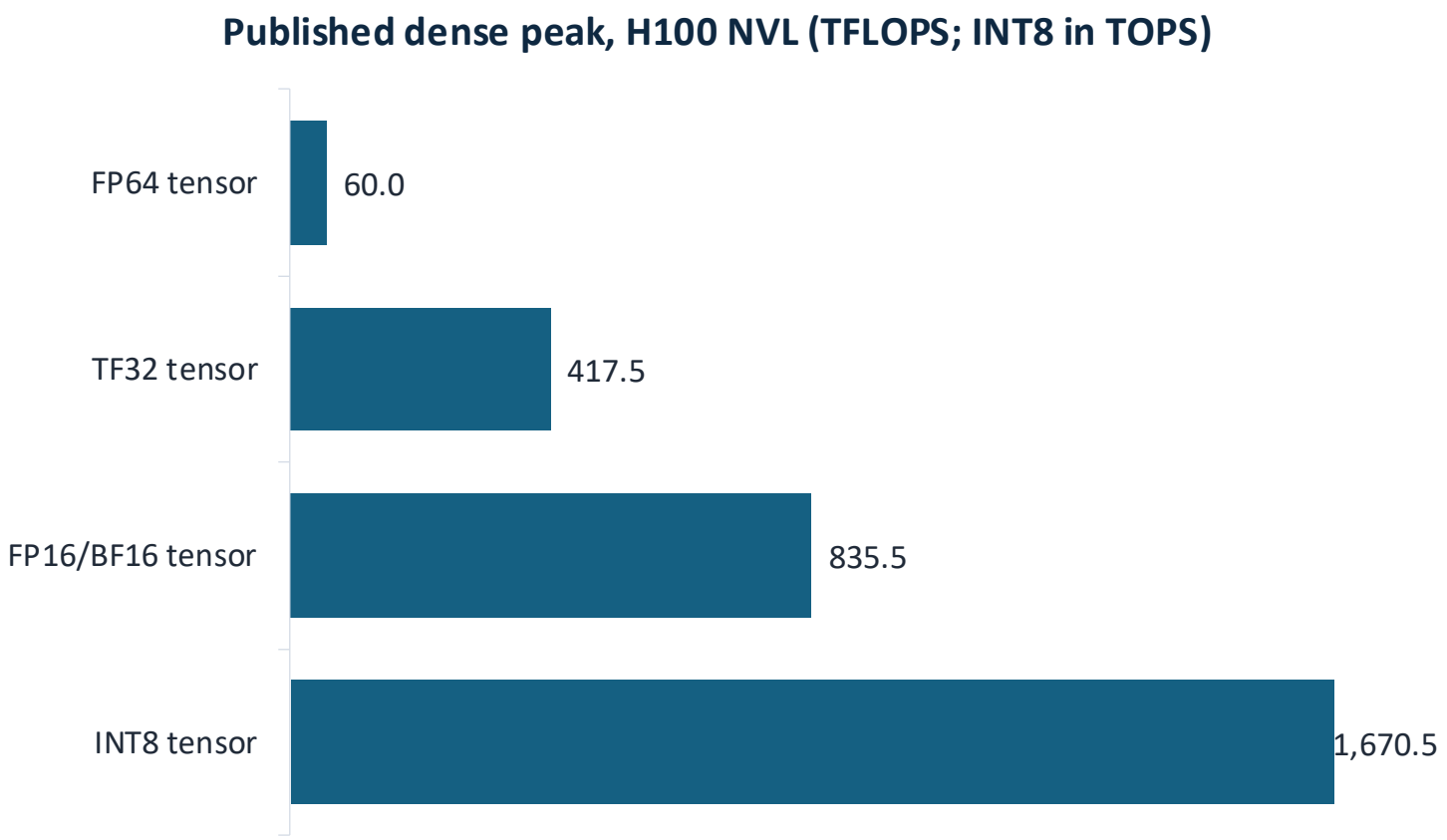
Architecture research
Hardware & Performance modeling and **Mixed precision Exploration**

Circuits & Device modeling
HW Modeling and simulation of Analog (ReRAM)/CMOS accelerators

Synthesis with EDA tools
Synthesis and evaluation with EDA flows

Computing Paradigms: HPC . Neuromorphic . Quantum

Why mixed-precision hardware matters



28x

INT8 over FP64 tensor-core peak

FP64 throughput grows slowly while low-precision capacity keeps growing.

Low-precision products accumulate in wider formats: INT8 into INT32, FP16/BF16/TF32 into FP32.

New class of algorithms are required which can guarantee accuracy for HPC, higher performance and lower energy consumption

SQUID at a glance

CONTEXT

Tensor cores favor low precision

On H100 NVL, INT8 tensor cores peak at about 28x the FP64 tensor-core rate.

DGEMM drives solvers and electronic-structure codes.

PROBLEM

FP64 results, FP64 cost

Exact kernels ignore the low-precision capacity.

ML quantizers accept percent-level error that scientific codes cannot.

STATE OF THE ART

Two extremes

- One extreme: Native DGEMM; Ozaki-I (45 INT8 GEMMs) and Ozaki-II (CRT) recover near-FP64 accuracy
- Other extreme: Machine-learning quantizers tolerate percent-level error that scientific workloads cannot accept

PROPOSAL

SQUID variants using low precision Tensor cores

- Scale row blocks of A and column blocks of B, run one low-precision GEMM, and re-quantize each output tile.
- Accuracy, Time, Energy, Memory study under wide dynamic range
- Experiments on Nvidia H100 NVL and AMD MI300A
- Forward error ratios of 3.85×10^{-6} and 3.08×10^{-5}

State of the art methods

Method	Idea	Cost or limit
cuBLAS DGEMM	Native FP64, can use FP64 tensor cores	50.6 TFLOP/s at N=8192 on H100
Mixed-precision solvers	Low-precision inner steps plus iterative refinement	Solver-level, not an individual GEMM
TF32 / TF64 emulation	Several reduced-precision products rebuild higher precision	Multiple GEMMs per product
ML quantizers	Runtime scaling for inference or training (LLM.int8 and others)	Percent-level error is tolerated
cuBLAS fixed-mantissa emulation	Fixed-point FP64 emulation with shared scales (cuBLAS 13.1)	Accuracy-performance control; not measured here
Ozaki-I and Ozaki-II	Many INT8 GEMMs give near-FP64 accuracy	Next slide

Ozaki-I and Ozaki-II: High Precision Matrix Multiplication using Low Precision Hardware

[Ozaki 2024, IJHPCA]

[Ozaki II 2026, IJHPCA]

OZAKI I • Floating-point splitting

1 Decompose	$A \approx \Sigma A_i$, $B \approx \Sigma B_j$ Exponent-aware floating-point slices
2 Multiply	Compute $A_i \times B_j$ using low-precision GEMM
3 Accumulate	Sum selected partial products with controlled rounding
4 Recover	Approximate high-precision C = AB

https://github.com/enp1s0/ozIMMU

Core arithmetic: floating-point tensor/GEMM

OZAKI II • Integer / fixed-point splitting

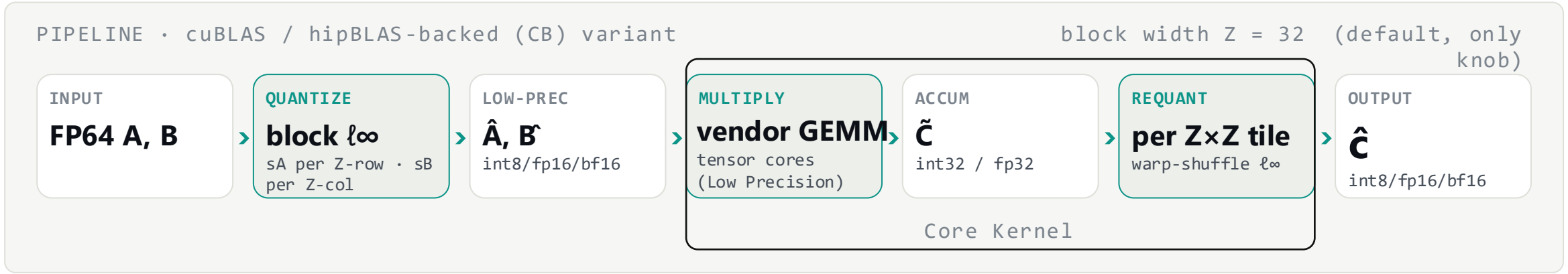
1 Scale & split	Convert inputs to integer slices with shared scaling factors
2 Integer GEMM	Compute slice products using INT8 or other integer arithmetic
3 Reconstruct	Combine exact integer partial sums; CRT may be used for modular variants
4 Rescale	Apply inverse scaling to recover C ≈ AB

https://github.com/RIKEN-RCCS/GEMMul8

Core arithmetic: integer tensor/GEMM

Key distinction: Ozaki I exploits floating-point slices; Ozaki II exploits integer slices and reconstruction.

SQUID pipeline



Runtime, not calibrated

Each Z-row block of A and Z-column block of B gets its own ℓ∞ scale, computed on the fly — fit for matrices that change every solver iteration.

Vendor tensor cores

The core multiply is a single cuBLAS / hipBLAS call in the target format — so the design ports NVIDIA→AMD unchanged, no custom kernel.

Per-tile, first of its kind

A lightweight pass rescales each Z×Z output tile. To our knowledge the first approximate GPU DGEMM to combine block-symmetric quant with per-tile requantize.

VARIANTS

- SQ_I8_CB
- SQ_FP16_CB
- SQ_BF16_CB
- SQ_TF32_CB
- SQ_FP32_CB
- cuBLAS DGEMM
- Ozaki

Quantized data enables to transfer less number of bits and optimizes communication cost

Variant: SQ_I8_CB (signed INT8)

Aspect	SQ_I8_CB
Quantized range	[-127, 127], -128 excluded for symmetry
Core GEMM	cublasGemmEx, INT8 inputs, INT32 output
Operand layout	K contiguous on both inputs (TN)
Requantization	k_requantize_i32 rescales in FP64, one thread block per tile
Thread blocks	(N/Z)^2 for N = M = K

Highest measured throughput of all variants, with larger error than the floating-point paths.

Variants: floating-point CB paths

SQ_FP16_CB

FP16 tensor cores

10-bit mantissa, 5-bit exponent.
FP16 to FP32 tensor cores; tiles stored in FP16.

SQ_BF16_CB

BF16 tensor cores

7-bit mantissa, 8-bit exponent (FP32 range, coarser mantissa).
Tiles stored in BF16.

SQ_TF32_CB

TF32 tensor cores

10-bit mantissa, 8-bit exponent.
Tiles stored in FP32. NVIDIA only.

SQ_FP32_CB

FP32 CUDA cores

Accuracy reference for tile normalization without reduced-precision products.

Experimental setup and metrics

HARDWARE

H100 NVL and MI300A

H100: CUDA 13.1, cuBLAS, clock locked at 1785 MHz.

MI300A: ROCm 7.2, hipBLAS.

THROUGHPUT

Effective TFLOP/s

P_eff = 2MNK / (1e12 · t), the same numerator for every method regardless of internal arithmetic.

ACCURACY

RMS forward error

eps = ||C_approx - C_ref||_F / ||A||B| ||_F against an FP64 reference.

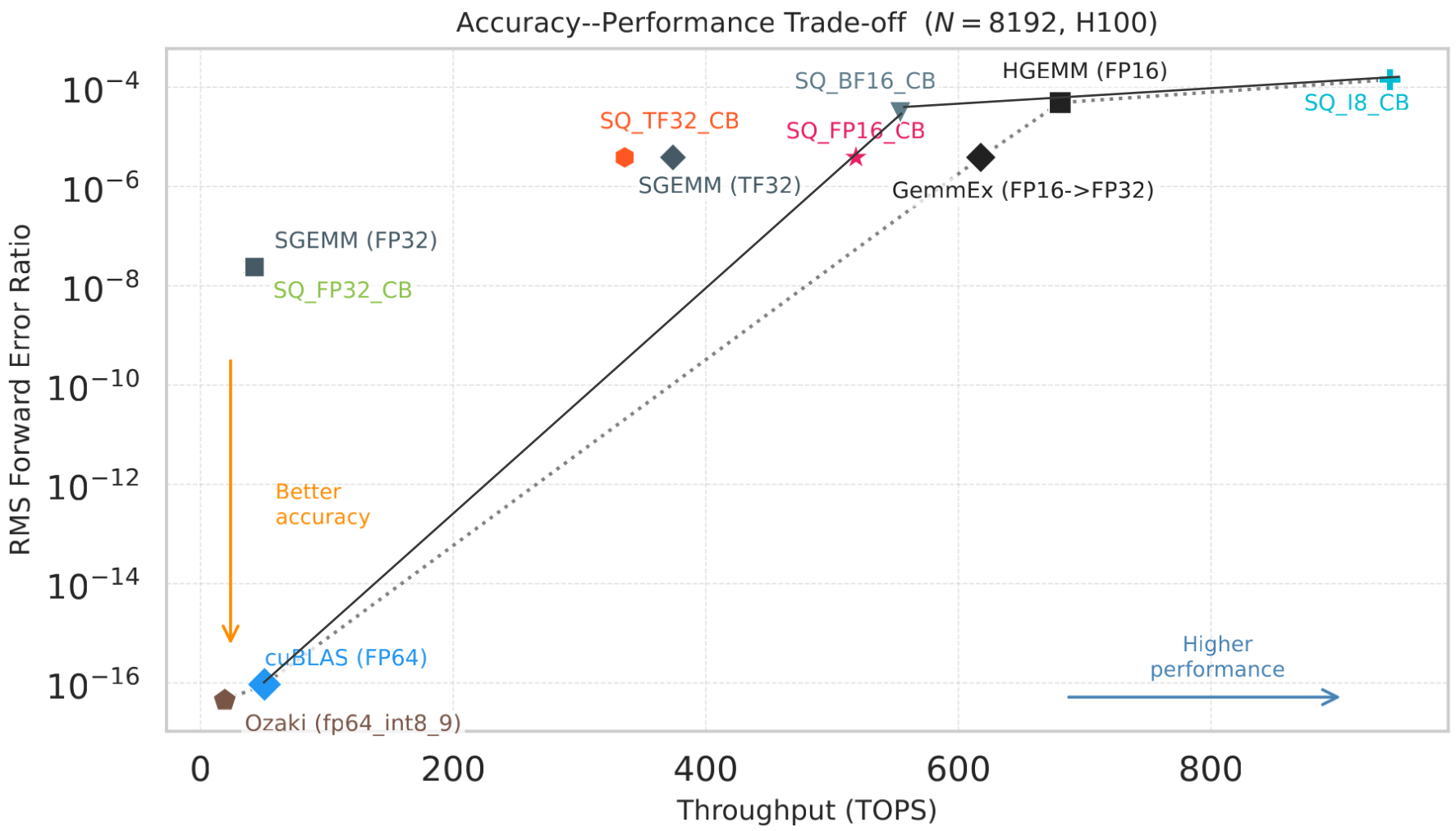
BOUNDARY

Timed region

Timed: core GEMM plus output requantization. Excluded: input quantization, transfers, INT8 transpose.

Energy from NVML or ROCm SMI sampled every 5 ms. Default Z = 32.

Throughput and accuracy at N = 8192



SPEEDUP

10.9x BF16

554.0 vs 50.6 TFLOP/s DGEMM

FASTEST

INT8: 941.7

TFLOP/s, error 1.42e-4

FP16 / BF16 CORE

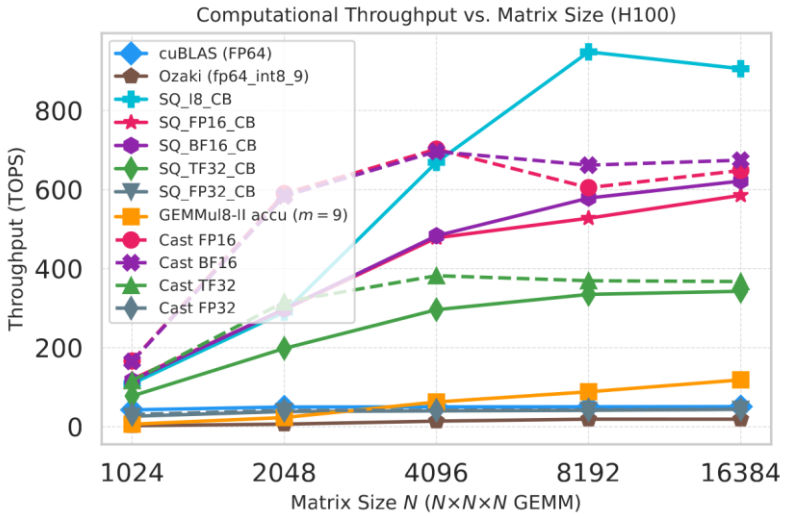
3.85e-6 / 3.08e-5

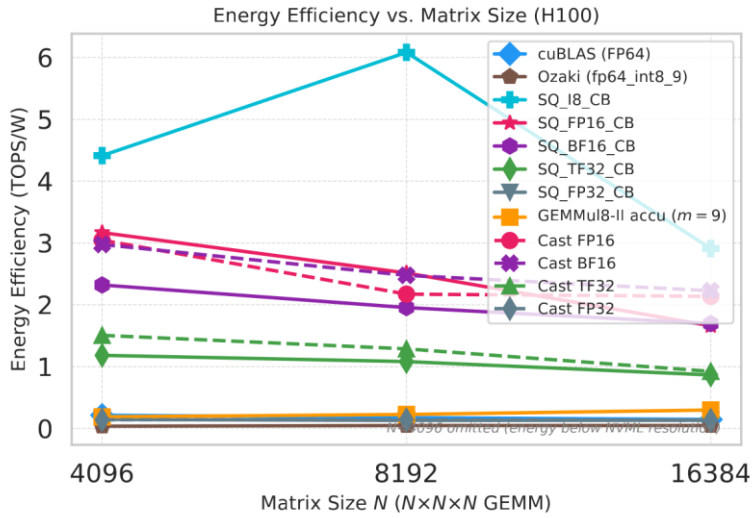
error before requantization

caveat **SQUID variants provide near single-precision accuracy for DGEMM**

payoff **It buys robustness under wide dynamic range** — where naive casts fail

Scaling with matrix size and call time





Method (N = 8192)	Call time (ms)
Native cuBLAS DGEMM	21.721
Ozaki-I (k=9)	56.953
SQUID INT8	1.159
SQUID FP16	2.111
SQUID BF16	2.000
SQUID TF32	3.278
SQUID FP32	26.346

INT8 PEAK

948.3 TFLOP/s

18.7x cuBLAS at N=8192

N = 16384

BF16 621.6, FP16 585.4

12.2x and 11.5x cuBLAS

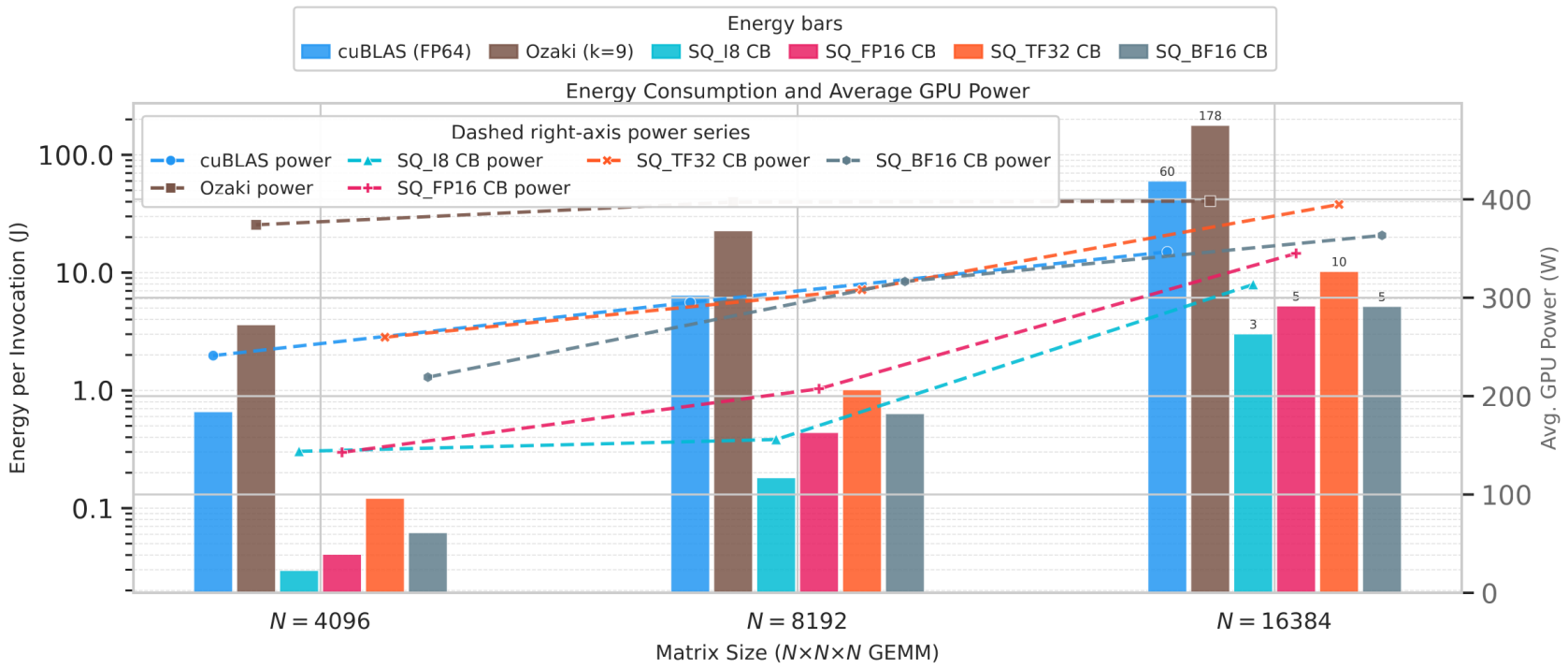
BASELINES

DGEMM 42.8 to 50.9

Ozaki-I 2.85 to 19.22 TFLOP/s

OAK RIDGE National Laboratory

Energy per invocation



Shortest calls (about 1.2 ms) are below the 5 ms sampling interval, so per-call energy is an estimate. Input preparation is excluded.

Fast paths finish sooner and draw less power — energy compounds the win.

INT8

1/35.5 of DGEMM

0.18 J vs 6.41 J per call

FP16

1/14.7 of DGEMM

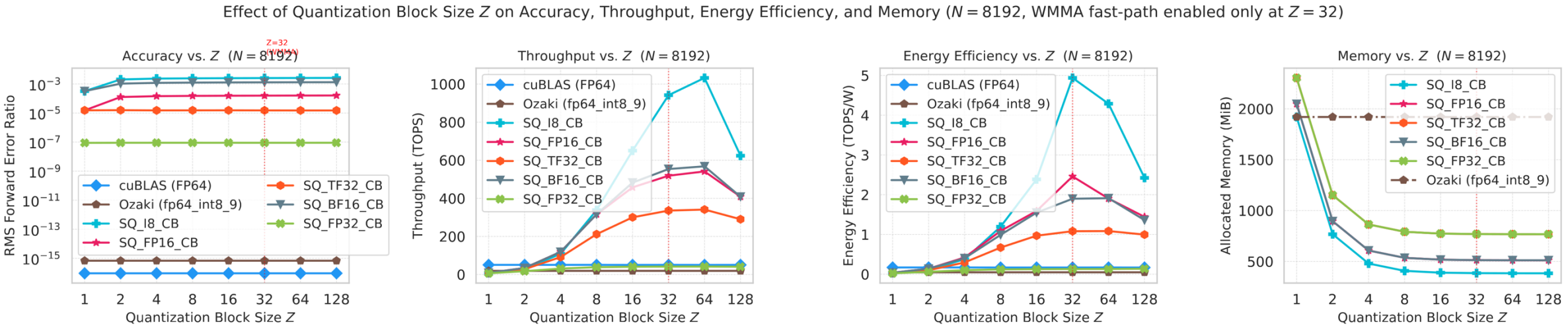
0.44 J per call

BF16

1/10.1 of DGEMM

0.63 J per call

Effect of block width Z



Variant (TFLOP/s)	Z = 32	Z = 64 (peak)	Z = 128
SQ_I8_CB	941.6	1032.4	623.8
SQ_FP16_CB	518.8	541.4	408.5
SQ_BF16_CB	553.9	568.2	408.5

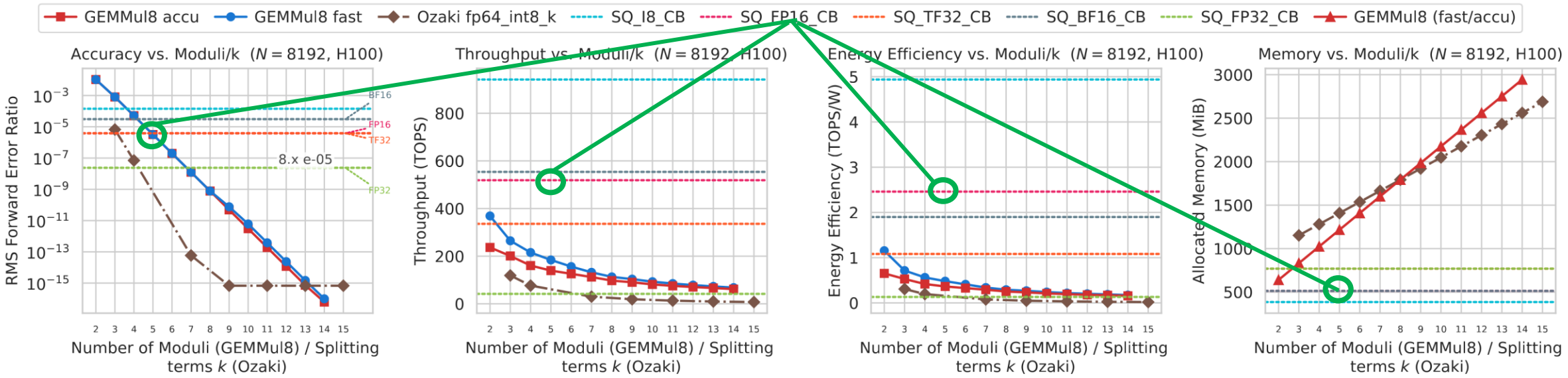
DEFAULT

Z = 32

Within about 9% of peak throughput; one quarter of the Z=64 tile buffer.

FP16 error 1.5e-5 to 1.6e-4 (N=512).

Ozaki-I, Ozaki-II and SQUID: cost of accuracy



OZAKI-II, M = 14

61.5 vs 50.6 TFLOP/s

Faster than native DGEMM at the FP64 floor (6.14e-17).

SQUID INT8

941.7 TFLOP/s, 1.42e-4

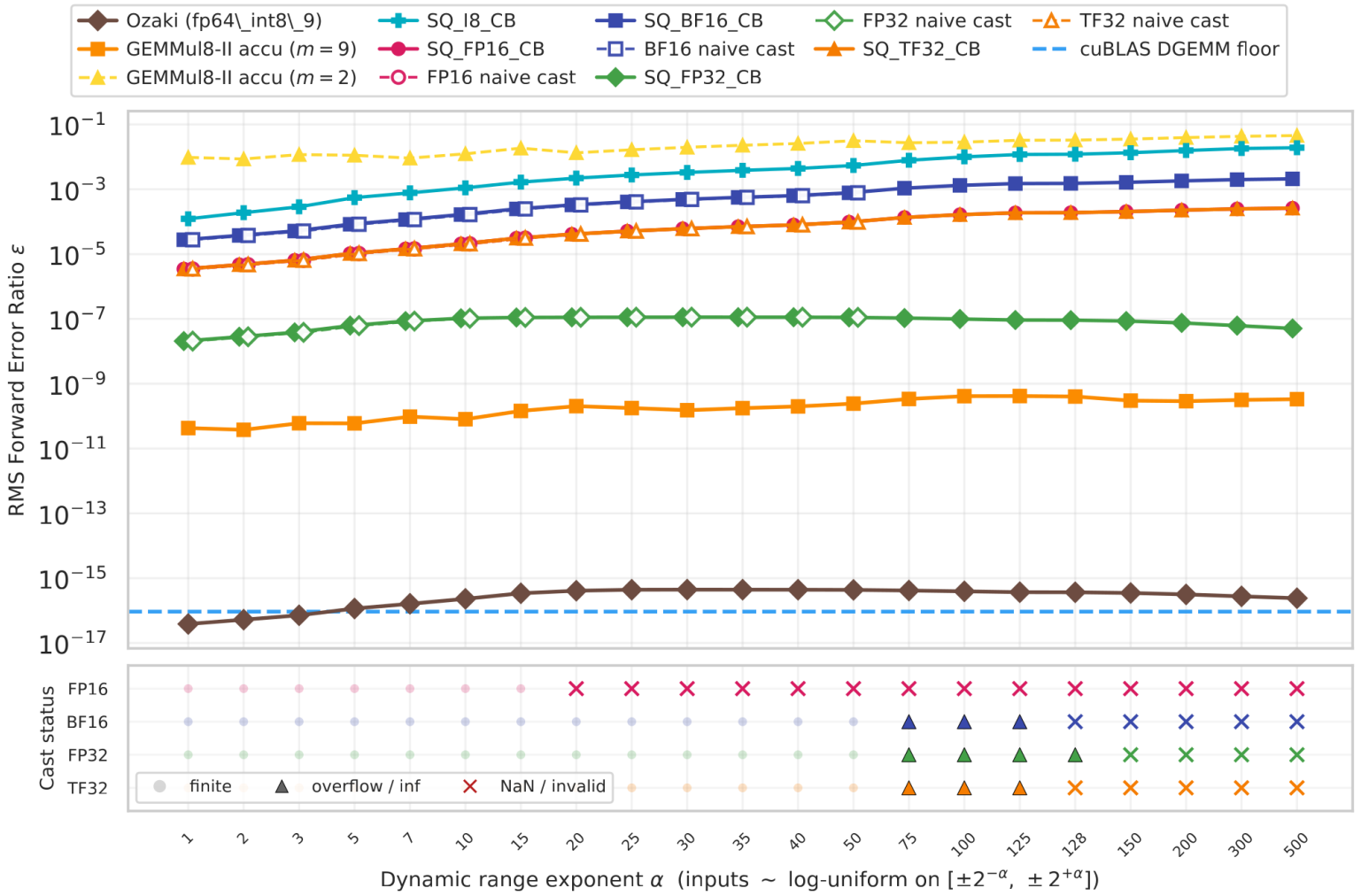
GEMMul8 m=2: 236.6 and 1.05e-2.
m=4: 160.5 and 5.08e-5.

OZAKI-I, K = 9

19.2 TFLOP/s

Error stops falling at 6.81e-16; k=3 reaches 119.0 TFLOP/s.

Accuracy under wide dynamic range



DIRECT CASTS

NaN at alpha = 20

FP16 first; BF16, FP32 and TF32 by alpha = 75.

NORMALIZED CORE

Finite through alpha = 500

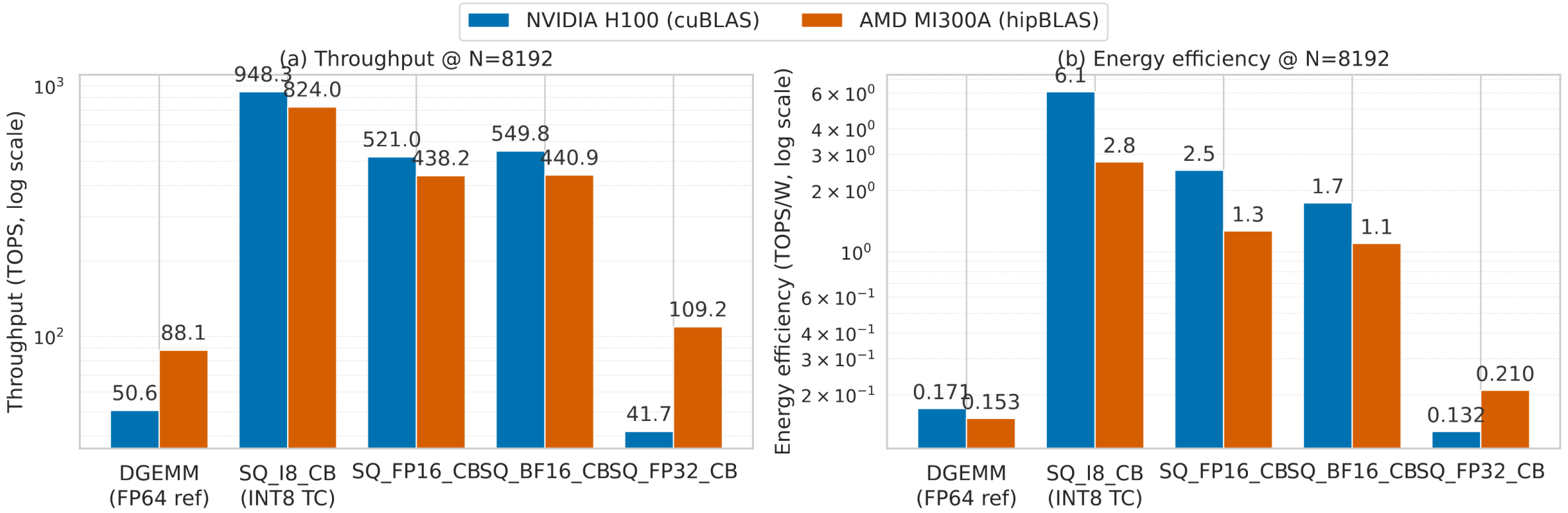
A 2^1000 input range; FP16 error 2.61e-4, BF16 2.09e-3.

SQUID INT8

1.23e-4 to 1.91e-2

About 156x growth; small entries round to zero.

Portability: H100 NVL vs MI300A



SQ_*_CB variants use vendor BLAS (cuBLAS on H100, hipBLAS on MI300A) for the core GEMM plus a requantize pass.
SQ_TF32_CB omitted (TF32 is NVIDIA-specific; NOT_SUPPORTED on gfx942). H100 measured at locked full clock (1785 MHz).

INT8 is 18.7x DGEMM on H100 and 9.4x on MI300A; FP16/BF16 about 10 to 11x and 5x. Power: GPU on H100, APU package on MI300A.

Low/Mixed-Precision Level 1 BLAS: AXPY on NVIDIA H100

Y = αX + Y · 400M–800M elements · FP16/BF16 and multi-word mixed precision

Half precision (FP16/BF16)

- FP16 is ~8×more accurate than BF16 (4.28e-4 vs. 3.42e-3 error).
- Tensor Cores add nothing for AXPY: no extra precision, ~10% slower than custom haxpy, higher energy.
- haxpy on standard cores: >2×faster than FP64, ~15% more energy-efficient than cublasAxpyEx.

Multi-word mixed precision (FP16)

- X ≈ X1 + X2, with X1 = fp16(X), X2 = fp16(X − X1).
- ~6×lower error than pure FP16 (7.07e-5); still above FP32 (2.31e-8).
- GPU offload: up to 1.45×faster, up to 34.2% less energy, half the memory.
- End-to-end time and energy comparable to FP32 cuBLAS.

Forward error	FP32 (Saxpy)	FP16 (haxpy)	BF16 (haxpy)	Mixed FP16 (mxaxpy_3)
Value	2.31e-8	4.28e-4	3.42e-3	7.07e-5

Takeaway: for simple memory-bound Level 1 kernels, standard low-precision cores beat Tensor Cores; multi-word FP16 recovers much of the lost accuracy at FP32-like cost.

* *Pedro Valero-Lara*, Maria Patrou, Piyush Sao, Keita Teranishi, Jeffrey S. Vetter, Narasinga Rao Miniskar, Oscar Hernandez, Sudip Seal, **Low/Mixed-Precision for Level 1 BLAS, a Test Case: AXPY,** 30th Annual IEEE High Performance Extreme Computing (HPEC) Conference, Sep 2026. (In Proceedings).

Conclusion

MIDDLE GROUND

SQUID: One GEMM + requantization

Between exact DGEMM/ Ozaki-I (45 GEMMs) / Ozaki-II and ML based quantization algorithms

ON H100

~10x faster,

~1/10 energy

FP16/BF16 vs DGEMM, excluding input preparation. INT8 is fastest, with larger error.

ROBUST TO RANGE

Finite where casts overflow

Normalized core paths stay finite across a 2^1000 input range.

PORTABLE

cuBLAS and hipBLAS

Same throughput ordering on MI300A; speedups differ.

Future Plans: Providing these variants in Julia ; Exploring it for other BLAS algorithms ; More block patterns can be explored ; Autotuning of variant / precision

Source code: https://code.ornl.gov/hippo/squid

(Will be released soon with opensource license)

Contact: miniskarnr@ornl.gov

This research is sponsored by the FORESEE Initiative through the High-Performance and -Precision Algorithms Using Low-Precision Hardware (HiPPo) project, as part of the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the US Department of Energy (DOE) under contract DE-AC05-00OR22725.

Thank you

Q&As