Joint Matrix: A Unified SYCL Extension for Matrix Hardware Programming (Part 2)

Mehdi Goli

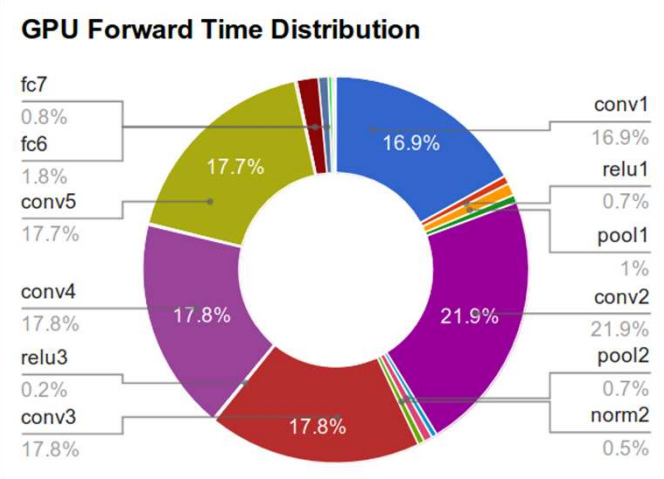
07/06/2023

Outline

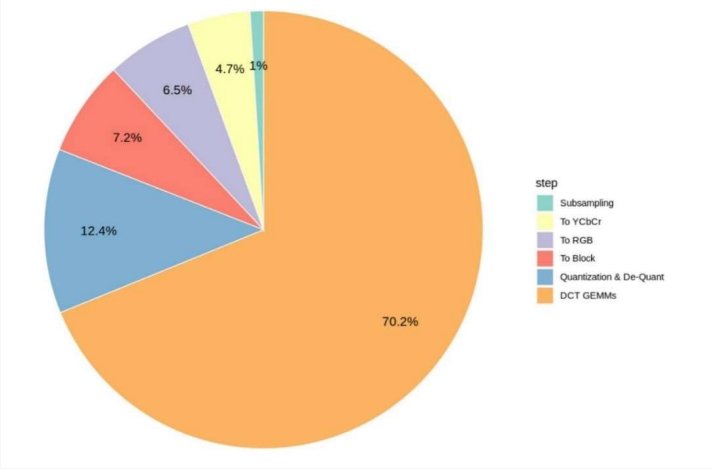
- Importance of GEMM
- Joint Matrix: What and Why?
- Tensor cores on NVIDIA GPU
- Tensor cores offload: WMMA vs MMA
- Performance Analysis
- Summary & Open Questions

Importance of GEMM

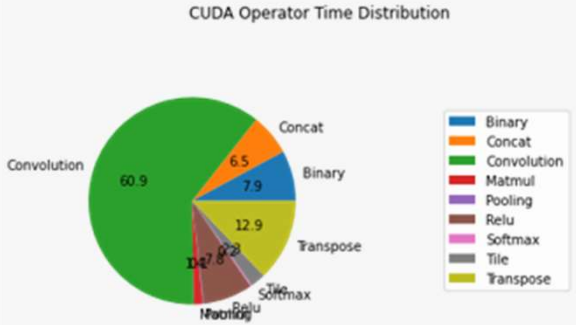
- GEMM forms the backbone of many modern technologies
  - Linear Algebra, Deep Learning, Code Theory, Image/Video Processing, Medical Imaging etc



AlexNet Performance Distribution*



JPEG Conversion



Point Net Performance distribution^

*: picture borrowed from AlexNet Performance Distribution
^: picture borrowed from IWOCL 2023: Accelerating Edge Device with SYCL

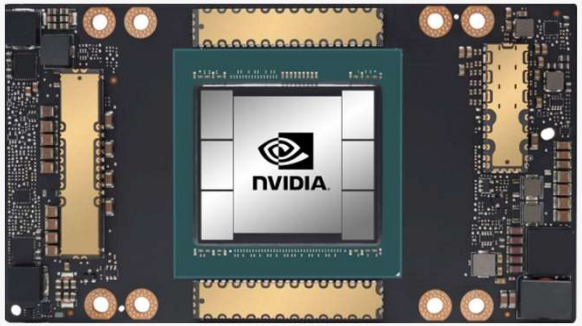
Joint matrix: What and Why?

- What: hardware extension designed to allow developers to make use of the underlying fast fused matrix multiply accumulate units (NVIDIA Tensor cores, Intel AMX and XMX, etc.) through SYCL
- Why: "... to provide a **unified interface** that is **portable** but also benefits from the **maximum performance** these different hardware can offer."

Source: https://github.com/intel/llvm/blob/sycl/sycl/doc/extensions/experimental/sycl_ext_oneapi_matrix/sycl_ext_oneapi_matrix.asciidoc#introduction

Joint Matrix Performance analysis

- Two Nvidia Architectures:
  - Nvidia Jetson Xavier, (SM72 capabilities)
  - Nvidia A100, (SM80 capabilities)



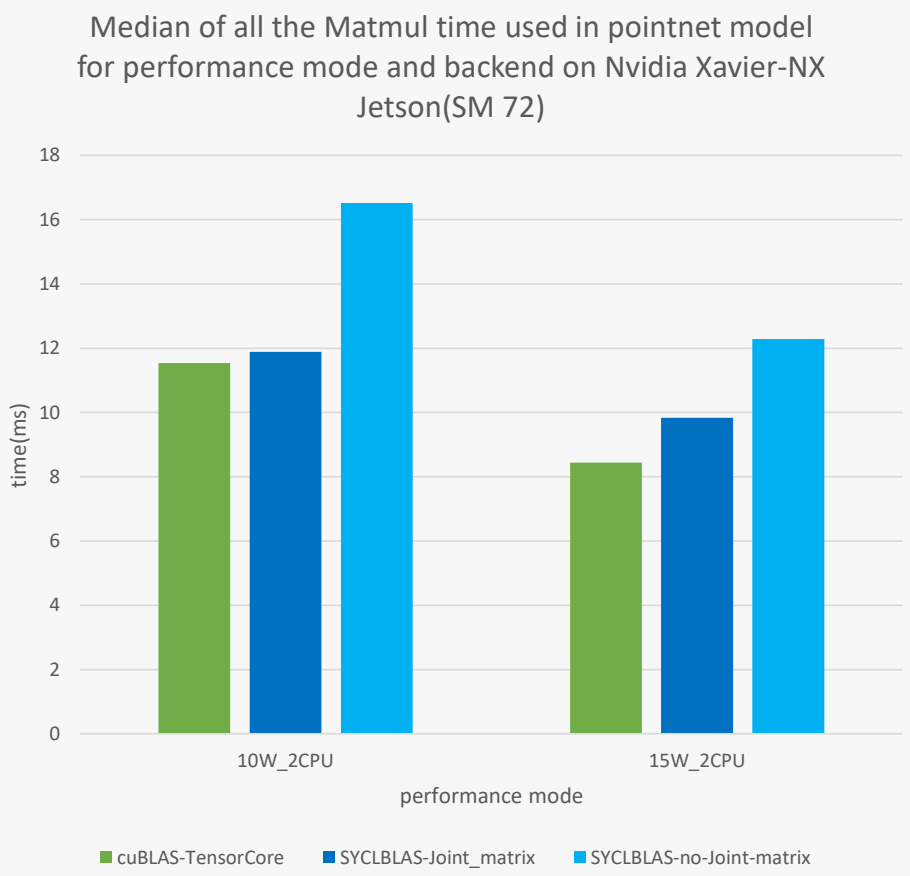
"Nvidia Jetson Xavier NX for Embedded & Edge Systems." *NVIDIA*, https://www.nvidia.com/en-au/autonomous-machines/embedded-systems/jetson-xavier-nx/.
Nvidia A100. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

Benchmark Configurations (Jetson Xavier)

- Configurations Details and Workload Setup:
  - Hardware:
    - 384-core NVIDIA Volta™ GPU with 48 Tensor Cores
    - 6-core NVIDIA Carmel ARM®v8.2 64-bit CPU
    - 8 GB 128-bit LPDDR4x
  - Software:
    - Ubuntu 18.04.5 LTS
    - Kernel Version: 4.9.201-tegra
    - L4T 32.5.1 [JetPack 4.5.1]
    - CUDA 10.2.89
    - CUDNN: 8.0.0.180
    - cuBLAS: 11
    - SYCL Open-Source Clang 17.0.0 : https://github.com/intel/llvm, Commit Hash 4a4702e2e992, compiled with AArch64 as the host target and CUDA as the device target
    - CUDA version 10.2
      - Compiler switches: -fsycl-targets=nxptx64-nvidia-cuda
    - NVIDIA-NVCC V10.2.89
      - Compiler switches: -O3 –gencode –arch=compute_72, code=sm_72
- GEMM operations for the Point-net Benchmarks were run for 1000 iterations on the 15W 2CPU configuration mode unless otherwise specified.
- Results were compared to the CUDA backend of ONNX runtime to the SYCL backend targeting the GPU through DPC++.
- Testing Date: Performance results are based on testing by Codeplay as of April 6th, 2023, and may not reflect all publicly available updates
- Performance varies by use, configuration and other factors.

Performance & Energy comparison SM72

- Joint matrix SYCL backend achieved 97% of the performance of the CUDA backend in 10W mode vs the 86% performance in the 15W mode
- Joint matrix SYCL backend in 10W had a 28% improvement from its baseline
- Joint matrix SYCL backend in 10W outperformed the 15 mode SYCL backend without Joint matrix



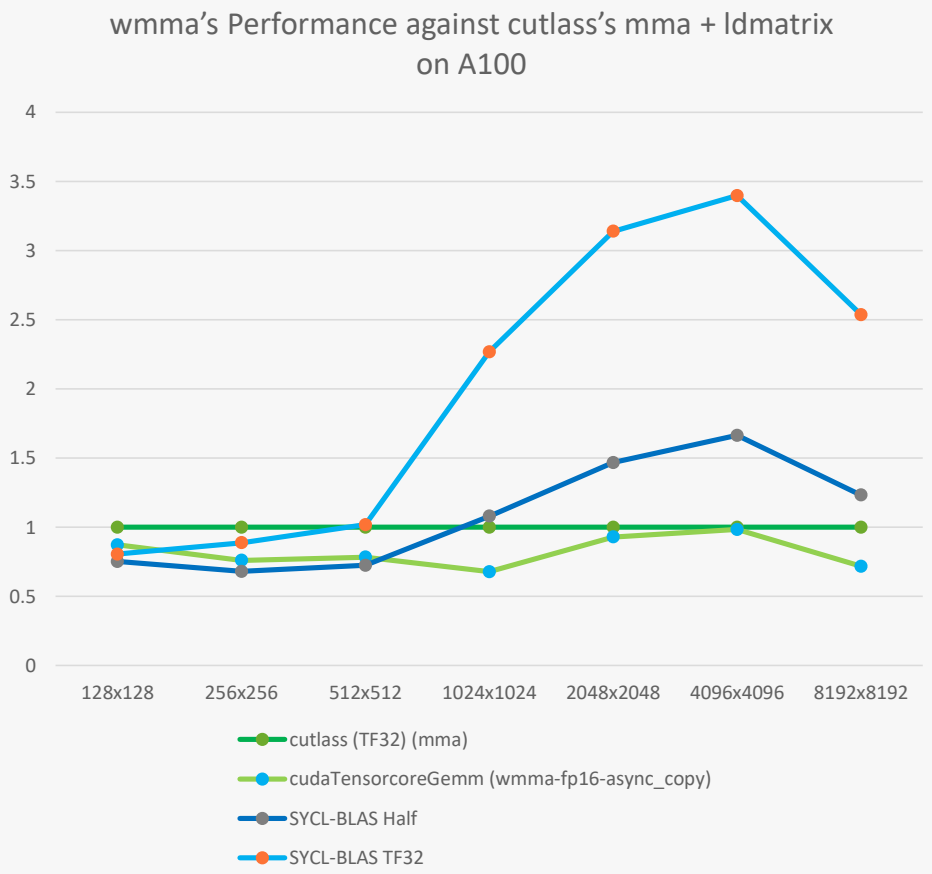
Source: IWOCL 2023: Accelerating Edge Device with SYCL

Benchmark Configurations (A100)

- Configurations Details and Workload Setup:
  - Hardware:
    - 6912-core NVIDIA A100™ GPU with 4 Tensor Cores per SM
    - 80 GB HBM2e
  - Software:
    - Ubuntu 22.04.2 LTS
    - Kernel Version: 5.15.0-69-generic
    - CUDA 11.7.1
    - Cutlass sample: https://github.com/NVIDIA/cutlass/blob/main/examples/14_ampere_tf32_tensorop_gemm/ampere_tf32_tensorop_gemm.cu
    - Cuda sample: https://github.com/NVIDIA/cuda-samples/blob/master/Samples/3_CUDA_Features/cudaTensorCoreGemm/cudaTensorCoreGemm.cu
    - SYCL Open-Source Clang 16.0.0 : https://github.com/intel/llvm, Commit Hash 23b68dbdfec7, compiled with AArch64 as the host target and CUDA as the device target
    - SYCL-BLAS: https://github.com/codeplaysoftware/sycl-blas
    - CUDA version 11.7.1
      - Compiler switches: -fsycl-targets=nxptx64-nvidia-cuda --cuda-gpu-arch=sm_80
    - NVIDIA-NVCC V11.7.1
      - Compiler switches: -O3 –gencode –arch=compute_80, code=sm_80
    - Nsight Compute V11.7.1
- Results were compared for SYCL and CUDA by using joint_matrix implementation in SYCL-BLAS and cuda/cutlass samples available online by setting the **persistence mode = 1** and setting **CLOCKs** to **max (Graphics/SM = 1410 MHz, Memory/Video = 1275 MHz)**
- GEMM operations performance collection schema - using SYCL-BLAS Benchmarks (with googlebench) | using cutlass/cuda github samples
  - Run 10 warmup iterations
  - Run 1000 performance iterations, synchronize after every iteration, and compute average iteration time
- Testing Date: Performance results are based on testing by Codeplay as of February 3rd, 2023, and may not reflect all publicly available updates
- Performance varies by use, configuration and other factors.

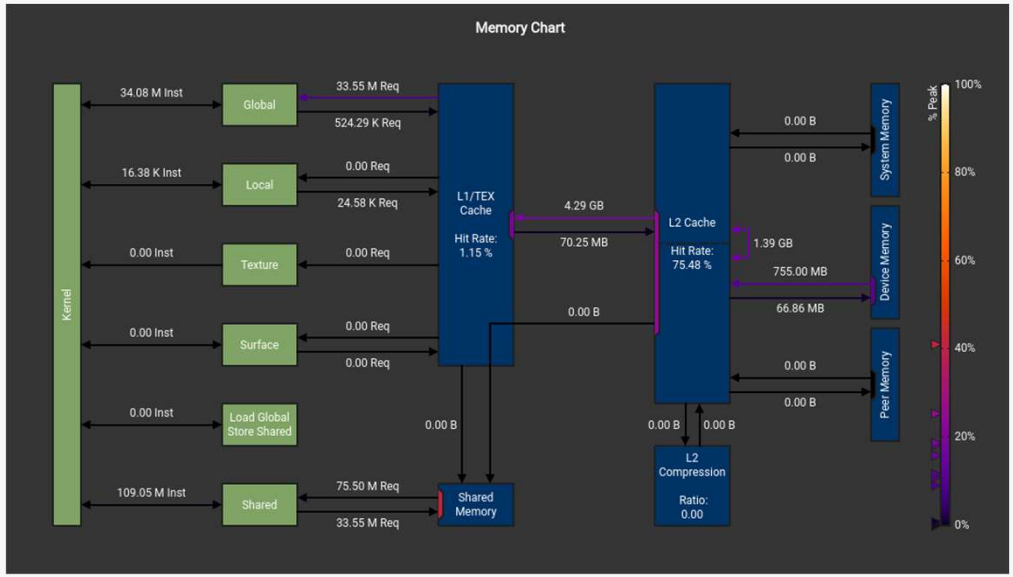
Performance comparison SM80

- GEMM Case Study on NVIDIA A100 using joint_matrix with DPC++
  - Two possible identified reasons for performance gap
    - Memory access
    - TensorCore PTX mapping

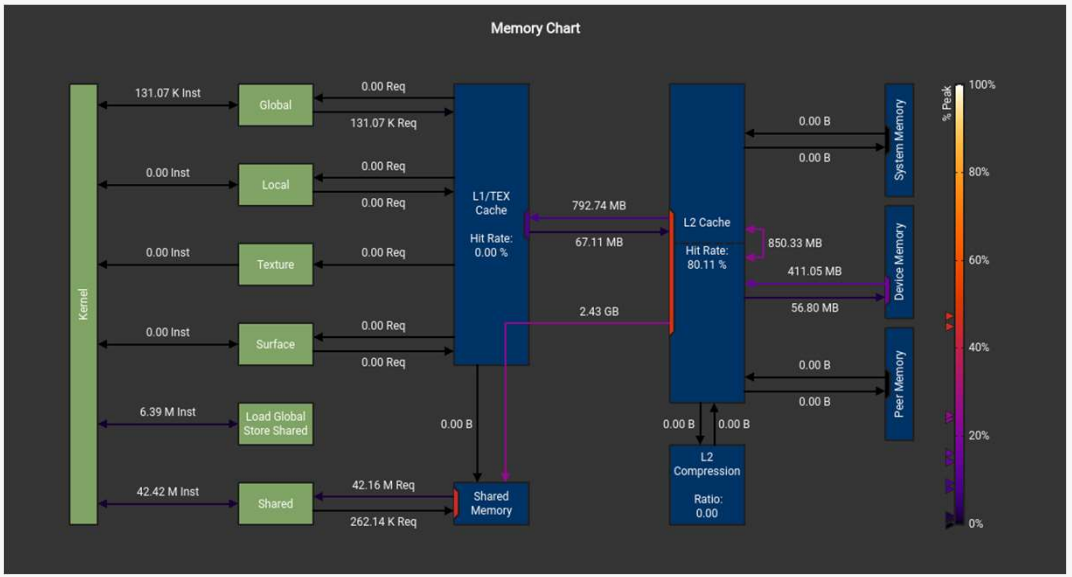


Memory Access

- Shared Load Matrix (ldmatrix)
- Shared Store From Global Load (cp.async)

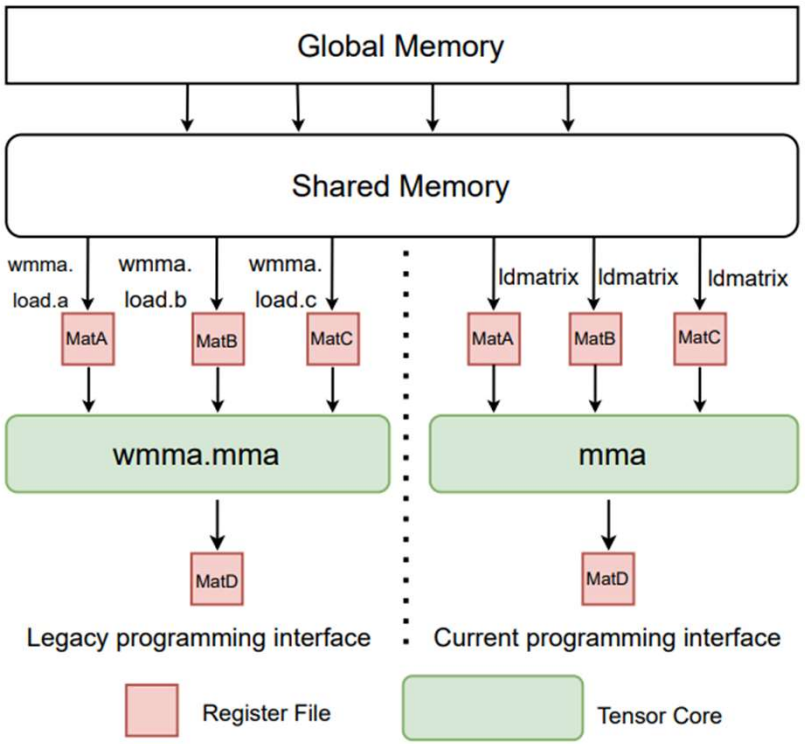


SYCL-BLAS Memory Chart (4k x 4k)



Cutlass Memory Chart (4k x 4k)

TensorCore Support in different Nvidia Architectures

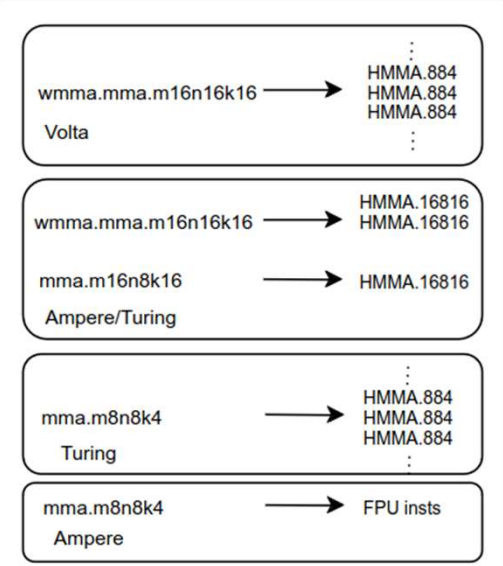


Architecture	Representative prodcuts	Numeric types	Programmability
Volta	V100, Jetson Xavier	FP16	Wmma, mma
Turing	T4, RTX20x	FP16, INT8, INT4, Binary	Wmma, ldmatrix, mma
Ampere	A100, RTX30x, Jetson Orin	FP16, BF16, TF32, FP64, INT8, INT4, Binary	Wmma, ldmatrix, mma, mma.sp
Hopper	H100	FP16, BF16, TF32, FP64, FP8, INT8	Wmma, ldmatrix, mma, mma.sp, wgmma

Pictures borrowed from Dissecting Tensor Cores via Microbenchmarks: Latency, Throughput and Numeric Behaviors Wei Sun, Ang Li, Tong Geng, Sander Stuijk, Henk Corporaal, IEEE Transactions on Parallel and Distributed Systems (Volume: 34, Issue: 1, 01 January 2023). https://arxiv.org/pdf/2206.02874.pdf

Tensor Core Instructions

- **WMMA**
  - May lead to multiple calls to HMMA based on the architecture and instruction config



- **MMA**
  - Based on the MMA instruction, it would map to
    - Either a single **hmma.m16n8k16** executed on tensor cores
    - Or multiple **hmma.m8n8k4** instructions executed on (slow) CUDA cores
  - MMA instruction interface wasn't available pre-Turing GPUs.

Pictures borrowed from Dissecting Tensor Cores via Microbenchmarks: Latency, Throughput and Numeric Behaviors Wei Sun, Ang Li, Tong Geng, Sander Stuijk, Henk Corporaal, IEEE Transactions on Parallel and Distributed Systems (Volume: 34, Issue: 1, 01 January 2023). https://arxiv.org/pdf/2206.02874.pdf

Comparison in lowering WMMA vs MMA

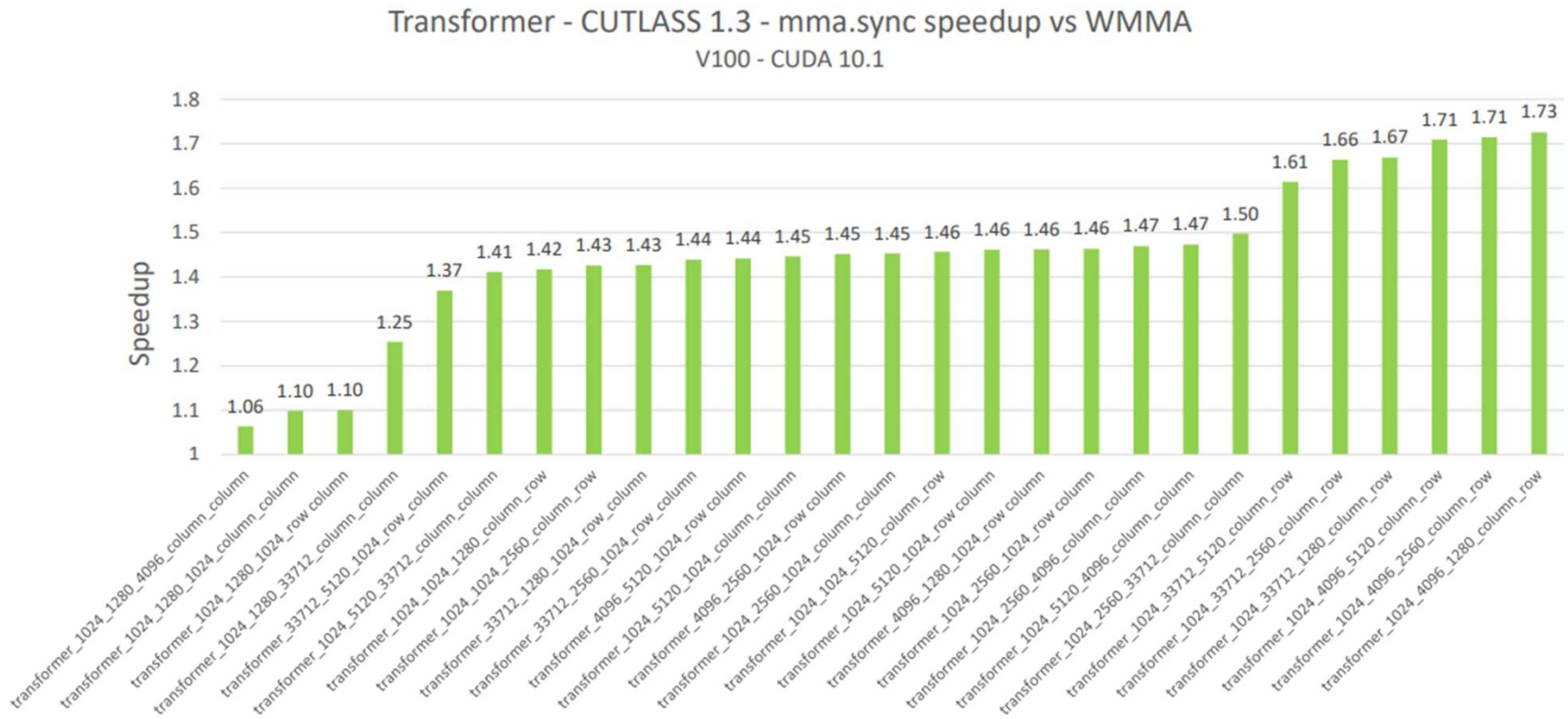
WMMA

- Portable abstraction layer over tensor cores
- Requires less programming effort
  - **wmma.load** can handle input layout for certain shape
- Does not have support for sparse matrix
- Have access to strict features of Tensor core
- Compatible for all Nvidia Devices supporting Tensor Core
- Used by **joint_matrix** extension inside the CUDA backend in DPC++

MMA

- Provides direct access to Tensor cores
- Can support Sparse Matrix
- Can support more matrix operand shape
- Used by CUTLASS/cuBLAS

SPEEDUP RELATIVE TO WMMA



Slides borrowed from**: Andrew Kerr, Timmy Liu, Mostafa Hagog, Julien Demouth, John Tran "PROGRAMMING TENSOR CORES: NATIVE VOLTA TENSOR CORES WITH CUTLASS", GTC 2019**
https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9593-cutensor-high-performance-tensor-operations-in-cuda-v2.pdf

Summary

- Supporting cp.async instruction in DPCPP can exploit architecture-specific features.
  - Challenges:
    - portability : Should mimic the instruction through software if the hardware does not support it
    - Performance: May affect the performance portability if other architectures do not support it
- Exposing the mma instruction in DPC++ CUDA backend would help bridge the performance gap between SYCL code and CUDA code
  - Pros:
    - Noticeable performance gain on Nvidia architectures
    - Nvidia's libraries at the very least for SM80 onwards rely on mma (cuBLAS/cuDNN/cuSparse uses it via CUTLASS) for supported architectures
    - Enables Sparse matrix operation on TensorCores
  - Cons:
    - There are a much larger number of instructions, so the SYCL interface would be much broader
    - Each architecture could require a different layout which would mean using a different joint_matrix type per architecture
    - The instructions take pointers to individual fragments, which means more engineering effort to write a kernel

Thank you!