LLNL-CONF-862461

Characterizing the Performance, Power Efficiency, and Programmability of AMD Matrix Cores

G. Schieffer, D. Medeiros, J. Faj, A. Marathe, I. Peng

April 2, 2024

2024 IEEE International Symposium on Performance Analysis of Systems and Software
Indianapolis, IN, United States
May 4, 2024 through May 7, 2024

Characterizing the Performance, Power Efficiency, and Programmability of AMD Matrix Cores

Gabin Schieffer[*], Daniel Medeiros[*], Jennifer Faj[*], Aniruddha Marathe[†], Ivy Peng[*]
[*]*Department of Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden*
[†]*Lawrence Livermore National Laboratory, Livermore, CA, USA*
{gabins, dadm, faj}@kth.se, marathe1@llnl.gov, ivybopeng@kth.se

***Abstract*—Matrix multiplication is a core computational part of deep learning and scientific workloads. The emergence of Matrix Cores in high-end AMD GPUs, a building block of Exascale computers, opens new opportunities for optimizing the performance and power efficiency of compute-intensive applications. This work provides a timely, comprehensive characterization of the novel Matrix Cores in AMD GPUs. We develop low-level micro-benchmarks for leveraging Matrix Cores at different levels of parallelism, achieving up to 350, 88, and 69 TFLOPS for mixed, float, and double precision on one GPU. Using results obtained from the micro-benchmarks, we provide a performance model of Matrix Cores that can guide application developers in performance tuning. We also provide the first quantitative study and modeling of the power efficiency of Matrix Cores at different floating-point data types. Finally, we evaluate the high-level programmability of Matrix Cores through the rocBLAS library in a wide range of matrix sizes from 16 to 64K. Our results indicate that application developers can transparently leverage Matrix Cores to deliver more than 92% peak computing throughput by properly selecting data types and interfaces.**

***Index Terms*—AMD GPU, Matrix Core, Tensor Core, AMD MI250, A100**

I. Introduction

AMD GPUs have emerged as a fundamental building block for exascale computing, exemplified by Frontier – the top 1 supercomputer in the world and the first supercomputer to deliver throughput in the magnitude of ExaFlops [1]. High-end AMD GPUs, in particular the Instinct MI200/300 series that power up current exascale supercomputers, are considered to be strong contenders with Nvidia's high-end GPUs, represented by the Volta V100 GPU [2], which was used to build pre-exascale supercomputers, and the Ampere A100 GPU, the successor of the V100. Besides the standard computing units and memory components on AMD and Nvidia GPUs, specialized hardware units for mixed-precision matrix-multiplication operations, namely Matrix Cores in AMD GPUs and Tensor Cores in Nvidia GPUs, are becoming one highly sought-after feature that may determine their adoption and success in the market with increasing demand on computing power and power efficiency.

The growing societal and economic interests for deep learning (DL) applications have influenced the design of hardware. The need for high-performance power-efficient matrix operations in DL applications has driven the development of specialized hardware-level matrix computation units on GPUs. For scientific workloads in high-performance computing (HPC), leveraging this specialized hardware and unlocking its potential in accelerating floating-point computations could bring significant application-level performance and system-level power saving. Therefore, this work provides a timely, in-depth characterization of AMD's latest Matrix Core in MI250X GPUs to prepare application developers for Frontier-like HPC supercomputers.

Extensive works have explored the Tensor Core on Nvidia GPUs [2]–[5]. Multiple works propose techniques for precision refinement to meet the need of accuracy in scientific workloads [2], [3]. Other works proposed important parallel algorithms, such as scan and reduction, tailored for the characteristics of Tensor Cores [4], [5]. Findings from these studies are likely to be applicable to AMD's Matrix Cores, given the similarity in their capability and programming interfaces. However, most of these works have certain dependency on low-level architectural characteristics. Therefore, our work provides an in-depth study of AMD's Matrix Core characteristics to identify opportunities and gaps of leveraging this novel hardware unit.

We first develop a set of low-level micro-benchmarks in the rocWMMA APIs and the `V_MFMA_*` instructions for utilizing Matrix Cores at configurable levels. After validating the micro-benchmarks, we quantify the achievable peak floating-point matrix multiplication operations throughput in different precisions and at different levels of Matrix Core utilization. Further, we establish and validate the performance model that correlates the peak performance with hardware usage. As we focus on HPC workloads, we specifically evaluate single- and double-precision datatypes in addition to mixed precision that is widely used in machine learning. On one AMD GPU, Matrix Cores achieve up to 350, 88, and 69 TFLOPS for mixed, float, and double precision. In contrast, our benchmarks achieve up to 290 and 19.4 TFLOPS for mixed and double precision on Tensor Cores in Nvidia A100 (float is not supported).

We leverage the micro-benchmarks and further develop a power sampling tool to understand the power efficiency of Matrix Cores. Combining the two, we are able to quantify the power consumption at different levels of throughput in different precisions on Matrix Cores. From the results, we are able to quantify the idle and dynamic power consumption. We identify a linear correlation between the power consumption

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-XXXXXX).

and delivered throughput, indicating that for each additional TFLOPS, additional 5.8, 2.1, and 0.61 Watts are consumed for double, single, and mixed precision, respectively. We note that the power consumption at the peak of double-precision operations on Matrix Cores could nearly reach the power cap, while the peak throughput is about 76% of the theoretical peak. The power efficiency of Matrix Cores is promising in reducing the energy cost of compute-intensive workloads.

While the low-level micro-benchmarks identify the potential of delivering high throughput from Matrix Cores, wide adoption in applications is only feasible through high-level programming interfaces. To understand the impact of programmability on Matrix Cores in real applications, we evaluate the rocBLAS library, whose current implementation leverages an internal two-level tile strategy that may divide matrices up to map computations onto Matrix Cores. However, as the tiling decision is only implicitly determined at runtime by the library, it is difficult to directly quantify the utilization of Matrix Cores. To address this problem, we derive two metrics that leverage hardware counters collected by *rocprof*. With this profiling method, we compare the number of floating-point operations respectively delivered from Matrix Cores and SIMD units, when using matrix dimensions ranging from 16 to 65K. Our results indicate that application developers can transparently leverage Matrix Cores to deliver more than 95% peak computing throughput by properly selecting data types and interfaces.

Our contributions are summarized as follows:

- We provide an in-depth understanding of the architecture and programming interfaces of latest AMD Matrix Cores;
- We develop low-level micro-benchmarks for controlling Matrix Cores utilization and leverage them for deriving a performance model;
- We provide a quantitative study and modelling of the power efficiency of Matrix Cores in different data precision;
- We evaluate and confirm the efficiency of high-level programmability through the rocBLAS library in utilizing Matrix Cores.

II. The Architecture of Matrix Core

The AMD Instinct GPU is a critical building block of current Exascale computers. For instance, the Frontier supercomputer consists of 37,000 MI250X GPUs and delivers 1.1 ExaFlops throughput [1]. The AMD MI250X GPU is based on the CDNA2 GPU architecture [6]. GPUs from this architecture – the MI200 series – aim at accelerating deep learning applications and HPC workloads. In this work, we focus on the low-level characteristics of Matrix Core units in AMD MI250X GPUs as it is a major component delivering high-throughput matrix multiplication operations, critical for compute-intensive machine learning and HPC workloads.

Each AMD MI250X GPU consists of two graphics compute dies (GCD) incorporated in a single physical package. The two GCDs are logically presented to users as two separate devices. Each GCD is equipped with 64 GB HBM2e memory and the two GCDs within one package are interconnected by four links of the cache-coherent Infinity Fabric. Thus, one

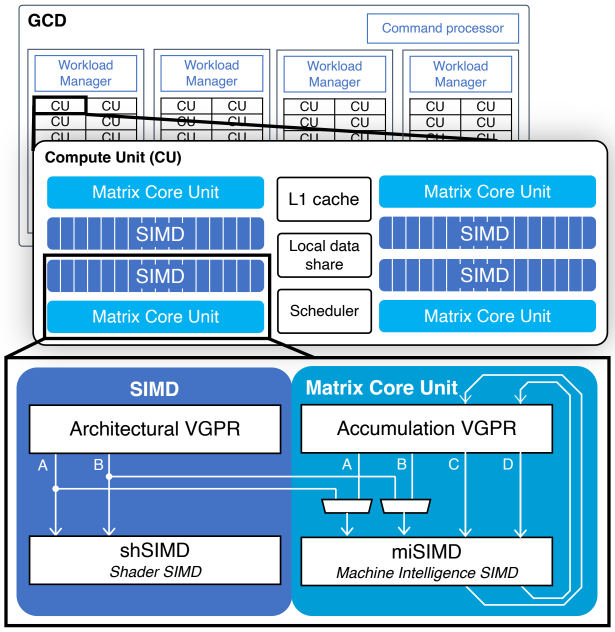

Fig. 1: An overview of the architecture of AMD's Matrix Cores, Compute Unit (CU), and SIMD units in AMD CDNA2 Graphics Compute Die (GCD).

GPU package has 128 GB HBM2e in total to deliver up to 3.2 TB/s bandwidth. Each GCD consists of 110 Compute Units (CU) as the basic processing unit. One CU consists of four Matrix Cores and four 16-wide single instruction multiple data (SIMD) units. Figure 1 illustrates the architecture of Matrix Cores and other important hardware units [6]–[8].

The Matrix Cores in this study are AMD's second generation matrix-specialized processing units for performing matrix fused multiply-add (MFMA) operations in high throughput. They are designed to execute the $D \leftarrow AB + C$ operation for specific matrix shapes and data types. A single MFMA operation is characterized by the datatypes and dimension $m \times n \times k$ of the matrices A, B, C, and D. In this notation, A is of shape $m \times k$, B is $k \times n$, and C and D are both of shape $m \times n$. Datatypes must be identical for A/B, and for C/D. In this paper, we use the notation *typeCD* ← *typeAB* to represent an MFMA operation, where C and D are of type *typeCD*, and A and B are *typeAB*.

Matrix Cores support six datatypes. In this work, we focus on evaluating Matrix Cores for HPC applications and thus evaluate the three IEEE 754 floating-point datatypes, i.e., FP16, FP32, and FP64, which are respectively designated as half-, single-, and double-precision. Matrix Cores also support 8-byte (INT8) and 32-byte (INT32) integer, along with the half-precision datatype bfloat16, which are specifically targeting machine learning workloads. Table I summarizes the supported floating-point datatypes and shapes on AMD MI250X's Matrix Cores, each line corresponding to a single assembly instruction (omitted in the table). AMD CDNA2 also supports smaller shapes, where a Matrix Core can execute up to four parallel MFMA operations on independent (A, B, C, D) matrices. For example, with the shape $16 \times 16 \times 4$, one can execute four parallel matrix FMA operations for the datatypes FP32 ← FP16.

TABLE I: The list of supported datatypes and shapes of MFMA operations ($D \leftarrow AB + C$) on Matrix Cores (AMD) and Tensor Cores (Nvidia) at the instruction level.

Types	**Shape**	
(C/D ← A/B)	**AMD CDNA2**	**Nvidia Ampere**
FP64 ← FP64	$16 \times 16 \times 4$	$8 \times 8 \times 4$
FP32 ← FP32	$16 \times 16 \times 4$ $32 \times 32 \times 2$	×
FP32 ← FP16	$16 \times 16 \times 16$ $32 \times 32 \times 8$	$16 \times 8 \times 8$ $16 \times 8 \times 16$
FP16 ← FP16	×	$16 \times 8 \times 8$ $16 \times 8 \times 16$

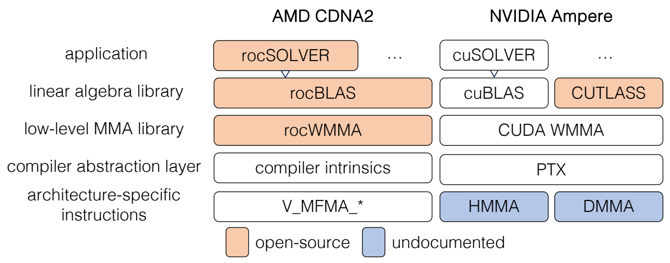

Fig. 2: The hierarchy of programming interfaces to AMD's Matrix Core and Nvidia's Tensor Core. A higher-level component typically relies on its direct lower-layer component.

AMD's Matrix Cores are considered a strong contender to Nvidia's Tensor Cores, which provide similar mixed-precision MMA computing capabilities. Matrix Cores, however, exhibit several higher-level characteristics, from the perspective of HPC computing, compared to the third-generation Tensor Cores from the Nvidia Ampere architecture. The first aspect lies in the 95.7 TFLOPS theoretical double-precision floating-point performance, as documented by AMD [6], about four times that of Nvidia's Tensor Cores (19.5 TFLOPS). As the use of double-precision is widespread in HPC applications, we see a particular opportunity to use AMD Matrix Cores in this context. In addition, AMD Matrix Cores offer more supported matrix shapes as shown in Table I, which can ease algorithmic adaptation and the utilization of Matrix Cores.

III. Programming AMD's Matrix Cores

Figure 2 presents a hierarchical view of programming interfaces to AMD Matrix Cores (based on MI250X) and Nvidia Tensor Cores (based on A100), where a higher-level programming interface relies on its direct lower-level interface. We briefly describe these approaches from low-level programming interfaces up to the application level as follows.

- *CDNA2 Instruction Set [8]:* As detailed in II, Matrix Cores support a variety of datatypes and matrix shapes. Each combination of datatype and matrix shape (i.e., each row in Table I) is supported by one instruction in the format `V_MFMA_[typeCD]_[MxNxK][typeAB]`. Those instructions read matrix elements from regular vector general-purpose registers ("Architectural VGPRs"), and store the result in a specific general-purpose registers file (the "Accumulation GPRs"). Read and write accesses to those registers are possible through specific instructions. Due to hardware limitations, several no-op instructions might be required before the result data can be read. Matrix Core instructions are executed collectively by all 64 threads of a wavefront, where wavefront is equivalent to warp in Nvidia's terminology. For specific matrix shapes, the `V_MFMA_*` instructions can perform one to four parallel MFMA operations, on independent groups of smaller matrices: $i \in [\![1, 4]\!], D_i \leftarrow A_i B_i + C_i$.

While Nvidia provides Tensor Cores with a cross-architecture low-level programming interface (PTX), the documentation of the architecture-specific instruction set (SASS) is not officially provided. In particular, the HMMA and DMMA instructions, which are equivalent to AMD's `V_MFMA_*` instructions, are not officially documented.

- *Compiler Intrinsics:* A set of intrinsics are provided in the LLVM compiler to program Matrix Cores at the programming language level. Those builtins are of format `__builtin_amdgcn_mfma_[typeCD]_[MxNxK][typeAB]`, and directly map to assembly-level Matrix Core-related instructions. In addition, AMD provides a Python script to get information on the mapping between matrix elements and registers [9], providing a way for users to develop C-level code, that can efficiently and directly leverage Matrix Cores. This ability to program Matrix Cores at the programming-language level facilitates compiler optimizations.

Nvidia provides extensive documentation on available instructions and register-element mapping for Tensor Cores, but no direct C-level programming interface is officially provided. Developers have to use either PTX assembly or higher-level interfaces to use Tensor Cores.

- *Low-level matrix multiply-accumulate APIs:* The *rocWMMA* C++ library provides a simplified way of programming Matrix Cores using C++ templated functions. WMMA stands for *Wave Matrix Multiply-Accumulate*, as this library relies on the cooperation of the 64 threads in a wavefront to perform a matrix multiply-accumulate (MMA) operation, in the same fashion as the underlying Matrix Core instructions. This library relies on the concept of a *fragment* to represent matrices. A fragment is a C++ object which abstracts the mapping of matrix elements to their respective registers. *rocWMMA* exposes functions to load and read data from/to fragments. These functions are translated at compile-time to register loads and stores, without any user-knowledge of the in-register data layout. In addition, a function `mma_sync` performs the fused multiply-and-add operation using Matrix Core instructions. *rocWMMA* is directly cross-compatible with Nvidia's counterpart, *CUDA WMMA*. However, as the available choices for matrix datatypes and shapes are different on Nvidia hardware, users must ensure that their use of the WMMA library complies with hardware limitations.
- *High-level Linear Algebra Library:* The Basic Linear Algebra Subprograms (BLAS) specification is widely used in machine learning and HPC applications. This specification defines a set of linear algebra routines. *rocBLAS* is AMD's implementation of the specification. *rocBLAS* tries to leverage Matrix Cores whenever they are available, with no option to opt-out at the user level. Such performance optimization requires the datatypes to be compatible with Matrix Core abilities, and is exploitable for specific operations, such as GEMM. To use Matrix Cores, the library chooses at runtime a strategy to map an operation on arbitrary-shaped matrices

to the constrained fixed-shape Matrix Core abilities. This is typically done by dividing the matrices into tiles, and executing $16 \times 16 \times 16$ operations on Matrix Cores. Nvidia provides a similar implementation called *cuBLAS* for Tensor Cores.

• *Applications and HPC Libraries:* Higher-level libraries can be built atop BLAS to implement general-purpose algorithms. An example is LAPACK (Linear Algebra PACKage) [10], which provides routines to solve systems of linear equations and find eigenvalues of matrices. This library delegates a significant amount of computation to the BLAS implementation. *rocSOLVER* is AMD's implementation of a subset of LAPACK routines. It relies on rocBLAS to execute matrix operations, which naturally leads to opportunistic leveraging of Matrix Cores in this high-level library. Finally, applications can leverage these libraries to eventually utilize Matrix Cores.

IV. Characterization Methodology

In this study, we conduct our experiments on an AMD testbed equipped with four AMD MI250X GPUs. From the vendor's datasheet, the maximum power consumption of a GPU package is 560 Watt [11]. Compilation tools, kernel drivers, and runtime libraries for the AMD platform are provided as part of the ROCm platform version 5.3.3. We also use an Nvidia testbed equipped with two Nvidia A100 GPUs for the comparative study. We use CUDA 11.5 for the Nvidia platform. For experiments on the AMD platform, our benchmarks run on a single graphics compute die (GCD), if not specified differently. When comparing with one Nvidia A100 GPU, we use one GPU package. We repeat experiments at least ten times for each reported metric and report error bounds if the variance exceeds 2%.

A. Benchmarks

We develop a set of micro-benchmarks that can be used to utilize the Matrix Cores at different levels and quantify the instruction latency for supported matrix shapes and datatypes. The benchmarking kernel has the 64 threads of a single wavefront perform collectively the same MMA operation within a loop of 40 million iterations. The average instruction latency is obtained by timing of the loop using the `clock64()` function inside device kernel code. When measuring instruction latency, one wavefront is used at a time. This micro-benchmark approach is similar to the approach for evaluating Tensor Cores on Nvidia GPUs [12], where evaluated instructions are isolated in a long-running loop. This benchmark excludes the impact of data transfer to registers as no load/store operations are performed. Thus, it allows getting latency information for each instruction. When measuring throughput, the number of wavefronts in the kernel can be configured at launch time.

The comparison between AMD and Nvidia GPUs is constrained by the notable difference on their hardware capability. The first difference comes from the supported datatypes – only matrix operations in mixed-precision F32 $\longleftarrow$ F16 and double-precision F64 $\longleftarrow$ F64 are supported on both platforms. In addition, supported instruction-level matrix shapes are different on the two platforms, which requires adaptation of the benchmark code. In the context of HPC workloads, we only focus on the three supported IEEE 754 floating-point datatypes – half-precision (FP16), single-precision (FP32), and double-precision (FP64).

We use the AMD-developed HIP C++ runtime API and kernel language to implement the benchmark. In particular, we use the rocWMMA API, which allows high-level C++-level access to Matrix Cores capabilities, while preserving the semantic of the code down to the level of assembly instructions. Since this API is cross-compatible with CUDA WMMA API, we can use the same codebase for both AMD and Nvidia platforms. To ensure that the compiled code is actually coherent with the goal of each experiment, we check the assembly-level instructions using the HIP compiler flag `-S` or the `cuobjdump` tool to verify the number of Matrix/Tensor Core instructions in use. We also ensure that compiler optimizations did not affect measurement correctness.

We anticipate that wide adoption of Matrix Cores in large-scale HPC applications is only feasible through high-level programming interfaces, such as math libraries. From the hierarchy of major programming approaches to Matrix Cores presented in Figure 2, the *rocBLAS* library is the fundamental building block for other high-level software approaches. Therefore, we evaluate the high-level programmability of Matrix Cores by assessing the efficiency of the *rocBLAS* library in utilizing Matrix Cores. In this study, we use the general matrix multiplication (GEMM) routine in the BLAS specification, which performs the operation $D \longleftarrow \alpha \cdot AB + \beta \cdot C$, where α and β are scalars, and A, B, C, D are arbitrary-size matrices. We evaluate all supported combination of floating-point data types in GEMM operations, including single (*SGEMM*), double (*DGEMM*), and mixed types (*HGEMM*, *HSS*, *HHS*).

We adapt the default GEMM use case provided by AMD in the rocBLAS library. The original code uses the `rocblas_{d,s}gemm` function to perform a DGEMM or SGEMM operation on GPU. We replace this function by the generic `rocblas_gemm_ex` function, which executes specific GEMM operations as identified by the datatype parameters. This allows running the mixed-precision GEMM operations, which are not exposed through dedicated functions. In our experiments, values in A and C are set to 1, while B is set to the identity matrix. The result in D should be a $n \times n$ matrix filled with 2, which makes the correctness of results easily verifiable.

B. Profiling Methods

For the micro-benchmarks written in WMMA APIs, as the utilization of Matrix Cores and Tensor Cores can be explicitly controlled, we can calculate FLOPS from the number of executed Matrix Core or Tensor Core instructions and the documented FLOPS per instruction from Nvidia and AMD's documentation, respectively.

As the documentation of rocBLAS indicates that no user action is required to leverage Matrix Cores, we still need to verify and quantify how Matrix Cores are being used. However, in applications based on *rocBLAS*, the number of floating-point operations cannot be trivially computed as in the micro-benchmarks, since the high-level libraries may not provide precise algorithmic descriptions, or unspecified compilation-time optimizations may be employed. Therefore, we use the performance counters provided by *rocprof* to measure the number of floating-point operations. In particular, non-zero values returned from counters related to Matrix Cores, i.e., `SQ_INSTS_VALU_MFMA_MOPS_F*`, would indicate that Matrix Cores are used in a rocBLAS-based application. We use the approach proposed in [13], [14] to derive the exact number of performed floating-point operations, from performance counters. Eq. 1 presents the formula used for double-precision floating-point operations. A similar formula can be derived for single and mixed-precision by substituting relevant counters for each datatype.

$$\begin{aligned}\text{TOTAL\_FLOPS\_F64} &= 512 \cdot \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_F64} \\ &+64 \cdot \text{SQ\_INSTS\_VALU\_ADD\_F64} + 64 \cdot \text{SQ\_INSTS\_VALU\_MUL\_F64} \\ &+128 \cdot \text{SQ\_INSTS\_VALU\_FMA\_F64} \qquad (1)\end{aligned}$$

The counters in Eq. 1 refer to the various types of floating-point operations that SIMDs and Matrix Cores can execute. `SQ_INSTS_VALU_MFMA_MOPS_F64` represents the number of floating-point operations performed by Matrix Cores; at hardware-level, this counter is incremented once every 512 operations. `SQ_INSTS_VALU_{ADD,MUL}_F64` designates the number of add/multiply operations, per-SIMD. This number needs to be multiplied by 64 to account for 64 parallel-running threads. Similarly, `SQ_INSTS_VALU_FMA_F64` counts the number of fused multiply-add (FMA) operations performed, per-SIMD. In this case, a factor of $2 \cdot 64 = 128$ needs to be added to account for the 64 parallel-running threads, each executing a fused multiply-add operation, which accounts for 2 floating-point operations. In addition, we use the individual values for each of those counters to identify the distribution of floating-point operations between regular SIMDs and Matrix Cores.

C. Power Measurements

Measurement of power consumption is done through the vendor-provided System Management Interface (SMI), namely ROCm SMI library. Similar approaches through nvidia-smi have been widely used on Nvidia GPUs [15], [16]. We sample the power consumption, in Watts, through the kernel execution lifespan. To collect this quantity on the AMD platform, we develop a background sampling process that calls `rsmi_dev_power_ave_get()` function to poll power periodically at a user-defined period. This function is part of the ROCm SMI library and provides identical results as when using the rocm-smi tool, while allowing to control sampling frequency. Our results are based on a sampling period of 100 ms and each kernel execution is controlled to run sufficiently long to gather at least 1000 samples. We also tried shorter sampling periods like 10ms, delivering similar results. In addition, we also validate our power measurements on our AMD testbed by comparing with the Cray power measurement counters dedicated to monitoring accelerator power consumption, accessible through the `/sys/cray/pm_counters` filesystem-based interface, as described in [17].

TABLE II: Measured latency of Matrix Core MFMA instructions on AMD MI250X GPU.

types (C/D ← A/B)	$m \times n \times k$	latency (cycles)
FP32 ← FP32	$32 \times 32 \times 2$	64.0
	$16 \times 16 \times 4$	32.0
FP32 ← FP16	$32 \times 32 \times 8$	64.0
	$16 \times 16 \times 16$	32.0
FP64 ← FP64	$16 \times 16 \times 4$	32.0

V. Computational Performance

In this section, we evaluate the performance of Matrix Cores in an MI250X GPU in three phases. In the first phase, we validate the accuracy of our micro-benchmarks, in measuring performance and controlling the utilization level of Matrix Cores. In the second phase, we derive a throughput model and compare the theoretical and actual reachable throughout on one GCD. Finally, we present an overall comparison with Nvidia A100's Tensor Cores.

A. Micro-benchmarking

We first use the micro-benchmark to quantify the latency of Matrix Core instructions, and then compare our measured results with vendor-provided datasheets for validating our micro-benchmark. Using the micro-benchmark approach presented in Section IV-A, we measure the latency of Matrix Core instructions using a single wavefront. The measured latency is presented in Table II. AMD documentations of values for Matrix Core performance are expressed in number of Matrix Core-performed floating-point operations per CU per cycle (noted FLOPS/CU/cycle) [6]. From our measurements of the latency c of an $m \times n \times k$ MFMA instruction, which performs $2mnk$ floating-point operations, we can deduce that a CU (with four Matrix Core units) is able to provide $8mnk/c$ FLOPS/CU/cycle. Using this relationship, we confirm that the measured latency is consistent with AMD's official data.

In the second experiment, we set up the micro-benchmark to increase the level of Matrix Core utilization and measure the obtained floating-point throughput at each configured level. In this test, inside one computational kernel, each wavefront iterates a number of 16×16×16 rocWMMA operations, which are mapped to underlying Matrix Core MFMA instructions at compilation time. We gradually increase the number of wavefronts in use when launching the kernel to eventually allows the benchmark to leverage all Matrix Cores on one GCD. We compute the number of floating-point operations performed on Matrix Cores as $2mnk \cdot N_{\text{iter}} \cdot N$, where N is the number of wavefronts, and N_{iter} is the number of $m \times n \times k$ MFMA operations performed by each wavefront, set to $N_{\text{iter}} = 10^7$. We measure the total execution time of the kernel by triggering HIP events, before and after the kernel launch, and use this value to derive the floating-point throughput. We are able to validate the micro-benchmark by comparing the measured results with the kernel profiling results obtained with rocprof.

B. Throughput Modelling

We leverage the micro-benchmarking results and the architectural information on the Matrix Cores to build a perfor-

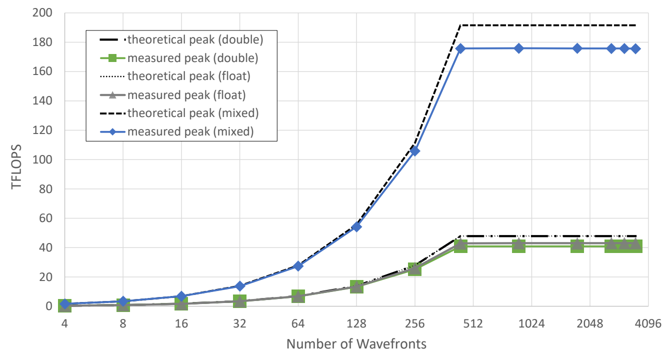

Fig. 3: A comparison of the measured and the predicted theoretical floating-point throughput on AMD Matrix Cores in one MI250X GCD for three floating-point datatypes.

mance model of peak floating-point throughput at various level of Matrix Core utilization. The model considers as input the latency c of a $m \times n \times k$ MFMA instruction, the clock frequency $f = 1700$ MHz of the hardware device, and the number N_{WF} of wavefronts set when launching the benchmark. This model is presented in Eq. 2. The threshold of 440 used for N_{WF} is the number of Matrix Cores in one GCD on MI250X, reflecting the fact that no more than 440 wavefronts can execute Matrix Core instructions at one time.

$$\text{FLOPS}(N_{\text{WF}}) = \frac{2mnk}{c} \cdot \min(N_{\text{WF}}, 440) \cdot f \qquad (2)$$

Figure 3 presents the results of achieved floating-point throughput at an increased number of wavefronts for the three floating-point datatypes (mixed, float, double) supported on Matrix Cores. We include in this figure the theoretical peak throughput, obtained using the model presented in Eq. 2 in dashed lines. We use values multiple of four for the number of wavefronts because four Matrix Cores are available on each CU. For the number of wavefronts ranging between 4 and 440, we increase at a doubling rate. However, as we reach 440 wavefronts, the number of wavefronts exceeds the number of available Matrix Cores in one GCD. Therefore, we use values multiple of 440; this is to avoid situations where the device is partially utilized for part of the kernel lifespan. For example, in a case where 660 wavefronts are launched, in a first phase, 440 will be able to execute immediately on the 440 Matrix Cores, and will terminate at the same time. The remaining 220 wavefronts will then execute in a second phase, during which half Matrix Cores will be idle.

Up to 440 wavefronts, we observe a linear increase of the floating-point throughput with the number of wavefronts, which is coherent with our model. Note that we use linear scale for the y-axis but logarithmic scale for the x-axis because the predicted and measured values are almost overlapping if the y-axis is in logarithmic scale. When exceeding 440 wavefronts, the throughput reaches a plateau and gives a sustained throughput of 175 TFLOPS for mixed-precision, 41 TFLOPS for double-precision, and 43 TFLOPS for single-precision. We observe that our benchmark achieves a high level of performance compared to the theoretical peak throughput – the double-, single- and mixed- precision throughput achieved respectively 85%, 90%, and 92% of the theoretical peak. These

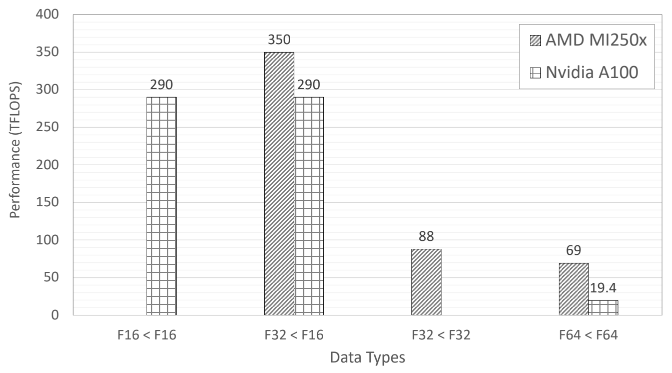

Fig. 4: The achieved floating-point throughput on Matrix Cores on AMD MI250X and Tensor Cores on Nvidia A100, for the four supported types in Table I.

close-to-peak performance on Matrix Cores are promising for application developers porting codes onto AMD MI250X GPUs to exploit the Matrix Cores' full potential.

C. *Comparison with Nvidia Tensor Cores*

To provide a direct comparison of AMD's Matrix Cores with Nvidia's Tensor Cores, we execute the same benchmark on Nvidia A100, using the CUDA WMMA API and by adapting the matrix shapes and datatypes to fulfill CUDA WMMA's requirements. In addition, to provide a fair comparison between the two vendors, we ensure that we evaluate the floating-point throughput of one AMD GPU package by executing the throughput benchmark in parallel on both GCDs in a single MI250X GPU. The results of our evaluation are presented in Figure 4, where the peak measurements are reported for all possible datatypes. Note that not all datatypes are supported by both platforms – while Nvidia A100 Tensor Cores do not support single-precision, AMD MI250X Matrix Cores do not support F16 $\longleftarrow$ F16 half-precision operations.

In these experiments, we achieve on Nvidia A100 a floating-point throughput of 290 TFLOPS for mixed-precision, and 19.4 TFLOPS for double-precision. Compared to the vendor's datasheet, this is equivalent to 93% of the 312 TFLOPS peak throughput in mixed-precision, and 99% of the 19.5 TFLOPS peak throughput in double-precision. For AMD MI250X, we achieve 350 TFLOPS in mixed-precision, which is 91% of the advertised 383 TFLOPS. In single-precision and double-precision, we reach 88 TFLOPS and 69 TFLOPS, respectively. As the theoretical peak for both single and double-precision is 95.7 TFLOPS, this represents 92% of the theoretical peak for single-precision and 72% for double-precision. The performance results indicate that MI250X Matrix Cores can provide higher floating-point performance than A100 Tensor Cores, which is critical to many HPC workloads.

In particular, the floating-point throughput for double-precision Matrix Cores is $\times 3.5$ higher on AMD MI250X than on Nvidia A100. One reason for this is that MI250X has more CU and a higher clock rate (1700 MHz) than Nvidia's A100 SM and clock rate (1410 MHz). Also, one A100 SM performs 128 FP64 FLOPS per cycle, while each MI250X CU performs 256. We further observe that when scaling the benchmark from one GCD to two GCDs, the floating-point throughput for double-precision does not scale accordingly. The percentage of the theoretical peak achieved in the two-GCD (72%) scenario is significantly lower than when using only one GCD (85%), which could be explained by the near-cap power consumption as detailed in Section VI.

AMD Matrix Cores outperform Nvidia Tensor Cores in three out of the four supported floating-point operations, giving 3.5× double-precision throughput. HPC workloads may significantly benefit from its floating-point throughput.

VI. Power Efficiency

We evaluate multiple aspects of power consumption of Matrix Cores, including static and dynamic power consumption, as well as their power efficiency. As the power measurement is instrumented at the whole physical package, we use both GCDs on the MI250X GPU by running one process per GCD. Figure 5 presents the power consumption at different levels of throughput. We also insert a projected peak throughput for two GCDs for the three floating-point data types, based on the previous section, and the power cap published by the vendor.

We quantify the idle power of a whole GPU package to be 88 W on the AMD MI250X. We formulate several hypotheses for the idle power use on the AMD GPU. First, the higher amount of HBM2e memory present on MI250X (128 GB) compared to A100 (40 GB) could be a significant factor. Also, the cache coherent interconnect, Infinity Fabric, could increase power consumption. In addition, differences in the system-specific setup of power management strategies, i.e., Nvidia A100 is kept in low-power mode while the AMD MI250X is probably kept in performance mode, could explain this difference.

We develop a model of power consumption for the three Matrix Core operation data types as a function of throughput in Eq. 3, where PC denotes the total power consumption in Watt and Th denotes the throughput in TFLOPS.

$$PC_{DT} = \begin{cases} 5.88 \cdot Th + 130 & \text{for } DT = double \\ 2.18 \cdot Th + 125.5 & \text{for } DT = float \\ 0.61 \cdot Th + 123 & \text{for } DT = mixed \end{cases} \tag{3}$$

By comparing the modelled power (dashed lines) and the measured power (solid lines) in Figure 5, we validate the accuracy of the power model. The 560 W GPU power cap, which limits the instantaneous GPU power draw, is also represented. Though mixed-precision (blue) and single-precision (gray) operations reach very different peak throughput, i.e., 350 and 88 TFLOPS, their peak power consumption are similar, i.e., 338 W for float and 319 W for mixed type. In contrast, the double-precision operation can reach a significantly higher power consumption, up to 541 W, at its peak throughput, approximating the 560 W-power cap closely.

To further understand the power impact of Matrix Core operations, we quantify the power efficiency for different data types. Power efficiency is computed as the average number of floating-point operations per second divided by the average power consumption. We quantify the power efficiency

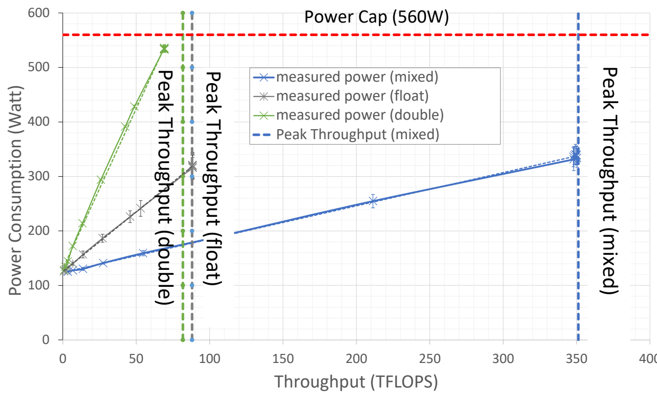

Fig. 5: Measured power consumption at increased obtained throughput for Matrix Core operations in three data types.

in TFLOPS per Watt for the various supported levels of precision. This performance per Watt metric is widely used to characterize power efficiency of computing systems [15], [16]. The highest power efficiency is achieved in the mixed precision operations, reaching 1020 GFLOPS/Watt.

In the previous section, single-precision and double-precision Matrix Cores operations both exhibit a similar level of floating-point arithmetic throughput. However, the power consumption measured when using double-precision is significantly higher than for single-precision. This observation translates to the power efficiency of single-precision Matrix Core operations (273 GFLOPS/Watt) to be approximately two times higher than for double-precision Matrix Core operations (127 GFLOPS/Watt). The use of single-precision Matrix Core operations should be preferred to double-precision, as this dramatically reduces the power consumption for similar levels of floating-point performance. The mixed-precision power efficiency (1020 GFLOPS/Watt) is 3.7× higher than the single-precision power efficiency (273 GFLOPS/Watt).

Power-efficient and energy-aware applications should leverage the high power efficiency of double-precision operations on Matrix Cores and further 4× and 8× power saving can be achieved when switching to single and mixed-precision.

VII. High-level Programmability

In this section, we evaluate the efficiency of utilizing Matrix Cores through high-level programming interfaces. In particular, we evaluate the *rocBLAS* library, a building block in other higher-level programming approaches, as presented in Figure 2. For this evaluation, we use the GEMM routine in five floating-point datapyes. We use matrices of dimension $N \times N$, $\alpha = \beta = 0.1$, and increase the value of N until exhausting the GPU memory at $N = 65000$.

Figure 6 presents the floating-point arithmetic throughput achieved for the GEMM operation in double-precision (DGEMM) and single-precision (SGEMM). We achieve a maximum of 43 TFLOPS in single-precision at $N = 8192$, and 37 TFLOPS in double-precision at $N = 4096$. From Section V, the reachable peak throughput on Matrix Cores is 43 and 41 TFLOPS in single- and double-precision, respectively.

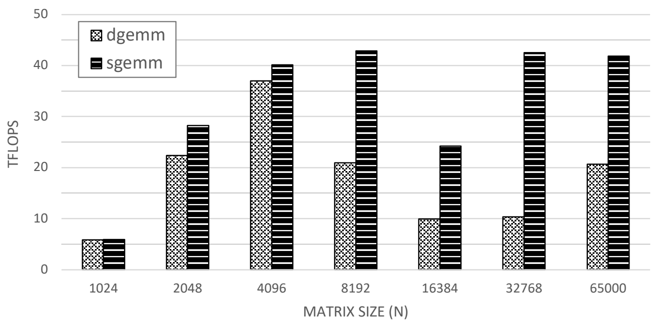

Fig. 6: Floating-point throughput achieved using rocBLAS for a $N \times N \times N$ GEMM operation, in single-precision (sgemm) and double-precision (dgemm).

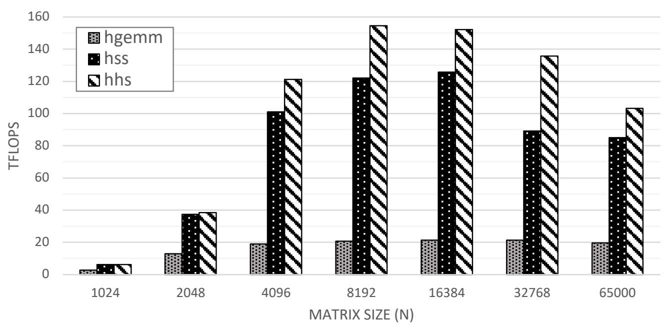

Fig. 7: Floating-point throughput achieved using rocBLAS for a $N \times N \times N$ GEMM operation in three mixed-precision datatypes (hgemm, hss, hhs).

Thus, the rocBLAS library reaches almost 100% and 90% of the peak performance in the two datatypes without imposing the low-level porting efforts onto programmers.

We observe a performance drop for SGEMM after N = 8192, and for DGEMM after N = 4096. One reason could be the high memory footprint and increased data movement above this point. This hypothesis is coherent with the fact that the double-precision DGEMM exhibits a drop earlier than single-precision SGEMM, due to the higher memory footprint of double-precision matrices. As we further increase the size of the matrices up to N = 65000, the performance for single-precision reaches values comparable to the observed peak, likely because the hardware can hide more latency-bound operations in this particular GEMM operation for larger problems.

Application developers can leverage the rocBLAS library to exploit near-peak floating-point throughput on Matrix Cores in single- and double-precision with minimal porting efforts.

We evaluate half- and mixed-precision GEMM operations in rocBLAS through HGEMM, HSS, and HHS operations. While these operations all operate on half-precision A and B matrices, the datatype for the matrix C and the output matrix D can be either half- or single-precision as detailed in Table III.

Figure 7 presents the measured throughput for the three operations. We observe that HHS outperforms HSS for all matrix sizes above 1024. A peak throughput of 155 TFLOPS is achieved for the HHS operation, which represents 88% of the peak throughput attainable on Matrix Cores in one GCD as measured in Section V. Note that HSS and HHS only differ in the datatype of C and D, where HHS has C and D in half-precision while HSS has them in single-precision. Thus, the performance gap between the two could come from type casting. As shown in Section II, the only operation supported by Matrix Cores on half-precision matrices is the mixed-precision FP32 ⟵ FP16 MFMA operation. Therefore, mapping operations to Matrix Cores could imply casting between single and half-precision, depending on the desired GEMM operation.

TABLE III: Datatypes for the three rocBLAS half- and mixed-precision GEMM operations in rocBLAS.

Operation	typeAB	typeCD	Compute type (α and β)
HGEMM	FP16	FP16	FP16
HHS	FP16	FP16	FP32
HSS	FP16	FP32	FP32

Similar to DGEMM and SGEMM, we also observe a performance decrease at N = 8192 for HHS and N = 16384 for HSS, which can be similarly explained by the cost of data movements being a limiting factor. One surprising result is that HGEMM, which only operates in FP16 values, is consistently outperformed by HSS and HHS for all matrix sizes. Our further profiling and analysis show that HGEMM does not utilize Matrix Cores at all in the executions, explaining the low performance.

Applications using the mixed-precision rocBLAS GEMM operations needs to use HSS and HHS to fully exploit Matrix Cores for delivering most floating-point operations.

Figure 8 presents the percentage of floating-point operations performed by Matrix Cores in a rocBLAS GEMM operation at various matrix sizes, as derived from counters in IV-B. We observe that for DGEMM, SGEMM, and HHS/HSS, more than 90% of floating-point operations are performed on Matrix Cores for matrix sizes with $N > 16$, and sustained above 99% for $N > 256$. As HGEMM does not utilize Matrix Cores, this indicates that matrix operations for the HGEMM routine are exclusively executed on SIMD units, which exhibit lower peak performance than Matrix Cores. Therefore, we can quantify the speedup achieved through the use of Matrix Cores in GEMM applications if we use HGEMM as the reference performance on SIMD units. Using values from Figures 6 and 7, this leads to 2.3×-7.5× speedup by Matrix Cores over SIMD units in mixed-precision, and up to 2.2× speedup in the single- and double-precision, with exception for double-precision at N = 16384 and N = 32768.

Our profiling results also show that HHS and HSS do not utilize Matrix Cores for the smallest N = 16 matrix. This is unexpected, as exactly one Matrix Core instruction would be required to perform a matrix FMA in mixed-precision for it. One reason could be that as a α/β scaling operation needs to be performed and cannot be mapped to Matrix Cores, the cost of running all operations on SIMD units is lower than distributing the matrix FMA and scaling operations between Matrix Cores and SIMD units.

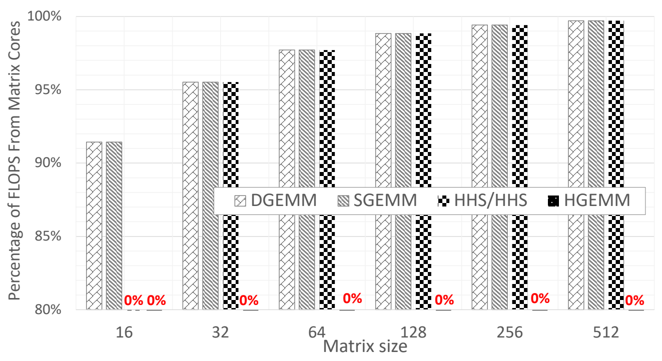

Fig. 8: The ratio of throughput in the rocBLAS GEMM routines delivered from Matrix Cores at increased matrix sizes.

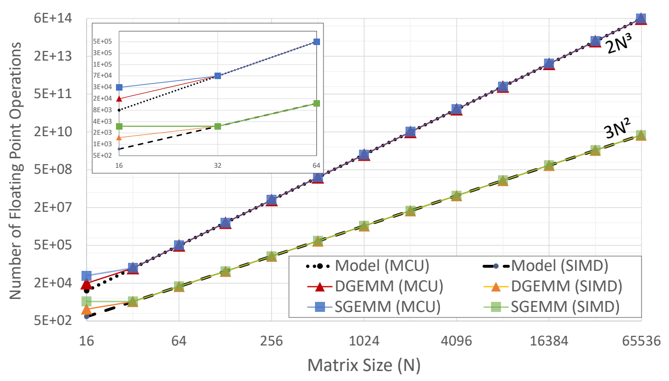

Fig. 9: The number of floating-point operations executed on Matrix Cores and SIMD units, respectively, for each matrix size in the rocBLAS GEMM routine.

To characterize the performance of rocBLAS, we further establish a model for the distribution of floating-point operations between SIMD units and Matrix Cores in rocBLAS GEMM. Figure 9 presents our model, along with measurements of the number of floating point operations performed respectively on SIMD units and Matrix Cores, for SGEMM and DGEMM. The number of floating-point operations required to perform a GEMM operation $\alpha AB + \beta C$ on matrices of shape N^2 is a polynomial of N. This polynomial is of degree three, due to the algorithmic complexity of the matrix multiplication AB.

We evaluate various values for the coefficients of this polynomial, and compare the resulting model with the measured numbers of floating-point operations performed respectively by SIMD units and Matrix Cores. We find that for one HGEMM, SGEMM, or HHS/HSS operation, $2N^3$ arithmetic floating-point operations are performed on Matrix Cores and $3N^2$ operations are performed on SIMD units. The number of Matrix Core operations is consistent with the $2N^3$ computational complexity of the matrix FMA operation performed by Matrix Cores. Moreover, we infer that the number of SIMD operations is induced by the α/β-scaling of matrix values, which cannot be performed on Matrix Cores, and is consistent with the N^2 term. The overlapping of the model and experimental values for $N \geq 32$ in Figure 9 validates our model. For $N = 16$, we observe deviation from the model likely because the optimization strategy used in rocBLAS differs for larger matrices.

For a GEMM operation, this model indicates that the number of floating-point operations performed on Matrix Cores is $\frac{2}{3}N$-times higher than the number of math SIMD operations. This model shows that the proportion of Matrix Core floating-point operations dominates the number of SIMD operations in rocBLAS GEMM. In particular, using this model, we find that for $N \geq 32$, more than 95% of floating-point operations are performed on Matrix Cores.

VIII. Related Works

System Characterization and Performance Evaluation. Leinhauser et al. [14] designed an instruction roofline model for AMD GPUs (specifically the AMD MI60, AMD MI100, and Nvidia V100 GPUs). They use rocprof to gather metrics and the kernel times and derive a model for instruction intensity and instructions per cycle. They demonstrate the model in a specific plasma simulation code. Godoy et al. [18] evaluated the DGEMM kernels from the perspective of portability in multiple frameworks and programming languages, including Julia, Python, and Kokkos, and used them in Nvidia A100 and AMD MI250X GPUs. They conclude that the Julia version is comparable to the HIP one. Eberius et al. [19] performed a strong scaling analysis on an Nvidia A100 and AMD MI250X GPUs together with an extended roofline model by adding a new metric of saturated problem size to allow more precise characterization of these GPU's performance.

Matrix Cores and Tensor Cores. Markidis et al. [2] evaluated the Tensor Cores in an Nvidia Tesla V100 GPU through the lens of programmability, performance, and precision. In particular, they noted that the memory traffic has a high impact on the overall performance of the matrix multiplications despite the integration of L1 data cache and shared memory subsystem in the GPU. A related analysis is done by Sun et al. [12], which uses microbenchmarks to evaluate the Tensor Cores available on Ampere architecture (A100 GPU). They find little differences between operations in FP16 or FP32. Other works also explored new algorithms in precision refinement and parallel reduction on Tensor Cores [3]–[5].

Applications. As Tensor Cores and Matrix Cores are relatively new, related applications are scarce. An application for the convolutional neural network (Winograd algorithm) was optimized by Guo et al. [20] using MI210 GPUs, with a speedup of 1.2x in comparison to the AMD's MIOpen baseline. Feng et al. [21] performed a study using Tensor Cores together with a neural network to allow the Tensor Cores to use arbitrary precisions, aside int1 and int4, testing with DGEMM and convolution kernels. Chalmers et al. [22] use hipBone, a Computational Fluid Dynamics application based on Nek5000/NekRS, to leverage the fine-graining present on Nvidia Tesla V100, AMD MI100 and AMD MI250X.

IX. Discussion and Conclusions

In summary, we provide a timely characterization of the floating-point operations on Matrix Cores on AMD GPUs, a building block of Exascale supercomputers. Our micro-benchmarking results show that the high floating-point throughput from Matrix Cores, reaching 350, 88, and

69 TFLOPS for mixed, float, and double precision on one AMD GPU, can provide the much-needed throughput in HPC workloads, where floating-point matrix operations are ubiquitous. Our analysis and modeling of the power consumption on Matrix Cores at different activity levels further confirm the high power efficiency of double-precision operations and a potential 4×-8× further reduction in power consumption when switching to single or mixed precision. Finally, as wide adoption in real applications is likely through high-level programming interfaces, we evaluate the efficiency of the rocBLAS library in utilizing Matrix Cores. Our results show that the rocBLAS library can deliver near-peak floating-point throughput on Matrix Cores in single- and double-precision in applications with minimal porting efforts.

References

[1] "Top500 list," 2023. [Online]. Available: https://www.top500.org/lists/top500/2023/06/

[2] S. Markidis, S. Chien, E. Laure, I. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & precision," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2018, pp. 522–531. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPSW.2018.00091

[3] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, "Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 603–613.

[4] C. A. Navarro, R. Carrasco, R. J. Barrientos, J. A. Riquelme, and R. Vega, "GPU Tensor Cores for Fast Arithmetic Reductions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 72–84, Jan. 2021.

[5] A. Dakkak, C. Li, J. Xiong, I. Gelado, and W.-m. Hwu, "Accelerating reduction and scan using tensor core units," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 46–57.

[6] AMD, "Introducing AMD CDNA2 architecture," AMD, Whitepaper, 2021. [Online]. Available: https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

[7] ——, "AMD instinct MI100 instruction set architecture," 2020. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/instruction-set-architectures/instinct-mi100-cdna1-shader-instruction-set-architecture.pdf

[8] ——, "AMD instinct MI200 instruction set architecture," 2022. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/instruction-set-architectures/instinct-mi200-cdna2-instruction-set-architecture.pdf

[9] ——, "Amd matrix instruction calculator," 2023. [Online]. Available: https://github.com/RadeonOpenCompute/amd_matrix_instruction_calculator

[10] "LAPACK – Linear Algebra PACKage." [Online]. Available: https://www.netlib.org/lapack/

[11] AMD, "Amd instinct mi250x datasheet," 2021. [Online]. Available: https://www.amd.com/en/products/accelerators/instinct/mi200/mi250x.html

[12] W. Sun, A. Li, T. Geng, S. Stuijk, and H. Corporaal, "Dissecting tensor cores via microbenchmarks: Latency, throughput and numeric behaviors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, 10 2022. [Online]. Available: https://www.osti.gov/biblio/1902259

[13] AMD, "Hierarchical roofline on AMD Instinct MI200 gpus," https://www.olcf.ornl.gov/wp-content/uploads/AMD_Hierarchical_Roofline_ORNL_10-12-22.pdf, 2022.

[14] M. Leinhauser, R. Widera, S. Bastrakov, A. Debus, M. Bussmann, and S. Chandrasekaran, "Metrics and design of an instruction roofline model for amd gpus," *ACM Trans. Parallel Comput.*, vol. 9, no. 1, jan 2022. [Online]. Available: https://doi.org/10.1145/3505285

[15] T. Patki, Z. Frye, H. Bhatia, F. Di Natale, J. Glosli, H. Ingolfsson, and B. Rountree, "Comparing gpu power and frequency capping: A case study with the mummi workflow," in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, 2019, pp. 31–39.

[16] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson, "Power aware computing on gpus," in *2012 Symposium on Application Accelerators in High Performance Computing*, 2012, pp. 64–73.

[17] A. Hart, H. Richardson, J. Doleschal, T. Ilsche, M. Bieler, and M. Kappel, "User-level power monitoring and application performance on cray xc30 supercomputers," in *Proceedings of the Cray User Group (CUG)*, 2014.

[18] W. F. Godoy, P. Valero-Lara, T. Dettling, C. Trefftz, I. Jorquera, T. Sheehy, R. G. Miller, M. Gonzalez-Tallada, J. S. Vetter, and V. Churavy, "Evaluating performance and portability of high-level programming models: Julia, python/numba, and kokkos on exascale nodes," in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 373–382.

[19] D. Eberius, P. Roth, and D. M. Rogers, "Understanding strong scaling on gpus using empirical performance saturation size," in *2022 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 2022, pp. 26–35.

[20] Y. Guo, L. Lu, and S. Zhu, "Novel accelerated methods for convolution neural network with matrix core," *The Journal of Supercomputing*, pp. 1–27, 2023.

[21] B. Feng, Y. Wang, T. Geng, A. Li, and Y. Ding, "Apnn-tc: Accelerating arbitrary precision neural networks on ampere gpu tensor cores," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476157

[22] N. Chalmers, A. Mishra, D. McDougall, and T. Warburton, "Hipbone: A performance-portable graphics processing unit-accelerated c++ version of the nekbone benchmark," *The International Journal of High Performance Computing Applications*, vol. 37, no. 5, pp. 560–577, 2023. [Online]. Available: https://doi.org/10.1177/10943420231178552