General-purpose processing on AI/ML accelerators *

1. INTRODUCTION

Artificial intelligence (AI) and Machine learning (ML) accelerators are ubiquitous in modern computers to match the demand of their target workloads. Famous examples include Google's Tensor Processing Units (TPUs) [18], Apple's Neural Engines [3], and Intel's Gaussian & Neural Accelerators (GNAs). Though various in accelerated functions and microarchitectures, these AI/ML accelerators are essentially matrix processors that perform operations on matrices. Therefore, another trend in accelerating AI/ML applications is providing critical mathematical functions, particularly matrix multiplications, in computer systems. Emerging industry examples include NVIDIA's Tensor Core Units (TCUs) [16], IBM Power 10 MMA units [20], Intel's AMX [13] and ARM SME [4].

Beyond the core training and inferencing processes in AI/ML workloads, a broad spectrum of matrix-based problems can also benefit from these AI/ML accelerators in theory. In fact, many of these problems are responsible for providing inputs to AI/ML training/inferencing or serving as the post-processing steps for AI/ML models. With modern AI/ML accelerators shifted the most time-consuming part to non-training/inferencing processes [10] and the significant under-utilization of AI/ML accelerators [25], democratizing AI/ML accelerators to accelerate non-training/inferencing processes becomes critical to improve the overall platform performance.

In this project, we will present our efforts in democratizing AI/ML accelerators. The first work is TCUDB project published in SIGMOD 2022 [12]. In TCUDB, we demonstrate matrix-based solutions to several frequently used database query patterns. In addition to transforming the desired query into matrix operations, TCUDB's query engine carefully gauges the value ranges and data intensity to ensure the quality and performance of results. TCUDB evaluates three real-world use cases, (1) linear algebra (LA) queries, (2) entity matching (EM), and (3) graph analytics. TCUDB demonstrates an outstanding performance advantage over a GPU-based engine by achieving up to 382× speedup.

The second work is TensorCV. In TensorCV, we revisited the design of the frequently-used, performance-critical non-training/inferencing functions in modern AI/ML pipelines. TensorCV presents matrix/tensor-based algorithms to enable the use of AI/ML accelerators in critical image processing steps, including rotation, cropping, resizing, normalization, color space conversions. Though these matrix-based algorithms have potentially higher algorithmic complexity than existing solutions, these algorithms can still supersede the performance of optimized implementations on modern general-purpose processors since AI/ML accelerators can execute our underlying operations efficiently. TensorCV shows our algorithms and implementations can achieve 7.25× speedup over conventional GPU-based implementations on the same architecture.

*The submission is based on SIMD^2: A Generalized Matrix Instruction Set for Accelerating Tensor Computation beyond GEMM. In The 49th Annual International Symposium on Computer Architecture (ISCA'22) and TCUDB: Accelerating Database with Tensor Processors (SIGMOD'22)

2. TCUDB

TCUDB is an analytic database query engine that explores opportunities of integrating AI/ML accelerators, particularly Tensor Core Units (TCUs) into a database engine's architecture. The increasing demand for native support of linear algebra queries (e.g., matrix multiplication itself) in SQL DB engines [11, 1, 8, 15, 7] and the observation that a large number of regular query operators can be cast into matrix multiplications, have made perfect sense to use TCUs for these database workloads. For example, one can show that the most commonly used natural joins [2, 6] and group-by aggregates can be encoded as matrix multiplications and enable TCUs to deliver exceptional performance.

Figure 1 provides an overview of the system architecture of TCUDB. TCUDB extends the common architecture of GPU-accelerated databases [9, 21, 23, 26, 5, 22, 24, 14, 17, 19] as a way to further accommodate executing query operators with TCU acceleration in the query analyzer, the query optimizer, the code generator, and the program driver.

To address the challenge of executing queries using matrix operations, we re-engineered a set of query operators that are theoretically feasible to be mapped to tensor/matrix algebra operations for TCUDB. The query operators cover many commonly used ones, including natural joins and group-by aggregates. TCUDB also features a code generator for generating executable code mapping input tables to tensor format and processes the query as matrix multiplication via WMMA or cuBLAS API calls. Depending on the data sparsity, TCUDB provides the option of sparse tensor encoding with sparse matrix multiplication. We developed the TCU-SpMM operator to support sparse matrix multiplication with TCU acceleration. Then, the TCUDB query analyzer can generate query plans that use these TCU-accelerated physical operators.

To resolve the challenge of limited precision and overhead in modern tensor processors, TCUDB's query optimizer carefully gauges the parameters in precision, data movement overhead, data transformation overhead, and computation throughput — as using lower data precision yields lower data movement overhead and higher computation throughput, but also take higher risks of leading to unacceptable answers as well as higher data transformation overhead. TCUDB presents an adaptive mixed-precision query optimization that dynamically selects the most appropriate precision in delivering the desired level of accuracy using the shortest end-to-end latency to handle queries.

We demonstrate the significant performance gain of TCUDB in a range of real-world applications, including entity matching, graph query processing, and matrix-based data analytics. Figure 2 presents the experimental result of running entity matching queries on iTunes-Amazon datasets. TCUDB achieves up to 288× speedup compared to a baseline GPU-based query engine (YDB). Please reference the TCUDB for more results [12].

3. TENSORCV

The non-AL/ML parts of the computer vision pipeline typically perform operations that help to enhance or extract the most critical part of images to enable more accurate and efficient ML results. Unfortunately, the conventional approach typically relies on CPU code whose performance can only scale with the relatively slow improvement through

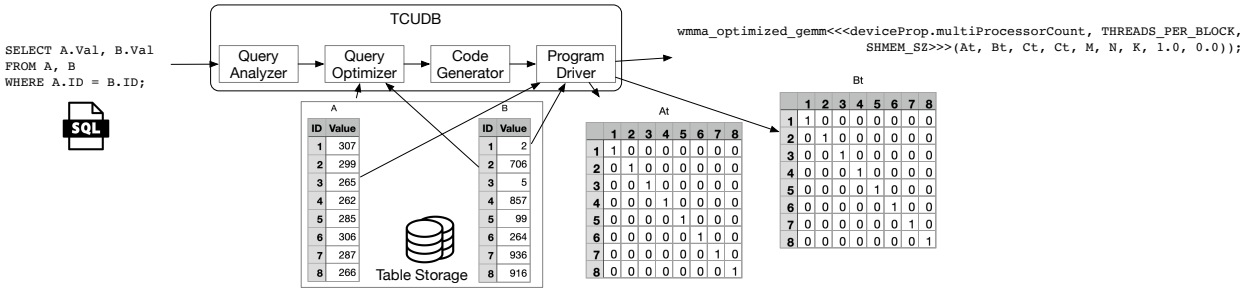

Figure 1: An overview of TCUDB's workflow.

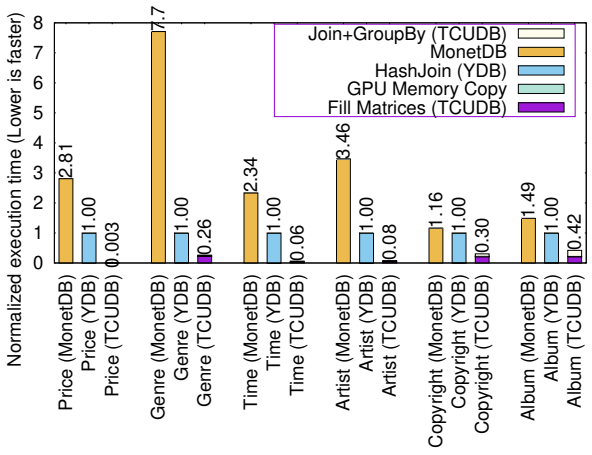

Figure 2: The relative runtime of the EM-blocking queries on TCUDB using the default deepmatcher-iTunes-Amazon datasets compared to MonetDB and YDB running the same query as the baseline.

Moore's Law, but not the rapid growth from emerging, innovative hardware accelerators. The inefficiency of these ML-adjacent stages will lead to under-utilized hardware accelerators and becomes the performance bottleneck as these stages cannot feed sufficient inputs to well-optimized ML models.

TensorCV aims at improving the under-rated performance issues in frequently used image processing functions in AI/ML-assisted computer-vision pipelines. Similar to the philosophy of TCUDB, the most critical task in TensorCV is revisiting entrenched implementations that optimize for conventional CPU/GPU models and designing algorithms that use matrix operations. In the following paragraphs, we will use the "resizing" function as an example of transforming from scalar algorithms to matrix-based algorithms.

In the conventional bi-linear resizing implementation, first, the code computes the relative ratio between input and output matrix sizes, which we call row and column scales. Then, the code calculates each output pixel values. However, this algorithm cannot exploit Tensor Cores since there is no matrix multiplications.

TensorCV transforms the conventional implementation into two matrix multiplications. Considering an m-by-$3n$ input and m'-by-$3n'$ output images, the proposed algorithm creates an m'-by-m matrix as L^{resize} and $3n$-by-$3n'$ matrix as R^{resize}. The content of L^{resize} and R^{resize} is only related to the target image's size; therefore, TensorCV only needs to create them once for every batch. In contrast, the conventional approach prevents the code from using Tensor Cores and recalculates weights for the same pixel position in each channel. The proposed algorithm will fill the content of L^{resize}

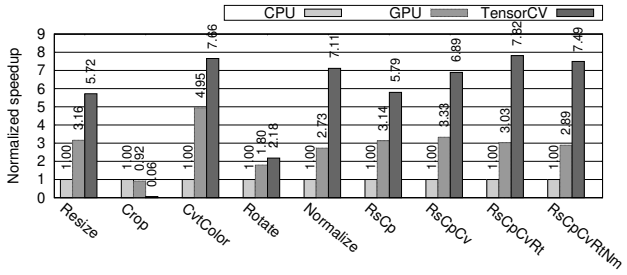

Figure 3: Performance of TensorCV compared with baseline OpenCV implementation using CPUs and GPUs

using the following formula:

$$L^{resize}_{m\times m'} \ni l^{resize}_{i,j} = \begin{cases} 1 - rowWeight_j & \text{if } \ j = top_i \\ rowWeight_j & \text{if } \ j = bot_i \\ 0 & \text{else} \end{cases} \quad (1)$$

R^{resize} holds column-related weights, and the proposed algorithm will fill the content of the matrix using the following formula:

$$R^{resize}_{3n\times 3n'} \ni r^{resize}_{i,j} = \begin{cases} 1 - colWeight_i & \text{if } \begin{array}{l} \lfloor i/3 \rfloor = left_i \\ \text{and } i\%3 = j\%3 \end{array} \\ colWeight_i & \text{if } \begin{array}{l} \lfloor i/3 \rfloor = right_i \\ \text{and } i\%3 = j\%3 \end{array} \\ 0 & \text{else} \end{cases} \quad (2)$$

After filling matrices L^{resize} and R^{resize} using Equation 1 and 2 at the beginning of each batch of images, the proposed algorithm can compute the output of each image resizing result as the following.

$$Output_{m'\times 3n'} = L^{resize}_{m'\times m} \cdot Input_{m\times 3n} \cdot R^{resize}_{3n\times 3n'} \quad (3)$$

Figure 3 compares the performance of TensorCV with conventional OpenCV implementations that can only use CPUs or CUDA cores. The baseline of Figure 3 is the CPU-based implementation. TensorCV's algorithm achieves up to 7.82× speedup in the RsCpCvRt function. However, using geometric-mean that discount the outliers, TensorCV still obtains 7.6× speedup on average. In contrast, on average, OpenCV-CUDA that uses GPU/CUDA cores can only speed up by 2×.

Another advantage of TensorCV is the ability to optimize across functions and combine several matrix multiplications into fewer ones. We presented four different use cases that combine multiple pre-processing functions. Compared with other implementations, TensorCV achieves 7.25× geometric mean in speedup. However, the performance is limited in the conventional GPU implementation, where cross-function optimization is complicated.

4. REFERENCES

[1] Christopher Aberger, Andrew Lamb, Kunle Olukotun, and Christopher Ré. Levelheaded: A unified engine for business intelligence and linear algebra querying. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 449–460. IEEE, 2018.

[2] Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *Proceedings of the 12th International Conference on Database Theory*, pages 121–126. Association for Computing Machinery, 2009.

[3] Apple Inc. Apple M1. `https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/`, 11 2020.

[4] Arm Corporation. Introducing the Scalable Matrix Extension for the Armv9-A Architecture. `https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/scalable-matrix-extension-armv9-a-architecture`, 2021.

[5] Sebastian Breß and Gunter Saake. Why it is time for a hype: A hybrid query processing engine for efficient gpu coprocessing in dbms. *Proc. VLDB Endow.*, 6(12):1398–1403, 2013.

[6] Shaleen Deep, Xiao Hu, and Paraschos Koutris. Fast join project query evaluation using matrix multiplication. In *SIGMOD*, pages 1213–1223. Association for Computing Machinery, 2020.

[7] Oksana Dolmatova, Nikolaus Augsten, and Michael H Böhlen. A relational matrix algebra and its implementation in a column store. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2573–2587, 2020.

[8] Joseph Vinish D'silva, Florestan De Moor, and Bettina Kemme. AIDA: Abstraction for advanced in-database analytics. *PVLDB*, 11(11):1400–1413, 2018.

[9] Naga K Govindaraju, Brandon Lloyd, Wei Wang, Ming Lin, and Dinesh Manocha. Fast computation of database operations using graphics processors. In *SIGMOD*, pages 215–226. ACM, 2004.

[10] Dongho Ha, Won Woo Ro, and Hung-Wei Tseng. Accelerating ML-adjacent Computation Using Tensor Processors. In *The 2023 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED 2023, 2023.

[11] Joseph M Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, et al. The madlib analytics library. *Proceedings of the VLDB Endowment*, 5(12), 2012.

[12] Yu-Ching Hu, Yuliang Li, and Hung-Wei Tseng. TCUDB: Accelerating Database with Tensor Processors. In *the 2022 ACM SIGMOD/PODS International Conference on Management of Data*, SIGMOD 2022, 2022.

[13] Intel Corporation. Intrinsics for Intel(R) Advanced Matrix Extensions (Intel(R) AMX) Instructions. `https://software.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/compiler-reference/intrinsics/intrinsics-for-intel-advanced-matrix-extensions-intel-amx-instructions.html`, 2021.

[14] Jing Li, Hung-Wei Tseng, Chunbin Lin, Yannis Papakonstantinou, and Steven Swanson. Hippogriffdb: Balancing i/o and gpu bandwidth in big data analytics. *PVLDB*, 9(14):1647–1658, 2016.

[15] S. Luo, Z. J. Gao, M. Gubanov, L. L. Perez, and C. Jermaine. Scalable linear algebra on a relational database system. *IEEE Transactions on Knowledge and Data Engineering*, 31(7):1224–1238, 2019.

[16] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 522–531. IEEE, 2018.

[17] Johns Paul, Jiong He, and Bingsheng He. GPL: A GPU-based pipelined query processing engine. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1935–1950, 2016.

[18] Kaz Sato, Cliff Young, and David Patterson. An in-depth look at google's first tensor processing unit (TPU). *Google Cloud Big Data and Machine Learning Blog*, 12, 2017.

[19] Anil Shanbhag, Samuel Madden, and Xiangyao Yu. A study of the fundamental performance characteristics of gpus and cpus for database analytics. In *SIGMOD*, pages 1617–1632, 2020.

[20] Brian W Thompto, Dung Q Nguyen, José E Moreira, Ramon Bertran, Hans Jacobson, Richard J Eickemeyer, Rahul M Rao, Michael Goulet, Marcy Byers, Christopher J Gonzalez, et al. Energy Efficiency Boost in the AI-Infused POWER10 Processor. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[21] Slawomir Walkowiak, Konrad Wawruch, Marita Nowotka, Lukasz Ligowski, and Witold Rudnicki. Exploring utilisation of gpu for database applications. *Procedia Computer Science*, 1(1):505–513, 2010.

[22] Kaibo Wang, Kai Zhang, Yuan Yuan, Siyuan Ma, Rubao Lee, Xiaoning Ding, and Xiaodong Zhang. Concurrent analytical query processing with gpus. *VLDB*, 7(11):1011–1022, 7 2014.

[23] Haicheng Wu, Gregory Diamos, Srihari Cadambi, and Sudhakar Yalamanchili. Kernel weaver: Automatically fusing database primitives for efficient gpu computation. In *MICRO*, pages 107–118. IEEE Computer Society, 2012.

[24] Haicheng Wu, D. Zinn, M. Aref, and S. Yalamanchili. Multipredicate join algorithms for accelerating relational graph processing on gpus. In *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, 09 2014.

[25] Abenezer Wudenhe and Hung-Wei Tseng. TPUPoint: Automatically Characterizing Hardware Accelerated Data Center Machine Learning Program Behavior. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS 2021, 2021.

[26] Yuan Yuan, Rubao Lee, and Xiaodong Zhang. The yin and yang of processing data warehousing queries on gpu devices. *Proceedings of the VLDB Endowment*, 6(10):817–828, 2013.