Brief Announcement: A Computational Model for Tensor Core Units

Rezaul Chowdhury
Stony Brook University
US
rezaul@cs.stonybrook.edu

Francesco Silvestri
University of Padova
Italy
silvestri@dei.unipd.it

Flavio Vella
Free University of Bolzen-Bolzano
Italy
flavio.vella@unibz.it

ABSTRACT

To respond to the need for efficient training and inference of deep neural networks, a plethora of domain-specific architectures have been introduced, such as Google Tensor Processing Units and NVIDIA Tensor Cores. A common feature of these architectures is the design for efficiently computing a dense matrix product of a given small size. In order to broaden the class of algorithms that exploit these systems, we propose a computational model, named the *TCU model*, that captures the ability to natively multiply small matrices. We then use the TCU model for designing fast algorithms for several problems, including dense and sparse matrix multiplication and the Discrete Fourier Transform. We finally highlight a relation between the TCU model and the external memory model.

CCS CONCEPTS

• **Theory of computation** → **Models of computation**; **Design and analysis of algorithms**.

KEYWORDS

Tensor core, computational model, hardware accelerators, efficient algorithms, linear algebra, graph problems

ACM Reference Format:
Rezaul Chowdhury, Francesco Silvestri, and Flavio Vella. 2020. Brief Announcement: A Computational Model for Tensor Core Units. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3350755.3400252

1 INTRODUCTION

Deep neural networks are nowadays used in several application domains where big data are available. The huge size of the data set, although crucial for improving neural network quality, gives rise to performance issues during the training and inference steps. In response to the increasing computational needs, several notable domain-specific hardware accelerators have been recently introduced, such as Google's Tensor Processing Units [5] and NVIDIA's Tensor Cores [6]. These compute units have been specifically designed for accelerating deep learning. Although such accelerators significantly vary in their design, they share circuits for efficiently multiplying small and dense matrices of fixed size, which is one of the most important computational primitives in deep learning. By using the terminology introduced in [3], we refer to all accelerators supporting hardware-level dense matrix multiplication as *Tensor Core Units (TCUs)* (or simply tensor units). By focusing on a specific computational problem, namely matrix multiplication, TCUs simultaneously exhibit both high performance and low energy consumption which set them apart from traditional CPU or GPU approaches [5]. Although TCUs were developed for domain-specific problems, it would be interesting and profitable to extend their application domain, for instance by targeting problems from linear algebra to graph analytics. A similar scenario appeared with the introduction of GPUs for general purpose computations. *Will TCUs have the same wide impact as GPUs?*

The goals of this paper are to present a framework for designing and analyzing efficient algorithms for TCUs, and to expand the class of algorithms that can exploit TCUs for better performance. We introduce a computational model for tensor core units, named (m, ℓ)*-TCU*, that captures the main features of tensor units. We then design TCU algorithms for dense and sparse matrix multiplication and for the Discrete Fourier Transform. Other algorithms, including graph algorithms, a class of stencil computations, integer multiplication, and polynomial evaluation are given in the extended version of this paper [2] (see Table 1 below). Finally, we observe that some lower bounds on the I/O complexity in the external-memory model [9] translate into lower bounds on TCU time.

Problem	**Time in (m, ℓ)-TCU**
Dense matrix multiplication	$O\left(\left(\frac{n}{m}\right)^{\omega_0}(m+\ell)\right)$
Sparse matrix multiplication	$O\left(\sqrt{\frac{n}{Z}}\left(\frac{Z}{m}\right)^{\omega_0}(m+\ell)+I\right)$
FFT	$O\left((n+\ell)\log_m n\right)$
(n, k)-Stencil computation	$O\left(n\log_m k+\ell\log k\right)$
Gaussian Elimination Paradigm	$\Theta\left(\frac{n^{3/2}}{m^{1/2}}+\frac{n}{m}\ell+n\sqrt{m}\right)$
Graph transitive closure	$\Theta\left(\frac{n^3}{\sqrt{m}}+\frac{n^2}{m}\ell+n^2\sqrt{m}\right)$
All pairs shortest distance	$O\left(\left(\frac{n^2}{m}\right)^{\omega_0}(m+\ell)\log n\right)$
Integer multiplication	$O\left(\left(\frac{n}{\kappa\sqrt{m}}\right)^{\log 3}\left(\sqrt{m}+\frac{\ell}{\sqrt{m}}\right)\right)$
Batch polynomial evaluation	$O\left(\frac{pn}{\sqrt{m}}+p\sqrt{m}+\frac{n}{m}\ell\right)$

Table 1: Our results (details are in [2]).

2 THE (m, ℓ)-TCU MODEL

We propose a computational model for tensor core units that captures the main features of tensor units.

Matrix acceleration. The hardware circuits implement a parallel algorithm to multiply two matrices of a fixed size, and the main cost is dominated by reading/writing the input and output matrices. For a given hardware parameter m, the multiplication of two matrices A

SPAA '20, July 15–17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-6935-0/20/07.
https://doi.org/10.1145/3350755.3400252

and B of size $\sqrt{m}\times\sqrt{m}$ each is implemented to execute in time $O(m)$. With time, we mean the running time as seen by the CPU clock and it should not be confused with the total number of operations executed by the unit, which is always $\Theta\left(m^{3/2}\right)$. Indeed, no existing tensor unit implements fast matrix multiplication algorithms, such as Strassen's. The matrix multiplication operation is called by an instruction specifying the addresses (in memory) of the two input matrices and of the output matrix where the result will be stored; data will be loaded/stored by the tensor unit.

Latency cost. A call to the tensor unit has a latency cost. As the state-of-the-art tensor units use systolic algorithms, the first output entry is computed after $\Omega\left(\sqrt{m}\right)$ time. There are also initial costs associated with activation, which can significantly increase when the unit is not connected to the CPU by the internal system bus or is shared with other CPUs. We thus assume that the cost of the multiplication of two matrices of size $\sqrt{m}\times\sqrt{m}$ is $O(m+\ell)$, where $\ell \geq 0$ is the latency cost.

Asymmetric behavior. As tensor units are designed for deep learning, the two matrices in the product $A \times B$ are managed differently. Matrix B represents the model (i.e., the weights of the deep neural network), while the rows of matrix A represent the input vectors to be evaluated. As the same model can be applied to k vectors, with $n >> \sqrt{m}$, it is possible to first load the weights in B and then to stream the n rows of A into the tensor unit, and thus reducing the latency cost. Hence, we assume in our model that two matrices of size $n \times \sqrt{m}$ and $\sqrt{m} \times \sqrt{m}$ are multiplied in time $O\left(n\sqrt{m}+\ell\right)$, where the number n of rows is specified by the algorithm and $n \geq \sqrt{m}$.

We define the *Tensor Computing Unit (TCU) model* as follows. The (m,ℓ)-*TCU* model is a standard RAM model where the CPU contains a circuit, named tensor unit, for performing a matrix multiplication $A \times B$ of size $n \times \sqrt{m}$ and $\sqrt{m} \times \sqrt{m}$ in time $O\left(n\sqrt{m}+\ell\right)$, where $m \geq 1$ and $\ell \geq 0$ are two model parameters and $n \geq \sqrt{m}$ is a value (possibly input dependent) specified by the algorithm. The matrix operation is initialized by a (constant size) instruction containing the addresses in memory of the two input matrices A and B, of the output matrix C, and the row number n of A. The *running time* (or simply time) of a TCU algorithm is given by the total cost of all operations performed by the CPU, including all calls to the tensor unit. We assume no concurrency between the tensor unit, memory and the CPU, and hence at most one component is active at any time. Each memory (and TCU) word consists of κ bits (in general, we denote $\kappa = \Omega(\log n)$ where n is the input size, that is enough for storing the input size in one word.)

2.1 Discussion on the model

Our goal is to understand how to exploit circuits of fixed size for matrix multiplication without including some characteristics of existing hardware accelerators (e.g., number of TUs or numerical precision). In the Google TPU, the right matrix B has size 256×256 words (i.e., $m = 65536$) [5]. The left matrix A is stored in the local unified buffer of $96k \times 256$ words; thus, TPUs can compute the product between two matrices of size 96k×256 and 256×256 in one (tensor) operation. The systolic array works in low precision with 8 bits per word ($\kappa = 8$). The bandwidth between CPU and TPU was limited in the first version (16GB/s), but it is significantly higher in more recent versions (up to 600 GB/s). Although TPU has a quick response time, the overall latency is high because the right hand matrix has to be suitably encoded via a TensorFlow function before loading it within the TPU. The high latency cost might mitigate the fact that our model does not capture limited bandwidth.

The Nvidia programming model allows multiplying matrices of size 16×16, although the hardware unit works on 4×4 matrices; we thus have $m = 256$. Memory words are of $\kappa = 16$ bits. Matrices can be loaded within TCs without a special encoding as in TPUs, since NVIDIA Volta natively provides support for matrix multiplication.

3 MATRIX MULTIPLICATION ALGORITHMS

Dense matrix multiplication. A Strassen-like algorithm for matrix multiplication is a recursive algorithm that utilizes as base case an algorithm $\mathcal{A}$ for multiplying two $\sqrt{n_0} \times \sqrt{n_0}$ matrices using p_0 element multiplications and $O(n_0)$ other operations (i.e., additions and subtractions) [1]; we assume $n_0 = O(p_0)$. Given two $\sqrt{n} \times \sqrt{n}$ matrices with $n > n_0$, a Strassen-like algorithm envisions the two $\sqrt{n} \times \sqrt{n}$ matrices as two matrices of size $\sqrt{n_0} \times \sqrt{n_0}$ each, where each entry is a submatrix of size $\sqrt{n/n_0}\times\sqrt{n/n_0}$: then, the algorithm recursively computes p_0 matrix multiplications on the submatrices (i.e., the p_0 element multiplications in $\mathcal{A}$) and then performs $O(n)$ other operations. For given parameters p_0 and n_0, the running time of the algorithm is $T(n) = O(n^{\omega_0})$, where[1] $\omega_0 = \log_{n_0} p_0$. By setting $n_0 = 4$ and $p_0 = 8$, we get the standard matrix multiplication algorithm ($\omega_0 = 3/2$), while with $n_0 = 4$ and $p_0 = 7$ we get Strassen's algorithm ($\omega_0 = \log_4 7 \sim 1.403$). Any fast matrix multiplication algorithm can be converted into a Strassen-like algorithm [7].

The TCU model can be exploited in Strassen-like algorithms by ending the recursion as soon as a subproblem fits the tensor unit: when $n \leq m$, the two input matrices are loaded into the tensor unit and their product is computed in $O(m)$ time. We assume $m \geq n_0$, otherwise the tensor unit would not be used.

Theorem 1. *Given a Strassen-like algorithm with parameters n_0 and p_0, there exists an algorithm that multiplies two $\sqrt{n}\times\sqrt{n}$ matrices on an (m,ℓ)-TCU model, with $m \geq n_0$, in $O\left(\left(\frac{n}{m}\right)^{\omega_0}(m+\ell)\right)$ time.*

The standard recursive matrix multiplication algorithm requires $O\left(n^{3/2}/m^{1/2} + (n/m)^{3/2}\ell\right)$ time. On the other hand, Strassen's algorithm takes $O\left(n^{1.4037}/m^{0.4037} + (n/m)^{1.4037}\ell\right)$ time.

We observe that the latency cost of the standard algorithm ($\omega_0 = 3/2$) can be further reduced to $(n/m)^{3/2}\ell$. The idea is to keep the right matrix B inside the tensor unit as much as possible. We split the left matrix A into $\sqrt{n/m}$ blocks A_i of size $\sqrt{n} \times \sqrt{m}$ (i.e., vertical strips of width $\sqrt{m}$) each and the right matrix B into square blocks $B_{i,j}$ of size $\sqrt{m}\times\sqrt{m}$ each, with $0 \leq i,j < \sqrt{n/m}$. Then, we compute $C_{i,j} = A_i \cdot B_{i,j}$ for each $0 \leq i,j < \sqrt{n/m}$ using the tensor unit in time $O\left(n\sqrt{m}+\ell\right)$. The final matrix C follows by computing the $\sqrt{n} \times \sqrt{m}$ submatrices $C_i = \sum_{j=0}^{\sqrt{n/m}-1} C_{i,j}$.

Theorem 2. *There exists an algorithm that multiplies two $\sqrt{n} \times \sqrt{n}$ matrices in the (m,ℓ)-TCU model in $\Theta\left(\frac{n^{3/2}}{m^{1/2}} + \frac{n}{m}\ell\right)$ time. The algorithm is optimal when only semiring operations are allowed.*

[1] We observe that ω_0 corresponds to $\omega/2$, where ω is the traditional symbol used for denoting the exponent in fast matrix multiplication algorithms.

Sparse matrix multiplication. A TCU algorithm to multiply two sparse matrices follows from the work [4] that uses as a black box a fast matrix multiplication algorithm for multiplying two $\sqrt{n} \times \sqrt{n}$ matrices in $O\left(n^{\omega/2}\right)$ time. Let Z be the number of non-zero entries in the output $C = A \cdot B$. We consider here the case where the output is balanced, that is there are $\Theta\left(Z/\sqrt{n}\right)$ non-zero entries in each row or column of C. The idea is to compress the rows of A and the columns of B from $\sqrt{n}$ to $\sqrt{Z}$ using a hash function or another compression algorithm to re-order the matrix A. Then the algorithm computes a dense matrix product between a $\sqrt{Z} \times \sqrt{n}$ matrix and a $\sqrt{n} \times \sqrt{Z}$ using the fast matrix multiplication algorithm. By replacing the fast matrix multiplication with the result of Theorem 1, we get the following:

Theorem 3. *Let A and B be two $\sqrt{n} \times \sqrt{n}$ matrices with at most I non-zero entries, and assume that $C = A \cdot B$ has at most Z non-zero entries evenly distributed among its rows and columns. Let $Z \geq m$ and let $\omega_0 = \log_{n_0} p_0$ be the exponent given by a Strassen-like algorithm; then there exists an algorithm for the (m, ℓ)-TCU model requiring $O\left(\sqrt{\frac{n}{Z}}\left(\frac{Z}{m}\right)^{\omega_0}(m+\ell) + I\right)$ time.*

4 DISCRETE FOURIER TRANSFORM

The Discrete Fourier Transform y of an n-dimensional (column) vector x can be defined as the matrix-vector product $y = x^T \cdot W$, where W is the Fourier matrix and T denotes the transpose of a matrix/vector. The Fourier matrix W is a symmetric $n \times n$ matrix where the entry at row r and column c is defined as: $W_{r,c} = e^{-(2\pi i/n)rc}$. The Cooley-Tukey algorithm is an efficient recursive algorithm for computing the DFT of a vector. The algorithm arranges x as an $n_1 \times n_2$ matrix X (in row-major order) where $n = n_1 \cdot n_2$; each column $X_{*,c}$ is replaced with its DFT and then each entry $X_{r,c}$ is multiplied by the twiddle factor w_n^{rc}; finally, each row $X_{r,*}$ is replaced by its DFT and the DFT of x is given by reading the final matrix X in column-major order.

To compute the DFT of x using an (m, ℓ)-TCU, we use the Cooley-Tukey algorithm where we set $n_1 = \sqrt{m}$ and $n_2 = n/\sqrt{m}$. We then use the tensor unit for computing the n_2 DFTs of size $n_1 = \sqrt{m}$ by computing $X^T \cdot W_{\sqrt{m}}$. Subsequently, we multiply each element in X by its twiddle factor and transpose X. Finally, we compute the n_1 DFTs of size n_2: if $n_2 > \sqrt{m}$, the DFTs are recursively computed; otherwise, if $n_2 \leq \sqrt{m}$, the n_1 DFTs are computed with the product $X^T \cdot W_{n_2}$ obtained using the tensor unit. For simplicity, we assume that the TCU model can perform operations on complex numbers (the assumption can be removed with a constant-factor slowdown).

Theorem 4. *The DFT of a vector with n entries can be computed on an (m, ℓ)-TCU in $O\left((n+\ell)\log_m n\right)$ time.*

We observe that the DFT algorithm generalizes the approach used in [8] on an NVIDIA Verdi architecture, which uses a Cooley-Tukey algorithm with $n_1 = 4$ and $n_2 = n/4$.

5 IMPACT ON EXTERNAL MEMORY MODEL

The time complexity of some of our TCU algorithms resemble the I/O complexity of the corresponding external-memory algorithms. For instance, the cost of dense matrix multiplication on an (m, ℓ)-TCU using only semiring operations (Theorem 2) is $O\left(n^{3/2}/\sqrt{m}\right)$ when $\ell = O(1)$, while the I/O complexity of computing the same dense matrix product in the (M, B) external-memory model is $O\left(n^{3/2}/\sqrt{M}\right)$ when $B = O(1)$ [9].

We observe that computing the product of two matrices of size $\sqrt{m} \times \sqrt{m}$ each requires $O(m)$ I/Os to load and store the input matrices in an internal memory of size $M = 3m$ and block size $B = O(1)$. Therefore any call to the tensor unit in a TCU can be simulated in the external memory of size $M = 3m$ with $\Theta(m)$ I/Os. Therefore, a lower bound in the external-memory model translates into a lower bound in a weaker version of the TCU model. In the *weak TCU model*, the tensor unit can only multiply matrices of size $\sqrt{m} \times \sqrt{m}$. All our TCU algorithms can be simulated in the weak version with a constant-factor slowdown when $\ell = O(m)$. We have the following result:

Theorem 5. *Consider a computational problem $\mathcal{P}$ with a lower bound $F_{\mathcal{P}}$ on the I/O complexity in an external memory with memory size $M = 3m + O(1)$ and block length $B = 1$. Then, any algorithm for $\mathcal{P}$ in the weak TCU model requires $\Omega(F_{\mathcal{P}})$ time.*

6 CONCLUSION

The paper leaves several open questions. It would be interesting to extend the class of problems that can be accelerated with TCUs, and to analyze whether existing algorithms for deep learning on tensor cores can be further improved. It is also crucial to validate the TCU model from an experimental point of view, and to extend the model by including parallel tensor accelerators and the low numerical precision.

ACKNOWLEDGMENTS

This work was partially supported by NSF grant CNS-1553510, UniPD SID18 grant, PRIN17 20174LF3T8 AHeAd, UniBZ-CRC 2019-IN2091 Project, and INdAM-GNCS Project 2020 NoRMA. Some results are based upon work performed at the AlgoPARC Workshop on Parallel Algorithms and Data Structures at the University of Hawaii at Manoa, in part supported by the NSF Grant CCF-1930579.

REFERENCES

[1] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. *J. ACM*, 59(6):32:1–32:23, 2013.
[2] R. A. Chowdhury, F. Silvestri, and F. Vella. A computational model for tensor core units, 2020. Arxiv 1908.06649.
[3] A. Dakkak, C. Li, J. Xiong, I. Gelado, and W.-M. Hwu. Accelerating reduction and scan using tensor core units. In *Proc. Int. Conf. on Supercomputing (ICS)*, pages 46–57, 2019.
[4] R. Jacob and M. Stöckel. Fast output-sensitive matrix multiplication. In *Proc. European Symposium on Algorithms (ESA)*, pages 766–778, 2015.
[5] N. P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. 44th Int. Symposium on Computer Architecture (ISCA)*, pages 1–12, 2017.
[6] Nvidia Tesla V100 GPU architecture. http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.
[7] R. Raz. On the complexity of matrix product. *SIAM Journal on Computing*, 32(5):1356–1369, 2003.
[8] A. Sorna, X. Cheng, E. D'Azevedo, K. Won, and S. Tomov. Optimizing the fast fourier transform using mixed precision on tensor core hardware. In *Proc. 25th Int. Conf. on High Performance Computing Workshops (HiPCW)*, pages 3–7, 2018.
[9] J. S. Vitter. Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006.