Tensor Core accelerated Database Operations*

*Note: Sub-titles are not captured in Xplore and should not be used

1st Akshaya Bindu Gowri
dept. name of organization (of Aff.)
Otto von Guericke University
Magdeburg, Germany
email address or ORCID

2nd Maneendra Hetti Perera
dept. name of organization (of Aff.)
Otto von Guericke University
Magdeburg, Germany
email address or ORCID

3rd Vegenshanti Valerian Dsilva
dept. name of organization (of Aff.)
Otto von Guericke University
Magdeburg, Germany
email address or ORCID

4th Akshata Balasaheb Bobade
dept. name of organization (of Aff.)
Otto von Guericke University
Magdeburg, Germany
email address or ORCID

5th Abhishek Digvijay Singh
dept. name of organization (of Aff.)
Otto von Guericke University
Magdeburg, Germany
email address or ORCID

6th Ayushi Dinesh Dani
dept. name of organization (of Aff.)
Otto von Guericke University
Magdeburg, Germany
email address or ORCID

***Abstract*—**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The current era is of Big Data where we have humongous amounts of data being collected and stored at a great speed everyday. Data is collected from multiple sources such as transactions, social media, electronic devices connected to the Internet etc. While accessing and processing this data efficiently is a challenge, research on special hardware to manage data has been ongoing[1]. We turn towards the use of matrix multiplication accelerator hardware in various deep learning architectures to speed up the performance. One such hardware unit is the Tensor Core which uses the mixed precision computing to accelerate throughput and at the same time maintain the accuracy[11] .In this paper we aim to use Tensor Cores to implement database operations such as Select-Where and Join operation.

A Tensor Core performs FP16 matrix multiplication and a FP16 or FP32 accumulate in a single step[9]. Currently, Tensor Cores are offered with the NVIDIA A100[11], NVIDIA Turing[11] and NVIDIA Volta [11] processors.

Tensor Cores can only be used at the moment using the GEMM algorithm. Our challenge here is to implement the aforementioned database operations using the GEMM algorithm.

II. Related Work

More than the Tensor Processing Unit, the capabilities of Graphics processing unit have been evaluated more often for accelerating database related operations. One recent work by Pedro and Hannes[8] that evaluated Tensor Processing Unit capabilities to speed up database operations, incorporated mapping relational operators to Tensorflow operations that are backed by TPUs. This study resulted in a conclusion that in comparison to GPUs, TPUs don't give a significant speed up for the relational query processing. The results of this study were not very lucid as several factors, such as, hardware session management, execution plan generation and data transfer were not taken into consideration.

In yet another research[6], the capabilities of Tensor Cores to perform matrix multiplication has been evaluated. This study discusses the pros and cons of performing operations in Tensor Cores, taking an advantage of the fact that TPUs are indeed performance boosters, while also considering that only matrix multiplication operations on comparatively smaller matrices is supported. Which implies that all kinds of operations have to be eventually converted into some form of matrix multiplication in order to utilise the performance boosting potential of Tensor Cores. NVIDIA's V100 TCUs have been used to perform these experiments. In addition to performance boosting, a decrease in power consumption has also been recorded while performing reduction and scan primitives.

A research by Roberto, Vega and Navarro[2], discusses the Tensor Processing Unit's capabilities to perform faster reductions. By "reduction", we imply at reduction of n number as a set of m x m MMA Tensor Core operations. This experiment compared TPU based reduction operations with the classic parallel GPU reduction. There was again a very significant performance improvement observed, however, the amount of precision loss for FP16 hasn't been quantified yet.

The join operations have been evaluated mainly using the Graphics Processing Unit. A research on Relational Joins on Graphic Processors[7], takes advantage of the new features of GPU. A set of data parallel primitives - split and sort, have been used to implement indexed or non indexed

Identify applicable funding agency here. If none, delete this.

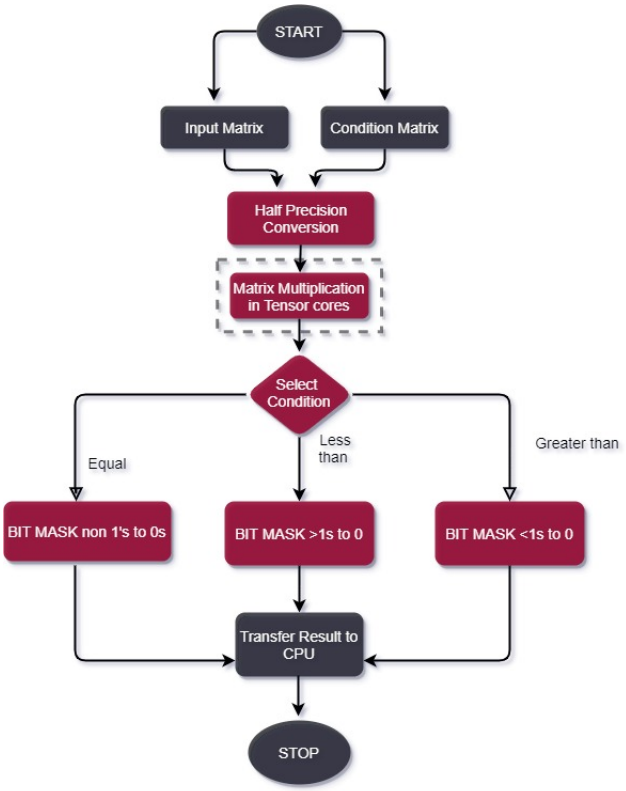

Figure 1: Select -Where Operation

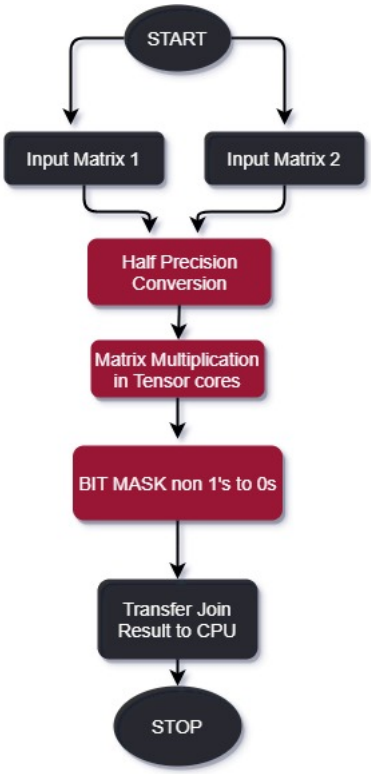

Figure 2: Join Operation

nested-loop, sort-merge and hash joins. Operations like Map, Scatter/Gather, Prefix Scan and Split are utilised to perform joins. As a result of this study, a performance improvement of around 10% was observed with respect to searching for the data in local memory, which is one of the main steps involved in Join operations. This was just one part of the complete join operation that showed significant performance improvement. The complete GPU based join operation could achieve a speedup of 2-27X in contrast with its optimized CPU based counterparts

III. Body

A. Conceptual flow

This section describes the conceptual flow for our program. Figure 1 and figure 2 show the flow chart for the Select-Where and Join operation. The steps executed in CPU and GPU have been marked with grey and pink colours respectively.

- Initially we have our input matrix and the relational operation that we want to perform.
- Depending on whether the relational operator is Select-Where or Join we create the condition matrix(Condition Matrix contains the reciprocal of the condition value and zeroes to ensure matrix multiplication compatibility) .
- We transfer the input matrix and condition matrix from CPU to GPU.
- Database operations using GEMM: In GPU we will perform matrix multiplication using GEMM algorithm.
- Bit Masking :The resulting matrix will be converted to a matrix containing only ones and zeroes using the bitmasking logic explained further.
- To obtain the desired rows, we multiply the input matrix with the result matrix and display the result.

B. Database Operations using GEMM

Proposed strategy: Computing Select-Where and Join with matrix multiplications in tensor cores:

Step 1: Here, We have our input matrix and condition matrix and the user decides whether the Select-where or Join operations has to be performed.

$$R = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \; S = \begin{bmatrix} 4 & 1 \\ 6 & 2 \end{bmatrix}$$

Step 2:Depending on the operation to be performed we create the condition matrix. For Select-Where we simply take the reciprocal of the condition value whereas for the Join operation we take the reciprocal of the join column. The condition matrix and the input matrix is padded with zeroes to make it compatible for matrix multiplication. Next we transpose the condition matrix. Finally the input matrix and condition matrix are transferred from CPU to GPU.

$$R = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} S = \begin{bmatrix} 4 & 1 \\ 6 & 2 \end{bmatrix} R * S = \begin{bmatrix} 0 & 4 \\ 0 & 5 \\ 0 & 6 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1/4 & 1/6 & 0 \end{bmatrix}$$

$$R * S = \begin{bmatrix} 1 & 4/6 & 0 \\ 5/4 & 5/6 & 0 \\ 6/4 & 1 & 0 \end{bmatrix}$$

Step 3: After performing step 1 and step 2 we get the matrix multiplication result. Then we need to flip non-ones into zeroes in every parallel row. If there is a presence of ones in matrix multiplication, it means join values are same and then we can combine them together.

$$R * S = \begin{bmatrix} 1 & 4/6 & 0 \\ 5/4 & 5/6 & 0 \\ 6/4 & 1 & 0 \end{bmatrix}$$

$$Row1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} Row2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} Row3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

C. Bit Masking

After performing the matrix multiplication, we have the result matrix with ones(result indices) and non ones(non relevant indices).Our task is to convert the non ones to zeroes and obtain a matrix with only ones and zeroes.We have considered three conditions i.e. equality(also used for join) , greater than and less than.

1) Equality: (ceil(abs(gemmresult[index]-1) && 1) XOR 1
2) Greater than: (ceil(gemmresult[index]-1) abs(ceil(gemmresult[index]-1))) && 1
3) Less than: (floor(gemmresult[index]-1) - abs(floor(gemmresult[index]-1)) && 1

Descriptions of the symbols and terms used:

- gemmresult: The result matrix after matrix multiplication using GEMM
- ceil : ceil() function that rounds a floating point value to the next highest integer
- abs: abs() function that gives the absolute value of the integer
- floor : floor() function rounds a floating point value to the next lowest integer
- index : Index of the element being considered
- XOR : XOR operation
- && : Logical AND operation

D. Computing Result Matrix

Step 1: After getting the positions of 1's in each row, we merge the corresponding rows of both the tables to get the join result.

Step 2: In the final step of join strategy, we transfer the join result of the matrices from GPU to CPU .

$$R * S = \begin{bmatrix} 1 & 4 & 1 \\ 3 & 6 & 2 \end{bmatrix}$$

E. Library Usage

We are using Nvidia GPU Volta's Tesla equipped with Tensor cores. [3] .Tensor cores only perform matrix multiplications, we use cuBlas [4] library for GEMM operations. We have not considered CPU to GPU transfer time for performance measure. For our implementation we use CUDA 9.2 API[10]. We have used methods cudaMalloc() [5] to allocate bytes size in linear memory on the device and cudaMemcpy()[5] to copy data between host and device.

REFERENCES

[1] E. Babb. "Implementing a relational database by means of specialzed hardware". In: *ACM Transactions on Database Systems (TODS)* 4.1 (Mar. 1979), pp. 1–29. ISSN: 15574644. DOI: 10.1145/320064.320065. URL: https://dl.acm.org/doi/10.1145/320064.320065.

[2] R. Carrasco, R. Vega, and C. A. Navarro. "Analyzing GPU Tensor Core Potential for Fast Reductions". In: *2018 37th International Conference of the Chilean Computer Science Society (SCCC)*. 2018, pp. 1–6.

[3] Pierre Comon. "Tensors : A brief introduction". In: *IEEE Signal Process. Mag.* (2014). ISSN: 10535888. DOI: 10.1109/MSP.2014.2298533.

[4] *cuBLAS — NVIDIA Developer*. URL: https://developer.nvidia.com/cublas (visited on 07/01/2020).

[5] *CUDA Runtime API :: CUDA Toolkit Documentation*. URL: https://docs.nvidia.com/cuda/cuda-runtime-api/group%7B%5C_%7D%7B%5C_%7DCUDART%7B%5C_%7D%7B%5C_%7DMEMORY.html%7B%5C#%7Dgroup%7B%5C_%7D%7B%5C_%7DCUDART%7B%5C_%7D%7B%5C_%7DMEMORY%7B%5C_%7D1g37d37965bfb4803b6d4e59ff26856356 (visited on 07/01/2020).

[6] Abdul Dakkak et al. "Accelerating Reduction and Scan Using Tensor Core Units". In: *Proceedings of the ACM International Conference on Supercomputing*. ICS '19. Phoenix, Arizona: Association for Computing Machinery, 2019, pp. 46–57. ISBN: 9781450360791. DOI: 10.1145/3330345.3331057. URL: https://doi.org/10.1145/3330345.3331057.

[7] Bingsheng He et al. "Relational Joins on Graphics Processors". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: Association for Computing Machinery, 2008, pp. 511–524. ISBN: 9781605581026. DOI: 10.1145/1376616.1376670. URL: https://doi.org/10.1145/1376616.1376670.

[8] Pedro Holanda and Hannes Mühleisen. "Relational Queries with a Tensor Processing Unit". In: *Proceedings of the 15th International Workshop on Data Management on New Hardware*. DaMoN'19. Amsterdam, Netherlands: Association for Computing Machinery, 2019. ISBN: 9781450368018. DOI: 10.1145/3329785.3329932. URL: https://doi.org/10.1145/3329785.3329932.

[9] Stefano Markidis et al. “NVIDIA tensor core programmability, performance & precision”. In: *Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018*. Institute of Electrical and Electronics Engineers Inc., Aug. 2018, pp. 522–531. ISBN: 9781538655559. DOI: 10.1109/IPDPSW.2018.00091. arXiv: 1803.04014.

[10] *Programming Tensor Cores in CUDA 9 — NVIDIA Developer Blog*. URL: https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/ (visited on 07/01/2020).

[11] *Tensor Cores: Versatility for HPC & AI — NVIDIA*. URL: https://www.nvidia.com/en-us/data-center/tensor-cores/ (visited on 06/28/2020).