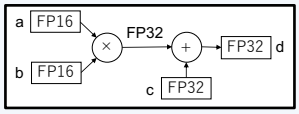
Accurate DGEMM using Tensor Cores

Daichi Mukunoki[1], Katsuhisa Ozaki[2], Takeshi Ogita[3]

1 RIKEN Center for Computational Science (JAPAN), 2 Shibaura Institute of Technology (JAPAN), 3 Tokyo Woman's Christian University (JAPAN)

Introduction

- As the demand for deep-learning increases, specialized hardware dedicated to low-precision matrix-multiplication, which is the kernel of those tasks, has been developing (e.g., NVIDIA Tensor Core, Google TPU, ARM Matterhorn's Matmul)
- **NVIDIA Tensor Cores** are a special processing unit that enables 4×4 matrix multiplication operations on FP16 inputs with FP32 precision and return the result on FP32. Besides, cuBLAS provides the cublasGemmEx routine using the above Tensor Core operation.



Tensor Core operation with FP32 accuracy

- Markidis et al. [1] proposed a method to achieve nearly the SGEMM equivalent accuracy on the matrices with the dynamic range supported on FP16 using the cublasGemmEx routine.
- In this poster, we present **an accurate DGEMM implementation, which has the DGEMM compatible interface and can be computed using cublasGemmEx on Tensor Cores**
- Our implementation returns the **correctly-rounded** result: it supports bit-level **reproducibility** and is compatible with other correctly-rounded GEMMs such as ExBLAS [2] and OzBLAS [3]

Data and computation precisions on GEMM routines

	Mat. A	Mat. B	Mat. C	Computation
cublasDgemm	FP64	FP64	FP64	FP64
cublasGemmEx (with Tensor Cores†)	FP16	FP16	FP32	FP32
our DGEMM	FP64	FP64	FP64	Correctly-rounded

†Other operations and data formats are also supported through the same interface without Tensor Cores

Method (Ozaki scheme)

- Our method is based on the Ozaki scheme [4], which is an accurate matrix-multiplication algorithm based on the error-free transformation for dot-product/matrix-multiplication.
- In this study, we modified the original version of the Ozaki scheme, which was designed to be computed using DGEMM, to be computed using Tensor Cores on FP64 input/output.

Ozaki scheme for Tensor Cores

function $(C \leftarrow \texttt{DGEMM-TC-CR}(m,n,k,A,B))$ // $A \in \mathbb{F}_{\texttt{FP64}}{}^{m\times k}$, $B \in \mathbb{F}_{\texttt{FP64}}{}^{k\times n}$

$(A_{\texttt{split}}[s_A], c_A[s_A]) \leftarrow \texttt{SplitA}(m,k,A)$ // $A_{\texttt{split}}$ is obtained on FP16

$(B_{\texttt{split}}[s_B], c_B[s_B]) \leftarrow \texttt{SplitB}(k,n,B)$ // $B_{\texttt{split}}$ is obtained on FP16

$r = 1$

for $(q = 1 : s_B)$ **do**

for $(p = 1 : s_A)$ **do**

This can be performed with cublasGemmEx using Tensor Cores

$C_{\texttt{tmp1}} = \texttt{GEMM}_{\texttt{FP32}}(m,n,k,A_{\texttt{split}}[p],B_{\texttt{split}}[q])$

$C_{\texttt{tmp2}}[r] = \texttt{ScaleUp}(C_{\texttt{tmp1}}, c_A[p] \times c_B[q]))$

ScaleUp computes element-wise scalar-multiplication of two matrices

$r += 1$

end for

end for

$C = \texttt{NearSum}(C_{\texttt{tmp2}}, r)$

end function

NearSum [5] computes the correctly-rounded summation

Spitting for a vector
(for matrix multiply, it is performed for inner-product wise direction)

function $((x_{\texttt{split}}[s_x], c[s_x]) \leftarrow \texttt{Split}(n,x))$

$\rho = \texttt{ceil}(\texttt{log2}(\mathbf{u}_{\texttt{FP64}}{}^{-1}) - (\texttt{log2}(\mathbf{u}_{\texttt{FP32}}{}^{-1}) - \texttt{log2}(n))/2)$

$\mu = \max_{0\le i\le n-1}(|x_i|)$

$j = 0$

while $(\mu \neq 0)$ **do**

$j = j + 1$

$\tau = \texttt{ceil}(\texttt{log2}(\mu))$

$\sigma = 2^{\rho+\tau}$

$x_{\texttt{tmp}} = \texttt{fl}_{\texttt{FP64}}((x+\sigma)-\sigma)$ // $x_{\texttt{tmp}}$ is the split vector on FP64

$x = \texttt{fl}_{\texttt{FP64}}(x - x_{\texttt{split}}[j])$

$\theta = \texttt{floor}(\texttt{log2}(65504) - \tau)$ // 65504 is maximum of the FP16 values

$c[j] = 2^{\theta}$ // this must be hold for scaling-up later

$x_{\texttt{split}}[j] = \texttt{fl}_{\texttt{FP16}}(x_{\texttt{tmp}} c[j])$ // Scaling-down and casting from FP64 to FP16

$\mu = \max_{0\le i\le n-1}(|x_i|)$

end while

$s_x = j$

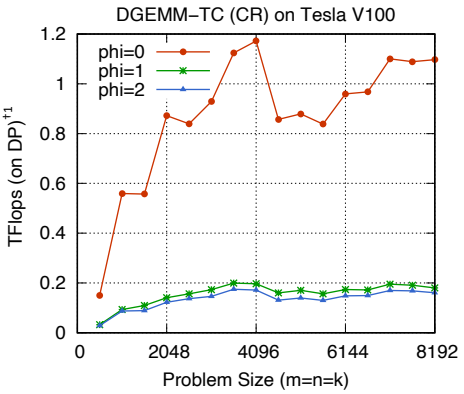
end function

$\mathbf{u}_{\texttt{FP64}}$ denotes the unit round-off of FP64: $\mathbf{u}_{\texttt{FP64}} = 2^{-53}$
$\mathbf{u}_{\texttt{FP32}}$ denotes the unit round-off of FP32: $\mathbf{u}_{\texttt{FP32}} = 2^{-24}$

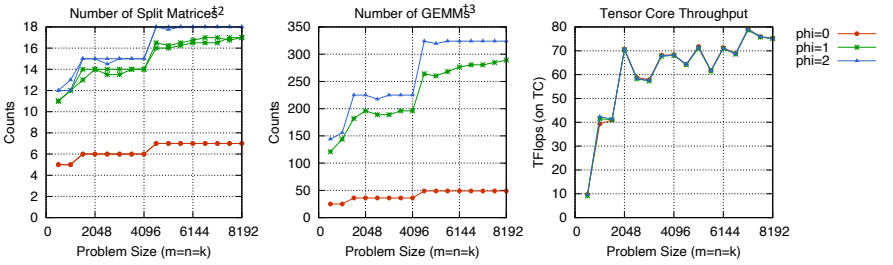
Note: In this method, ***the number of split matrices depends on the range of the absolute values of the input matrices***

Performance on Tesla V100

- Environment: NVIDIA Tesla V100 GPU (PCIe-32GB): the theoretical peak performance (with 1.38 GHz) is 7065.6 GFlops on FP64 and 113049.6 GFlops on Tensor Cores. CUDA 10.1, GPU driver version 418.39.
- **Performance depends on the input** (the number of split matrices depends on the range of the absolute values of the input matrices): matrices are initialized with (rand-0.5) × exp(φ × randn) to vary the floating-point range: 9.3E-10 – 5.0E-01 on $\varphi = 0$, 1.5E-09 – 1.6E+02 on $\varphi = 1$, and 1.3E-09 – 4.8E+04 on $\varphi = 2$.

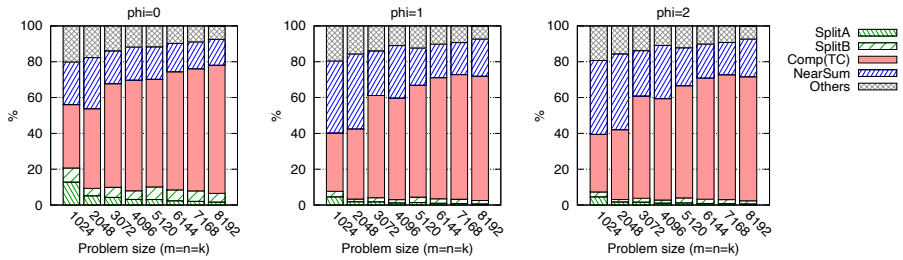


†1:"Flops (on DP)" is the number of FP64 floating-point operations, corresponding to the standard DGEMM, per second



Performance Details

†2: The values for matrices A and B are shown with the same line. †2†3: The average of all blocks.



Execution Time Breakdown

Conclusion

- We presented an accurate GEMM implementation with the DGEMM compatible interface using cublasGemmEx performed on Tensor Cores, which returns the correctly-rounded result on FP64
- The performance depends on the input matrices. For example, for matrices initialized with random numbers having the dynamic range of 1E+9, we achieved approx. 1.2 TFlops.
- Although our implementation brings no performance advantage against cuBLAS DGEMM on GPUs that support fast FP64 (1/16 of Tensor Cores), it can be beneficial on hardware with limited FP64 support such as NVIDIA Tesla T4, whose FP64 performance is 1/256 of Tensor Cores.
- Our approach opens the way to utilize AI-oriented hardware, which supports limited FP64 performance, for more general purposes and may impact the system co-design.
- The latest work including some extensions will be published soon.

Reference and Acknowledgement

[1] Markidis, S., Chien, S.W.D., Laure, E., Peng, I.B., Vetter, J.S.: NVIDIA Tensor Core Programmability, Performance Precision. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 522–531 (2018)
[2] S. Collange, D. Defour, S. Graillat, R. Iakymchuk: Numerical reproducibility for the parallel reduction on multi- and many-core architectures, Parallel Computing, Vol. 49, pp. 83-97, 2015.
[3] Mukunoki, D., Ogita, T., Ozaki, K.: Reproducible BLAS Routines with Tunable Accuracy Using Ozaki Scheme for Many-core Architectures. In: 13th International Conference on Parallel Processing and Applied Mathematics (PPAM2019) (2019), (to appear)
[4] K. Ozaki, T. Ogita, S. Oishi, S. M. Rump, "Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications," Numerical Algorithms, Vol. 59, No. 1, pp. 95-118, 2012.
[5] S. Rump, T. Ogita, S. Oishi, "Accurate floating-point summation part II: Sign, k-fold faithful and rounding to nearest," SIAM Journal on Scientific Computing, Vol. 31, No. 2, pp. 1269-1302, 2008.

* This research was partially supported by MEXT as "Exploratory Issue on Post- K computer" (Development of verified numerical computations and super high- performance computing environment for extreme researches) and the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number 19K20286. This research used computational resources of Cygnus provided by Multidisciplinary Cooperative Research Program in Center for Computational Sciences, University of Tsukuba.