

High Performance Implementation of Reproducible BLAS Routines with Tunable Accuracy using Ozaki Scheme

Daichi Mukunoki

*Graduate School of Science,
Tokyo Woman's Christian University
Suginami-ku, Tokyo
mukunoki@lab.twcu.ac.jp*

Takeshi Ogita

*Division of Mathematical Sciences,
Tokyo Woman's Christian University
Suginami-ku, Tokyo
ogita@lab.twcu.ac.jp*

Katsuhisa Ozaki

*Department of Mathematical Sciences,
Shibaura Institute of Technology
Saitama, Japan
ozaki@sic.shibaura-it.ac.jp*

Abstract—This study presents a high performance implementation of Basic Linear Algebra Subprograms (BLAS) routines supporting reproducibility and tunable accuracy using the Ozaki scheme, which is an accurate matrix-multiplication method proposed by Ozaki et al. in 2011. The method ensures reproducibility and realizes tunable accuracy, including correct-rounding, by eliminating the effect of rounding-error in the computation. The most advantage of the method is that the method can be constructed based on level-3 BLAS. In this study, we show the implementation of three routines from level 1–3 BLAS: inner-product (DOT), matrix-vector multiplication (GEMV), and matrix-matrix multiplication (GEMM), with several optimization techniques for reducing the memory consumption and improving the performance. The performance evaluation on Titan V GPU demonstrates that our implementation achieves more than 73% of the expected peak performance.

I. INTRODUCTION

Even though several Basic Linear Algebra Subprograms (BLAS) implementations supporting accurate and reproducible computations have been proposed such as ExBLAS [1] and ReproBLAS [2], the cost of the software development and the performance overhead against the normal BLAS implementations optimized for a certain platform, in particular on level-3 routines, are still high. To address those issues, this study spots on the Ozaki scheme. The Ozaki scheme is an accurate matrix-multiplication method proposed by Ozaki et al. in 2011 [3], which can achieve reproducible floating-point results as well as the tunable accuracy including correct-rounding. The most advantage of the Ozaki scheme is that the method is constructed on level-3 BLAS, a highly optimized implementation of which is usually provided by the system vendor on most platforms. Therefore, we expect that the Ozaki scheme is an effective approach to develop accurate and reproducible BLAS routines with high performance at low development cost for many platforms. In this study, we present a high performance implementation of three routines from level 1–3 BLAS, inner-product (DOT), matrix-vector multiplication (GEMV), and matrix-matrix multiplication (GEMM), using the Ozaki scheme. Our implementation employs several optimization techniques for reducing the memory consumption

and improving the performance. This paper provides the accuracy and performance evaluation results on a GPU.

II. OZAKI SCHEME

This section briefly introduces the overview of the Ozaki scheme. For more details, see the original paper [3]. Hereinafter, we explain the case of an inner-product of the two vectors $x \in \mathbb{F}^n$ and $y \in \mathbb{F}^n$, where $\mathbb{F}$ be the set of floating-point numbers. However, the method is applicable to the BLAS operations which is constructed with inner-product such as GEMV and GEMM. First, the input vectors are split into several split vectors as $x = x^{(1)} + x^{(2)} + \dots + x^{(s_x)}$, $s_x \in \mathbb{N}$, $x^{(p)} \in \mathbb{F}^n$ and $y = y^{(1)} + y^{(2)} + \dots + y^{(s_y)}$, $s_y \in \mathbb{N}$, $y^{(q)} \in \mathbb{F}^n$. Then, the Ozaki scheme computes the inner-product as $x^T y = (x^{(1)} + x^{(2)} + \dots + x^{(s_x)})^T (y^{(1)} + y^{(2)} + \dots + y^{(s_y)})$. The splittings are performed to meet the following two properties: (1) if $x^{(p)}(i)$ and $y^{(q)}(j)$ are non-zero elements, $|x^{(p)}(i)| \geq |x^{(p+1)}(i)|$ and $|y^{(q)}(j)| \geq |y^{(q+1)}(j)|$: the accuracy of the final result can be controlled by omitting some lower split vectors. (2) let $\text{fl}(\dots)$ denote a computation performed with floating-point arithmetic. $(x^{(p)})^T y^{(q)} = \text{fl}((x^{(p)})^T y^{(q)})$, $1 \leq p \leq s_x$, $1 \leq q \leq s_y$: the inner-product computations for the split vectors can be computed using a BLAS routine (DOT) performing standard floating-point operations since $\text{fl}((x^{(p)})^T y^{(q)})$ has no round-off error. In the cases of matrix-vector multiplication and matrix-matrix multiplication, the computations can be performed using GEMV and GEMM, respectively. The summation of the computation results of the split vectors can be computed by using some accurate and reproducible summation algorithm such as NearSum [4]. In this study, we introduce $d, d \in \mathbb{N}, d \geq 2$, as the degree of split vector products to control the accuracy of the final result. When d is specified, we omit the computations for $p+q > d$ in $(x^{(p)})^T y^{(q)}$. Consequently, at most $(d^2 - d)/2$ vector products are computed. The accuracy that can be obtained with a certain d depends on the length of the input vectors and the range of the absolute values in the input vectors: the degree of d required to achieve the best accuracy depends on the input vectors.

TABLE I: Maximum relative error against the correct-rounded result on GEMM ($n = m = k = 1000$)

ϕ	cuBLAS DGEMM	OzBLAS DGEMM			
		$d = 3$	$d = 5$	$d = 7$	$d = 9$
0	3.27E-09	9.39E-08	0	0	0
1	4.19E-10	2.23E-04	2.07E-16	0	0
2	6.99E-11	5.42E-01	8.34E-13	0	0
3	7.33E-11	1.17E+03	7.02E-09	0	0
4	5.84E-12	1.99E+04	2.09E-03	1.03E-15	0

III. IMPLEMENTATION

We implemented DOT ($r = x^T y$), GEMV ($y = Ax$), and GEMM ($C = AB$) which compute and return double-precision floating-point data. Currently, our implementation supports only the computations described above: some features including the scalar arguments in the BLAS interface (α, β) are not supported. We implemented them on NVIDIA GPUs using CUDA, but the method and the optimization techniques we applied are architecture-independent. For the summation of the split vectors or matrices, we used the NearSum, a summation algorithm which can achieve correct-rounding, by Rump et al. [4]. The Ozaki scheme consumes extra memory spaces for storing the split data. To reduce the memory consumption on GEMV and GEMM, we applied a blocking technique which divides a matrix into a rectangle shape in the cyclic manner. On DOT and GEMV, the computations of the split data can be performed using GEMM by storing the split vectors as a matrix. In this way, we can expect to improve the performance by increasing the data reuse since the performance of DOT and GEMV should be memory-bound. For that purpose, we used cublasDgemm on GEMV. However, we used an optimized GEMM implementation by us on DOT because the cublasDgemm is not optimized for the computation of skinny rectangular matrices appeared in the DOT computation. On the other hand, on GEMM, we utilized a batched GEMM routine (cublasDgemmBatched) to compute the split matrices efficiently and simultaneously.

IV. EVALUATION

We evaluated the performance on NVIDIA Titan V GPU with CUDA 9.1. The GPU has 7449.6 GFlops of theoretical peak performance on double-precision and 652.8 GB/s of theoretical peak bandwidth. Table I shows the maximum relative error against the correct-rounded result on GEMM ($n = m = k = 1000$). OzBLAS is our implementation based on the Ozaki scheme. ‘ d ’ corresponds the degree of split vector/matrix products to control the accuracy. The input matrices were initialized as $(\text{rand} - 0.5) \times \exp(\phi \times \text{randn})$, where rand is a uniform random number $[0, 1)$ and randn is a random number from the standard normal distribution. In this way, the range of the absolute value of the input can be controlled by ϕ : the larger ϕ is, the range becomes wider. In the table, zero means the result achieved correct-rounding.

Table II shows the throughputs and the relative execution times compared to the corresponding cuBLAS double precision routines when $\phi = 8$. The throughputs of OzBLAS

TABLE II: Performance on Titan V GPU

(a) DOT ($n = 2^{22}$)				
	Execution Time [sec]	Computation Throughput	Overhead to cuBLAS	
			Expected	Obtained
cuBLAS	1.01E-03	82.88 GFlops	1.0	1.0
OzBLAS($d = 3$)	8.95E-03	9.371 GOps	8.0	8.8
OzBLAS($d = 5$)	1.85E-02	4.540 GOps	16.0	18.3
OzBLAS($d = 7$)	2.79E-02	3.006 GOps	24.0	27.6
OzBLAS($d = 9$)	3.76E-02	2.232 GOps	32.0	37.1

(b) GEMV ($m = n = 10240$)				
	Execution Time [sec]	Computation Throughput	Overhead to cuBLAS	
			Expected	Obtained
cuBLAS	1.25E-03	168.2 GFlops	1.0	1.0
OzBLAS($d = 3$)	1.13E-02	18.64 GOps	8.0	9.0
OzBLAS($d = 5$)	2.29E-02	9.164 GOps	16.0	18.4
OzBLAS($d = 7$)	3.45E-02	6.083 GOps	24.0	27.7
OzBLAS($d = 9$)	4.61E-02	4.555 GOps	32.0	36.9

(c) GEMM ($m = n = k = 10240$)				
	Execution Time [sec]	Computation Throughput	Overhead to cuBLAS	
			Expected	Obtained
cuBLAS	3.01E-01	7144 GFlops	1.0	1.0
OzBLAS($d = 3$)	1.24E+00	1734 GOps	3.0	4.1
OzBLAS($d = 5$)	3.54E+00	606.6 GOps	10.0	11.8
OzBLAS($d = 7$)	7.95E+00	270.3 GOps	21.0	26.4
OzBLAS($d = 9$)	1.41E+01	152.3 GOps	36.0	46.9

are represented by “Ops”, which means the computation throughput per second calculated based on the number of mathematical arithmetic operations in the BLAS operation: for example on GEMM for $n \times n$ matrices, it is obtained as $2n^3/t$ [Ops] where t is the execution time. The expected overhead to cuBLAS routine was estimated based on the memory access cost on DOT and GEMV (memory-bound) and the computation cost on GEMM (compute-bound).

V. CONCLUSION

We demonstrated that the performance of DOT, GEMV, and GEMM using the Ozaki scheme, supporting reproducibility and tunable accuracy, on Titan V GPU. Our implementation attained more than 73% of the expected peak performance estimated based on the overhead compared to the cuBLAS double-precision routine. We will discuss the implementation, optimization, and performance as well as the other opportunities for improving the performance more in the full paper.

ACKNOWLEDGMENT

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).

REFERENCES

- [1] R. Iakymchuk, S. Collange, D. Defour, and S. Graillat, “ExBLAS – Exact BLAS,” <https://exblas.lip6.fr>.
- [2] P. Ahrens, H. D. Nguyen, and J. Demmel, “ReproBLAS – Reproducible Basic Linear Algebra Sub-programs,” <https://bebop.cs.berkeley.edu/reproblas>.
- [3] K. Ozaki, T. Ogita, S. Oishi, and S. M. Rump, “Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications,” *Numer. Algorithms*, vol. 59, no. 1, pp. 95–118, 2012.
- [4] S. Rump, T. Ogita, and 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, 2009.