HAL
open science

ExBLAS: Reproducible and Accurate BLAS Library

Roman Iakymchuk, Caroline Collange, David Defour, Stef Graillat

To cite this version:

Roman Iakymchuk, Caroline Collange, David Defour, Stef Graillat. ExBLAS: Reproducible and Accurate BLAS Library. 2015. hal-01202396v1

HAL Id: hal-01202396

https://hal.science/hal-01202396v1

Preprint submitted on 20 Sep 2015 (v1), last revised 21 Dec 2015 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ExBLAS: Reproducible and Accurate BLAS Library

Roman Iakymchuk*†, Sylvain Collange‡, David Defour§, Stef Graillat*
*Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005 Paris, France
Email: {roman.iakymchuk, stef.graillat}@lip6.fr
†Sorbonne Universités, UPMC Univ Paris 06, ICS, F-75005 Paris, France
‡INRIA – Centre de recherche Rennes – Bretagne Atlantique, Campus de Beaulieu, F-35042 Rennes Cedex, France
Email: sylvain.collange@inria.fr
§DALI–LIRMM, Université de Perpignan, 52 avenue Paul Alduy, F-66860 Perpignan, France
Email: david.defour@univ-perp.fr

***Abstract*—Due to non-associativity of floating-point operations and dynamic scheduling on parallel architectures, getting a bit-wise reproducible floating-point result for multiple executions of the same code on different or even similar parallel architectures is challenging. We address the problem of reproducibility in the context of fundamental linear algebra operations – like the ones included in the BLAS library – and propose algorithms that yield both reproducible and accurate results (correct rounding, except for triangular solver). We present implementations of these algorithms for BLAS routines along with the performance results in parallel environments such as Intel desktop and server CPUs, Intel Xeon Phi, and both NVIDIA and AMD GPUs.**

***Keywords*—BLAS library, reproducibility, accuracy, superaccumulator, floating-point expansions, error-free transformations, multi- and many-core architectures.**

I. Introduction

Exascale computing (10^{18} operations per second) is likely to be reached within a decade. This increased computational power of modern supercomputers enables us to solve more complex and larger problems. That, consequently, leads to the higher number of floating-point (FP) operations to be performed, each of them potentially causing a round-off error. As FP operations like addition and multiplication are non-associative, so that, for instance, parallel implementations of the BLAS routines become non-reproducible. We define reproducibility as an ability to obtain a bit-wise identical FP result from multiple runs of the same code on the same data. However, the result often varies from one parallel machine to another or even from one run to another. These discrepancies worsen on heterogeneous architectures such as clusters with co-processors or GPUs. Such non-determinism of FP calculations in parallel programs causes validation and debugging issues and may even lead to deadlocks. The problem of both non-determinism and non-reproducibility of FP computations becomes so important that it is listed among the top ten Exascale challenges [5].

Numerical reproducibility can be addressed by targeting either the order of operations or the error resulting from finite arithmetic. One solution consists in providing the deterministic control over rounding errors by, for example, enforcing the execution order for each operation. However, this approach is not portable and/or does not scale well with the number of processing cores. The other solution aims at avoiding cancellation and rounding errors by using an exact accumulation method such as the superaccumulator proposed by Kulisch [8]. This solution increases the accuracy at the price of more operations and memory transfers. Because of that, for a long time, it was considered too costly.

To enhance reproducibility, Intel proposed a "Conditional Numerical Reproducibility" (CNR) in its MKL. However, CNR does not ensure correct rounding and it induces large performance overhead. For instance, for large arrays the MKL's summation with CNR is $85-93\,\%$ slower than both the regular MKL's and our reproducible summation. Demmel and Nguyen introduced a family of algorithms for reproducible summation in FP arithmetic [3]. Their solution induces a twofold slowdown as data transfers and reductions need to be performed twice. They improved the algorithm [4] using a single reduction among nodes, which reduced the overhead to roughly $20\,\%$. Demmel and Nguyen have extended their concept to the other BLAS-1 routines, distributed as ReproBLAS. Arteaga et al. [1] used this approach with improved communication for summation and obtained the same accuracy with roughly $10\,\%$ overhead. Neal [10] considered scalar superaccumulators of different sizes for summation in the R software. Alternatively, we introduced [2] a multi-level approach to compute reproducible sums. This approach is based on FP expansions (FPEs) and superaccumulators. The proposed implementations showed that the numerical reproducibility and bit-perfect accuracy can be achieved at no additional cost for large sums with dynamic ranges of up to 90 orders of magnitude.

In this work, we address the problem of reproducibility of the BLAS routines due to cancellation and rounding errors that occur during FP operations. We begin with parallel reduction using our multi-level summation approach, which allows to perform accumulation in any order without loosing a bit of information. Then, we extend this approach to BLAS routines from level-1 to level-3. The paper is organized as follows. Section II reviews aspects of computer arithmetic. Section III presents our multi-level reproducible approach. We evaluate our implementations in Section IV and discuss conclusions in Section V.

II. Computer Arithmetic

FP arithmetic consists in approximating real numbers with a mantissa, an exponent, and a sign:

$$x = \pm \underbrace{x_0.x_1 \ldots x_{M-1}}_{mantissa} \times b^e, \quad 0 \le x_i \le b-1, \quad x_0 \neq 0$$

where b is the basis, M is the precision, and e is the exponent ($e_{\min} \leq e \leq e_{\max}$). The IEEE-754 standard specifies FP formats and operations as well as requires correctly rounded results for the basic FP operations $(+, -, \times, /, \sqrt{\ })$ and *fma*[1]. This means the operations are performed as if the result was computed with the infinite precision and rounded to the FP format. In this paper, we extend the correct rounding criterion to BLAS routines. We consider double precision format (`binary64`) and assume rounding-to-nearest.

Two approaches exist to perform one FP addition without introducing rounding error. The first solution aims at computing the error which occurred during rounding using FP expansions in conjunction with error-free transformations (EFTs). FPEs represent the result as an unevaluated sum of FP numbers, whose components are ordered by magnitude with minimal overlap to cover a wide range of exponents. FPEs of sizes 2 are described in [9]. When working with the rounding-to-nearest mode, the rounding error of addition and multiplication can be represented as a FP number. This error can be computed in FP arithmetic using EFTs: `TwoSum`, Alg. 1, for addition; `TwoProd`, Alg. 2, for multiplication.

Adding one FP number to an expansion is an iterative operation. The FP number is first added to the head of the expansion. The rounding error is recovered as a FP number, using an EFT such as `TwoSum`, and recursively accumulated to the remainder of the expansion. FPE computations are free of conditional branches and memory accesses, making fast pipelined and vectorized implementations practical. However, their complexity is linear with the number of terms in the FPE, so they are best suited for low dynamic range sums.

Algorithm 1: EFT for the sum of two FP numbers.

Function $[r, s]$ = *TwoSum*(a, b)
- $r \leftarrow a + b$
- $z \leftarrow r - a$
- $s \leftarrow (a - (r - z)) + (b - z)$

Algorithm 2: EFT for the product of two FP numbers.

Function $[r, s]$ = *TwoProd*(a, b)
- $r \leftarrow a \times b$
- $s \leftarrow fma(a, b, -r)$

The second solution exploits the finite range of representable FP numbers by storing every bit of the sum. It can be implemented as a large fixed-point accumulator (superaccumulator). This accumulator covers the range from the minimum representable FP value to the maximum value independently of the sign. We use a superaccumulator of 2098 (emin + emax + mantissa $= 1022 + 1023 + 53$) bits for double precision FP accumulation. Superaccumulator implementations involve irregular memory accesses, but the amortized complexity of accumulation does not depend on accumulator length, making it suitable for large dynamic range sums.

III. Multi-Level Reproducible Approach

We split the reproducible parallel reduction algorithm into five stages [2]: filtering, private superaccumulation, local superaccumulation, parallel reduction, and rounding. This decomposition is suitable for nested parallelism on modern architectures. The idea is to accumulate numbers with commonly-occurred exponents using a fast small FPE cache and to use slower superaccumulators only for numbers of unlikely exponents.

The first stage uses FPEs with EFTs for addition of two FP numbers, see Alg. 1. Each vector lane or GPU thread maintains its own FPE of size p. In case the accuracy provided by FPEs is not enough, the remaining rounding error is accumulated to private superaccumulators on the second stage. Private superaccumulators belong to a CPU thread or a GPU warp. Each private superaccumulator also receives the contents of the FPEs at the end of the summation. On the third stage, private superaccumulators are merged into a single local superaccumulator (one per a node or a GPU warp). The fourth stage performs a parallel reduction of local superaccumulators to a final superaccumulator. Finally, on the fifth stage, the result is rounded to the desired FP format.

We extended and adapted this approach to dot product, triangular solver [7], and matrix-matrix multiplication [6].

IV. Performance Experiments

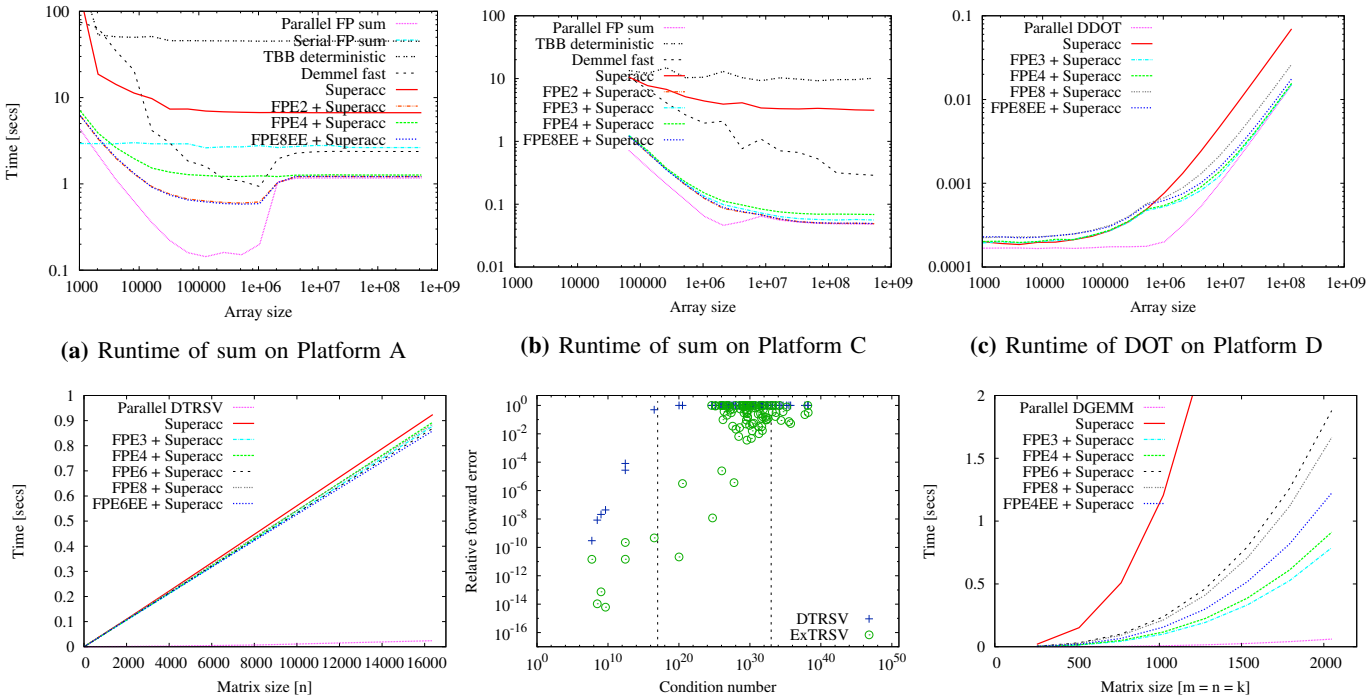
This section details our implementations of the multi-level approach and presents their evaluation on a range of parallel platforms, which are listed in Tab. I. We verify the correctness of our implementations against the generic multiple precision library MPFR, which is sequential and runs on CPUs only.

TABLE I: Hardware platforms employed in the evaluation.

A	Intel Core i7-4770	4 cores with Hyper-Threading	3.400 GHz
B	Intel Xeon E5-4650L	64 nodes with 2×8 cores	2.600 GHz
C	Intel Xeon Phi 5110P	60 cores $\times$ 4-way Multi-Threading	1.053 GHz
D	NVIDIA Tesla K20c	13 SMs $\times$ 192 CUDA cores	0.705 GHz
E	AMD Radeon HD 7970	32 CUs $\times$ 64 units	0.925 GHz

Our implementations take advantage of all resources on modern processors: SIMD instructions, fused-multiply-and-add (FMA) instruction, and multi-threading on multi-core CPUs and Xeon Phi; local memory and atomic instructions on GPUs. For example, on the Intel Haswell architecture, we use AVX intrinsics to benefit from the $4-$way SIMD and implement `TwoSum` by replacing half of the 6 additions/subtractions by FMAs (multiplying by one). The latter optimization allows to use both the FP adder and FMA units that can run in parallel on Haswell. Thread-level parallelism is exposed using OpenMP to benefit from multi-core and hardware multi-threading. The final inter-node reduction within a cluster is performed using MPI. On the Intel Xeon Phi co-processor, we benefit from 8-way SIMD by using 512-bit vector intrinsics. We perform explicit memory prefetching in order to maximize memory throughput. We develop hand-tuned OpenCL implementations for NVIDIA and AMD GPUs. They use multiple superaccumulators per work group, which are stored in local or global memory. In order to avoid bank conflicts, superaccumulators are interleaved together to spread their digits among different memory banks. Concurrency between multiple threads, which share single superaccumulator, is handled through atomics.

[1] fma(a, b, c) computes $a \times b + c$ with a single rounding.



(a) Runtime of sum on Platform A

(b) Runtime of sum on Platform C

(c) Runtime of DOT on Platform D

(d) Runtime of TRSV on Platform D

(e) Accuracy of TRSV

(f) Runtime of GEMM on Platform E

Fig. 1: ExBLAS performance results on various architectures from Tab. I.

As baselines, we consider our vectorized and parallelized non-deterministic algorithms that are implemented using vector intrinsics, optimized reduction, blocking, etc. In case of summation, we also use the deterministic Intel TBB reduction (referred as "TBB deterministic") and the Fast Deterministic Parallel Sum algorithm [3] ("Demmel fast").

Figs. 1a and 1b present the timings achieved by the summation algorithms as a function of the input dataset size N of double precision FP numbers. In the legends of all figures, "Superacc" corresponds to our algorithm that is solely based on superaccumulators; "FPEp + Superacc" stands for our algorithm with FPEs of size p $(p = 2 : 8)$ in conjunction with superaccumulators when needed; "FPE8EE + Superacc" represents our algorithm based on the expansion of size 8 and the early-exit optimization technique, meaning stop propagating errors when they are zeros. These plots show that on large datasets the performance of the baseline unordered sum as well as the expansion cache is constrained by the memory bandwidth on all platforms. For instance, these algorithms achieve 23.3 GB/s on platform A over the theoretical peak bandwidth of 25.6 GB/s. As expected, for small dataset that fits in cache, the non-reproducible parallel sum outperforms our implementations by a factor of 2 to 5. However, "TBB deterministic" and MPFR (not shown on the plot) run two and three orders of magnitude, accordingly, slower than our implementations. The Fast Deterministic Parallel Sum algorithm requires two passes on the data. In memory-bounded scenarios, its execution is two times slower than the unordered reduction and the other single-pass algorithms. In addition, the overhead of the second kernel call impacts the performance of the GPU implementation.

Fig. 2 compare the scaling of performance for the reduction algorithms on platform B. The number of MPI process varies from 1 to 64, each of them performing the summation of 16M double-precision FP numbers. This dataset size ensure that we fall in the out-of-cache case. For each process, we measure the local summation time, which is colored red, and the MPI reduction time, which is colored green. For the whole range of processors, the execution time of each algorithm is dominated by the local summation time because of the dataset size. In addition, due to the equal distribution of computations among MPI processes, the computation time is roughly equivalent on the whole range of MPI processes, while the reduction time changes according to the number of MPI processes involved. We normalize the total runtime of each algorithm by the total execution time of the parallel FP summation. The Fast Deterministic Parallel Sum algorithm is 4.72 times slower than the conventional parallel summation on 64 MPI processes. In contrast, our implementation ("FPE8EE + Superacc") delivers accurate and bit-wise reproducible results with the 10 % overhead compared to the conventional summation.

We have extended multi-level summation algorithm to dot product by applying `TwoProd` for exact multiplication of two FP numbers. Fig. 1c shows the performance results of dot product on platform D, see Tab. I. `TwoProd` induces small performance overhead due to summation of two arrays: one for the result and another for the error. Therefore, the performance penalties are slightly higher than in case of ExSUM. Nevertheless, for large array sizes, ExDOT delivers numerically reproducible results with no performance loses.

On the BLAS level-2, we propose a blocked multi-level reproducible algorithm for triangular solver (ExTRSV). The idea behind is to divide matrix by blocks and apply ExTRSV on diagonal blocks and matrix-vector multiplication (ExGEMV) on sub/upper diagonal blocks; the later is beneficial for the per-

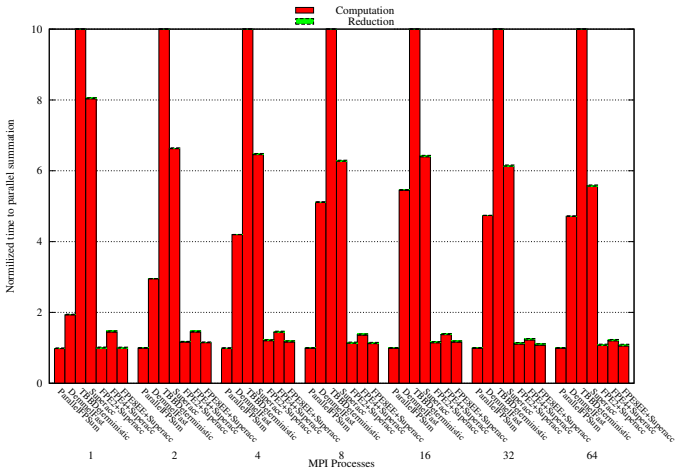


Fig. 2: Performance scaling of parallel reduction on platform B.

formance. Both ExTRSV and ExGEMV are built upon ExSUM and ExDOT. Fig. 1d presents the measured time achieved by the substitution algorithm as a function of the matrix size n on platform D, see Tab. I. The implementations with expansions deliver better performance than with superaccumulators only. However, due to switches to superaccumulators at the end of computing each element of the solution as well as when the accuracy provided by expansions is not sufficient, the benefit of using expansions is small. Moreover, the division by diagonal elements and the following accumulated errors have negative impact on the final accuracy of the solution, see Fig. 1e, but not on reproducibility. As an enhancement of the accuracy, we propose to use one step of iterative refinement, which is based on ExGEMV, ExTRSV, and ExAXPY. Although this solution will induce larger performance overhead, the accuracy of the results will be improved significantly.

We also tackle the core of BLAS – matrix-matrix multiplication (GEMM) kernel – by providing a blocked multi-level reproducible ExGEMM algorithm. Fig. 1f presents the performance results achieved by GEMM implementations as a function of the matrix size n on platform E, see Tab. I. The performance of ExGEMM, even with FPEs in conjunction with superaccumulators, is limited, because the non-deterministic double precision GEMM already squeezes the architecture performance and does not leave much resources for our approach. That leads to $12-16$ times performance overhead on NVIDIA and AMD GPUs. We consider to ameliorate the performance of both ExTRSV and ExGEMM in order to be within the 10 times performance overhead.

V. Conclusions and Future Work

We presented a multi-level approach that delivers both accurate and bit-wise reproducible results for fundamental linear algebra operations such as the ones included in the BLAS library. We derived an algorithm for parallel FP reduction, which is a common operation among the BLAS routines, and provided implementations on multi- and many-core architectures. For large sums with more than 10^7 elements of moderate dynamic range, which cover the most common cases in practice, the proposed implementations deliver comparable performance to the highly optimized parallel FP summations along with bit-wise reproducible results on a large range of platforms. Moreover, our summation algorithm outperforms the existing alternatives.

We extended the multi-level approach to dot product, triangular solver with one right-hand side, and matrix-matrix multiplication. ExDOT delivers similar performance results to ExSUM. Even though the performance of ExTRSV and ExGEMM lag behind their non-reproducible counterparts, their outputs are consistently reproducible and accurate, in terms of rounding-to-nearest in case of the later, independently of threads scheduling and data partitioning. We proposed to use the iterative refinement technique in order to improve accuracy of ExTRSV, which is saturated by the division.

Our roadmap includes deriving a complete set of reproducible, accurate, and fast BLAS routines on parallel architectures from desktop and server processors to Intel Xeon Phi co-processors and GPU accelerators. We plan to conduct a priori error analysis of the derived ExBLAS (Exact BLAS) routines. More information on the ExBLAS project as well as its sources can be found at https://exblas.lip6.fr/.

Acknowledgement

This work undertaken in the framework of CALSIMLAB is supported by the public grant ANR-11-LABX-0037-01 and used the HPC resources of ICS funded by Region Île-de-France and the project Equip@Meso (ANR-10-EQPX-29-01) both overseen by the French National Research Agency (ANR) as part of the "Investissements d'Avenir" program (ANR-11-IDEX-0004-02). This work was partially supported by the project FastRelax (ANR-14-CE25-0018-01).

References

[1] Andrea Arteaga, Oliver Fuhrer, and Torsten Hoefler. Designing bit-reproducible portable high-performance applications. In *Proceedings of IPDPS'14, Washington, DC, USA*, pages 1235–1244, 2014.

[2] Sylvain Collange, David Defour, Stef Graillat, and Roman Iakymchuk. Numerical Reproducibility for the Parallel Reduction on Multi- and Many-Core Architectures. Technical Report hal-00949355, version 3, INRIA, DALI–LIRMM, LIP6, ICS, February 2015.

[3] James Demmel and Hong Diep Nguyen. Fast reproducible floating-point summation. In *Proceedings of the 21st IEEE Symposium on Computer Arithmetic, Austin, Texas, USA*, pages 163–172, 2013.

[4] James Demmel and Hong Diep Nguyen. Parallel Reproducible Summation. *IEEE Transactions on Computers*, 64(7):2060–2070, 2015.

[5] Robert Lucas et al. Top Ten Exascale Research Challenges. Technical report, DOE ASCAC, February 2014.

[6] Roman Iakymchuk, David Defour, Sylvain Collange, and Stef Graillat. Reproducible and Accurate Matrix Multiplication for GPU Accelerators. Technical Report hal-01102877, LIP6, ICS, DALI–LIRMM, INRIA, January 2015.

[7] Roman Iakymchuk, David Defour, Sylvain Collange, and Stef Graillat. Reproducible Triangular Solvers for High-Performance Computing. Technical Report hal-01116588, version 2, LIP6, ICS, DALI–LIRMM, INRIA, February 2015.

[8] Ulrich Kulisch and Van Snyder. The Exact Dot Product As Basic Tool for Long Interval Arithmetic. *Computing*, 91(3):307–313, March 2011.

[9] Xiaoye S. Li, James W. Demmel, David H. Bailey, Greg Henry, Yozo Hida, Jimmy Iskandar, William Kahan, Suh Y. Kang, Anil Kapur, Michael C. Martin, Brandon J. Thompson, Teresa Tung, and Daniel J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Softw.*, 28(2):152–205, 2002.

[10] Radford M. Neal. Fast exact summation using small and large superaccumulators. Technical report, University of Toronto, May 2015.