Towards a Reproducible Solution of Linear Systems

Roman Iakymchuk, Stef S Graillat, David Defour, Erwin Laure, Enrique S Quintana-Ortí

Towards a Reproducible Solution of Linear Systems

Roman Iakymchuk*, Stef Graillat†, David Defour‡, Erwin Laure*, and Enrique S. Quintana-Ortí§
*KTH Royal Institute of Technology, 100 44 Stockholm, Sweden. {riakymch,erwinl}@kth.se
†Sorbonne Universités, UPMC Univ Paris VI, 75005-Paris, France. stef.graillat@lip6.fr
‡Université de Perpignan, 66860-Perpignan, France. david.defour@univ-perp.fr
‡Universidad Jaime I, 12.071-Castellón, Spain. quintana@uji.es

***Abstract*—Solving a linear system of equations is an important underlying part of numerous scientific applications. In this article, we address the issue of non-deterministic and, therefore, non-reproducible solution of linear systems and propose an approach to ensure its reproducibility. Our approach is based on the hierarchical and modular structure of linear algebra algorithms. Consequently, we divide computations into smaller logical blocks – such as a blocked LU factorization, triangular system solve, and matrix-matrix multiplication – and ensure their reproducible results. In this manner, we also split the blocked LU factorization into the unblocked LU and the BLAS-3 routines; the former is built on top of scaling a vector and outer product of two vectors routines from BLAS-1 and -2, accordingly. In this work, our focus is on constructing these building blocks that eventually lead, as we will prove, to the reproducible solution of linear systems.**

***Keywords*-Linear system, reproducibility, long accumulator, error-free transformation.**

I. Introduction

In many fields of science and engineering, the process of finding the solution for a specific problem requires to solve a system of linear equations. The common approach is to develop solvers for those tasks alone and then spend a considerable amount of time on tuning them. However, the best practice suggests to use already optimized solution-routines contained in linear algebra libraries. Those routines naturally form a hierarchical (layered) and modular structure. A good example of such structure is the Basic Linear Algebra Subprograms (BLAS) library, which is divided into three distinguishable levels depending on the operands involved: BLAS-1 handles vector-scalar operations; BLAS-2 is in change of matrix-vector computations; BLAS-3 – matrix-matrix computations.

Finding a solution of a linear system with a single right-hand side can be expressed in this hierarchical and modular manner, involving the LU factorization, as depicted in Fig. 1, where TRSV corresponds to the triangular solver; L is a unit lower triangular, while U is a non-unit upper triangular matrices. Both matrices are usually computed by employing the high-performance blocked LU factorization. The latter also follows the hierarchical pattern, engaging routines from all three BLAS levels:

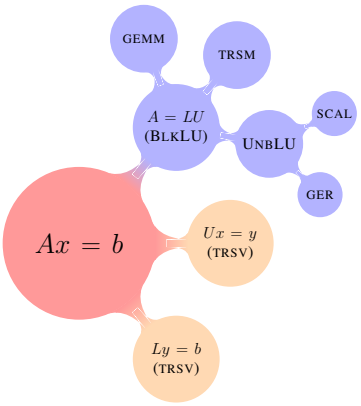
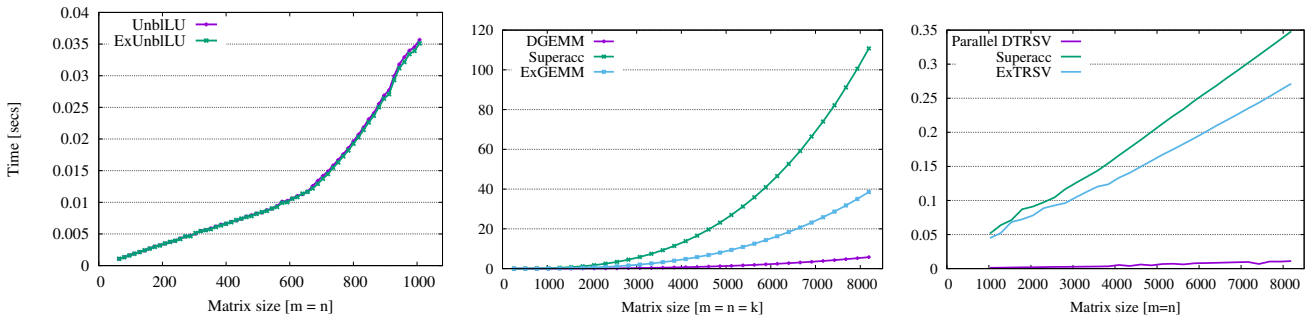


Figure 1: An example of a kernel-wise representation for the linear system solver.

TRSM to solve a unit lower triangular system with multiple right-hand sides; GEMM to compute matrix-matrix product; UNBLU for the unblocked algorithmic variant of the LU factorization with partial pivoting for stability. UNBLU can be formulated in terms of the Level-1/2 BLAS kernels, namely the vector scaling (SCAL) and the rank-1 update of a matrix (GER). Additionally, to find a pivot, we search for a maximum element in the column (MAX) and, if applicable, swap rows (SWAP).

Solving a linear system requires $2n^3/3$ flops for the blocked LU factorization and $2n^2$ for two triangular solve. Each of these operations is performed inexactly, meaning rounded to a certain value with the cascade of the follow-up errors; in this work we assume rounding-to-nearest. Although there are implementations of BLAS routines and both blocked and unblocked LU factorization that deliver high-performance on a wide range of architectures, including GPUs, their reproducibility and accuracy cannot be simply assured. That is mainly due to the non-associativity of floating-point operations as well as dynamic thread and warp scheduling on CPUs and GPUs, accordingly.

We have been working on addressing the issues of non-reproducibility for many of the above-mentioned building blocks. We started with the parallel reduction [1] and extended this work to the triangular solve and matrix-matrix multiplication. This was followed by preparing building blocks for the blocked LU factorization as well as its underlying unblocked variant. as well as its underlying unblocked variant. In this work, we aim to aggregate and augment all the above-mentioned building blocks to obtain the reproducible solver for systems of linear equations. Furthermore, we plan to provide proofs regarding reproducibility of the computed results.



(a) UNBLU on P100 **(b)** GEMM on K80 **(c)** TRSV on K80

Figure 2: Performance results.

II. APPROACH

Numerical reproducibility can be addressed by targeting either the order of operations or the error resulting from the finite computer arithmetic. One solution consists in providing the deterministic control over rounding errors by, for example, enforcing the execution order for each operation. However, these 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, for instance, a long accumulator (aka superaccumulator) such as the one proposed by Kulisch [2]. This solution increases the accuracy at the price of more operations and memory transfers per output data.

In order to tackle both drawbacks, we propose to couple the superaccumulator with a lightweight approach – such that floating-point expansions (FPEs) with error-free transformations like `TwoSum` [3] and `TwoProd` [4] – which utilizes a vector of several floating-point numbers in order to store the result of computation. Such FPEs are owned by each thread in a warp or a group of thread, so that a warp, half of a warp, or a group of threads share one superaccumulator, which is invoked rarely as FPEs are large enough to host the results for moderate ill-conditioned problems during the computations. This strategy led us to fruitful results with up to 8 % performance overhead for memory-bound problems such as parallel reduction and dot product. For the compute-bound operations – such as triangular solve and matrix-matrix multiplication – the approach always delivers reproducible results. However, there is still space for improvements in terms of performance although we already employ techniques like blocking and loop unrolling.

III. EXPERIMENTAL RESULTS

In this section, we focus on the performance results of the unblocked LU factorization, TRSV, and GEMM. We conduct experiements on[1]: an NVIDIA P100 GPU with 2560 CUDA cores @ 1,822 MHz in the boost mode; an NVIDIA K80 GPU with 4,992 CUDA cores with a dual-GPU design @ 875 MHz in the boost mode. We verify the accuracy of the obtained solutions by comparing against the multiple precision sequential library MPFR.

Fig. 2a shows the results for the unblocked LU factorization. In the figures, results with the prefix "Ex" correspond to our "exact" algorithms. For the unblocked LU factorization, we carefully conduct arithmetic operations in order to avoid intermediate rounding errors for SCAL by proposing EXINVSCAL, where division is performed directly. Due to that, we even observe a small gain in performance. Fig. 2b demonstrates the timings for the non-deterministic and reproducible implementations of GEMM. The overhead of EXGEMM is roughly 6x slower than GEMM, but it is also about 3x faster than "Superacc"; "Superacc" stands for the implementation that is solely based on long accumulators. Fig. 2c presents the performance results achieved by the triangular solve algorithm for a non-unit upper triangular matrix. The performance results for a lower unit triangular matrix show a similar pattern. Despite some performance overheads, the computed results are always reproducible.

IV. CONCLUSIONS

We have presented our approach to ensure a reproducible solution of linear systems by employing and extending the ExBLAS library (exblas.lip6.fr) as well as exploiting the hierarchical and modular structure of linear algebra operations. We are working on finalizing all the components as in Fig. 1 and deriving theoretical proofs.

[1]These resources were kindly provided by the Czestochowa University of Technology during the PPAM conference (ppam.pl) and by the Swedish National Infrastructure for Computing at PDC Center for HPC, KTH.

REFERENCES

[1] S. Collange, D. Defour, S. Graillat, and R. Iakymchuk, "Numerical reproducibility for the parallel reduction on multi- and many-core architectures," *Parallel Computing*, vol. 49, pp. 83–97, 2015.

[2] U. W. Kulisch, *Computer arithmetic and validity*, 2nd ed., ser. de Gruyter Studies in Mathematics. Berlin: Walter de Gruyter & Co., 2013, vol. 33, theory, implementation, and applications.

[3] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, third ed.* Addison-Wesley, 1997.

[4] T. Ogita, S. M. Rump, and S. Oishi, "Accurate Sum and Dot Product," *SIAM J. Sci. Comput*, vol. 26, p. 2005, 2005.