

# Performance and Numerical Aspects of Decompositional Factorizations with FP64 Floating-Point Emulation in INT8

Piotr Luszczek, Vijay Gadepally, LaToya Anderson, William Arcand, David Bestor, William Bergeron, Alex Bonn, Daniel J. Burrill, Chansup Byun, Michael Houle, Matthew Hubbell, Michael Jones, Peter Michaleas, Guillermo Morales, Julia Mullen, Andrew Prout, Albert Reuther, Antonio Rosa, Charles Yee, and Jeremy Kepner  
*LLSC, MIT Lincoln Laboratory, Lexington, MA, USA*

**Abstract**—Mixing precisions for performance has been an ongoing trend as the modern hardware accelerators started including new, and mostly lower-precision, data formats. The advantage of using them is a great potential of performance gain and energy savings. The disadvantage are the numerical issues not present in the standard-mandated floating-point formats. Split integer emulation of FP64 takes this to an extreme with the computation performed only by fixed-point tensor core units. We present the new issues the emulation faces for practical cases involving dense linear solver. We show extensive numerical tests indicating the effect of extended numerical range of matrix entries. We also scaled the input sizes to study the performance and numerical profiles on the NVIDIA Hopper GPUs.

**Index Terms**—mixed-precision, numerical linear solvers, floating-point emulation.

## I. INTRODUCTION

Mixed-precision methods [1] continue to proliferate in computational and data sciences due to the many hardware accelerators supporting unconventional number formats [2]. This spurred the development of emulation techniques that deliver the accuracy of standard floating-point types while using compute units that lack support for essential features from the IEEE 754 specification. These approaches straddle the trade-off between the reduction of accuracy and the gain in performance and energy efficiency. This originates in the use of tensor or matrix units for low-precision computations as opposed to vector and/or scalar units for the high-precision equivalents. The performance benefit is often much larger than the raw bit-count mount suggest: the 16-bit computations with FP16 or BFloat16 are more than 4 times as efficient as the 64-bit operations on FP64 data. The two terms: accuracy and precision will be clarified below. Here, we only point out that we focus on the use of these terms in numerical analysis rather than in data science and machine learning, which often have the flexibility of incorporating lesser formats into their

Research was sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

algorithmic frameworks as indicated by many generative and large language models.

The methods that especially stand out among the FP64 emulation efforts are those that use only the integer units, INT8 in particular. They come to prominence due to the inflection point brought about by the introduction of NVIDIA’s Blackwell design. For the first time in the high end compute line of GPUs, the FP64 performance decreased while all other formats continue the drastic pace of multi-fold improvements. The previous generation chip, NVIDIA Hopper, is rated at over 50 Tflop/s in FP64 and over 1500 Top/s in INT8. The hardware’s sparsity feature raises it over 3000 Tera-op/s. Both of these ratings take advantage of the tensor core units. By comparison, Blackwell is rated at 40 Tflop/s in FP64 because the tensor core unit instructions are relegated to the fused multiply-add units. At the same time, the INT8 tensor cores are rated at nearly 5 Peta-op/s (or 10 Peta-op/s with sparsity). To summarize, the advantage of INT8 over FP64 grows from 30-fold in Hopper to well over 100-fold in Blackwell. This creates an opportunity to exploit this imbalance and emulate FP64 using the INT8 tensor core units.

In this paper, we investigate the numerical and performance aspects of floating-point emulation. We focus specifically on the dense methods for linear systems where calculations using FP64 are always in a great need of hardware acceleration.

## II. EMULATION OF FP64 AND INT8 SPLITTING

The basic theory of emulation is based on the error-free transformations [3] and predates the existence of fast tensor core units in hardware. The implementations of this scheme came later targeting low-precision floating-point formats [4] and the use of integer units followed [5]. Without going into details, the first step is to split the product of dense matrices  $A, B \in \mathbb{R}^{n \times n}$  into individual components that are mapped onto separate compute units:

$$AB = (A_1 + A_2)(B_1 + B_2) = A_1B_1 + A_1B_2 + A_2B_1 + A_2B_2 \quad (1)$$

where matrices  $A_1, A_2, B_1, B_2$  are derived from  $A$  and  $B$ , respectively, by splitting the 53 bits of FP64 significand (or mantissa) to fit the target compute units. For example, splitting

onto 32-bit units, we would use a multiplier of  $2^{32}$  so that  $\|A_1\| \approx 2^{32}\|A_2\|$  meaning that  $A_1$  stores the higher bits of the mantissa and  $A_2$  stores the lower ones. For 8-bit splitting, we need to use more components to account for a shorter elements of the hardware tensor cores. In general, the  $k$ -way splitting can be represented as

$$AB = \left( \sum_{i=1}^k A_i \right) \left( \sum_{i=1}^k B_i \right) = \sum_{i,j=1\dots k} A_i B_j. \quad (2)$$

As multiplication produces extra bits on output (up to twice as many bits as either of the inputs) they must be rounded which happens automatically in FP64 hardware. For splitting schemes such as (1), the rounding is induced by truncating the summation early and not performing the dropped matrix products such as  $A_2 B_2$ . Or for the general formula (2), the rounding would set a threshold  $t$  and drop the components of the sum for which  $i + j > t$ . The additional details involving numerical aspects of rounding are out of scope here.

The important aspect of split-scheme emulation is its lack of exponent because all bits in the split form come from the original significands (mantissae) thus drastic difference in number ranges of matrix elements causes representation issues. Consider a trivial 2-by-2 matrix multiply with identity

$$AB \equiv \begin{bmatrix} 2^{+53} & 2^{+53} \\ 2^{-53} & 2^{-53} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (3)$$

with a wildly changing numerical range between the 1<sup>st</sup> and 2<sup>nd</sup> row of matrix  $A$ . Any fixed-width storage scheme such as the one used for the split-scheme emulation of FP64 would round down the second row of  $A$  to 0 due to its limited number of allocated space for the bits of split matrices. In this exaggerated example the loss of accuracy due to the dropped values seems small compared to the large norm of  $A$  but in practice this effect compounds for larger matrices as we show below in our numerical experiments. Here, we only indicate how this problem with numeric range could result from poor scaling or even a permutation of rows and/or columns that cluster together matrix entries in a way that causes truncation of mantissa bits when switching to the split integer representation. Consider the original matrix product with scaling applied to both arguments on both sides:

$$(L_A A R_A)(L_B B R_B) \quad (4)$$

where we select the left and right scaling matrices to increase the dynamic range of the entries in either  $A$  or  $B$ . This could be performed with trivial diagonal matrices that have a diagonal entries varying widely in dynamic range just as the rows of matrix  $A$  in (3). Another possibility is to choose the left/right matrices, say  $L_A$  and  $R_A$ , to be a permutation matrices that group together rows and columns of  $A$  by increasing their dynamic range. Either diagonal scaling or permutation matrices are cheap to apply and invert so they are rarely considered to be an issue for calculations in FP64.

The issues with split integer emulation presented so far were applied to a generic matrix product. While they are concerning in general, they may not be of practical importance for the common dense operations involving matrix product. We next turn our attention to the case of dense linear solvers where the numeric range issues could be masked for some matrices, yet the problems persists if they are introduced through a particular selection of matrix elements.

### III. RELATED WORK

The foundational work that defined the error-free computations is now commonly referred as the Ozaki Scheme [3]. In order to make it more widely applicable, the constant values used for splitting were subsequently generalized [6]. An implementation was later revised to further improve the accuracy profile for the 2-slice scenario [7]. To determine the best splitting in practice, a method of was proposed to both test and validate the various choices [8]. Unlike the original work, there is also a potential to apply integer splitting to the case of sparse matrix multiplication [9]. The double precision matrix-matrix multiply or DGEMM() emulation was the original application but the use of FP16 Tensor Cores was also possible leveraging another piece of computational hardware units inside modern accelerators [4]. Another extension of the integer splitting approach was to extend the precision of the emulated format focusing on quadruple precision or FP128 for the matrix multiplication operation [10]. An alternative approach to error-free transformation used both the Cauchy–Schwarz inequality and Karatsuba algorithm to arrive at a matrix multiplication algorithm [11]. The derivation of the best splitting was further enhanced with an iterative procedure [12]. An arbitrary length precision was applied to both inner products and sparse matrix-vector multiplications [13]. Improved accuracy enhancing efficiency for symmetric rank- $k$  operations were done in the context of eigenvalue and singular value solvers [14]. The recent efforts implemented DGEMM() using emulation with INT8 Tensor Cores [15]. Further acceleration of DGEMM() emulation was achieved by the means of combing INT8 Tensor Cores with application of error-free computation method [16]. Finally, for the case of unequal sizes of integer splits, an accuracy extension was proposed to handle the original error-free transformation [17].

### IV. DENSE NUMERICAL SOLVERS USING FP64 EMULATION

Decompositional factorizations are commonly used in computational and data sciences as both efficient and numerically stable methods for obtaining a solution of a system of linear equations. In its generic form written as

$$Ax = b \quad (5)$$

where  $A \in \mathbb{R}^{n \times n}$  and  $x, b \in \mathbb{R}^n$ . We are concerned here with general non-singular matrices but it is possible our methods would equally apply to symmetric cases either positive definite or indefinite. Given a two-by-two partitioning  $A$  from (5)

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (6)$$

we can use the decompositional approach by factoring  $A$  into a product of triangular matrices producing the LU factorization as is the case for the HPL benchmark [18]. In this process, all 3<sup>rd</sup> order flops are concentrated in the Schur complement:

$$A'_{22} \leftarrow A_{22} - A_{21}A_{11}^{-1}A_{12} = A_{22} - A_{21}(L_{11}U_{11})^{-1}A_{12} \quad (7)$$

which is a perfect target for fast matrix-matrix multiplication such as the one implemented by the split-scheme emulation of FP64. The question is then if there is a potential of adverse scaling from (4) to enter the factorization process thus forcing truncation and increasing the errors beyond standard floating-point roundoff. The basic observation is that the update formula (7) has self-stabilizing property stemming from the pivoting process that chooses the diagonal elements for scaling  $A_{11}$  to remain numerically stable under mild assumptions about the initial conditioning of  $A$ . Using the 2-by-2 partitioning of (6), partial pivoting (the most common pivoting scheme for LU factorization) selects pivots from  $[A_{11}A_{21}]^T$  and produces triangular  $L_{11}$  that has 1's on the diagonal and entries of magnitude less than 1 below the diagonal. This leaves  $U_{11}$  to be the potential source of issues with large disparity in numeric range as this upper triangular factor remains unscaled by the pivot value used for  $L_{11}$ . However, the application of  $U_{11}^{-1}$  to  $A_{21}$  occurs inside the panel factorization (the LU factorization of a rectangular portion of  $A$ ) and it is done by a custom code that is unlikely to benefit from increased performance of the split-scheme emulation. As a result, it remains challenging to observe problematic scaling inside the Schur complement, which leaves us to consider a different type of matrices that may potentially create issues for the split-scheme emulation.

In order to measure the numerical errors in the computed solution, we will be using the *scaled residual* defined as

$$\text{Scaled residual} = \frac{\|Ax - b\|_{\infty}}{(\|A\|_{\infty}\|x\|_{\infty} + \|b\|_{\infty})n\epsilon} \quad (8)$$

which needs to be below 16 for a valid HPL run. [18]

## V. PARAMETRIC FAMILY OF MATRICES WITH CHALLENGING NUMERICAL RANGE

We turn now to a classic example of matrix showing exponential element growth when using partial pivoting. It is commonly attributed to Wilkinson [19], [20] and it is simply an almost lower-triangular matrix with 1's on the diagonal, -1's below the diagonal, and an extra 1's above the diagonal in the last column. A 5-by-5 example looks as following:

$$\text{Wilkinson}_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}. \quad (9)$$

Factorizing the Wilkinson matrix of size  $n$  using partial pivoting produces  $2^n$  element growth and thus it requires

complete pivoting to limit the pivot growth to a 3<sup>rd</sup> degree polynomial [21]. This is impractical because of how complete pivoting inhibits parallelism. But the element growth of Wilkinson matrices is the property allowing us to arbitrarily increase the numerical range of the matrix elements, which is challenging to handle by the split-scheme emulation of FP64.

We first limit the element growth in the Wilkinson matrices by observing that the inverse of the Turing matrix (no extra 1's in the last column) has the largest element  $2^{n-2}$  [22] in the bottom row and it is the consequence of the summation of all previous column elements of the inverse:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 \\ 4 & 2 & 1 & 1 & 0 \\ 8 & 4 & 2 & 1 & 1 \end{bmatrix}. \quad (10)$$

This exponential growth can be mitigated by limiting how many -1's are below the diagonal and indicate this with parameter  $d$ . For  $n = 5$  and  $d = 2$  we have

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & -1 & -1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 \\ 3 & 2 & 1 & 1 & 0 \\ 5 & 3 & 2 & 1 & 1 \end{bmatrix}, \quad (11)$$

which leads us to observe that the numbers in columns form the Fibonacci sequence:  $F_n = F_{n-1} + F_{n-2}$ . This is still an exponentially growing sequence but it is also exponentially slower than the original. In fact, we can regulate the base of the exponent by adjusting the  $d$  parameter because the growth of values in the inverse matrix is determined by a generalized Fibonacci sequence  $G_n^{(d)}$ :

$$G_n^{(d)} = \sum_{i=1}^d G_{n-i}. \quad (12)$$

It is easy to establish that  $F_n \equiv G_n^{(2)}$  and  $G_n^{(n)} = 2^n$  when we set  $G_0^{(n)} = 1$  and  $G_i^{(n)} = 2^{i-1}$  for  $1 \leq i \leq n$ .

The next characteristic of the Wilkinson growth is a long running sequence of 1's that allows expansion of a long sequence with geometric doubling. By using the  $d$  parameter we are able to limit the exponent base by choosing the right  $G_n^{(d)}$  sequence. To limit the sequence's length, consider an example with  $n = 5$ ,  $d = 5$  and a blocking factor  $b = 2$ :

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ -1 & 1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & -\frac{1}{5} & -\frac{1}{5} & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{5} & -\frac{1}{5} \\ 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{5} \\ \frac{1}{2} & \frac{1}{5} & -\frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix}, \quad (13)$$

where the blocking factor  $b$  becomes the length of the sequence with exponential growth in the matrix inverse.

TABLE I

SCALED RESIDUAL ERROR FOR A SOLVE OF A SYSTEM OF LINEAR EQUATIONS WITH A NEW DATA GENERATION SCHEME PARAWILK<sub>256</sub>( $d = 4, b = 15, \alpha = 1/2$ ). PASSING VALUE IS ASSUMED TO BE 16.0 AND REQUIRES AT LEAST 7 INT8 SPLITS. THE BOTTOM ROW SHOWS RESULT WITH HARDWARE-NATIVE FP64.

| INT8 splits   | Scaled residual     |
|---------------|---------------------|
| 3             | 26464646.4755162261 |
| 4             | 341348.2056036110   |
| 5             | 3798.4563646804     |
| 6             | 39.1393959359       |
| 7             | 0.3245150561        |
| 8             | 0.0030077355        |
| 9             | 0.0001860455        |
| FP64 (cuBLAS) | 0.0002377248        |

Finally, we observe that the geometric sequences formed in our parametrized form of the Wilkinson matrix depend on the initial value of the in the upper part of the matrix. We thus replace those upper-part 1's with the third and final parameter  $\alpha$ . We arrive at the 3-parameter formulation that we refer to as ParaWilk <sub>$n$</sub> ( $d, b, \alpha$ ).

## VI. RANDOMIZATION FOR PARAMETRIC MATRICES

The common problem with parametrized matrix families is that they may admit closed-form inverses and thus are problematic for data generators in the context of scalable benchmarks. [23] This leads us down the path of randomization, which is already used by HPL [18] with the matrix elements drawn from a uniform distribution ranging between  $-1/2$  and  $1/2$  denoted as  $U(-1/2, 1/2)$ . Such random distributions tend to generate matrices with favorable numerical properties for solvers based on the standard floating-point solvers. [24] However, having matrix entries with alternating signs creates a greater likelihood for cancellation of growth in number range when performing the Schur complement (7). To counteract this, we switch our matrix generator to produce only positive random entries skewed towards larger values. In particular we used  $2U_n^2(0, 1)$  to produce the results in the numerical experiments section. However, the randomization is not applied uniformly in order to preserve the numerical pattern created by the ParaWilk formula from Section §V. We use nonzero pattern function

$$\text{nnzpattern}(A) = \begin{cases} 1 & \text{if } a_{ij} \neq 0, \\ 0 & \text{if } a_{ij} = 0. \end{cases} \quad (14)$$

and only add the random entries where the ParaWilk is 0:

$$A \sim (1 - \text{ParaWilk}(d, b, \alpha)) \odot 2U^2(0, 1) + \text{ParaWilk}(d, b, \alpha) \quad (15)$$

where  $\odot$  is Hadamard product or element-wise multiply.

## VII. NUMERICAL PROPERTIES OF LINEAR SOLVER WITH INT8 EMULATION

We begin the numerical results by showing in Tab. I the increasing number of INT8 splits and the corresponding values of the scaled residual given by (8): the main metric that determines correctness of an HPL run [18]. For the results

TABLE II

SELECTED PARAMETER VALUES OF THE PARAWILK <sub>$n$</sub> ( $d, b, \alpha$ ) FAMILY OF MATRICES AND THE SCALED RESIDUAL VALUE AFTER LU FACTORIZATION AND SOLVE USING THE INTEGER SPLITTING EMULATION OF FP64. WE USED  $\alpha = 1$  FOR EXPERIMENTS IN THE TABLE.

| $n$  | 6 splits |     |                 | 7 splits |     |                 |
|------|----------|-----|-----------------|----------|-----|-----------------|
|      | $d$      | $b$ | Scaled residual | $d$      | $b$ | Scaled residual |
| 256  | 4        | 15  | 39.1            | 10       | 15  | 17.3            |
| 384  | 4        | 18  | 19.3            | 11       | 12  | 18.9            |
| 512  | 6        | 16  | 19.6            | 12       | 16  | 17.6            |
| 640  | 7        | 22  | 21.7            | 13       | 16  | 24.1            |
| 768  | 6        | 19  | 17.6            | 13       | 25  | 19.3            |
| 896  | 8        | 15  | 40.3            | 14       | 28  | 19.0            |
| 1024 | 8        | 23  | 22.4            | 15       | 19  | 16.3            |
| 1152 | 8        | 15  | 40.5            | 16       | 18  | 18.2            |
| 1280 | 8        | 16  | 25.8            | 16       | 30  | 18.9            |
| 1408 | 9        | 16  | 22.2            | 18       | 22  | 24.6            |
| 1536 | 9        | 28  | 27.5            | 17       | 23  | 26.3            |
| 1664 | 11       | 19  | 40.6            | 17       | 26  | 16.7            |
| 1792 | 10       | 23  | 16.5            | 19       | 23  | 61.6            |
| 1920 | 10       | 15  | 17.0            | 18       | 20  | 38.8            |
| 2048 | 11       | 23  | 20.7            | 19       | 26  | 19.3            |

in the table, the data generator in the tests produced the following matrix: ParaWilk<sub>256</sub>( $d = 4, b = 15, \alpha = 1/2$ ). There are a few conclusions that can be reached from this basic test. Firstly, it is possible to use INT8 splitting to emulate FP64 for this matrix. However, it required as many as 7 splits to reach the passing value of the scaled residual, which is prohibitive from the performance standpoint as shown in the next section. Additionally, as many as 9 splits were needed to reach the same value of the scaled residual as was achieved with hardware-native FP64 which is shown at the bottom row of the table. Thus, our parametric data generator can be used as a detector for using FP64 emulation with integer splitting.

Next we turn to more comprehensive numerical results shown in Fig. 1, which intends to shed more light on the scaling properties of our proposed parametric family of matrices as their sizes increase. The figure shows a variety of numerical and statistical metrics to indicate their behavior in the tested range of matrix dimensions as the implementation of the LU factorization changes from using the hardware-native FP64 at the top to using either 6 or 7 INT8 splits shown on two bottom plots, respectively. The matrix used in these tests was generated with the standard HPL generator that used uniform RNG centered around 0:  $A \sim U(-1/2, 1/2)$ .

Finally, in Tab. II we show results from a scaling experiments that indicate it is possible to further extend the numerical accuracy effects in the LU factorization when choosing the right parameters of our family of matrices given by the ParaWilk formulation presented in Section §V. In order to limit the number of experiments, we only present the first set of parameters we encountered that caused the scaled residual exceed its maximum value of 16. Generally speaking, the values of  $d$ , or the number of elements in the  $G_n^{(d)}$  sequence (12), are slowly monotonically increasing with  $n$  in a faster-than-logarithmic rate. The blocking parameter  $b$  is non-monotonic but its growth has the same order. This is

## VIII. PERFORMANCE RESULTS

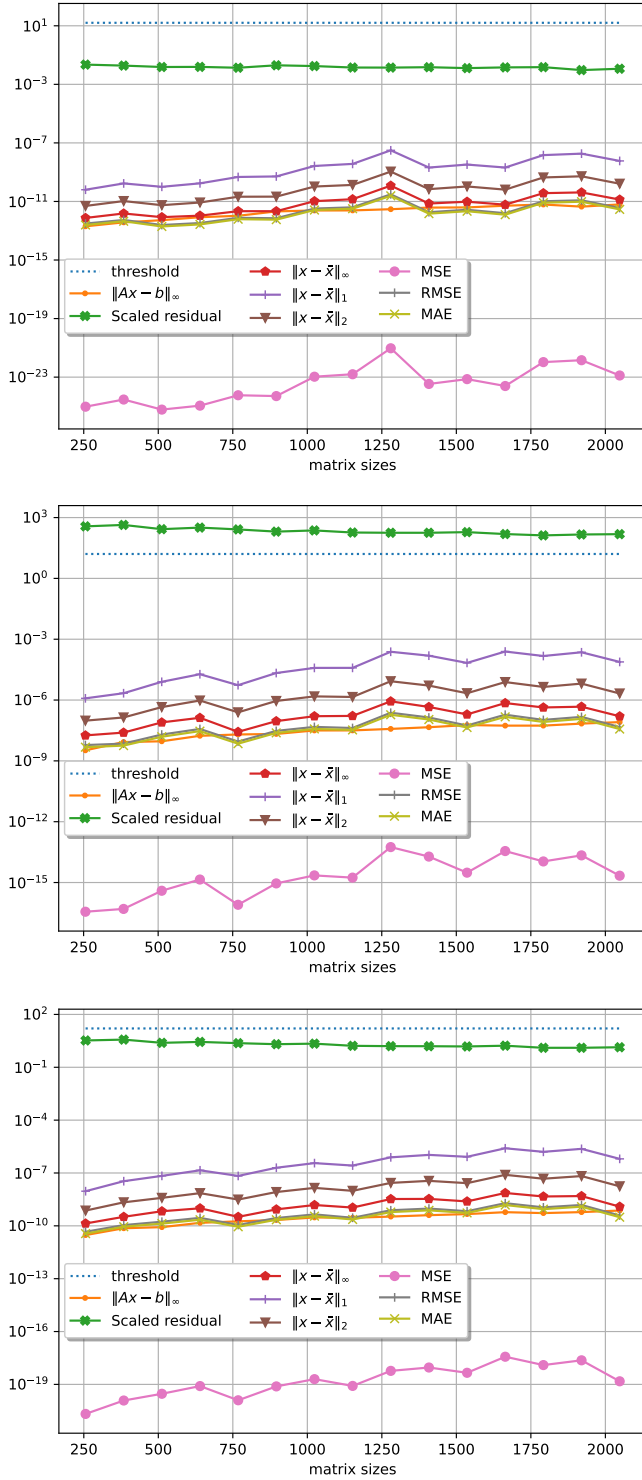


Fig. 1. Comparison of numerical and statistical metrics for the solution of a system of linear equations of increasing sizes with  $A \sim U(-1/2, 1/2)$ . The results from using FP64 hardware in cuBLAS is shown in the top. The bottom two plots show 6 and 7 INT8 splits, respectively, and only the latter achieves the passing values of the scaled residual: 16 (dotted line on all plots).

a positive indicator that the data generator is suitable to use with larger matrix sizes.

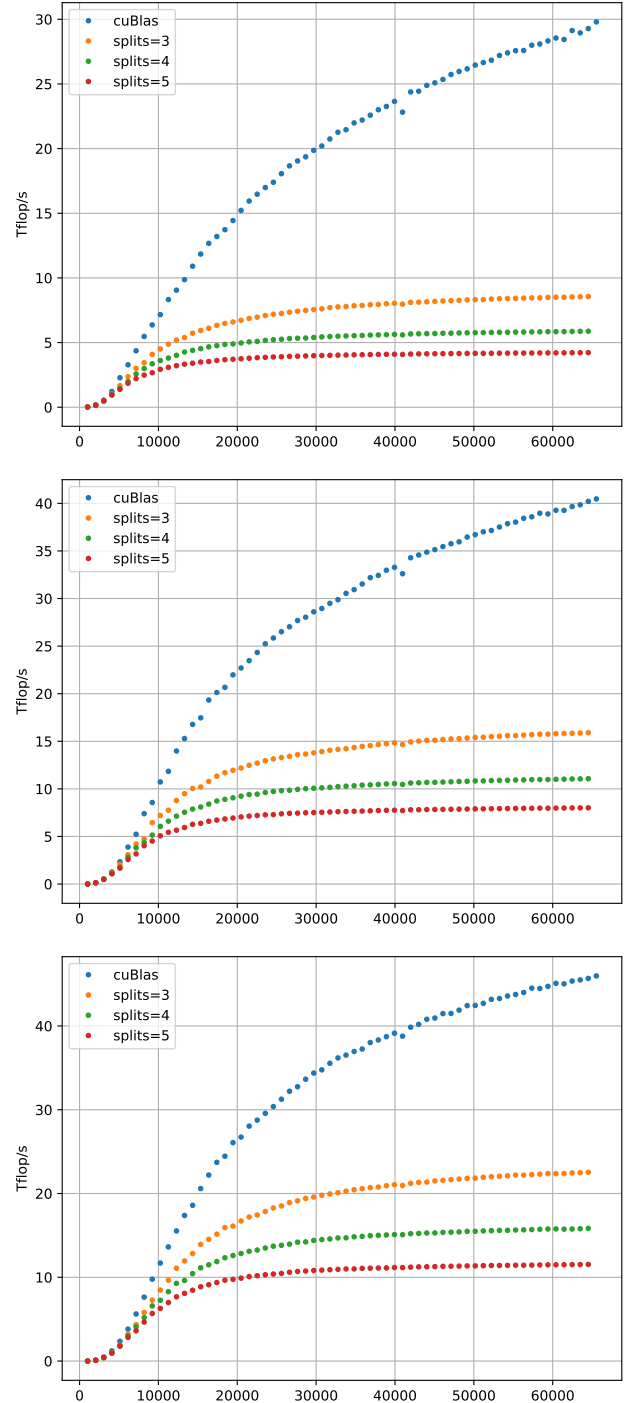


Fig. 2. Performance of cuBLAS in FP64 compared to split integer implementation with varying number of splits and blocking factors: 256 (top), 512 (center), and 768 (bottom).

In this section, we focus on the performance results to indicate the benefits of INT8 split emulation of FP64 on a widely available NVIDIA Hopper card. In particular, we used PCIe version 4 model of this GPU with 114 Streaming

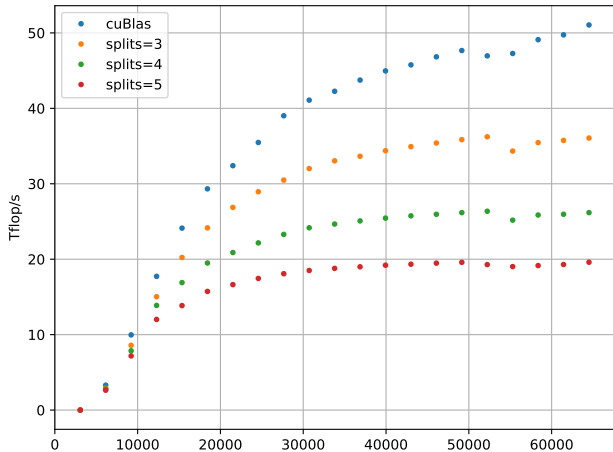
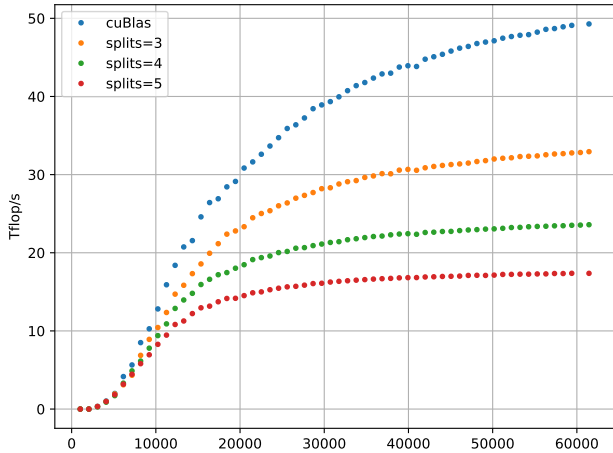
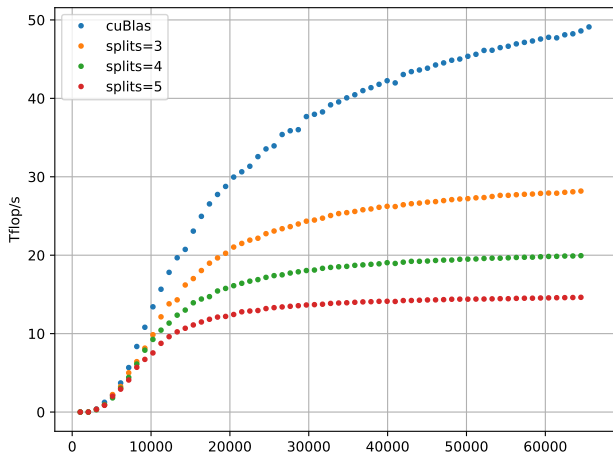


Fig. 3. Performance of cuBLAS in FP64 compared to split integer implementation with varying number of splits and blocking factors: 1024 (top), 1280 (center), and 1536 (bottom).

Multiprocessors and over 55 Tflop/s of FP64 performance at the Lincoln Laboratory Supercomputing Center. [25]

For our software implementation, we used the integer split emulation code available in the Microsoft Github repository under handle `enp1s0/ozIMMU` which is the basis for the official RIKEN repository Microsoft Github located at RIKEN-

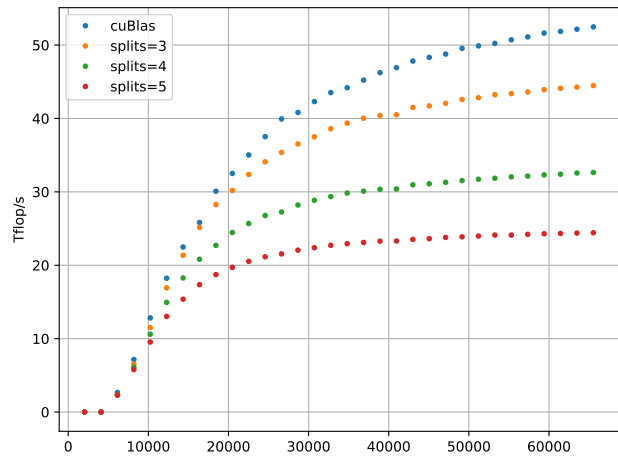


Fig. 4. Performance of cuBLAS in FP64 compared to split integer implementation with varying number of splits and blocking factor of 2048.

### RCCS/accelerator\_for\_ozIMMU.

In order to focus the obtained performance values on the performance obtained from the subsequent Schur complements (7), we simplified the implementation to primarily use Level 3 BLAS without using memory-bound operations such as pivoting. This made our results higher than many available factorization codes thus maximizing the influence of the matrix product implemented with INT8 splitting emulation. Fig. 2 shows results for the blocking factors 256, 512, and 768. Fig. 3 shows results for the blocking factors 1024, 1288, and 1536. Fig. 4 shows results for the blocking factor 2048. Some of the blocking factors have prime factors different than 2 and thus they divide fewer matrix sizes, which is reflected in the plots with less measurement points. This is especially visible for blocking factor 1536 in Fig. 3.

### IX. CONCLUSIONS AND FUTURE WORK

We presented the numerical and performance results from running HPL’s LU factorization using hardware-native FP64 and INT8 splitting emulation. We showed how current random number generator poses little problems for the split integer emulation and we proposed an alternative data generator that is capable of posing more challenging numerical range of entries. This resulted in a greater potential of numerical failures of the emulation and may be indicative of issues possible in applications using similar dense solvers.

We envision a number of extensions of our work including additional solver types, matrix properties [26], and data formats in need of input data triggering numerical inaccuracies.

### ACKNOWLEDGMENTS

The authors wish to acknowledge the following individuals for their contributions and support: Koley Borchard, Chris Bernardi, Bob Bond, Alan Edelman, Peter Fisher, Jeff Gottschalk, Chris Hill, Charles Leiserson, Kirsten Malvey, Sanjeev Mohindra, Heidi Perry, Christian Prothmann, Steve Rejto, Scott Ruppel, Daniela Rus, Mark Sherman, Marc Zissman.

## REFERENCES

- [1] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, B. F. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y. M. Tsai, and U. M. Yang, "A survey of numerical linear algebra methods utilizing mixed-precision arithmetic," *The International Journal of High Performance Computing Applications*, 2021. [Online]. Available: doi.org/10.1177/109434202111003313
- [2] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–9. [Online]. Available: doi.org/10.1109/HPEC.2019.8916327
- [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," *The Numerical Algorithms*, vol. 59, no. 1, pp. 95–118, 2012.
- [4] D. Mukunoki, K. Ozaki, T. Ogita, and T. Imamura, "DGEMM using tensor cores, and its accurate and reproducible versions," in *High Performance Computing*, P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief, Eds. Springer International Publishing, Cham, 2020, pp. 230–248.
- [5] H. Ootomo and R. Yokota, "Recovering single precision accuracy from tensor cores while surpassing the FP32 theoretical peak performance," *The International Journal of High Performance Computing Applications*, vol. 36, no. 4, pp. 475–491, 2022. [Online]. Available: doi.org/10.1177/10943420221090256
- [6] K. Ozaki, T. Ogita, S. Oishi, and S. M. Rump, "Generalization of error-free transformation for matrix multiplication and its application," *Nonlinear Theory and Its Applications*, vol. 4, no. 1, pp. 2–11, 2013.
- [7] K. Ozaki, T. Ogita, and S. Oishi, "Improvement of error-free splitting for accurate matrix multiplication," *Journal of computational and applied mathematics*, vol. 288, p. 127–140, 2015.
- [8] —, "Error-free transformation of matrix multiplication with a posteriori validation," *Numerical Linear Algebra with Applications*, vol. 23, no. 5, p. 931–946, 2016.
- [9] S. Ichimura, T. Katagiri, K. Ozaki, T. Ogita, and T. Nagai, "Threaded accurate matrix-matrix multiplications with sparse matrix-vector multiplications," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, p. 1093–1102. [Online]. Available: doi.org/10.1109/IPDPSW.2018.00168
- [10] D. Mukunoki, K. Ozaki, T. Ogita, and T. Imamura, "Accurate matrix multiplication on binary128 format accelerated by ozaki scheme," in *Proceedings of the 50th International Conference on Parallel Processing (Lemont, IL, USA) (ICPP '21)*. New York, NY, USA: Association for Computing Machinery, 2021, article 78, 11 pages. [Online]. Available: doi.org/10.1145/3472456.3472493
- [11] M. Lange and S. M. Rump, "An alternative approach to ozaki's scheme for error-free transformation of matrix multiplication," in *International Workshop on Reliable Computing and Computer-Assisted Proofs*, 2022.
- [12] K. Ozaki, D. Mukunoki, and T. Ogita, "Error handling and applications of iterative error-free transformations for matrix multiplication (in Japanese)," in *The 18th Joint Meeting of JSIAM Activity Groups*, 2022.
- [13] D. Mukunoki, K. Ozaki, T. Ogita, and T. Imamura, "Infinite-precision inner product and sparse matrix-vector multiplication using ozaki scheme with DOT2 on manycore processors," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2022, p. 40–54.
- [14] Y. Uchino, "Study on high-reliability numerical methods for eigenvalue/singular value decomposition and matrix multiplication (in Japanese)," Ph.D. dissertation, Shibaura Institute of Technology, 2024.
- [15] H. Ootomo, K. Ozaki, and R. Yokota, "DGEMM on integer matrix multiplication unit," *The International Journal of High Performance Computing Applications*, vol. 38, no. 4, pp. 297–313, 2024. [Online]. Available: doi.org/10.1177/10943420241239588
- [16] Y. Uchino, K. Ozaki, and T. Imamura, "Performance enhancement of the Ozaki scheme on integer matrix multiplication unit," *The International Journal of High Performance Computing Applications*, vol. 39, no. 3, pp. 462–476, 2025. [Online]. Available: doi.org/10.1177/10943420241313064
- [17] K. Ozaki, D. Mukunoki, and T. Ogita, "Extension of accurate numerical algorithms for matrix multiplication based on error-free transformation," *Japan Journal of Industrial and Applied Mathematics*, vol. 42, no. 1, pp. 1–20, 2025.
- [18] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: Past, present, and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, August 10 2003, DOI: 10.1002/cpe.728.
- [19] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. Princeton, NJ, USA: Prentice-Hall, 1963.
- [20] —, *The Algebraic Eigenvalue Problem*. London, UK: Oxford University Press, 1965.
- [21] L. N. Trefethen and R. S. Schreiber, "Average-case stability of Gaussian elimination," *SIAM Journal on Matrix Analysis and Applications*, vol. 11, no. 3, pp. 335–360, Jul. 1990.
- [22] A. M. Turing, "Rounding-off errors in matrix processes," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 1, no. 1, pp. 287–308, Jan. 1948.
- [23] M. Fasi and N. J. Higham, "Matrices with tunable infinity-norm condition number and no need for pivoting in lu factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 42, no. 1, pp. 417–435, 2021. [Online]. Available: doi.org/10.1137/20M1357238
- [24] P. Luszczek, Y. Tsai, N. Lindquist, H. Anzt, and J. Dongarra, "Scalable data generation for evaluating mixed-precision," in *Proceedings of IEEE HPEC'20: High Performance Extreme Computing*, Waltham, MA, USA, 2020.
- [25] A. Reuther, J. Kepner, C. Byun, S. Samsi, W. Arcand, D. Bestor, B. Bergeron, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, L. Milechin, J. Mullen, A. Prout, A. Rosa, C. Yee, and P. Michaleas, "Interactive supercomputing on 40,000 cores for machine learning and data analysis," in *2018 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2018, pp. 1–6. [Online]. Available: doi.org/10.1109/HPEC.2018.8547629
- [26] G. Zielke, "Testmatrizen mit maximaler Konditionszahl," *Computing*, vol. 13, no. 1, pp. 33–54, Mar. 1974. [Online]. Available: doi.org/10.1007/BF02268390