

# Applications of Mathematics

---

Katsuhisa Ozaki; Takeshi Terao; Takeshi Ogita; Takahiro Katagiri  
Verified numerical computations for large-scale linear systems

*Applications of Mathematics*, Vol. 66 (2021), No. 2, 269–285

Persistent URL: <http://dml.cz/dmlcz/148723>

## Terms of use:

© Institute of Mathematics AS CR, 2021

Institute of Mathematics of the Czech Academy of Sciences provides access to digitized documents strictly for personal use. Each copy of any part of this document must contain these *Terms of use*.



This document has been digitized, optimized for electronic delivery and stamped with digital signature within the project *DML-CZ: The Czech Digital Mathematics Library* <http://dml.cz>

## VERIFIED NUMERICAL COMPUTATIONS FOR LARGE-SCALE LINEAR SYSTEMS

KATSUHISA OZAKI, Saitama, TAKESHI TERAOKA, Saitama,  
TAKESHI OGITA, Tokyo, TAKAHIRO KATAGIRI, Nagoya

Received November 25, 2019. Published online January 15, 2021.

*Abstract.* This paper concerns accuracy-guaranteed numerical computations for linear systems. Due to the rapid progress of supercomputers, the treatable problem size is getting larger. The larger the problem size, the more rounding errors in floating-point arithmetic can accumulate in general, and the more inaccurate numerical solutions are obtained. Therefore, it is important to verify the accuracy of numerical solutions. Verified numerical computations are used to produce error bounds on numerical solutions. We report the implementation of a verification method for large-scale linear systems and some numerical results using the RIKEN K computer and the Fujitsu PRIMEHPC FX100, which show the high performance of the verified numerical computations.

*Keywords:* verified numerical computation; floating-point arithmetic; high-performance computing; large-scale linear system

*MSC 2020:* 65G20, 65G50, 65Y05

### 1. INTRODUCTION

The computational speed of supercomputers, e.g. floating-point operations per second, and the total amount of memory increase rapidly. Hence, the treatable problem size in scientific computing is getting larger. Floating-point numbers and their arithmetic are widely used for numerical computations. Today's computers support standard floating-point formats, such as binary64, binary32, and more seldom

---

This research was supported by MEXT as “Exploratory Challenge on Post-K computer” (Development of verified numerical computations and super high-performance computing environment for extreme researches) using computational resources of the K computer at RIKEN R-CCS and Fujitsu FX100 at Nagoya University through the HPCI System Research Project (Project ID: hp190192).

also binary16 as defined in IEEE 754 [6]. The computational precision supported by standard hardware has not increased over 30 years. The larger the problem size, the more number of floating-point operations are involved. Then the problems caused by rounding errors can become crucial.

Many scientific problems are boiled down to linear systems. Therefore, it is important to obtain an accurate numerical solution of a linear system

$$(1.1) \quad Ax = b$$

and to guarantee its accuracy. So-called verified numerical computations produce a pair of an approximate solution and a corresponding error bound using only floating-point arithmetic. Fast and efficient verification methods have been proposed for linear systems, e.g. [10], [13], [18], [14], [17], [15], [21], [25], [26]. However, there are few results of verified numerical computations for linear systems in massively parallel and distributed computing environments. Kolberg et al. reported that interval linear systems with 100 000 unknowns were solved and the enclosure of the solution set can be obtained [8].

We implemented an algorithm for accurate matrix-vector products and verified numerical computations for linear systems and report some numerical results over 1 000 000 unknowns. Although the method is not new, this paper, first in the world, reports numerical results of such large-scale linear systems. Our goal is to obtain an accurate numerical result and its tight error bound. Each section in this paper answers the following questions for the large-scale linear systems:

- ▷ Section 2: Are rounding errors an actual issue? And is accuracy assurance necessary?
- ▷ Section 3: Is it possible to obtain accurate numerical results using iterative refinements?
- ▷ Sections 4, 5: Is it possible to obtain tight error bounds for accurate numerical results?

In Section 2, we show actual relative errors for the large-scale linear systems. In Section 3, it is shown that an iterative refinement works very efficiently to produce accurate numerical solutions of large-scale linear systems. In Section 4, we introduce several verification methods and discuss implementation issues on super computers. The implemented verification method requires between 4 and 8 times as many node hours as calculating a numerical solution by PDGESV in ScaLAPACK [1] on super-computers as shown in Section 5, where ScaLAPACK is a library of high-performance linear algebra routines for parallel distributed memory machines.

## 2. NOTATION AND ERROR OF FLOATING-POINT ARITHMETIC

Let  $\mathbb{F}$  be a set of binary floating-point numbers as defined in the IEEE 754 standard [6]. The notation  $\mathbf{fl}(\cdot)$ ,  $\mathbf{fl}_{\nabla}(\cdot)$  and  $\mathbf{fl}_{\triangle}(\cdot)$  indicate a result of floating-point arithmetic with rounding to nearest (roundTiesToEven), rounding downward (roundTowardNegative) and rounding upward (roundTowardPositive), respectively. For readability, we omit to use the notation  $\mathbf{fl}$  for each operation in floating-point arithmetic. For example,  $\mathbf{fl}((a + b) + c)$  means  $\mathbf{fl}(\mathbf{fl}(a + b) + c)$ . Let  $u$  be the roundoff unit, for example,  $u = 2^{-53}$  for binary64. The notation  $|\cdot|$  for a vector and a matrix is to take absolute values componentwise. For example, for  $a \in \mathbb{R}^n$ ,  $|a| = (|a_1|, |a_2|, \dots, |a_n|)^\top$ . In the following,  $I$  is used to denote the  $n$ -by- $n$  identity matrix. In our numerical examples, we used the K computer at RIKEN R-CCS and the Fujitsu PRIMEHPC FX100 (hereafter FX100) at Nagoya University. Specifications of these supercomputers are shown in Table 1.

	K computer	FX100
CPU	SPARC64 VIIIfx	SPARC64 XIfx
Peak (GFLOPS)	128	1 126
Number of Cores	8	32
Memory	16 GB	32 GB
Maximum Number of Nodes	82 944	2 880

Table 1. The specification of 1 node of the K computer at RIKEN [23] and the Fujitsu PRIMEHPC FX100 [3] at Nagoya University. GFLOPS in this table indicates Giga Floating-point number Operations Per Second.

We first demonstrate the influence of rounding errors. Using binary64, we produced a linear system whose exact solution is  $x = (1, 1, \dots, 1)^\top$  by using Algorithm 3 in [19].  $A$  is generated as an  $n$ -by- $n$  random matrix.  $A'$  ( $\approx A$ ) is obtained using Algorithm 3 in [19]. Some of lower bits in the significant of the elements in  $A'$  are zeros, so that there is no rounding error in  $b := A'(1, 1, \dots, 1)^\top$ . Hence,  $(1, 1, \dots, 1)^\top$  is the exact solution of  $A'x = b$ . Note that there is a case  $A(1, 1, \dots, 1)^\top \notin \mathbb{F}^n$ . We obtain  $\hat{x}$  as an approximate solution by PDGESV in ScaLAPACK on the K computer. Then, the relative error for  $\hat{x}_i$  is

$$\frac{|x_i - \hat{x}_i|}{|x_i|} = |1 - \hat{x}_i|.$$

Each node has a 40 000-by-40 000 matrix, and we set the block-cyclic size to 200, that is set in array descriptor for block-cyclically distributed matrix in ScaLAPACK. Table 2 shows the average and the maximum relative error for various  $n$ . A floating-point number in binary64 has 53 significant bits including the implicit one. For

$n \geq 320\,000$ , about half of the significant bits of the floating-point result are incorrect because  $2^{-26} \approx 10^{-8}$ . It means that binary64 has approximately 16 decimal digits but only 7–8 decimal digits of the numerical results are correct. When we generate other random matrices, the results have the same tendency.

$n$	Number of Nodes	Average	Maximum
80 000	4	$4.02e - 10$	$2.23e - 09$
160 000	16	$2.94e - 09$	$1.57e - 08$
320 000	64	$1.08e - 08$	$7.17e - 08$
640 000	256	$1.64e - 07$	$1.15e - 06$

Table 2. Average and maximum relative errors.

### 3. ITERATIVE REFINEMENT

First, we show how to obtain an accurate numerical solution of (1.1). We employ iterative refinements, which are well summarized in [4], Chapter 12. Rounding error analysis for the iterative refinements using double of working precision is introduced in [27]. Verification methods for linear system using iterative refinement are proposed in [15], [20]. The steps of the iterative refinement are described as follows:

- 1) solve  $Ax = b$  numerically and obtain an approximation  $\hat{x}$ ,
- 2) compute the residual  $r \approx b - A\hat{x}$ ,
- 3) solve  $Ay = r$  numerically and obtain an approximation  $\hat{y}$ ,
- 4) update  $\hat{x} := \text{fl}(\hat{x} + \hat{y})$ .

We repeat 2), 3), 4) until we obtain an accurate numerical solution. If we perform LU decomposition at 1), we can reuse the LU factors at 3) for obtaining  $\hat{y}$ . Hence, the computational costs for 2), 3) and 4) are almost negligible. In step 2), we need higher-precision computations in the cases where heavy cancellation leads to large relative errors. Note that this partly depends on the condition number of  $A$ , but always depends on the condition numbers of the individual dot products. There are libraries that support accurate matrix-vector product, e.g. XBLAS [9] and MPACK [12]. Because we need an enclosure of matrix-vector product parallel and distributed computing, on the basis of Algorithm 5.3 (Dot2) and Algorithm 5.8 (Dot2Err) in [16], we implemented accurate algorithms for a matrix-vector product for parallel and distributed computing. These are extensions of algorithms in [29]. The cost of an accurate matrix-vector product using Algorithm 5.3 (Dot2) in [16] with Fused Multiply-Add is  $8n^2 - 5n$  flops. The notation ‘‘flops’’ means floating-point operations. Additional computing time using the accurate algorithm for a matrix-vector product is negligible compared to the cost of  $\mathcal{O}(n^3)$  flops for the LU decomposition.

Therefore, we only show the flow of the algorithm, and we omit to explain the detail of the implementation in this paper. Figure 1 shows the image of the accurate dot product for parallel and distributed computing. We applied Algorithm 3.3 (PDotK,  $K = 2$ ) in [29] in each node. Then, the results are transferred to the first node. Finally, we apply Algorithm 4.4 (Sum2) in [16] and obtain the final result in the first node.

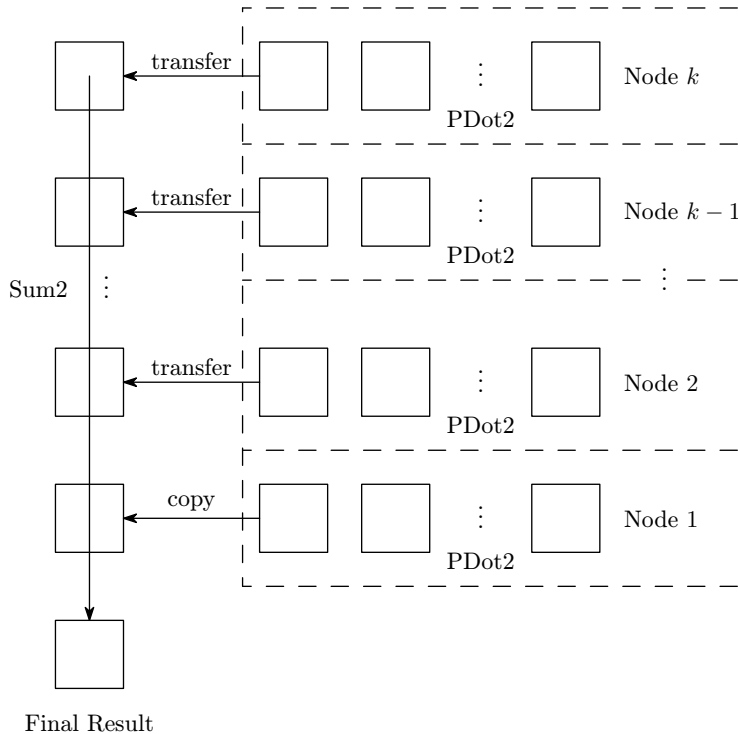


Figure 1. Image of accurate dot product for parallel and distributed computing. The boxes show floating-point numbers.

To evaluate the efficiency of the iterative refinement, we generate a matrix  $A$  whose elements are binary64 floating-point numbers with the condition number  $\|A^{-1}\|_2\|A\|_2$  being approximately  $10^{10}$ . The singular values of  $A$  are geometrically distributed. We use Algorithm 3 in [19] and generate a linear system  $Ax = b$  whose exact solution is  $x = (1, \dots, 1)^\top$ . Tables 3 and 4 show the maximum relative error with iterative refinement for various  $n$ . The results in Table 3 are obtained not using accurate matrix-vector product but PDGEMV in PBLAS. An initial approximate solution  $\hat{x}$  is obtained at 1) using PDGESV in ScaLAPACK. Table 4 indicates that the exact solution can be obtained in three iterations. When we generate other matrices with the condition number  $10^{10}$ , the results have the same tendency. Therefore, we

can see that the iterative refinement with the accurate matrix-vector product works well for such large-scale problems.

$n$	Iteration Number	0	1	2
	160 000	2.69e - 05	7.43e - 09	9.30e - 09
	320 000	5.56e - 04	1.58e - 08	8.28e - 08
	640 000	9.82e - 04	8.65e - 08	9.37e - 08
	1 280 000	2.68e - 02	2.72e - 07	1.85e - 07

Table 3. The maximum relative error using PDGESV in PBLAS.

$n$	Iteration Number	0	1	2	3
	160 000	2.69e - 05	2.89e - 13	0.00e + 00	0.00e + 00
	320 000	5.56e - 04	1.31e - 10	0.00e + 00	0.00e + 00
	640 000	9.82e - 04	4.38e - 10	0.00e + 00	0.00e + 00
	1 280 000	2.68e - 02	1.40e - 07	1.23e - 12	0.00e + 00

Table 4. The maximum relative error using the accurate algorithm for the matrix-vector product.

#### 4. VERIFICATION METHODS FOR LINEAR SYSTEMS

We briefly review a class of verification methods for linear systems. Let  $R \in \mathbb{F}^{n \times n}$ , which is usually given as an approximate inverse of  $A$ . Suppose

$$(4.1) \quad \|RA - I\|_\infty \leq \alpha \in \mathbb{F}, \quad \|R(A\hat{x} - b)\|_\infty \leq \beta \in \mathbb{F}.$$

If  $\alpha < 1$ , then

$$\|\hat{x} - A^{-1}b\|_\infty \leq \frac{\beta}{1 - \alpha}$$

is satisfied [18], [14], [17], [21]. There are several verification methods for computing  $\alpha$  and  $\beta$ . Some of them are listed in Table 5. The cost in the table is the number of floating-point operations including that for the LU decomposition. We omit  $\mathcal{O}(n^2)$  terms in the column “Cost” for simplicity. Note that “Cost” depends on the computation of  $\alpha$ , because only matrix-vector products are necessary for  $\beta$ .

There is a trade-off between the cost of computing and the quality of the upper bound  $\alpha$  in the methods listed in the table. It means that the faster the method, the more likely to fail in verifying  $\alpha < 1$  for ill-conditioned problems. Whether the value of  $\alpha$  is  $10^{-8}$  or 0.1 makes little difference for the inclusion of the quality. So we are interested in the cheapest verification method that is still able to verify  $\alpha < 1$ . The

Method	Developers	Cost	Year
1	Oishi-Rump	$4/3n^3$	2002 [18]
2	Ozaki et al.	$7/3n^3$	2010 [21]
3	Ogita-Oishi	$10/3n^3$	2005 [14]
4	Ogita-Rump-Oishi	$4n^3$	2005 [17]
5	Oishi-Rump	$6n^3$	2002 [18]

Table 5. List of verified numerical computations for linear systems.

comparison of the value of  $\alpha$  computed via the listed methods is given in [21]. Here, we introduce Method 4, which yields the second smallest value to  $\alpha$  in the listed methods. If  $(3n + 2)u < 1$ , Method 4 obtains  $\alpha$  as

$$(4.2) \quad \alpha := \mathbf{fl}\left(\frac{\alpha_1 + \tilde{\gamma}_{3n+2}(\alpha_2 + 2)}{1 - 2u}\right),$$

where

$$(4.3) \quad \alpha_1 := \mathbf{fl}(\|RA - I\|_\infty), \quad \alpha_2 := \mathbf{fl}(\|R\|(|A|e)\|_\infty), \quad \tilde{\gamma}_n := \mathbf{fl}\left(\frac{nu}{1 - nu}\right).$$

We can obtain  $\alpha$  in (4.2) without directed rounding. The main cost for Method 4 is computations of  $R$  and  $\mathbf{fl}(RA)$ . We need  $4n^3 + O(n^2)$  flops, because cost for obtaining  $R$  and  $\mathbf{fl}(RA)$  is  $2n^3 + O(n^2)$  flops. Because

$$(4.4) \quad \alpha \approx \|(3n + 2)u(|R||A| + I)\|_\infty$$

in many cases, large  $n$  immediately produces  $\alpha > 1$ . The constant  $3n + 2$  in (4.4) can be reduced down to  $n$  as introduced in Appendix<sup>1</sup>. However, it is still proportional to  $n$ , and it turns out that the bound (4.2) cannot be applied for large  $n$ . Thus, we straightforwardly adopt Algorithm 4.5 in [15] as an established method for computing  $\alpha$  to show some numerical examples in this section.

Table 6 shows  $\alpha$  obtained by (4.2) using the K computer. The column ‘‘Cnd’’ refers to the condition number of the coefficient matrix. If the computation yields  $\alpha \geq 1$ , then the verification failed. We use ‘-’ to denote this case. While Method 4 produces  $\alpha < 1$  for  $n = 100$  with the condition number being  $10^{12}$ , it returns  $\alpha > 1$  for  $n = 1\,280\,000$  with the condition number being  $10^4$ . In Methods 1–4, one or another way, the error bound is derived using a worst case error analysis. This reduces the computational complexity but leads to rough enclosures for the actual product  $RA$ . The error estimate increases linearly with  $n$ . Together with the overestimation by

---

<sup>1</sup> Thanks to the reviewer for pointing this out.



the use of the infinity norm, this yields the results presented in Table 6. Therefore, a more robust method is necessary for large-scale problems, and we focus on Method 5 in this paper. Remark that the method by Kolberg et al. is based on the enclosure theory in [24]. The method is essentially the same as Method 5.

Cnd	$n$	100	10 000	1 280 000
$10^2$		1.70e - 10	1.18e - 06	1.80e - 02
$10^4$		1.15e - 07	7.18e - 04	-
$10^6$		5.23e - 06	3.26e - 02	-
$10^8$		6.65e - 05	4.89e - 01	-
$10^{10}$		4.86e - 04	-	-
$10^{12}$		2.75e - 03	-	-

Table 6. Upper bound  $\alpha$  of  $\|RA - I\|_\infty$ .

Oishi and Rump proposed Method 5 obtaining  $\alpha$  and  $\beta$  in (4.1), which exploits directed rounding [18]. They take  $R \in \mathbb{F}^{n \times n}$  as an approximate inverse matrix of  $A$ . Computing  $RA - I$  with rounding downwards and upwards, respectively, we have

$$(4.5) \quad \mathbf{fl}_{\nabla}(RA - I) \leq RA - I \leq \mathbf{fl}_{\Delta}(RA - I).$$

The procedure for (4.5) is to switch a rounding mode to RoundTowardNegative, call PDGEMM, again switch a rounding mode to RoundTowardPositive, and call PDGEMM. To change a rounding mode, the function `fesetround` is supported in C language standardized in C99. We compute a matrix  $T \in \mathbb{F}^{n \times n}$  as

$$(4.6) \quad t_{ij} := \max(|\mathbf{fl}_{\nabla}(RA - I)|_{ij}, |\mathbf{fl}_{\Delta}(RA - I)|_{ij})$$

and  $\alpha$  as

$$(4.7) \quad \alpha := \mathbf{fl}_{\Delta}(\|T\|_\infty).$$

The main cost for computing  $\alpha$  is the computation of  $R$ ,  $\mathbf{fl}_{\nabla}(RA)$  and  $\mathbf{fl}_{\Delta}(RA)$ . Each of them requires  $2n^3 + O(n^2)$  flops, so totally  $6n^3 + O(n^2)$  flops are necessary. Using a probabilistic error bound by Higham and Mary [5], we can expect

$$(4.8) \quad \alpha \lesssim \|\lambda\sqrt{n+1}u(|R||A| + I)\|_\infty + O(u^2),$$

where  $\lambda$  is a constant of  $O(1)$ . Compared to  $\alpha$  from Method 4 in (4.4), Method 5 produces much smaller  $\alpha$  for large  $n$ , so we implement Method 5 for the supercomputers.

Next, we adapt a method in [20] for computing  $\beta$ . Let  $\hat{x} \in \mathbb{F}^n$  be an approximate solution of (1.1). We then compute an enclosure for the residual  $A\hat{x} - b$ . Applying Dot2Err to matrix-vector product, we get  $c_1, r_1 \in \mathbb{F}^n$  such that

$$c_1 - r_1 \leq A\hat{x} - b \leq c_1 + r_1,$$

where all elements in  $r_1$  are non-negative. The vectors  $c_1$  and  $r_1$  mean center and radius, respectively. Using Dot2Err,  $r_1 \ll |c_1|$  is expected. Similarly, an enclosure of  $Rc_1$  is obtained as

$$c_2 - r_2 \leq Rc_1 \leq c_2 + r_2.$$

Then

$$\beta := \|\mathbf{f}1_{\Delta}(|c_2| + r_2 + |R|r_1)\|_{\infty}.$$

The cost for computing  $\beta$  is  $\mathcal{O}(n^2)$  and negligible compared to that for  $\alpha$ .

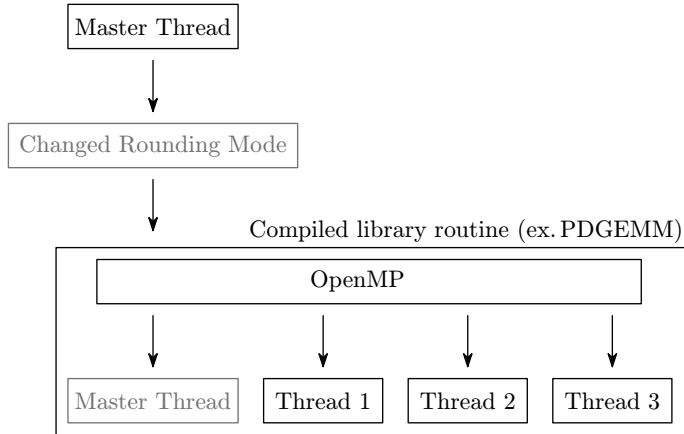


Figure 2. Change of the rounding mode.

Here, we discuss pitfalls for switches of rounding modes on parallel and distributed computers.

- (P1) Even if the rounding mode is changed in the master thread, the rounding mode in the other threads is not changed (Fig. 2).
- (P2) We use reduction functions in Message Passing Interface (MPI), for example, `MPI_Reduce` and `MPI_AllReduce`. The sum is computed not on CPU, but InterConnect Controller (Fig. 3) on the K computer and the FX100, i.e., switches of rounding modes by the function `fesetround` cannot work correctly in this case.
- (P3) If an alternative algorithm for matrix multiplication, such as Strassen [28], is used in PDGEMM, the inequality (4.5) is not generally satisfied.

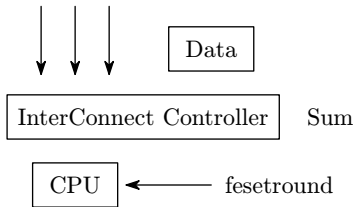


Figure 3. Computation in the reduction.

One may think that we can switch the rounding mode after threading by OpenMP for (P1). However, it is impossible to do it in compiled library routines (Fig. 2), for example, PDGEMM in PBLAS [22] and PDLANGE in ScaLAPACK, which are used for (4.6) and (4.7), respectively. Because MPI\_Reduce is used for both PDGEMM and PDLANGE, (P2) is actually a problem for the verification method.

## 5. NUMERICAL RESULTS

We report several numerical examples using the K computer and the FX100. Thanks to Fujitsu, problems (P1) and (P3) have been solved.

- ▷ Fujitsu provided the function `f1lib_omp_fesetround_`. This works for both the K computer and the FX100. Using the function, the rounding mode of floating-point arithmetic in PBLAS and ScaLAPACK can be changed for all threads in all nodes.
- ▷ Fujitsu confirmed that an alternative algorithm for matrix multiplication, such as Strassen [28], is not applied for PDGEMM in PBLAS.

Besides, if we set the `mca` option for `mpixec` as

```
--mca:coll_base_reduce_commute_safe1,
```

then the order of the reduction is fixed, and floating-point arithmetic is executed using the only CPU. Using this option tends to slow down the performance, but we checked that this is less than 5 % in terms of the overall computing time. With this option being used, changes of the rounding mode are also valid for MPI\_Reduce and MPI\_AllReduce. Therefore, all issues from (P1) to (P3) are solved.

We generate test matrices, whose singular values are geometrically distributed. For a generated matrix  $A$ , let  $t = (1, 1, \dots, 1)^\top$  and compute the right-hand side vector  $b = f1(At)$ . Then  $x \approx (1, 1, \dots, 1)^\top$  for  $Ax = b$  can be expected. We apply three iterative refinements to obtain  $\hat{x}$ . The matrix  $R$  is obtained by solving  $AX = I$  using PDGESV in ScaLAPACK. We set the block-cyclic size to 200. Table 7 shows  $f1_\Delta(\beta/f1_\nabla(1 - \alpha))$  for various condition numbers for  $n = 10\,000$  and  $n = 1\,000\,000$  on the K computer. If the computation yields  $\alpha > 1$ , then the verification failed.

We use ‘-’ to denote this case. The results indicate that tight error bounds can be obtained for large  $n$ .

Cnd	$n$	10 000	1 000 000
	$10^2$	$1.11\text{e} - 16$	$1.11\text{e} - 16$
	$10^4$	$1.11\text{e} - 16$	$1.11\text{e} - 16$
	$10^6$	$1.11\text{e} - 16$	$1.13\text{e} - 16$
	$10^8$	$1.11\text{e} - 16$	-
	$10^{10}$	$1.17\text{e} - 16$	-
	$10^{12}$	-	-

Table 7. Comparison of  $fl_{\Delta}(\beta/fl_{\nabla}(1 - \alpha))$ .

$n$	No. of Nodes	Time (APP)	Time (VNC)	Ratio of Node-hours
300 000	400	631	4 474	7.07
600 000	1 600	1 440	9 116	6.32
1 200 000	6 400	3 575	19 066	5.33
2 400 000	25 600	10 025	40 827	4.07

Table 8. Comparison of node-hours (K computer).

$n$	No. of Nodes	Time (APP)	Time (VNC)	Ratio of Node-hours
120 000	36	111	569	5.09
240 000	144	239	1 137	4.74
360 000	324	392	1 768	4.50
480 000	576	583	2 402	4.11

Table 9. Comparison of node-hours (FX100).

Next, we show computational performance on the K computer and the FX100. We use APP and VNC to denote the approximate computation (PDGESV) and the verified numerical computations including iterative refinements, respectively. Note that the mca option is not set for the numerical example of APP. Tables 8 and 9 show computing times (Time, sec) and ratio(s) of node hours (VNC / APP) on the K computer and the FX100. Note that the theoretical ratio is nearly nine, because the cost of the LU decomposition is  $\frac{2}{3}n^3 + O(n^2)$  flops, and that of the verification is  $6n^3 + O(n^2)$  flops.

There is a difference in the amount of working memory between APP and VNC. For APP without the iterative refinement, only a matrix is stored, because  $A$  is overwritten to the LU factors and an approximate solution is obtained by backward and forward substitutions. On the other hand, four matrices are stored for VNC:  $A$ ,  $R$ ,

$\text{fl}_{\nabla}(RA - I)$  and  $\text{fl}_{\Delta}(RA - I)$ . Therefore, the number of nodes can be reduced for APP, and it makes efficiency of the parallelization better. The ratios of node hours for the reduced number of nodes are shown in Tables 10 and 11. The individual number of nodes is given in the respective columns. In this case, the performance is still better than the theoretical ratio. In particular, the ratio for FX100 is much better than nine. The reason is the difference in any or all of the following factors: the instruction level parallelism (ILP), floating-point operations performance, and throughput of LU decomposition and matrix multiplications.

$n$	No. of Nodes	Time	No. of Nodes	Time	Ratio of
	APP	APP	VNC	VNC	Node-hours
300 000	100	2 203	400	4 470	8.12
600 000	400	4 513	1 600	9 116	8.07
1 200 000	1 600	9 624	6 400	19 066	7.92
2 400 000	6 400	21 843	25 600	40 827	7.47

Table 10. Comparison of node-hours (K computer).

$n$	No. of Nodes	Time	No. of Nodes	Time	Ratio of
	APP	APP	VNC	VNC	Node-hours
120 000	9	483	36	568	4.97
240 000	36	876	144	1 136	5.19
360 000	81	1 309	324	1 768	5.40
480 000	144	1 802	576	2 402	5.33

Table 11. Comparison of node-hours (FX100).

## 6. CONCLUSION

We reported the implementation and performance of verified numerical computations for linear systems on supercomputers. Further results for other verification methods, memory reduced implementation, and implementation of accurate algorithms for a matrix-vector product will appear in forthcoming papers. We will implement recursive block matrix multiplication [2] for the improvement of (4.2). If the dimension of the matrix is 1 000 000, and the condition number of the coefficient matrix is  $10^8$ , the method based on [18] failed to produce the error bound. We will try to produce error bounds for the linear system implementing the methods in [10], [25], [26] as future work. These methods can be applied for the linear system with more ill-conditioned coefficient matrices. Moreover, the linear system with structured matrices such as symmetric positive definite matrix will be the focus in future project on verified numerical computations using supercomputers.

APPENDIX

We introduce a better upper bound compared to (4.2). Let  $u_s$  be the smallest positive floating-point number in  $\mathbb{F}$ . Assume

$$(6.1) \quad (n+3)u < 1, \quad \frac{1}{2}n^2u_s + \frac{1}{2}n^2u \cdot u_s + \frac{1}{2}(1+u)^{n-2}u_s < u.$$

Using (2.1) and (2.2) in [17] yields

$$(6.2) \quad a + b \leq (1+u)\mathbf{fl}(a+b),$$

$$(6.3) \quad ab \leq (1+u)\mathbf{fl}(ab) + \frac{1}{2}u_s$$

for  $0 \leq a, b \in \mathbb{F}$ . If underflow does not occur in  $\mathbf{fl}(ab)$ , the term  $\frac{1}{2}u_s$  in (6.3) can be omitted. From (2.8) in [17],

$$(6.4) \quad (1+u)^n \leq \frac{1}{1-nu}$$

is satisfied. The inequality

$$(6.5) \quad \frac{a}{1-nu} \leq \mathbf{fl}\left(\frac{a}{1-(n+1)u}\right)$$

is given in (2.14) in [17], where  $u^{-1} \cdot u_s < a \in \mathbb{F}$ . Let  $T := \mathbf{fl}((RA) - I)$ . From Lemma 4 in [11], if  $\mathbf{fl}(\|T\|_\infty) < 1$ ,

$$(6.6) \quad |\mathbf{fl}((RA) - I) - (\mathbf{fl}(RA) - I)|_{ii} \leq u$$

is satisfied for all  $i$ . Application of Theorem 4.2 in [7] with care of underflow gives

$$(6.7) \quad |\mathbf{fl}(RA) - RA| \leq nu|R||A| + \frac{1}{2}nu_sE,$$

where all elements in  $E \in \mathbb{F}^{n \times n}$  are ones. From (6.6) and (6.7) we obtain

$$\begin{aligned} T - (RA - I) &= \mathbf{fl}(\mathbf{fl}(RA) - I) - (RA - I) = \mathbf{fl}(RA) - I + \Delta_1 - (RA - I) \\ &= \mathbf{fl}(RA) - RA + \Delta_1 = \Delta_2 + \Delta_1, \end{aligned}$$

where  $|\Delta_1| \leq uI$  and  $|\Delta_2| \leq nu|R||A| + \frac{1}{2}nu_sE$ . Hence,

$$(6.8) \quad |RA - I|e \leq |T|e + |\Delta_2|e + |\Delta_1|e \leq |T|e + nu|R||A|e + \frac{1}{2}n^2u_s e + ue,$$

where  $e := (1, 1, \dots, 1)^\top$ . Applying (6.2) and (6.3), we obtain

$$(6.9) \quad |T|e \leq (1+u)^{n-1} \mathbf{f}1(|T|e)$$

and

$$(6.10) \quad \begin{aligned} nu|R||A|e &\leq nu(1+u)^{n-1}(|R|\mathbf{f}1(|A|e)) \\ &\leq nu\left((1+u)^{2n-1}\mathbf{f}1(|R|(|A|e)) + \frac{1}{2}nu_s e\right) \\ &= nu(1+u)^{2n-1}\mathbf{f}1(|R|(|A|e)) + \frac{1}{2}n^2u \cdot u_s e. \end{aligned}$$

Therefore, (6.8), (6.9) and (6.10) yield

$$(6.11) \quad \begin{aligned} |RA - I|e &\leq |T|e + nu|R||A|e + \frac{1}{2}n^2u_s e + ue \\ &\leq (1+u)^{n-1}\mathbf{f}1(|T|e) + nu(1+u)^{2n-1}\mathbf{f}1(|R|(|A|e)) \\ &\quad + \frac{1}{2}n^2u \cdot u_s e + \frac{1}{2}n^2u_s e + ue. \end{aligned}$$

From (6.4), (6.5) and (6.3), we obtain an upper bound of  $nu(1+u)^{2n-1}\mathbf{f}1(|R|(|A|e))$  as

$$(6.12) \quad \begin{aligned} nu(1+u)^{2n-1}\mathbf{f}1(|R|(|A|e)) &= (1+u)^{n-2}nu(1+u)^{n+1}\mathbf{f}1(|R|(|A|e)) \\ &\leq (1+u)^{n-2}\frac{nu}{1-(n+1)u}\mathbf{f}1(|R|(|A|e)) \\ &\leq (1+u)^{n-2}\mathbf{f}1\left(\frac{nu}{1-(n+2)u}\right) \cdot \mathbf{f}1(|R|(|A|e)) \\ &\leq (1+u)^{n-1}\mathbf{f}1\left(\frac{nu}{1-(n+2)u}(|R|(|A|e))\right) + \frac{1}{2}(1+u)^{n-2}u_s e. \end{aligned}$$

Then, using (6.11), (6.12), (6.1), (6.2), (6.4) and (6.5) yields

$$\begin{aligned} |RA - I|e &< (1+u)^{n-1}\left(\mathbf{f}1(|T|e) + \mathbf{f}1\left(\frac{nu}{1-(n+2)u}(|R|(|A|e))\right)\right) + 2ue \\ &< (1+u)^{n-1}\left(\mathbf{f}1(|T|e) + \mathbf{f}1\left(\frac{nu}{1-(n+2)u}(|R|(|A|e))\right) + 2ue\right) \\ &\leq (1+u)^{n+1} \cdot \mathbf{f}1\left(|T|e + \frac{nu}{1-(n+2)u}(|R|(|A|e)) + 2ue\right) \\ &\leq \frac{1}{1-(n+1)u} \cdot \mathbf{f}1\left(|T|e + \frac{nu}{1-(n+2)u}(|R|(|A|e)) + 2ue\right) \\ &\leq \mathbf{f}1\left(\frac{|T|e + (nu/(1-(n+2)u))(|R|(|A|e)) + 2ue}{1-(n+2)u}\right). \end{aligned}$$

Therefore, we finally have

$$(6.13) \quad \|RA - I\|_\infty \leq \mathbf{fl} \left( \frac{\| |T|e + (nu/(1 - (n + 2)u))(|R|(|A|e)) + 2ue\|_\infty}{1 - (n + 2)u} \right).$$

This upper bound is almost three time better than (4.2).

Let

$$\alpha' := \mathbf{fl} \left( \frac{\| |T|e + (nu/(1 - (n + 2)u))(|R|(|A|e))\|_\infty}{1 - (n + 1)u} \right).$$

From a similar discussion, we produce

$$\|RA - I\|_\infty \leq \alpha' + 2u.$$

Therefore,

$$1 - \|RA - I\|_\infty \geq 1 - 2u - \alpha'$$

is satisfied. Here, from (2.1) in [17] and  $\mathbf{fl}(1 - 2u) = 1 - 2u$  we derive








$$1 - \|RA - I\|_\infty \geq (1 - u)\mathbf{fl}((1 - 2u) - \alpha').$$

This is applicable to [17], Theorem 3.2.

The alternative method using the unit in the first place is proposed in [11].

**Acknowledgments.** The authors wish to thank the anonymous reviewers for comments on earlier version of this paper and suggestions for future work. We sincerely express our thank to Fujitsu Limited for developing the function of switches of rounding modes and giving us the fruitful information of BLAS functions. Thanks to Mr. Ryota Ochiai, Mr. Atsushi Sakamoto, and Mr. Ryota Kobayashi, former students in Shibaura Institute of Technology for the assistance of coding and numerical tests.

### References

- [1] *L. S. Blackford, J. Choi, A. Cleary, E. Dazevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley: ScaLAPACK - Scalable Linear Algebra PACKage.* Available at <http://www.netlib.org/scalapack/> (2019). 
- [2] *A. M. Castaldo, R. C. Whaley, A. T. Chronopoulos: Reducing floating point error in dot product using the superblock family of algorithms.* SIAM J. Sci. Comput. *31* (2008), 1156–1174.   
- [3] *FUJITSU: FUJITSU Supercomputer PRIMEHPC FX100.* Available at <https://www.fujitsu.com/global/products/computing/servers/supercomputer/primehpc-fx100/> (2020).
- [4] *N. J. Higham: Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, 2002.   



- [5] *N. J. Higham, T. Mary*: A new approach to probabilistic rounding error analysis. *SIAM J. Sci. Comput.* *41* (2019), A2815–A2835. [zbl](#) [MR](#) [doi](#)
- [6] *IEEE Computer Society*: IEEE Standard for Floating-Point Arithmetic: IEEE Std 754-2008. IEEE, NewYork, 2008. [doi](#)
- [7] *C.-P. Jeannerod, S. M. Rump*: Improved error bounds for inner products in floating-point arithmetic. *SIAM J. Matrix Anal. Appl.* *34* (2013), 338–344. [zbl](#) [MR](#) [doi](#)
- [8] *M. Kolberg, G. Bohlender, L. G. Fernandes*: An efficient approach to solve very large dense linear systems with verified computing on clusters. *Numer. Linear Algebra Appl.* *22* (2015), 299–316. [zbl](#) [MR](#) [doi](#)
- [9] *X. Li, J. Demmel, D. Bailey, Y. Hida, J. Iskandar, A. Kapur, M. Martin, B. Thompson, T. Tung, D. Yoo*: XBLAS - Extra Precise Basic Linear Algebra Subroutines. Available at <https://www.netlib.org/xblas/> (2008). [sw](#)
- [10] *A. Minamihata, K. Sekine, T. Ogita, S. M. Rump, S. Oishi*: Improved error bounds for linear systems with  $H$ -matrices. *Nonlinear Theory Appl., IEICE* *6* (2015), 377–382. [doi](#)
- [11] *Y. Morikura, K. Ozaki, S. Oishi*: Verification methods for linear systems using upf estimation with rounding-to-nearest. *Nonlinear Theory Appl., IEICE* *4* (2013), 12–22. [doi](#)
- [12] *M. Nakata*: The MPACK: Multiple Precision Arithmetic BLAS (MBLAS) and LAPACK (MLAPACK). Available at <http://mplapack.sourceforge.net/> (2011). [sw](#)
- [13] *A. Neumaier*: A simple derivation of the Hansen-Blikle-Rohn-Ning-Kearfott enclosure for linear interval equations. *Reliab. Comput.* *5* (1999), 131–136. [zbl](#) [MR](#) [doi](#)
- [14] *T. Ogita, S. Oishi*: Fast verification for large-scale systems of linear equations. *IPJS Trans.* *46* (2005), 10–18. (In Japanese.)
- [15] *T. Ogita, S. Oishi, Y. Ushiro*: Computation of sharp rigorous componentwise error bounds for the approximate solutions of systems of linear equations. *Reliab. Comput.* *9* (2003), 229–239. [zbl](#) [MR](#) [doi](#)
- [16] *T. Ogita, S. M. Rump, S. Oishi*: Accurate sum and dot product. *SIAM J. Sci. Comput.* *26* (2005), 1955–1988. [zbl](#) [MR](#) [doi](#)
- [17] *T. Ogita, S. M. Rump, S. Oishi*: Verified Solution of Linear Systems Without Directed Rounding: Technical Report No. 2005-04. Advanced Research Institute for Science and Engineering, Waseda University, Tokyo, 2005.
- [18] *S. Oishi, S. M. Rump*: Fast verification of solutions of matrix equations. *Numer. Math.* *90* (2002), 755–773. [zbl](#) [MR](#) [doi](#)
- [19] *K. Ozaki, T. Ogita*: Generation of linear systems with specified solutions for numerical experiments. *Reliab. Comput.* *25* (2017), 148–167. [MR](#)
- [20] *K. Ozaki, T. Ogita, S. Miyajima, S. Oishi, S. M. Rump*: A method of obtaining verified solutions for linear systems suited for Java. *J. Comput. Appl. Math.* *199* (2007), 337–344. [zbl](#) [MR](#) [doi](#)
- [21] *K. Ozaki, T. Ogita, S. Oishi*: An algorithm for automatically selecting a suitable verification method for linear systems. *Numer. Algorithms* *56* (2011), 363–382. [zbl](#) [MR](#) [doi](#)
- [22] *A. Petitet*: PBLAS - Parallel Basic Linear Algebra Subprograms. Available at [http://www.netlib.org/scalapack/pblas\\_qref.html](http://www.netlib.org/scalapack/pblas_qref.html). [sw](#)
- [23] *RIKEN Center for Computational Science*: What is K? Available at <https://www.r-ccs.riken.jp/en/k-computer/about/> (2019).
- [24] *S. M. Rump*: Kleine Fehlerschranken bei Matrixproblemen: PhD Thesis. Universität Karlsruhe, Karlsruhe, 1980. (In German.) [zbl](#) [doi](#)
- [25] *S. M. Rump*: Accurate solution of dense linear systems I: Algorithms in rounding to nearest. *J. Comput. Appl. Math.* *242* (2013), 157–184. [zbl](#) [MR](#) [doi](#)
- [26] *S. M. Rump*: Accurate solution of dense linear systems II: Algorithms using directed rounding. *J. Comput. Appl. Math.* *242* (2013), 185–212. [zbl](#) [MR](#) [doi](#)
- [27] *R. D. Skeel*: Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comput.* *35* (1980), 817–832. [zbl](#) [MR](#) [doi](#)

- [28] *V. Strassen*: Gaussian elimination is not optimal. *Numer. Math.* *13* (1969), 354–356. [zbl](#) [MR](#) [doi](#)
- [29] *N. Yamanaka, T. Ogita, S. M. Rump, S. Oishi*: A parallel algorithm for accurate dot product. *Parallel Comput.* *34* (2008), 392–410. [MR](#) [doi](#)

*Authors' addresses*: *Katsuhisa Ozaki* (corresponding author), *Takeshi Terao*, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-shi, Saitama 337-8570, Japan, e-mail: [ozaki@sic.shibaura-it.ac.jp](mailto:ozaki@sic.shibaura-it.ac.jp), [nb17105@shibaura-it.ac.jp](mailto:nb17105@shibaura-it.ac.jp); *Takeshi Ogita*, Tokyo Woman's Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan, e-mail: [ogita@lab.twcu.ac.jp](mailto:ogita@lab.twcu.ac.jp); *Takahiro Katagiri*, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, Aichi 464-8601, Japan, e-mail: [katagiri@cc.nagoya-u.ac.jp](mailto:katagiri@cc.nagoya-u.ac.jp).