

IEEE Copyright Notice

Copyright © 2020 IEEE

Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Accepted to be published in: 2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)

Cite as:

T. Iwashita, K. Suzuki and T. Fukaya, “An Integer Arithmetic-Based Sparse Linear Solver Using a GMRES Method and Iterative Refinement,” 2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), GA, USA, 2020, pp. 1-8, doi: 10.1109/ScalA51936.2020.00006.

BibTex:

```
@INPROCEEDINGS{9308712, author={T. {Iwashita} and K. {Suzuki} and T. {Fukaya}}, booktitle={2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)}, title={An Integer Arithmetic-Based Sparse Linear Solver Using a GMRES Method and Iterative Refinement}, year={2020}, volume={}, number={}, pages={1-8}, doi={10.1109/ScalA51936.2020.00006}}
```

An Integer Arithmetic-Based Sparse Linear Solver Using a GMRES Method and Iterative Refinement

Takeshi Iwashita
Information Initiative Center
Hokkaido University
Sapporo, Japan
Email: iwashita@iic.hokudai.ac.jp

Kengo Suzuki
Department of Electrical Engineering
Hokkaido University
Sapporo, Japan
Email: kiken50627@eis.hokudai.ac.jp

Takeshi Fukaya
Information Initiative Center
Hokkaido University
Sapporo, Japan
Email: fukaya@iic.hokudai.ac.jp

Abstract—In this paper, we develop a (preconditioned) GMRES solver based on integer arithmetic, and introduce an iterative refinement framework for the solver. We describe the data format for the coefficient matrix and vectors for the solver that is based on integer or fixed-point numbers. To avoid overflow in calculations, we introduce initial scaling and logical shifts (adjustments) of operands in arithmetic operations. We present the approach for operand shifts, considering the characteristics of the GMRES algorithm. Numerical tests demonstrate that the integer arithmetic-based solver with iterative refinement has comparable solver performance in terms of convergence to the standard solver based on floating-point arithmetic. Moreover, we show that preconditioning is important, not only for improving convergence but also reducing the risk of overflow.

Index Terms—Fixed point number, GMRES method, Integer arithmetic, Iterative linear solver, Iterative refinement

I. INTRODUCTION

In recent years, it has become difficult to improve the performance of processors, particularly their energy efficiency. The main reason is the decline in lithographic scaling, which threatens the well-known techno-economic model for the IT industry, that is, Moore’s Law [1], [2]. Thus, new computing technologies and devices based on different physics from CMOS technology are being widely investigated. Although quantum computing is a typical example for these technologies, some technologies aim to develop an ultra low-power but high-performance computer that is operated by instructions similar to conventional computers, for example computing devices based on single-flux-quantum (SFQ) circuits [3], [4]. However, these new types of computers may support only integer arithmetic in the early stage of research and deployment, because circuits for floating-point (FP) arithmetic are more complex and power consuming than those for integer arithmetic. Accordingly, we attempt to evaluate the potential of integer arithmetic computing for scientific computing. Specifically, we focus on iterative methods that are widely used in various scientific simulations, and investigate an integer arithmetic-based iterative linear solver, in which only integer arithmetic is used in the main iteration loop.

While there is a wide variety of iterative solvers, we develop a generalized minimal residual (GMRES) solver [5] using

integer (fixed-point number) arithmetic that is denoted by int-GMRES. The GMRES method is a Krylov subspace method and is used as a standard solver for a linear system that has an unsymmetric coefficient matrix. In our solver, the iterative refinement technique is used with the GMRES solver based on integer arithmetic to obtain a solution vector with the same accuracy as the output of a standard FP arithmetic solver. Although the technique is classical, it is useful for mixed-precision computing [6]. In this paper, we introduce the iterative refinement framework for an integer arithmetic-based solver and present the details of the implementation of the int-GMRES solver.

In Sections II and III, we introduce some notation and problem definitions, including the initial scaling of the linear system to be solved. In Section IV, we describe the iterative refinement framework for the solver based on integer arithmetic. In Sections V and VI, we present the details of the implementation of (preconditioned) int-GMRES. In Section VII, we present the numerical results. In Sections VIII and IX, we describe related works and summarize the paper.

II. NOTATION

In this paper, we discuss a linear solver in which integer arithmetic is mainly used. In the program, some variables and elements of arrays are declared as integer numbers, and they are treated as fixed point numbers in the analysis. We use Q_{d_m, d_f} notation for the fixed-point number. Q_{d_m, d_f} denotes a number with d_m integer and d_f fractional bits. The word length WL is $d_m + d_f + 1$, because a sign bit is used. The entire word is a two’s complement integer.

In the following, we denote the i -th row j -th element of matrix Z by z_{ij} or $Z(i, j)$. We denote the i -th element of vector z by z_i . When matrices, vectors, and variables have a *bar*, such as $\bar{A}^{(k)}$, this indicates that their elements or values are fixed-point or integer numbers and are stored using the intWL type in the program.

III. PROBLEM AND INITIAL SCALING

In this paper, we consider the following n -dimensional linear system of equations:

$$\hat{A}\hat{x} = \hat{b}. \quad (1)$$

The elements of $\hat{\mathbf{A}}$ and $\hat{\mathbf{b}}$ are given by FP numbers. Typically, they are double precision. We need to solve (1) with sufficient accuracy; that is, the relative residual norm calculated using (double-precision) FP arithmetic must be smaller than a given tolerance. The final value of each element of $\hat{\mathbf{x}}$ is given by an FP number.

First, the linear system (1) is scaled using FP arithmetic as follows:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \quad (2)$$

where $\mathbf{A} = \hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}$ and $\mathbf{b} = \hat{\mathbf{D}}^{-1}\hat{\mathbf{b}}$. When we intend to preserve a particular property of the coefficient matrix, such as symmetry, the scaled linear system can be written as follows:

$$\mathbf{A} = \hat{\mathbf{D}}_1^{-1}\hat{\mathbf{A}}\hat{\mathbf{D}}_2^{-1}, \quad \mathbf{x} = \hat{\mathbf{D}}_2\hat{\mathbf{x}}, \quad \mathbf{b} = \hat{\mathbf{D}}_1^{-1}\hat{\mathbf{b}}. \quad (3)$$

In (2) and (3), $\hat{\mathbf{D}}$, $\hat{\mathbf{D}}_1$, and $\hat{\mathbf{D}}_2$ are diagonal matrices. In the present analysis, the i -th diagonal element of $\hat{\mathbf{D}}$ is given by

$$\hat{d}_{ii} = \max_j |a_{ij}|/2^{\bar{\alpha}_a}. \quad (4)$$

When the linear system (2) is solved mainly using integer arithmetic, the setting of $\bar{\alpha}_a$ can be an important issue, and depends on the solver implementation. Based on our preliminary tests, we suggest that $\bar{\alpha}_a = WL/4$, whereas a larger value can be set for a preconditioned solver.

IV. ITERATIVE REFINEMENT

We use an iterative refinement technique, which is slightly adjusted for iterative linear solvers based on integer arithmetic. In the technique, we refine the approximate solution vector by solving the residual equation. We assume that we obtain sufficiently accurate solution vector by k_t times refinements. In each refinement, a linear system of equations is approximately solved. Then, the solution vector (or its sufficiently accurate approximation) \mathbf{x} is written as

$$\mathbf{x} = \tilde{\mathbf{x}}^{(1)} + \tilde{\mathbf{x}}^{(2)} + \dots + \tilde{\mathbf{x}}^{(k_t)}. \quad (5)$$

In our technique, the approximate solution vector $\tilde{\mathbf{x}}^{(k)}$ for the k -th refinement is obtained by (approximately) solving the linear system of equations:

$$\bar{\mathbf{A}}^{(k)}\mathbf{x}^{(k)} = \mathbf{b}^{(k)}. \quad (6)$$

In (6), each element of $\mathbf{x}^{(k)}$ and $\mathbf{b}^{(k)}$ is given by an FP number.

A. Setting of the Right-Hand Side and Solution Vector

Before the k -th refinement process, we calculate

$$\mathbf{b}'^{(k)} = \mathbf{b} - \mathbf{A}\left(\sum_{l=1}^{k-1} \tilde{\mathbf{x}}^{(l)}\right) \quad (7)$$

using FP arithmetic. We note that $\mathbf{b}'^{(1)} = \mathbf{b}$. Although $\tilde{\mathbf{x}}^{(k)}$ can be determined by solving $\bar{\mathbf{A}}^{(k)}\tilde{\mathbf{x}}^{(k)} = \mathbf{b}'^{(k)}$, we solve its scaled system (6) considering the use of an integer arithmetic-based solver and representation range of a fixed-point number. Using FP arithmetic, we calculate the scaled vector $\mathbf{b}^{(k)}$ of $\mathbf{b}'^{(k)}$ using

$$\mathbf{b}^{(k)} = \frac{1}{\gamma^{(k)}}\mathbf{b}'^{(k)}, \quad (8)$$

and

$$\gamma^{(k)} = \max_i |b_i^{(k)}|. \quad (9)$$

Then, the vector for the k -th refinement $\tilde{\mathbf{x}}^{(k)}$ is written by

$$\tilde{\mathbf{x}}^{(k)} = \gamma^{(k)}\mathbf{x}^{(k)}. \quad (10)$$

When the entire refinement process works, we can expect that the scaling factor $\gamma^{(k)}$ decreases as k increases.

B. Coefficient Matrix

In this subsection, we describe the setting of $\bar{\mathbf{A}}^{(k)}$. Each element of matrices used in the iterative linear solver is given by an integer number without fractional bits. After the initial scaling of the original linear system, we cast each element of \mathbf{A} to an integer number and obtain $\bar{\mathbf{A}}_0$. Next, we calculate $\bar{\mathbf{A}}_1$ using $\bar{\mathbf{A}}_1 = \mathbf{A} - \bar{\mathbf{A}}_0$ with FP arithmetic. Then, we determine a scaling factor $\bar{\alpha}_1$ as follows:

$$\bar{\alpha}_1 = \bar{\alpha}_a - \lfloor \log_2 \max_{ij} |A_1(i, j)| \rfloor. \quad (11)$$

After each element of $\bar{\mathbf{A}}_1$ is multiplied by $2^{\bar{\alpha}_1}$, it is cast to an intWL number to obtain $\bar{\bar{\mathbf{A}}}_1$. After the same scaling and casting processes are performed repeatedly, the coefficient matrix can be written as

$$\mathbf{A} = \bar{\mathbf{A}}_0 + \frac{1}{2^{\bar{\alpha}_1}}\bar{\bar{\mathbf{A}}}_1 + \frac{1}{2^{\bar{\alpha}_2}}\bar{\bar{\mathbf{A}}}_2 + \dots + \frac{1}{2^{\bar{\alpha}_p}}\bar{\bar{\mathbf{A}}}_p, \quad (12)$$

because each element of \mathbf{A} is an FP number with a finite word length. Each element of $\bar{\bar{\mathbf{A}}}_l (l = 0, \dots, p)$ is an integer number (no fractional bits). It holds that $\bar{\alpha}_1 < \bar{\alpha}_2 < \dots < \bar{\alpha}_p$. In the k -th refinement process, we use a limited number of terms on the right-hand side of (12); that is

$$\bar{\mathbf{A}}^{(k)} = \bar{\mathbf{A}}_0 + \sum_{l=1}^{s(k)} \frac{1}{2^{\bar{\alpha}_l}}\bar{\bar{\mathbf{A}}}_l, \quad (13)$$

where $s(k)$ is a parameter for the solver. When $s(k) = 0$, we only use $\bar{\mathbf{A}}_0$ in the refinement process; that is, $\bar{\mathbf{A}}^{(k)} = \bar{\mathbf{A}}_0$.

C. Refinement Process

Finally, we introduce an iterative refinement framework for iterative linear solvers that mainly use integer arithmetic, as shown in Fig. 1. In Fig. 1, $\tilde{\mathbf{x}}$ is the approximation of \mathbf{x} and S is the maximum value of $s(k)$. We assume that no FP arithmetic is used in the main loop of the iterative solver used in the framework. Table I lists the arguments of the iterative linear solver based on integer arithmetic. The input parameter d_f is the number of fractional bits for fixed-point numbers involved in the iterative solver. In the program, the input data of the coefficient matrix are represented by integer numbers. The input of $\mathbf{x}^{(k)}$ is an initial guess for the iterative solver. The output of $\mathbf{x}^{(k)}$ is the (approximate) solution vector of (6), each element of which is an FP number.

```

Initial scaling
Calculate  $\bar{\mathbf{A}}_0, \bar{\mathbf{A}}_1, \dots, \bar{\mathbf{A}}_S, \bar{\alpha}_1, \bar{\alpha}_2, \dots, \bar{\alpha}_S$ 
for  $k = 1, 2, \dots$ 
  if  $(\|\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}\|/\|\mathbf{b}\| < \epsilon)$  break
  Calculate  $\mathbf{b}'^{(k)}$ 
  Calculate  $\gamma^{(k)}$  and  $\mathbf{b}^{(k)}$ 

Integer_arithmetic_based_linear_solver( arguments )
// to solve  $\bar{\mathbf{A}}^{(k)} \mathbf{x}^{(k)} = \mathbf{b}^{(k)}$ 

 $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} + \gamma^{(k)} \mathbf{x}^{(k)}$ 
endfor

```

Fig. 1. Iterative refinement framework using the iterative linear solver based on integer arithmetic

V. GMRES SOLVER USING INTEGER ARITHMETIC (INT-GMRES)

A. Overview and Data Types of int-GMRES

In this section, we introduce the GMRES solver based on integer arithmetic that is used in the iterative refinement framework. We denote the solver by int-GMRES in this paper.

In our solver, each element of the coefficient matrix is given by an integer number (no fractional bits). The elements of vectors and variables used in the main GMRES iteration loop are given by fixed-point numbers in the $Qd_m.d_f$ format.

In the following sections, we use the term "bit shift." In this paper, left and right shifts with β bits refer to multiplication by 2^β and division by 2^β , respectively. These operations for signed integer numbers can be implemented using the *shift* operation when the used computer supports a *logical shift*. In this paper, we assume the use of this type of computer. However, in some computational environments, the result of a shift operation for a signed integer number is "undefined."

Figure 2 shows the algorithm for the int-GMRES solver of m iterations. In a practical application, iteration can be terminated when $|\bar{g}_{j+1}|$ is sufficiently small. In the figure, (FP) represents the statement or calculation based on FP arithmetic, whereas (INT) represents integer arithmetic. In the following subsections, we explain for the basic arithmetic of fixed-point numbers and kernels of GMRES, and then present the implementation details.

B. Basic Arithmetic of Fixed-Point Numbers

In this subsection, we describe the implementation of four basic arithmetics of fixed-point numbers in the $Qd_m.d_f$ format.

1) *Addition and Subtraction*: The addition and subtraction of two fixed point numbers of $Qd_m.d_f$ are straightforwardly implemented using the integer addition instruction. The obtained integer value directly represents the result in the $Qd_m.d_f$ format.

TABLE I
TYPES OF ARGUMENTS

Arrays, variables	I/O	Number type
$\mathbf{A}_0, \dots, \mathbf{A}_s$	Input	Integer
$\bar{\alpha}_1, \dots, \bar{\alpha}_s$	Input	Integer
d_f	Input	Integer
$\mathbf{b}^{(k)}$	Input	Floating point
$\mathbf{x}^{(k)}$	Input / Output	Floating point

2) *Multiplication*: The multiplication of fixed-point numbers is required in various parts of the GMRES program that include calculations of inner products and norms. Let us consider the multiplication of two fixed-point numbers in the $Qd_m.d_f$ format: \bar{t}_1 and \bar{t}_2 . We denote the integer representation of \bar{t}_1 and \bar{t}_2 in the program by t_1 and t_2 , respectively; that is, $t_1 = 2^{d_f} \cdot \bar{t}_1$ and $t_2 = 2^{d_f} \cdot \bar{t}_2$. The multiplication procedure for $\bar{t}_r = \bar{t}_1 \bar{t}_2$ is given as follows: After two integer numbers t_1 and t_2 are divided by 2^{β_1} and 2^{β_2} , respectively, they are multiplied using the integer instruction. The obtained value corresponds to \bar{t}_r in the $Qd'_m.d'_f$ format, where $d'_f = 2d_f - \beta_1 - \beta_2$ and $d'_m = WL - d'_f - 1$. When we need the result represented as a $Qd_m.d_f$ number, the value is divided by $2^{(d'_f - d_f)}$. Figure 3 demonstrates the multiplication procedure of fixed-point numbers. When the computer supports a *logical* shift operation, using a C language-like representation, the multiplication in the program, in which the result is represented in the $Qd_m.d_f$ format, is written as

$$t_r = ((t_1 \gg \beta_1) * (t_2 \gg \beta_2)) \gg (d_f - \beta_1 - \beta_2), \quad (14)$$

where t_r is the integer representation in the program for \bar{t}_r .

3) *Division*: The division of \bar{t}_1 by \bar{t}_2 is implemented as follows: After the first source operand t_1 is multiplied by 2^{β_1} and the second source operand t_2 is divided by 2^{β_2} , the first operand is divided by the second operand using the integer division instruction. The resultant variable is multiplied by $2^{(d_f - \beta_1 - \beta_2)}$ and the final result in the $Qd_m.d_f$ format is obtained.

4) *Square Root*: The calculation of a square root is required for the GMRES algorithm. In this subsection, we describe the calculation of the square root of a fixed-point number \bar{t}_s in the $Qd'_m.d'_f$ format, where $d'_m + d'_f = WL - 1$. Let t_s denote the integer representation of \bar{t}_s in the program. We apply the Babylonian square root algorithm for t_s using integer arithmetic. The obtained value $t_r = \sqrt{t_s}$ is multiplied by $2^{(d_f - d'_f/2)}$. The final result provides the integer representation of the square root of \bar{t}_s in the $Qd_m.d_f$ format.

C. Kernels of the GMRES Method

In this subsection, we describe the implementation of three computational kernels of the GMRES method.

1) *Inner product*: We consider the inner product of two vectors, each element of which is a fixed-point number in the $Qd_m.d_f$ format. Using the multiplication and addition

1. Compute $\mathbf{r}_0 = \mathbf{b}^{(k)} - \bar{\mathbf{A}}^{(k)} \mathbf{x}^{(k)}$,
 $\mathbf{v}_1 = \mathbf{r}_0 / \|\mathbf{r}_0\|$ // (FP)
2. Cast \mathbf{v}_1 to $\bar{\mathbf{v}}_1$
3. $\bar{\mathbf{g}} = (1, 0, \dots, 0)^\top$
4. For $j=1, 2, \dots, m$
5. Compute $\bar{\mathbf{w}}_{j+1} = \bar{\mathbf{A}}^{(k)} \bar{\mathbf{v}}_j$ // (INT)
6. For $i = 1, \dots, j$
7. $\bar{h}_{i,j} = (\bar{\mathbf{w}}_{j+1}, \bar{\mathbf{v}}_i)$ // (INT)
8. $\bar{\mathbf{w}}_{j+1} = \bar{\mathbf{w}}_{j+1} - \bar{h}_{i,j} \bar{\mathbf{v}}_i$ // (INT)
9. Endfor
10. $\bar{h}_{j+1,j} = \|\bar{\mathbf{w}}_{j+1}\|$ // (INT)
11. $\bar{\mathbf{v}}_{j+1} = \bar{\mathbf{w}}_{j+1} / \bar{h}_{j+1,j}$ // (INT)
12. For $i = 1, \dots, j-1$
13.
$$\begin{pmatrix} \bar{h}_{i,j} \\ \bar{h}_{i+1,j} \end{pmatrix} = \begin{pmatrix} \bar{c}_i & \bar{s}_i \\ -\bar{s}_i & \bar{c}_i \end{pmatrix} \begin{pmatrix} \bar{h}_{i,j} \\ \bar{h}_{i+1,j} \end{pmatrix}$$
 // (INT)
14. Endfor
15. $\bar{t}_{mp} = \sqrt{\bar{h}_{j,j}^2 + \bar{h}_{j+1,j}^2}$ // (INT)
16. $\bar{c}_j = \frac{\bar{h}_{j,j}}{\bar{t}_{mp}}, \bar{s}_j = \frac{\bar{h}_{j+1,j}}{\bar{t}_{mp}}$ // (INT)
17. $\bar{g}_j = \bar{c}_j * \bar{g}_j, \bar{g}_{j+1} = -\bar{s}_j * \bar{g}_j$ // (INT)
18. $\bar{h}_{j,j} = \bar{t}_{mp}$
19. $\bar{h}_{j+1,j} = 0$ // (INT)
20. Endfor
21. Cast $\bar{\mathbf{g}}$ to \mathbf{g} , and $\bar{\mathbf{v}}_i$ to \mathbf{v}_i
22. $\mathbf{y} = \|\mathbf{r}_0\| \mathbf{H}_m^{-1} \mathbf{g}$ // (FP)
23. $\mathbf{x}^{(k)} = \mathbf{x}^{(k)} + \sum_{i=1}^j y_i \mathbf{v}_i$ // (FP)

Fig. 2. Algorithm for the int-GMRES method (m iterations)

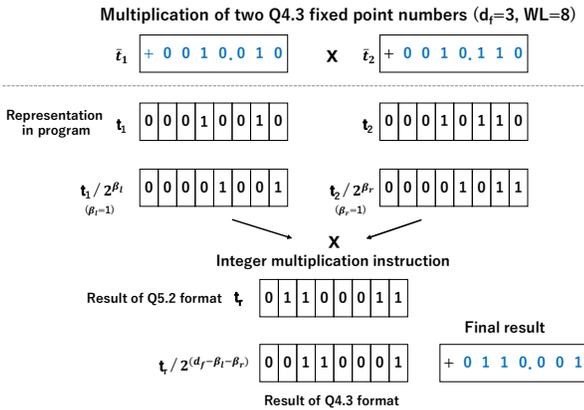


Fig. 3. Multiplication of fixed-point numbers

operations for fixed-point numbers described in Section V-B, we obtain the result of the inner product as a number in the $Qd'_m.d'_f$ format. To obtain a better accuracy in calculations, we typically set d'_f to be larger than d_f . Therefore, to obtain a result in the $Qd_m.d_f$ format, the result variable is divided by $2^{d_f - d'_f}$. Figure 4 shows a sample code for the inner product. In the figure, $b1$ and $b2$ correspond to β_1 and β_2 in the procedure for the multiplication, respectively.

```
cs=0;
for (l=0; l<n; l++) {
    cs=cs+(v[l] >> b1)*(w[l] >> b2) ; }
```

Fig. 4. Calculation of the inner product with the setting $d'_f = 2d_f - \beta_1 - \beta_2$ (only on computers that support a logical shift for a signed integer number)

2) *Norm*: When we calculate a vector norm, we first calculate the inner product of the vector and itself. Using the procedure described above, we obtain the result of the inner product in the $Qd'_m.d'_f$ format. Then, we calculate its square root using the procedure described in Section V-B4. Finally, we obtain the norm of the vector which is represented in the $Qd_m.d_f$ format.

3) *Matrix Vector Multiplication*: Matrix vector multiplication is a main kernel of Krylov subspace methods, in which the GMRES method is classified. From (13), the kernel consists of $s+1$ matrix vector multiplications:

$$\bar{\mathbf{A}}^{(k)} \bar{\mathbf{v}} = \bar{\mathbf{A}}_0 \bar{\mathbf{v}} + \sum_{l=1}^{s(k)} \frac{1}{2^{\bar{\alpha}_l}} \bar{\mathbf{A}}_l \bar{\mathbf{v}}, \quad (15)$$

where $\bar{\mathbf{v}}$ is an n -dimensional source vector. In our implementation, each element of the matrices is given by an integer number, which has no fraction bits. The element of the source and resultant vectors is a fixed-point number in the $Qd_m.d_f$ format. Consequently, each matrix vector multiplication $\bar{\mathbf{A}}_l \bar{\mathbf{v}}$ can be performed by a simple integer matrix vector multiplication program. Each element of $\bar{\mathbf{A}}_l \bar{\mathbf{v}}$ is divided by $2^{\bar{\alpha}_l}$, and then added to the corresponding element of the resultant vector.

In the above procedure, it is implied that the result of $\bar{\mathbf{A}}_l \bar{\mathbf{v}}$ does not contribute to the final result when $\bar{\alpha}_l$ is substantially large. Consequently, we estimate that s must be at most 3 or 4 in a practical scenario. When we require more accuracy for the matrix vector multiplication, we should use multiple words for each element of the resultant vector.

D. Implementation Details of int-GMRES and Setting of the Operand Shifts

In this subsection, we present the details of the int-GMRES solver while paying special attention to setting the parameters in fixed-point number arithmetic.

1) *Cast of \mathbf{v}_1 to $\bar{\mathbf{v}}_1$* (l. 2 in Fig. 2): Each element of \mathbf{v}_1 is multiplied by 2^{d_f} using FP arithmetic. Then, it is cast to an intWL number. The obtained integer array that corresponds to $\bar{\mathbf{v}}_1$ consists of fixed-point numbers in the $Qd_m.d_f$ format.

2) *Arnoldi Process* (l. 4-11 in Fig. 2) :

a) *Line 5 (matrix vector multiplication)*: Line 5 is matrix vector multiplication, which we implement using the method described in Section V-C3.

b) *Line 7 (inner product)*: Line 7 in Fig. 2 is the calculation of an inner product, which we implement using the method described in Section V-C1. We suggest a special setting for the operand shift in the multiplication involved in the calculation. Because $\bar{\mathbf{v}}_i$ is a normalized vector, the upper $WL - d_f - 2$ bits of each element of $\bar{\mathbf{v}}_i$ are always zero.

Considering this feature, we only shift the first source operand which corresponds to \bar{w}_{j+1} ; that is $\beta_2 = 0$.

c) *Line 8*: Line 8 involves the multiplication of a vector element by a scalar value and subtraction between two vectors. Like the inner product in line 7, we only shift the first source operand in the multiplication, considering the profile of \bar{v}_i .

d) *Line 10 (norm)*: Line 10 is the calculation of the norm, which we implement using the procedure described in Section V-C2. In the multiplication, we naturally set $\beta_1 = \beta_2$.

e) *Line 11*: Line 11 is the division of a vector by a scalar number. We use the procedure for division described in Section V-B3.

3) *Givens Rotation (l. 12-20 in Fig. 2)* :

a) *Line 13*: We regard the statement as the inner product of the vectors of two elements. Therefore, we use the procedure for the inner product. For the multiplication involved in the procedure, we also use a special setting for the operand shift. Because the absolute value of \bar{c}_i and \bar{s}_i is not larger than one, we only shift the second operand corresponding to $\bar{h}_{i,j}$ or $\bar{h}_{i+1,j}$, that is, $\beta_1 = 0$.

b) *Line 15*: We can implement the statement as the calculation of the norm of the vector of two elements.

c) *Line 16*: We use the procedure for the division of a fixed-point number by another fixed-point number.

d) *Line 17*: Line 17 consists of the multiplication of scalar values. The absolute values of \bar{c}_i and \bar{s}_i are bounded by one, and g_j monotonically decreases as the iteration count j increases. Thus, we do not shift the operands in the multiplication because of the low risk of overflow.

4) *Update of the Solution Vector (l. 21-23 in Fig. 2)* :

a) *Line 21*: We cast each element of the integer arrays for \bar{g} and \bar{v}_i to an FP number, which we then divide by 2^{d_f} . In the practical implementation, we combine these casting operations with the following computations (l. 22-23) to avoid an additional array allocation.

b) *Lines 22 and 23*: We update the output of the int-GMRES solver, that is, $\mathbf{x}^{(k)}$ using FP arithmetic.

5) *Summary of Setting the Parameters*: Table II summarizes the type of fixed-point numbers, that is, the number of fractional bits, and the quantity of *shift* in arithmetic involved in the int-GMRES solver. In the table, the line number corresponds to the line of the statement in Fig. 2, and #fb represents the number of fractional bits of the fixed-point number used for vectors and variables.

VI. PRECONDITIONING

Preconditioning is a practically important technique to accelerate the convergence of an iterative solver. To apply a preconditioning technique to the GMRES solver, we replace two statements (lines 1 and 5) in Fig. 2 by the following statements:

Line 1'. Compute $\mathbf{r}_0 = \bar{\mathbf{M}}^{-1}(\mathbf{b}^{(k)} - \bar{\mathbf{A}}^{(k)}\mathbf{x}^{(k)})$,
 $\mathbf{v}_1 = \mathbf{r}_0 / \|\mathbf{r}_0\|$ // (FP)

Line 5'. Compute $\bar{w}_{j+1} = \bar{\mathbf{M}}^{-1}\bar{\mathbf{A}}^{(k)}\bar{v}_j$ // (INT)

Typically, the preconditioner matrix $\bar{\mathbf{M}}$ well approximates the coefficient matrix. In this paper, we report the application

of a standard incomplete LU (ILU), which is precisely ILU(0) preconditioning. In our solver, the element of the preconditioner matrix is given by an integer number (no fractional bits), which is the same as the coefficient matrix.

A. ILU preconditioning

In ILU preconditioning, we use the incomplete factorized matrix of the coefficient matrix. Using FP arithmetic, we incompletely factorize the coefficient matrix as

$$\mathbf{A} \simeq \mathbf{LDU}, \quad (16)$$

where \mathbf{L} and \mathbf{U} have ones for their diagonal elements. Next, we define two diagonal matrices as follows:

$$\mathbf{D}_l \mathbf{D}_r = \mathbf{D}, \quad (17)$$

$$D_l(i, i) = |d_{ii}|^{1/2}, \quad (18)$$

and

$$D_u(i, i) = \text{sgn}(d_{ii})|d_{ii}|^{1/2}. \quad (19)$$

Then, we introduce two matrices:

$$\tilde{\mathbf{L}} = \mathbf{LD}_l \quad (20)$$

and

$$\tilde{\mathbf{U}} = \mathbf{D}_u \mathbf{U}. \quad (21)$$

We apply the type cast from float/double to int for each element of $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$, and then we obtain lower and upper triangular matrices, $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$, respectively. Then, the preconditioner matrix $\bar{\mathbf{M}}_{ilu}$ is given by

$$\bar{\mathbf{M}}_{ilu} = \bar{\mathbf{L}}\bar{\mathbf{U}}. \quad (22)$$

The ILU preconditioning step corresponding to lines 1' and 5' is given by forward and backward substitutions. We can simply use a program for the substitutions in which integer arithmetic is used; that is, if we have a program for substitution based on FP arithmetic, we only change the data type for the matrix and vectors (float/double to int) in the program.

VII. NUMERICAL RESULT

A. Computation Environment and Test Problems

We conducted numerical tests to evaluate the developed int-GMRES solver. We evaluated the convergence of the relative residual norm of the solver in comparison with a standard GMRES solver using FP arithmetic. We performed numerical tests on a node of Fujitsu CX2550 (M4) at the Information Initiative Center, Hokkaido University. The node was equipped with two Intel 20-core Xeon (Gold 6148) processors and 384 GB shared memory. We wrote the program code in C and used an Intel compiler for the analysis. Logical shift was supported on the computer for a signed integer number.

In the integer arithmetic-based solver, the linear system (6) for the refinement was approximately solved using m iterations of int-GMRES. For comparison, we also used a standard double precision GMRES(m) solver. We set the convergence criterion as the relative residual norm being less than 10^{-8} . The relative residual norm was calculated every m iterations

TABLE II
NUMBER OF FRACTIONAL BITS OF THE INPUT AND OUTPUT VARIABLES AND OPERAND SHIFTS IN THE CALCULATION

Line #	Kernel	Arithmetic	Input		Output		
			1st (or single) operand	2nd operand	Result		
			#fb	Shift	#fb	Shift	#fb
Line 5	Matrix vector	Multiplication	0	No	d_f	No	d_f
		Multiplication	d_f	No	d_f	No	d_f
Line 7	Inner Product	Multiplication	d_f	$/2^{\beta_1}$	d_f	No	$2d_f - \beta_1$
		Addition	$2d_f - \beta_1$	No	$2d_f - \beta_1$	No	$2d_f - \beta_1$
		Shift	$2d_f - \beta_1$	$/2^{d_f - \beta_1}$	-	-	d_f
Line 8	Vector update	Multiplication	d_f	$/2^{\beta_1}$	d_f	No	d_f
		Subtraction	d_f	No	d_f	No	d_f
Line 10	Norm	Multiplication	d_f	$/2^{\beta_1}$	d_f	$/2^{\beta_1}$	$2d_f - 2\beta_1$
		Addition	$2d_f - 2\beta_1$	No	$2d_f - 2\beta_1$	No	$2d_f - 2\beta_1$
		Square root	$2d_f - 2\beta_1$	No	-	-	d_f
Line 11		Division	d_f	$\times 2^{\beta_1}$	d_f	$/2^{\beta_2}$	d_f
		Multiplication	d_f	No	d_f	$/2^{\beta_2}$	$2d_f - \beta_2$
Line 13	Inner Product	Addition	$2d_f - \beta_2$	No	$2d_f - \beta_2$	No	$2d_f - \beta_2$
		Shift	$2d_f - \beta_2$	$/2^{d_f - \beta_2}$	-	-	d_f
		Multiplication	d_f	$/2^{\beta_1}$	d_f	$/2^{\beta_1}$	$2d_f - 2\beta_1$
Line 15		Addition	$2d_f - 2\beta_1$	No	$2d_f - 2\beta_1$	No	$2d_f - 2\beta_1$
		Square root	$2d_f - 2\beta_1$	No	-	-	d_f
		Division	d_f	$\times 2^{\beta_1}$	d_f	$/2^{\beta_2}$	d_f
Line 17		Multiplication	d_f	No	d_f	No	d_f

TABLE III
MATRIX INFORMATION FOR THE TEST PROBLEMS

Data set	Problem type	Dimension	# nonzero
atmosmodj	CFD	1,270,432	8,814,880
atmosmodl	CFD	1,489,752	10,319,760
cage14	Graph	1,505,785	27,130,349
CoupCons3D	Structural problem	416,800	17,277,420
epb2	Thermal problem	25,228	175,027
majorbasis	Optimization problem	160,000	1,750,416
memchip	Circuit simulation	2,707,524	13,343,948
stomach	Electro-physical model	213,360	3,021,648
torso3	Finite difference model	259,156	4,429,042
wang3	Semiconductor analysis	26,064	177,168

using FP arithmetic in both the standard and integer arithmetic-based solvers. The comparison of the convergence properties of the solvers is performed every m iterations.

For the test problems, we selected ten linear systems from the SuiteSparse Matrix Collection [7]. We selected unsymmetric matrices with various sizes from the collection, for which the standard GMRES solver based on double-precision FP arithmetic worked. Table III lists the properties of selected matrices. The right-hand side vector was given by a vector of ones.

B. Results for the Non-Preconditioned GMRES Solver

The int-GMRES solver based on integer arithmetic requires parameters to be set. The number of fractional bits d_f was given by 30. The word length WL was 64, and the 64bit integer (int64) type was used for both fixed-point and integer numbers used in the solver. The parameter for the coefficient matrix s was given by zero; that is, we only used \bar{A}_0 in the test. These settings were also used in the numerical test of the preconditioned GMRES solver. Table IV lists the setting (number of bits) for operand shifts involved in the calculation of int-GMRES. Moreover, α_a was set to 16 for the non-

preconditioned solver and 32 for the ILU preconditioned solver.

Table V shows the number of iterations of a standard GMRES solver using double-precision FP arithmetic and the int-GMRES solver, where "Double" denotes the standard solver. In the numerical tests, int-GMRES, in addition to the standard solver could solve the problem. When $m = 10$, int-GMRES unexpectedly converged faster in three test cases. In the *wang3* test, which was the worst case for int-GMRES, the solver only required 20% more iterations than the standard solver.

When $m = 30$, the convergence rates of the standard and integer arithmetic based solvers were comparable for the test cases, except for *cage14* and *wang3*. However, int-GMRES only required more iterations for an additional refinement step than the standard solver to converge in *cage14*. In the *wang3* test, which was regarded as the worst case for int-GMRES, the solver only required 24% more iterations than the standard solver. Figure 5 shows the comparison of the convergence rates of the standard double-precision and integer arithmetic-based GMRES solvers. For the *atmosmodj* dataset, the two solvers had an identical convergence rate, which means that the loss of accuracy in int-GMRES did not have a significant influence on solver performance. In contrast to the result of the *atmosmodj* test, the int-GMRES solver had a lower convergence rate than the standard solver in the *wang3* test.

C. Results for the Preconditioned GMRES Solver

When preconditioning is applied to an iterative solver, the convergence rate generally improves. Thus, we can expect that the residual norm is relatively small and the risk of overflow in the calculation is reduced. When ILU preconditioning is used, we can avoid the operand shift that sacrifices the accuracy of arithmetics in the solver. We note that the first source operand shift in the division operation is necessary for improving the

TABLE IV

NUMBER OF FRACTIONAL BITS OF THE INPUT AND OUTPUT VARIABLES, AND OPERAND SHIFTS FOR THE GMRES SOLVER (NO PRECONDITIONING)

Line #	Arithmetic	Setting for operand shift
Line 7	Multiplication	$\beta_1 = 16$
Line 8	Multiplication	$\beta_1 = 16$
Line 10	Multiplication	$\beta_1 = 16$
Line 11	Division	$\beta_1 = 16, \beta_2 = 14$
Line 13	Multiplication	$\beta_2 = 16$
Line 15	Multiplication	$\beta_1 = 16$
Line 16	Division	$\beta_1 = 16, \beta_2 = 14$

TABLE V

NUMBER OF ITERATIONS IN NO PRECONDITIONING CASE

Data set	$m=10$		$m=30$	
	Double	int-GMRES	Double	int-GMRES
atmosmodj	5,820	5,850	2,100	2,100
atmosmodl	880	840	420	420
cage14	20	20	30	60
CoupCons3D	430	430	360	360
epb2	820	730	540	540
majorbasis	90	100	90	90
memchip	460	380	300	300
stomach	310	310	180	180
torso3	150	150	150	150
wang3	720	860	510	630

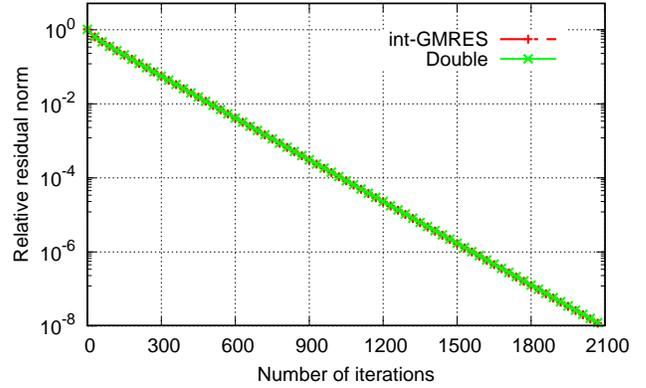
calculation accuracy. Table VI lists the settings of the operand shifts in the preconditioned solver based on integer arithmetic, which is denoted by int-ILU-GMRES.

Table VII shows the number of iterations of the ILU-GMRES solver using double-precision FP arithmetic and the int-ILU-GMRES solver. When compared with the non-preconditioned solver, both solvers attained significant improvement in convergence. Moreover, the convergence rates of the two solvers were comparable. The int-ILU-GMRES only required more iterations for an additional refinement step than the standard solver in some test cases. Figure 6 shows that both solvers had identical convergence behavior of the relative residual norm in the *wang3* test.

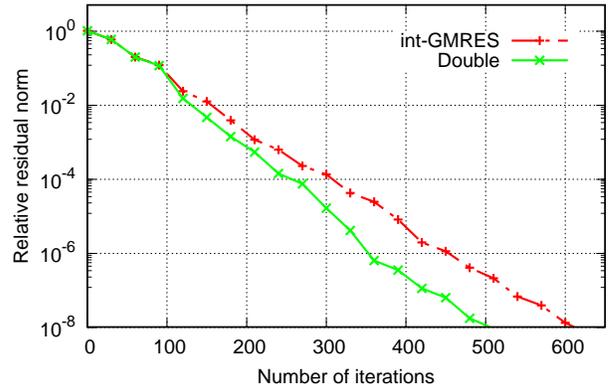
D. Discussions

1) *Preconditioning*: Preconditioning is important in the context of iterative solvers based on integer arithmetic, because it reduces the risk of overflow. Consequently, we can decrease the number of bits of the operand shift, which improves the accuracy of arithmetic. For the non-preconditioned solver, we investigated an auto-tuning technique for the shift. However, it proved to be hardly necessary in the preconditioning case. In the implementation of int-ILU-GMRES, we could avoid using the operand shift which sacrificed the accuracy. A similar effect was also confirmed in the Gauss–Seidel preconditioning case. (Because of the page limit, the numerical result is not shown in this paper.)

2) *Condition of the Problems*: Because we selected test problems (matrices) for which a non-preconditioned GMRES solver using FP arithmetic attained convergence, the problems were not heavily ill-conditioned. Consequently, the int-GMRES solver also solved the problems. It is possible that



(a) atmosmodj test



(b) wang3 test

Fig. 5. Comparison of the convergence behaviors of standard and int-GMRES solvers without preconditioning when m is 30

TABLE VI

NUMBER OF FRACTIONAL BITS OF THE INPUT AND OUTPUT VARIABLES AND OPERAND SHIFTS FOR THE ILU-GMRES SOLVER

Line #	Arithmetic	Setting for operand shift
Line 7	Multiplication	$\beta_1 = 0$
Line 8	Multiplication	$\beta_1 = 0$
Line 10	Multiplication	$\beta_1 = 0$
Line 11	Division	$\beta_1 = 30, \beta_r = 0$
Line 13	Multiplication	$\beta_2 = 0$
Line 15	Multiplication	$\beta_1 = 0$
Line 16	Division	$\beta_1 = 30, \beta_2 = 0$

problems exist that the standard FP arithmetic solver can solve but int-GMRES cannot. However, as far as we have tested, it seems not to be an easy task to seek such a problem; that is, the int-GMRES solver used with the iterative refinement technique may have comparable solver performance to the standard FP arithmetic solver.

VIII. RELATED WORKS

In this section, we introduce several papers that discuss mixed-precision linear solvers using the iterative refinement technique. The survey paper [6] by D. G6ddecke et al. provides a good introduction to the mixed-precision iterative

TABLE VII
NUMBER OF ITERATIONS IN THE ILU PRECONDITIONING CASE

Data set	$m=10$		$m=30$	
	Double	int-GMRES	Double	int-GMRES
atmosmodj	610	610	300	300
atmosmodl	140	140	120	120
cage14	10	20	30	60
CoupCons3D	140	150	30	60
epb2	50	50	60	60
majorbasis	20	20	30	60
stomach	20	20	30	60
torso3	40	40	30	60
wang3	180	180	120	120

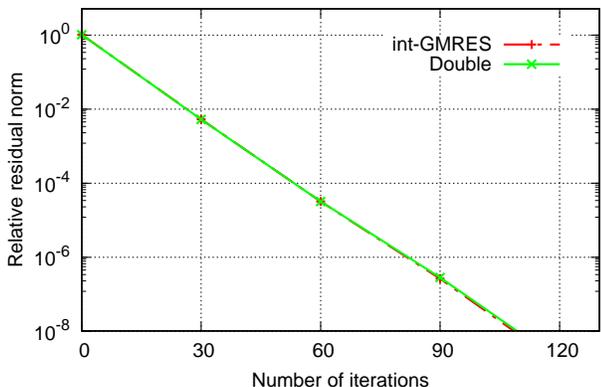


Fig. 6. Comparison of the convergence behaviors of standard and int-GMRES solvers with ILU preconditioning in the wang3 test when m is 30

refinement algorithm framework. The paper [8] by Anzt et al. is another early work on a mixed-precision linear solver, in which the authors reported a GPU implementation of an error correction solver using the GMRES method and showed the effectiveness of their approach in CFD applications. A. Haidar et al. reported the development of an architecture-specific algorithm and highly tuned implementations for the latest GPUs of mixed-precision iterative refinement solvers in [9]. Their solver that involved LU factorization was targeted at a linear system with a dense coefficient matrix. Carson et al. presented a general algorithm for iterative refinement with three precisions and its error analysis in [10]. Moreover, the Exascale Computing Project Multiprecision Effort Team (Lead: Hartwig Anzt) recently opened its technical report to the public, which provides a comprehensive review of mixed-precision computing [11].

Next, we briefly mention analyses based on integer arithmetic (fixed-point numbers). Currently, integer arithmetic is often used in machine learning and artificial intelligence applications. LU factorization based on integer arithmetic for these applications is given in [11]. Numerical linear algebra algorithms based on fixed-point numbers have also been investigated in the context of signal processing [12], [13]. The difference between the present research and these papers is in the investigation and development of the GMRES method based on integer arithmetic.

IX. CONCLUSIONS

In this paper, we developed a GMRES solver based on integer arithmetic, denoted by int-GMRES. The int-GMRES solver was used with an iterative refinement technique to attain a solution as accurate as that of a normal linear solver based on FP arithmetic. We also developed an ILU preconditioned int-GMRES solver. In integer arithmetic (fixed-point number) computing, it is important to avoid overflow in calculations. We explained how the operands are adjusted (logically shifted) in the calculation considering the characteristics of the GMRES method. We conducted numerical tests using matrices from SuiteSparse Matrix Collections. The numerical results demonstrated that the int-GMRES solver had comparable solver performance in terms of convergence to the standard FP solver. Moreover, we found that preconditioning was important for the solver using integer arithmetic to avoid overflow.

In the future, we will evaluate solver performance in terms of timing on the model of new computing devices in which integer arithmetic has advantages over conventional computing devices for calculation speed or power consumption.

REFERENCES

- [1] J. S. Vetter, E. P. DeBenedictis and T. M. Conte, "Architectures for the Post-Moore Era," *IEEE Micro*, vol. 37, pp. 6–8, 2017.
- [2] J. Shalf, "The future of computing beyond Moore's law," *Phil. Trans. R. Soc. A.*, 378 (2166), 20190061, 2020.
- [3] R. Sato, Y. Hatanaka, Y. Ando, M. Tanaka, A. Fujimaki, K. Takagi, N. Takagi, "High-speed operation of random-access-memory-embedded microprocessor with minimal instruction set architecture based on rapid single-flux-quantum logic," *IEEE Trans. Appl. Supercond.*, vol. 27, pp. 1–5, 2017.
- [4] K. Ishida, M. Tanaka, T. Ono, K. Inoue, "Towards ultra-high-speed cryogenic single-flux-quantum computing," *IEICE Trans. Electron.*, vol. E101-C, pp. 359–369, 2018.
- [5] Y. Saad, *Iterative Methods for Sparse Linear Systems*, (Second ed.). Philadelphia, PA, SIAM, 2003.
- [6] D. Göddeke, R. Strzodka, and S. Turek, "Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations," *Int. J. Parallel Emergent Distrib. Syst.*, vol. 22, pp. 221–256, 2007.
- [7] T. A. Davis, and Y. Hu, "The university of Florida sparse matrix collection," *ACM Trans. Math. Software*, 38, pp. 1–25, 2011.
- [8] H. Anzt, V. Heuveline, and B. Rucker, "An error correction solver for linear systems: evaluation of mixed precision implementations," *VECPAR 2010, LNCS*, vol. 6449, Springer, 2011.
- [9] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, "Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers," *Proc. SC18, Intl. Conf. High Performance Comput., Networking, Storage and Analysis*, pp. 603–613, 2018.
- [10] E. Carson, and N. J. Higham, "Accelerating the solution of linear systems by iterative refinement in three precisions," *SIAM J. Sci. Comput.*, vol. 40, pp. A817–A847, 2018.
- [11] A. Abdelfattah, H. Anzt, E.G. Boman, E. Carson, T. Cojean, J. Dongarra, M. Gates, T. Grützmacher, N.J. Higham, S. Li, N. Lindquist, Y. Liu, J. Loe, P. Luszczek, P. Nayak, S. Pranesh, S. Rajamanickam, T. Ribizel, B. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y.M. Tsai, I. Yamazaki, U.M. Yang, "A Survey of Numerical Methods Utilizing Mixed Precision Arithmetic," *arXiv preprint arXiv:2007.06674*, 2020.
- [12] Z. Nikolić, H. T. Nguyen, and G. Frantz, "Design and implementation of numerical linear algebra algorithms on fixed point DSPs," *EURASIP Journal on Advances in Signal Processing*, 087046, 2007.
- [13] T. Pradhan, B. Kabi, and A. Routray, "Fixed-point Hestenes algorithm for singular value decomposition of symmetric matrices," *Proc. 2013 Intl. Conf. Electronics, Signal Processing and Communication Systems*, 2013.