AIR: Iterative Refinement Acceleration using Arbitrary Dynamic Precision: Iterative refinement acceleration using arbitrary dynamic precision

AIR: Iterative Refinement Acceleration using Arbitrary Dynamic Precision☆

JunKyu Lee[a,*], Gregory D. Peterson[b], Dimitrios S. Nikolopoulos[a], Hans Vandierendonck[a]

[a]*The Institute of Electronics, Communications and Information Technology, Queen's University Belfast, Belfast BT7 1NN, U.K.*
[b]*The Electrical Engineering and Computer Science, The University of Tennessee, Knoxville, TN 37996 USA*

Abstract

The increased degree of concurrent operations by lower precision arithmetic enables high performance for iterative refinement. Most of related work present statically defined mixed precision arithmetic approaches, while adapting a level of arithmetic precision dynamically in a loop with one-bit granularity can further improve the performance. This paper presents Arbitrary Dynamic Precision Iterative Refinement algorithm (AIR) that minimizes the total significand bit-width to solve iterative refinement. AIR detects the number of cancellation bits dynamically per iteration and uses the information to provide the least sufficient significand bit-width for the next iteration. We prove that AIR is a backward stable algorithm and can bring up to $2-3\times$ speedups over a mixed precision iterative refinement depending on the characteristics of hardware. Our software demonstration shows that AIR requires only 83% of the significand bits required by mixed precision iterative refinement that solve linear systems for double precision accuracy for backward error with 32×32 standard normally distributed matrices.

Keywords: Acceleration, High performance, Adaptive system, Iterative refinement, Arbitrary precision, Dynamic precision

1. Introduction

Computational precision impacts performance and energy efficiency. For instruction-driven architectures such as CPUs and GPUs, lower precision arithmetic brings speedup by improving data transfer rate under a fixed memory bandwidth and by increasing word-level parallelism for SIMD architectures. In such computing platforms, linear speedup can be expected by employing lower arithmetic precision if single precision arithmetic is twice faster than double precision arithmetic. In contrast, for data-driven architectures such as FPGAs and ASICs, smaller ALUs employing a lower precision arithmetic increase the parallelism for the computation with the additional ALUs that can be fit in an area. For example, the number of multipliers can be quadradically increased with decreasing significand bit-width [1]. Therefore, quadratic speedup is expected by decreasing significand bit-width in such computing platforms, if multiplications are main computing kernels for an application. The speedup factor is related to the increased degree of concurrent operations by decreasing arithmetic precision and depends on computer architecture.

Many applications such as deep learning and scientific computation provide the opportunities to utilize variable precision arithmetic in order to gain speedup and energy reduction [2, 3, 4]. The statically defined mixed precision approaches were intensively investigated for iterative refinement [2, 3, 5, 6], assuming that the $O(n^2)$ refinement procedure would be trivial in terms of run time, compared to the $O(n^3)$ matrix decomposition, where n is a matrix size. However, if a matrix size is small or accuracy requirement is high, the $O(n^2)$ refinement is not trivial in terms of run time, resulting in the need to minimize not only the time cost for $O(n^3)$ matrix decomposition but also the $O(n^2)$ refinement procedure. Kiełbasiński proposed an iterative refinement exploiting one-bit granularity dynamically for the $O(n^2)$ refinement procedure in 1981 [7]. However, the condition number of a matrix should be known prior to computation, limiting application of the algorithm. (The condition number of a matrix defines the perturbation magnitude in the solution according to the perturbation in the inputs and it generally requires $O(n^3)$ computation to compute a condition number.) In [8], the iterative refinement of [7] was amended, but the amended algorithm still required the information of the number of iterations taken prior to the computation, still limiting its application. The question arises how to exploit arbitrary precision arithmetic whose arithmetic granularity is one bit to maximize speedup and energy savings for practical applications.

This paper presents the first Arbitrary Dynamic Precision Iterative Refinement algorithm (AIR) that solves a linear system with the lowest significand cost compared to previously proposed iterative refinements by exploiting dynamic information of the number of cancellation bits per iteration in the residual. It is the adaptive precision capability of AIR to minimize the total significand bits further compared to previously proposed iterative refinements. As consequences, speedups and energy savings follow. The main contributions of this paper are two folds:

☆This project has received funding by the European Commission Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 798209 (Entrans) and the grant agreement No. 732631 (OPRECOMP).

*Corresponding author

Email addresses: junkyu.lee@qub.ac.uk (JunKyu Lee), gdp@utk.edu (Gregory D. Peterson), d.nikolopoulos@qub.ac.uk (Dimitrios S. Nikolopoulos), h.vandierendonck@qub.ac.uk (Hans Vandierendonck)

- To the best of our knowledge, AIR is the first practical algorithm that solves linear systems with the lowest significand cost by using arbitrary dynamic precision arithmetic, compared to previously proposed iterative refinements. We demonstrate that AIR solves linear systems of $n = 32$ with 0.83 significand cost compared to a mixed precision iterative refinement of [6] and with 0.81 significand cost compared to a previously proposed arbitrary precision iterative refinement of [8] for double precision solution accuracy for backward error.

- This paper includes theoretical framework for AIR including the description of the algorithm, its achievable accuracy, its required number of iterations, its achievable speedups, and application conditions required to maximize its speedups.

We discuss conventional iterative refinement and related work in Section 2; the description of AIR and its numerical properties in Section 3; the performance analysis of AIR in Section 4 ; experimental evaluation in Section 5; discussion in Section 6; and finally, concludes the paper in Section 7. This article is the first summarized version of chapter 5 in [9], but with amendment of AIR algorithm, enhanced related work with recent publications, and enhanced experimental evaluation.

2. Iterative refinement and related work

Linear solvers (i.e., $A\mathbf{x} = \mathbf{b}$) appear in many applications such as electromagnetic simulations, quantum scattering, least-square problems, signal processing, partial differential equations, and computational chemistry [10, 29]. Performing matrix decomposition followed by solving triangular systems can seek the approximation of the exact solution $\mathbf{x}$ (i.e., direct method) [12]. However, the direct method requires $O(n^3)$ matrix decomposition computation with original precision arithmetic (i.e., precision used for data representation) and generally requires a conventional iterative refinement [13] to improve the solution accuracy further. A mixed precision iterative refinement can solve the system with lower significand cost without losing solution accuracy by employing lower precision arithmetic for the $O(n^3)$ matrix decomposition [2].

2.1. Iterative refinement

Algorithm 1 describes the iterative refinement [13]. In this paper, we employ the tilde ($\tilde{}$) notation to indicate quantities computed by a floating point arithmetic. The ϵ_i represents a machine epsilon for an arithmetic precision used for step i and P is the partial pivoting matrix generated by LU decomposition with Partial Pivoting (LUPP); a machine epsilon represents the maximum relative error bound for individual floating point arithmetic [12]. For example, the machine epsilon for a round nearest mode is 2^{-24} for single precision arithmetic and 2^{-53} for double precision arithmetic [14]. The LUPP is performed in step 1 and the lower triangular matrix L and the upper triangular matrix U are used for step 3; thus, the precision applied for step 1 is often used for step 3. Step 2 seeks a residual:

step 1: generate approximator: $L \cdot U = P \cdot A$ with ϵ_1
for i = 0, 1, 2, ... ($\mathbf{x}^{(0)} = \mathbf{0}$)
  step 2: compute the residual: $\tilde{\mathbf{r}}^{(i)} = A\tilde{\mathbf{x}}^{(i)} - \mathbf{b}$ with ϵ_2
  step 3: seek the approx error: $A\tilde{\mathbf{z}}^{(i)} = \tilde{\mathbf{r}}^{(i)}$ with ϵ_3
  step 4: deduct the approx error: $\tilde{\mathbf{x}}^{(i+1)} = \tilde{\mathbf{x}}^{(i)} - \tilde{\mathbf{z}}^{(i)}$ with ϵ_4
end for

Algorithm 1: Iterative refinement

$\tilde{\mathbf{r}}^{(i)} = A(\mathbf{x} + \delta\mathbf{x}^{(i)}) - A\mathbf{x} = A\delta\mathbf{x}^{(i)}$, where $\delta\mathbf{x}^{(i)}$ is the error for the computed solution at the i^{th} iteration. The $\delta\mathbf{x}^{(i)}$ is sought in step 3: $\tilde{\mathbf{z}}^{(i)} = A^{-1}(A\delta\mathbf{x}^{(i)}) = \delta\mathbf{x}^{(i)}$. The error found is subtracted in step 4 : $\tilde{\mathbf{x}}^{(i+1)} = (\mathbf{x} + \delta\mathbf{x}^{(i)}) - \delta\mathbf{x}^{(i)} = \mathbf{x}$. Without rounding errors from step 2 to 4, just one iteration can seek the exact solution. Due to rounding errors, the procedure from step 2 to 4 performs iteratively in order to reduce the error in the computed solution; the decreasing rate of the error is known as the convergence rate. In this paper, the procedure from step 2 to 4 is denoted as *the refinement procedure*. The refinement procedure continues until the solution accuracy is numerically satisfied. Any approximation methods for step 3 is allowed for iterative refinement as long as its relative error in $\tilde{\mathbf{z}}$ in step 3 is less than unity [15]. We employ LUPP approximator for step 3 in this paper. In iterative refinement, the original precision (i.e., precision used to represent the input matrix A and vector $\mathbf{b}$) solution accuracy can be generally obtained for backward error by employing the original precision arithmetic for steps 2 and 4 [2] and for forward error by employing doubled precision arithmetic (i.e., the precision arithmetic having doubled significand bits compared to the original precision) for steps 2 and 4 [16, 17]. We often mention the notations specified in Algorithm 1. We also mention higher precision arithmetic if the precision is higher than the original precision and lower precision if the precision is lower than the original precision from this point forward. In this paper, we refer to A as an ill-conditioned matrix if $\kappa(A) > 1/\epsilon_1$, where $\kappa(A)$ is an infinity norm condition number of the matrix A, and as a well-conditioned matrix if $\kappa(A) < \sqrt{1/\epsilon_1}$. Notice that any mixed precision iterative refinements employing a lower precision arithmetic for step 1 is not recommended to ill-conditioned systems in practice.

2.2. Related work

2.2.1. Iterative refinements

In 1948, Wilkinson first proposed iterative refinement [13] employing the original precision arithmetic for steps 1, 3 and 4 and doubled precision arithmetic for step 2; thus, the iterative refinement required extra time for the refinement procedure compared to the direct method, while improving solution accuracy for the forward error. In order to minimize the run time for the iterative refinement without losing accuracy, Kiełbasiński proposed Binary Cascade Iterative Refinement (BCIR) based on Woźniakowski suggestion of employing an arbitrary lower precision arithmetic for step 1 (i.e., $O(n^3)$) [7]. BCIR employs an arbitrary lower precision arithmetic for steps 1 and 3 and an arbitrary higher precision arithmetic for steps 2 and 4 per iteration. However, BCIR had limitations for practical use, since the required number of iterations (or condition number of the matrix) should be known prior to computation [7, 8]. As a practical

Table 1: Precision utilisation for iterative refinements

IRs	Step 1	Step 2	Step 3	Step 4
XMIR	LP	AP	LP	AP
DynIR	SP	DP to DDP	SP	DDP
AIR	LP	APs (LP to HAP)	LP	HAP

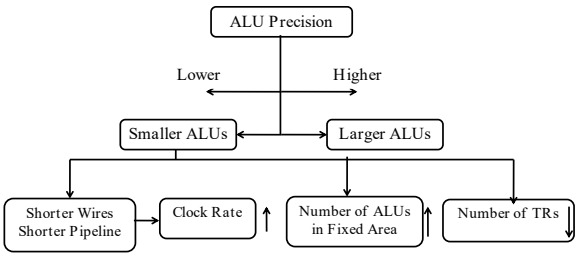

Figure 1: Impact of lower precision ALUs

algorithm, Langou et al. demonstrated that applying single precision arithmetic for steps 1 and 3 and double precision arithmetic for steps 2 and 4 on a CPU minimized run time without losing double precision solution accuracy for backward error [2]. Lee and Peterson proposed eXtended Mixed-precision Iterative Refinement (XMIR) employing an arbitrary static precision arithmetic for steps 2 and 4 to achieve an arbitrary precision accuracy on an FPGA [6]. In 2017, Carson and Higham discussed iterative refinements utilizing three types of precision arithmetic in terms of accuracy and run time [3]. In 2018, Lee et al. proposed a Dynamic precision arithmetic Iterative Refinement (DynIR) that achieves double precision solution accuracy in the forward error, while minimizing run time further compared to statically defined mixed precision iterative refinements [18]. The iterative refinements considered in [3] selected precision statically, using three types of precision for steps 2, 3, and 4 respectively. In contrast, our algorithm AIR adapts precision of steps 2 and 4 dynamically. DynIR dynamically selects either double or double-double precision for steps 2 and 4. AIR extends DynIR by allowing multiple arbitrary precisions dynamically for steps 2 and 4. Table 1 describes the precision utilisation for XMIR, DynIR, and AIR according to the steps in Algorithm 1. Table 1 uses LP for a lower precision, AP for an arbitrary precision, SP for single precision, DP for double precision, DDP for double-double precision, and HAP for the highest precision out of multiple types of arbitrary precision in AIR. We chose XMIR of [6] as a state of the art mixed precision iterative refinement to compare its accuracy and significand cost with AIR, since employing the same arithmetic precision for step 4 as step 2 has extra accuracy benefits with negligible performance loss [17].

2.2.2. *Exploiting the increased parallelism on FPGA and ASIC*

The computational precision impacts the resulting performance and energy efficiency as shown in Fig. 1. Lower precision ALUs can be implemented using shorter pipelines and wires so that the applied clock rate can be improved. Since a lower precision ALU is smaller than a higher precision ALU, relatively larger numbers of lower precision ALUs can be employed within the same area as for a few higher precision ALUs;

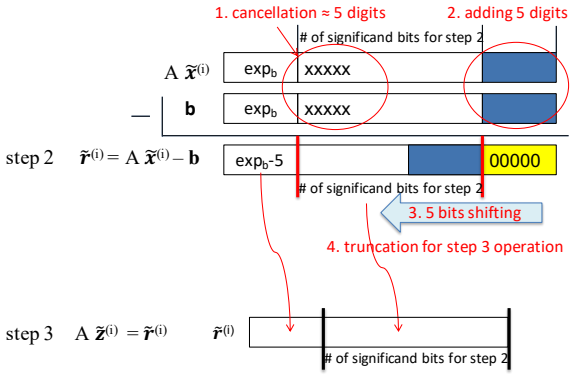

Figure 2: Numerical property of cancellation error

the exploitable parallelism for the computation can be increased with the additional ALUs. Hence, employing lower precision ALUs can achieve higher performance. The number of transistors in a lower precision ALU is less than for a higher precision ALU, implying that a lower precision ALU requires less power. Therefore, employing the least sufficient precision that produces the prescribed solution accuracy can result in higher performance without increasing power consumption.

Dynamically configured ALUs according to a level of precision arithmetic were proposed in [19, 20, 21, 22, 23, 24] to exploit such lower precision arithmetic benefits. In [19], a 64-bit multiply accumulator can be configured according to multiple precisions to compute one 64x64, two 32x32, four 16x16, or eight 8x8 unsigned/signed multiply-accumulations using shared segmentation. In [20], a quadruple precision multiplier is proposed that also can compute two double precision multiplications. In [21], a dual-mode floating-point divider is proposed that is capable of performing two parallel double-precision divisions or one quadruple-precision division. In [22, 23], a novel ALU architecture has been proposed that performs arbitrary precision arithmetic operations according to the user preference. In [24], ALUs employing dynamic bit-width truncation have been proposed. Increasing the number of the Processing Elements (PEs) on FPGA or ASIC by employing lower precision ALUs can improve performance. FPGA can also utilize its reconfigurablity to deal with the dynamically varying number of PEs whenever the precision requirement is changed.

3. Arbitrary dynamic precision iterative refinement

AIR employs a static lower precision arithmetic for steps 1 and 3 and an arbitrary dynamic precision arithmetic for steps 2 and 4 per iteration (i.e., one static precision for step 3 and the other dynamically varying precision for steps 2 and 4 per iteration).

3.1. *Motivation of AIR*

Fig. 2 describes the numerical property occurred in transferring the residual computed from step 2 to step 3. For example, if $|\tilde{r}_j^{(i-1)}| = 1$ and $|\tilde{r}_j^{(i)}| = 2^{-5}$ at step 2 for the two consecutive iterations, where $\tilde{r}_j^{(i)}$ represents the j^{th} component of the

residual $\tilde{\mathbf{r}}^{(i)}$ at the i^{th} iteration, the absolute value of the difference between the two exponents is 5. In this case, if step 2 employs additional 5 bits for the significand at the i^{th} iteration, the resulting residual does not lose any information for step 3 after the truncation. The decreasing ratio $|\tilde{r}_j^{(i)}|/|\tilde{r}_j^{(i-1)}|$ is a component-wise convergence rate for the residual. Assuming that $|\tilde{r}_j^{(i+1)}|/|\tilde{r}_j^{(i)}| = |\tilde{r}_j^{(i)}|/|\tilde{r}_j^{(i-1)}|$, the least sufficient precision arithmetic for steps 2 and 4 at the $(i+1)^{th}$ iteration requires the machine epsilon $2^{-5}\epsilon_2^{(i)}$, where $\epsilon_2^{(i)}$ is a precision machine epsilon for step 2 at the i^{th} iteration. In such case, the significand bit-width required for the j^{th} component in step 2 at the $(i+1)^{th}$ iteration, $t_{j,2}^{(i+1)}$, can be represented:
$t_{j,2}^{(i+1)} = t_1 + \lceil log_2(|b_j|/|\tilde{r}_j^{(i)}|)\rceil + \lceil log_2(|\tilde{r}_j^{(i-1)}|/|\tilde{r}_j^{(i)}|)\rceil$,
where t_1 is the number of significand bits applied for step 1 and b_j is the j^{th} component of $\mathbf{b}$. Notice that $\mathbf{b} = \tilde{\mathbf{r}}^{(0)}$. In other words, the number of significand bits for step 2 at the next iteration is represented by the significand bit-width of the initial precision for LUPP (i.e., t_1) augmented by the number of cancellation bits at the current iteration and the number of cancellation bits between the two consecutive residuals.

Determining precision arithmetic based on the component-wise convergence rate is not appropriate in practice, since the step 2 implementation using n different types of precision arithmetic makes the data path on the FPGA (or ASIC) complicated. Therefore, one representable precision arithmetic is desirable for step 2 and we consider an infinity norm convergence rate instead of multiple component-wise convergence rates. An infinity norm represents the maximum absolute value out of all components in the vector. We denote $\|\cdot\|$ for the infinity norm instead of $\|\cdot\|_\infty$ in this paper for simplification.

3.2. Description of AIR

Algorithm 2 describes AIR that determines an arbitrary precision arithmetic for steps 2 and 4 adaptively per iteration. If convergence occurs (i.e., $\|\tilde{\mathbf{r}}^{(i)}\|/\|\tilde{\mathbf{r}}^{(i-1)}\| < 0.5$) at the current iteration i, the number of significand bits for step 2 at the next iteration, $t_2^{(i+1)}$, is adaptively chosen based on the dynamic information of the approximate number of cancellation bits at the current iteration (i.e., $\lceil log_2(\|\mathbf{b}\|/\|\tilde{\mathbf{r}}^{(i)}\|)\rceil$) and the approximate number of cancellation bits between the two consecutive residuals of $\tilde{\mathbf{r}}^{(i)}$ and $\tilde{\mathbf{r}}^{(i-1)}$ (i.e., $\lceil log_2(\|\tilde{\mathbf{r}}^{(i-1)}\|/\|\tilde{\mathbf{r}}^{(i)}\|)\rceil$). Therefore, in Algorithm 2, the next significand bit-width for step 2, $t_2^{(i+1)}$ is:
$t_2^{(i+1)} = t_1 + \lceil log_2(\|\mathbf{b}\|/\|\tilde{\mathbf{r}}^{(i)}\|)\rceil + \lceil log_2(\|\tilde{\mathbf{r}}^{(i-1)}\|/\|\tilde{\mathbf{r}}^{(i)}\|)\rceil$. If $t_O > 2t_1$, $2t_1$ is chosen for $t_2^{(1)}$ since it guarantees the residual accuracy [13, 16]. If the number of cancellation bits is low, it is possible that $t_2^{(2)} < t_2^{(1)}$. Notice that step 2 computation is not required when $i = 0$, since $\mathbf{x}^{(0)} = \mathbf{0}$.

AIR determines the level of the precision arithmetic for the next iteration at the current iteration. The precision machine epsilon chosen by AIR for the next iteration can be represented using convergence rates. For example, the machine epsilon, $\epsilon_2^{(i+1)} = \tilde{\sigma}_b^{(i+1)}\epsilon_1$, where $\tilde{\sigma}_b^{(i+1)} = \sigma_b^{(i)}\sigma_b^{(i)}\sigma_b^{(i-1)}\ldots\sigma_b^{(1)}$ and $\sigma_b^{(i)} = \|\tilde{\mathbf{r}}^{(i)}\|/\|\tilde{\mathbf{r}}^{(i-1)}\|$ representing the convergence rate at the i^{th} iteration. Notice that the determination for $\epsilon_2^{(i+1)}$ employs two $\sigma_b^{(i)}$s in $\tilde{\sigma}_b^{(i+1)}$, implying that AIR uses the information of the current convergence rate to support additional sufficient significand bits for the next step 2.

step 1: generate approximator : $L \cdot U = P \cdot A$ with t_1
$\|\tilde{\mathbf{r}}^{(0)}\| = \|\mathbf{b}\|$;
$A\tilde{\mathbf{x}}^{(1)} = \mathbf{b}$ with t_3;
if ($t_O > 2t_1$) $t_2^{(1)} = 2t_1$; else $t_2^{(1)} = t_O$; end if

for i = 1, 2, ...
step 2: compute the residual : $\tilde{\mathbf{r}}^{(i)} = A\tilde{\mathbf{x}}^{(i)} - \mathbf{b}$ with $t_2^{(i)}$
if ($\|\tilde{\mathbf{r}}^{(i)}\|/\|\tilde{\mathbf{r}}^{(i-1)}\| < 0.5$) //convergence
$t_2^{(i+1)} = t_1 + \lceil log_2(\|\mathbf{b}\|/\|\tilde{\mathbf{r}}^{(i)}\|)\rceil + \lceil log_2(\|\tilde{\mathbf{r}}^{(i-1)}\|/\|\tilde{\mathbf{r}}^{(i)}\|)\rceil$;
else //no convergence
$t_2^{(i+1)} = t_2^{(i)} + 1$;
end if
if ($t_2^{(i)} > t_O$) $t_2^{(i)} = t_O$; end if
step 3: seek the approximated error : $A\tilde{\mathbf{z}}^{(i)} = \tilde{\mathbf{r}}^{(i)}$ with t_3
step 4: deduct the approximated error : $\tilde{\mathbf{x}}^{(i+1)} = \tilde{\mathbf{x}}^{(i)} - \tilde{\mathbf{z}}^{(i)}$ with $t_4^{(i)}$
end for

Note.
$t_2^{(i)}(= t_4^{(i)})$:the number of significand bits for step 2 (and 4) at the i^{th} iteration.
$t_1(= t_3)$:the number of significand bits for step 1.
t_O: the number of significand bits for the original precision (i.e., prescribed solution accuracy precision).

Algorithm 2: Arbitrary dynamic precision iterative refinement

3.3. Computational cost for adaptive precision determination

For instruction-driven architectures (e.g., CPU), the time cost overhead of n comparisons for $\|\cdot\|$ for the precision determination by AIR is theoretically trivial compared to total run time consisting of $O(n^3)$ step 1 and $O(n^2)$ step 2 if the matrices are not too small. The two logs followed by ceiling operations per iteration are also negligible compared to $O(n^2)$ step 2 operations per iteration. However, notice that in parallel architectures such as GPUs, global reduction comparison can be expensive.

For data-driven architectures (e.g., FPGA), the precision determination overhead is also negligible in terms of time cost and area since the exponent values of the residuals can be extracted to determine an arithmetic precision as described in Eq. (1) requiring just five integer arithmetic:

$$t_2^{(i+1)} = t_1 + max(|exp_{\mathbf{b}}|) + max(|exp_{\tilde{\mathbf{r}}^{(i-1)}}|) - 2max(|exp_{\tilde{\mathbf{r}}^{(i)}}|) + 2 \quad (1)$$

where the last term '2' is used to replace the two ceiling operations used in Algorithm 2 and $max(|exp_{\mathbf{b}}|)$ represents absolute exponent value of the maximum absolute component in $\mathbf{b}$.

3.4. Numerical properties of AIR

The correctness of AIR is described in Numerical Property 1 (NP1) and Numerical Property 2 (NP2). The proofs for NP1 and NP2 are described in Appendix A and Appendix B respectively.

Numerical Property 1. *Accuracy of AIR for forward error*
AIR produces the solution accuracy for forward error:
$\|\tilde{\mathbf{x}}^{(m+1)} - \mathbf{x}\|/\|\mathbf{x}\| \leq \mu_1 \kappa(A)\epsilon_O$ *if* $\kappa(A)\epsilon_2^{(i)} \leq 1/2$,
where μ_1 *is a positive value depending on the matrix size of A (e.g., in practice,* $\mu_1 \leq \sqrt{n}$*) and m is the required number of iterations to achieve the prescribed accuracy. [12].*

Numerical Property 2. *Accuracy of AIR for backward error*
AIR produces the original precision solution accuracy for backward error (i.e., AIR is a backward stable algorithm.):
$\|b - A\tilde{\mathbf{x}}^{(m+2)}\|/\|A\|\|\tilde{\mathbf{x}}^{(m+2)}\| \leq \mu_2\epsilon_O$ *if* $\kappa(A)\epsilon_2^{(i)} \leq 1/2$ *and* $q\kappa(A)$ *is less than* 10 *[15], where q is a relative error of a LUPP solver and* μ_2 *is a positive value depending on the matrix size of A (e.g., in practice,* $\mu_2 \leq \sqrt{n}$*).*

Notice that AIR in Algorithm 2 employs the original precision arithmetic for the highest precision arithmetic for steps 2 and 4. Employing doubled precision arithmetic for steps 2 and 4 can make AIR obtain the original precision solution accuracy for forward error [13, 16]. Its proof is the same as Appendix A, but with setting $\epsilon_2^{(m)} = \epsilon_O^2$ in Appendix A.

The number of iterations required for AIR should not be increased substantially compared to XMIR of [6] which constantly employs the original arithmetic precision for steps 2 and 4. Numerical Property 3 (NP3) describes that the required number of iterations for AIR is equivalent to XMIR for well-conditioned matrices. The proof is in Appendix C.

Numerical Property 3. *Required number of additional iterations for AIR*
If a matrix is well-conditioned (i.e., $\kappa(A) < \sqrt{10/\epsilon_1}$*), AIR requires 1 additional iteration at most compared to XMIR.*

4. Performance properties of AIR

4.1. Hardware characteristics based on Roofline

4.1.1. Instruction-driven architectures

For insturction-driven architectures, Roofline model of [25] provides a plot explaining attainable performance according to the operational intensity defined as the number of floating point operations per byte access from DRAM. A computing kernel having a low operational intensity is not able to achieve a peak performance defined by hardware specification since the data supply rate from DRAM to CPU cannot catch up with the data consumption rate by arithmetic operations. Such kernels are called memory bound kernels in [25]. Therefore, the attainable performances for memory bound kernels are limited by the memory bandwidth capability. Other type kernels are named compute bound kernels that can approach the peak performance defined by hardware specification. Utilising reduced precision arithmetic can improve the computational performance linearly for both memory bound kernels by improving the data supply rate from DRAM to CPU and compute bound kernels by increasing word-level parallelism on SIMD architectures.

4.1.2. Speedup factor (p)

We denote p as a speedup factor by employing reduced precision arithmetic. For linear speedup, $p = 1$ (e.g., if single precision arithmetic is twice as fast as double precision, $p \approx 1$.) and for quadratic gain, $p = 2$. Therefore, p depends on hardware architectures. For example, $p \approx 1$ for both memory and compute bound kernels for NVidia Pascal GPUs since the operational intensity increases linearly by accessing reduced precision data, and the computational performance also increases linearly as arithmetic precision is reduced either from double to single or from single to half.

4.1.3. Data-driven architectures

The Roofline of [26] for reconfigurable computing is deviated from the traditional Roofline of [25], since the hardware circuit can be reconfigurable during run time so that attainable performance can be dynamic. The attainable performance for compute bound kernels on FPGAs can be approximately estimated as : the number of the PEs $\times$ the number of floating point operations per PE $\times$ clock frequency [26, 27]. Therefore, unless the allowable clock frequency is affected, the attainable performance is improved in proportion to the increased ratio of the number of PEs for compute bound kernels. For memory bound kernels on a reconfigurable computing platform employing direct memory access such as a Cray XD1, the memory bandwidth between DRAM and FPGA is the main factor limiting computational performance [28]. In such cases, $p \approx 1$ even for data-driven architectures. It is highly probable that solving smaller matrices by AIR could belong to memory bound kernels, resulting in $p \approx 1$ for data-driven architectures.

4.2. Performance modelling

It is natural to consider a run time modeling as a function of significand bit-width, since significand bit-width is directly related to time cost for a floating point operation [1]. Therefore, we consider the average run time per floating point operation as a function of significand bit-width discussed in [6] in order to explore the theoretical speedups by AIR. The average run time modeling assumes that the time costs for addition, subtraction, multiplication, and division are equal for iterative refinements. Even though it takes longer for division than other floating point operations, the portion of divisions in iterative refinement is relatively small. We also assume that the time cost from the adaptive precision determination is trivial, compared to the total time cost to solve a linear system; considering the time cost from the adaptive precision determination, the total time cost of each iterative refinement should be increased by $\eta \times n \times$ (number of iterations), where the η is a variable depending on the implementation of the precision determination on a computing platform. Based on the assumption, the average run time per floating point operation is:

$$T(t) = a \cdot t^p, \text{where } p = \{1, 2\} \tag{2}$$

The t is the number of significand bits for precision arithmetic, the a is a scaling factor to make Eq. (2) hold. We employ p to present the performance variation *theoretically* with a simple

function of precision. The more realistic performance model would be more complicated since it needs to consider arithmetic unit architecture, memory organisation structure and etc.

4.3. Speedup by XMIR over direct method

First, we explore the time cost for XMIR that produces the original precision accuracy for backward error using Eq. (2). At $i = 0$ in Algorithm 1, only steps 1 and 3 are performed : this process is called the direct method and actual refinement procedure starts when $i = 1$. Ignoring the $O(n)$ step 4, the theoretical time cost for XMIR is described in (3): $2n^3/3$ operations are required for step 1 and $2n^2$ each for step 2 and 3.

$$T_{XMIR} = T(t_O)(2/3n^3 \times \gamma^p + 2n^2(m_X + 1)(1 + \gamma^p)) \quad (3)$$

where γ represents the ratio of the required significand bits between lower and original precision arithmetic (i.e., $\gamma = t_1/t_O$) and m_X is the required number of iterations for XMIR. The γ^p is the key parameter to determine the achievable speedup. For example, CPU or GPU has the fixed γs such as $1/2$ (e.g., t_{single}/t_{double}) or $1/4$ (e.g., t_{half}/t_{double}) and $p \approx 1$, however FPGA has a variable γ due to arbitrary precision support and $p \gtrsim 1$ (i.e., linear speedup for adders and quadradic speedup for multipliers [5]). When $\gamma^p \to 1$, XMIR is not beneficial in terms of the speedup, since it still needs $O(n^3)$ original precision computation. In other words, when $\gamma^p \to 0$, the impact of XMIR on the speedup over the direct method using LUPP can be maximized, since XMIR requires only $O(n^2)$ original precision computation. The theoretical speedup of XMIR over an LUPP direct method is as follows:

$$S_{X/D} = (1 + 3/n)/(\gamma^p + 3(m_X + 1)(1 + \gamma^p)/n) \quad (4)$$

Eq.(4) leads to Performance Property 1 (PP1).

Performance Property 1. *Speedup by XMIR over LUPP direct method is:* $S_{X/D} = (1 + 3/n)/(\gamma^p + 3(m_X + 1)(1 + \gamma^p)/n)$
The ratio of the significand cost of XMIR to direct method is: $(\gamma + 3(1 + \gamma)(m_X + 1)/n)/(1 + 3/n)$
When $\gamma^p \to 0$*, the speedup of XMIR over direct method is:* $(n + 3)/(3(m_X + 1))\times$*, implying a greater speedup for a larger matrix requiring less number of iterations.*
When $n \to \infty$*, the speedup of XMIR over direct method is:* $(t_O/t_1)^p$*, implying a greater speedup for a larger p and a larger significand gap between* t_O *and* t_1*. For most current instruction-driven architectures having* $p \approx 1$*, the maximum speedup by XMIR over direct method will approach* $(t_O/t_1)\times$ *when* $n \to \infty$*.*

Notice that PP1 assumes that direct method produces the prescribed solution accuracy without refinement procedure even though a few iterations are usually required for ill-conditioned matrices; we will see it shortly in the experimental evaluation section.

4.4. Speedup by AIR over XMIR

The theoretical achievable speedup by AIR over XMIR is described in Performance Property 2 (PP2). The proof is shown in Appendix D.

Table 2: Number of significand bits for the iterative refinements

Iterative Refinements	Step 1	Step 2	Step 3	Step 4
UIR	53	53	53	53
XMIR	12	53	12	53
BCIR(nested loop)	12	24-53	12	24-53
AIR	12	13-53	12	13-53

Performance Property 2. *Max speedup by AIR over XMIR for backward error*
When $\gamma^p \to 0$*, the achievable speedup of AIR over XMIR approaches* $(p + 1)\times$.

Based on PP2, the greater speedup by AIR over XMIR occurs if a user requires a higher accuracy. The maximum energy efficiency of AIR over XMIR can also approach $(p + 1)\times$ if power consumption is constant during execution. For large problems, the speedup by AIR over direct method approaches the speedup by XMIR over direct method as the factorization dominates overall run time. For a large matrix, the speedup by AIR over XMIR becomes higher when the prescribed accuracy becomes higher, resulting in more iterations required.

5. Experimental evaluation

The aim of this section is to demonstrate that AIR solves linear systems with the lowest significand cost without losing solution accuracy, compared to XMIR, BCIR, and Uni-precision Iterative Refinement (UIR). The UIR employs 53 significand bits including 1 bit for implicit significand for all steps in Algorithm 1 (i.e., the iterative refinement followed by direct method). Table 2 shows the numbers of the significand bits utilized in the iterative refinements for our experiments. For the demonstration, we implement the four types of arbitrary precision floating point operations for addition, subtraction, multiplication, and division in C; AIR, XMIR, BCIR, and UIR are implemented with the arbitrary precision arithmetic functions. Then, we measure the numbers of iterations taken for AIR, XMIR, BCIR, and UIR and use them to estimate the total significand costs for the iterative refinements.

5.1. Experimental setup

- Test samples: 100 32×32 matrices with elements from standard normal distribution; the condition numbers of the matrices follow: $log(\kappa(A)) \approx log(n) + 1.537$ [29].
- Test algorithms: AIR, XMIR of [6], BCIR of [8], UIR.
- Approximator (step 1): LUPP approximator with $t_1 = 12$.
- Precision setting: $t_O = 53$ and $t_3 = t_1$ for all iterative refinements.
- Accuracy requirement: double precision solution accuracy for backward error. We terminate the iterations if the following accuracy requirement is met:

$$\frac{\|\tilde{\mathbf{r}}^{(i)}\|_\infty}{\|A\|_\infty \|\tilde{\mathbf{x}}^{(i)}\|_\infty} < \sqrt{n}\, 2^{-53} \quad (5)$$

We skip forward error tests to avoid redundancy, since forward error accuracy is bounded [30]:

$$\text{forward error} \lesssim \text{backward error} \times \kappa(A) \quad (6)$$

In forward error tests with the $2^{-41}(= 2^{-53}2^{t_1})$ accuracy requirement, we observed the similar pattern for the number of iterations to the backward error tests.
- Condition number estimation: the infinity norm condition numbers measured by *dgecon* function from Math Kernel Library.
- Implementation of arbitrary precision: the implemented arbitrary functions truncate the significand bits of the two operands based on the prescribed arbitrary precision, perform double precision arithmetic on the two truncated operands, and return outcomes after re-truncation.

5.1.1. Estimation for total significand cost

We estimate the total significand bit-width required for AIR, XMIR, BCIR, and UIR, using Eq. (7), (8), (9), and (10) respectively.

$$TM_{AIR} = 2n^3t_1/3 + 2n^2\Sigma_{i=1}^{m_A}(t_2^{(i)} + t_3) + 2n^2t_2^{(m_A)} \quad (7)$$

$$TM_{XMIR} = 2n^3t_1/3 + 2n^2\Sigma_{i=1}^{m_X}(t_O + t_3) + 2n^2t_O \quad (8)$$

$$TM_{BCIR} = 2n^3t_1/3 + 2n^2\Sigma_{i=1}^{m_B}(2^{(i-1)}t_2^{B(m_B-i+1)}) + 2n^2(2^{m_B}t_3) + 2n^2t_2^{B(m_B)} \quad (9)$$

$$TM_{UIR} = (2n^3/3 + 2n^2)t_O + m_U2n^2(t_O + t_3) + 2n^2t_O \quad (10)$$

where m_A, m_X, m_B, m_U are the required number of iterations for AIR, XMIR, BCIR, and UIR respectively, and $t_2^{(i)}$ and $t_2^{B(i)}$ are the significand bit-width for step 2 at the i^{th} iteration for AIR and BCIR respectively. Notice that the significand bit-width for step 4 is ignored due to its trivial $O(n)$ computation. The right most part of each of Eq. (7), (8), (9), and (10) is required for the final accuracy check. For the BCIR, $t_2^{B(1)} = 2^1 \times t_1$, $t_2^{B(2)} = 2^2 \times t_1$, ..., $t_2^{B(i)} = 2^i \times t_1$ [8]. If $t_2^{B(i)} > 53$, we employ $t_2^{B(i)} = 53$. We continue to run the BCIR by increasing the required number of iterations until double precision accuracy has been achieved for the backward error. In other words, we assume that the required number of iterations for the BCIR has been known prior to computation. Once the BCIR terminates the iteration, we use the number of iterations taken to measure the total significand bit-width. The XMIR employs $t_2 = t_4 = 53$ for the entire iteration, while for the AIR, $t_2^{(i)}(= t_4^{(i)})$ is dynamically chosen per iteration based on Algorithm 2.

5.2. Number of iterations

Fig. 3 shows the numbers of iterations required to solve the linear systems by AIR, XMIR and UIR. The x-axis represents the Log_2-based infinity norm condition numbers of the matrices and the y-axis represents the number of iterations taken. We set 29 for the maximum number of iterations for AIR, XMIR and UIR. AIR shows one additional iteration at most for all matrices having $\kappa(A) < 2^{10.8}$ (87% of the matrices) compared to XMIR and requires more than one iteration only for the three matrices of $Log_2(\kappa(A)) = 2^{10.8}$, $2^{14.4}$, and $2^{16.3}$. This experiment supports NP3 empirically. In other words, if a matrix is well-conditioned (e.g., $\kappa(A) < \sqrt{10 \times 2^{12}} = 2^{7.7}$), AIR requires one additional iteration at most. AIR also shows one less iteration than XMIR for the three matrices of $Log_2(\kappa(A)) = 2^{9.0}$, $2^{9.97}$, and $2^{9.98}$. Notice that NP3 holds for all well-conditioned matrices based on the theoretical proof but AIR requires more than one iterations only for the three matrices in our experiments, implying that 97% of the test matrices can hold NP3 over the entire range of the condition numbers of the test matrices. AIR and XMIR did not achieve the prescribed accuracy for nine matrices in 29 iterations that are shown with the marks at 30 at the y-axis in Fig. 3. Particularly for the ill-conditioned matrices (i.e., $\kappa(A) > 2^{13}$), AIR produces the prescribed accuracy for one out of the four matrices; increasing t_1 can make AIR produce the prescribed accuracy for the matrices. However, both AIR and XMIR produce the prescribed double precision accuracy for all well-conditioned matrices. UIR requires at most one iteration to achieve the prescribed solution accuracy in our experiments.

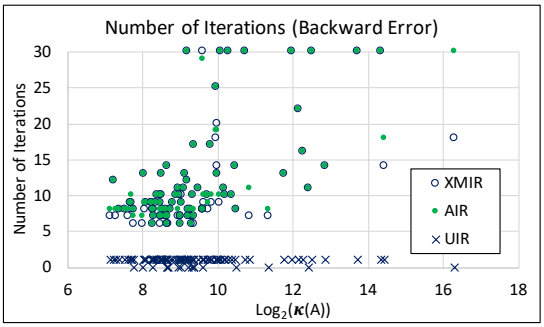

Figure 3: Numbers of iterations for AIR and XMIR

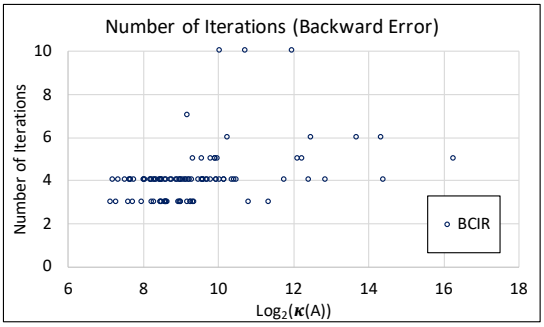

Figure 4: Numbers of iterations for BCIR

Fig. 4 shows the numbers of iterations required for BCIR. Notice that BCIR is a recursive iterative refinement; therefore, the numbers of iterations are fewer than XMIR and AIR. Please refer to [7] or [8] for the details of the BCIR. We set nine for the maximum number of iterations for BCIR. The BCIR requires three to seven iterations for most matrices. The BCIR did not achieve the prescribed accuracy in nine iterations for the three matrices marked at ten in the y-axis.

5.3. Significand bit-width variation of AIR

Fig. 5 shows the numbers of significand bits used per iteration for AIR for the four sample matrices for which both AIR and XMIR show the same numbers of iterations. A matrix of a higher condition number generally requires more iterations. Except the matrix of $\kappa(A) = 2^{8.25}$, the other three matrices show $t_2^{(1)} > t_2^{(2)}$ due to the low convergence rates. The significand bit-width generally is increased as iteration proceeds. Fig. 6

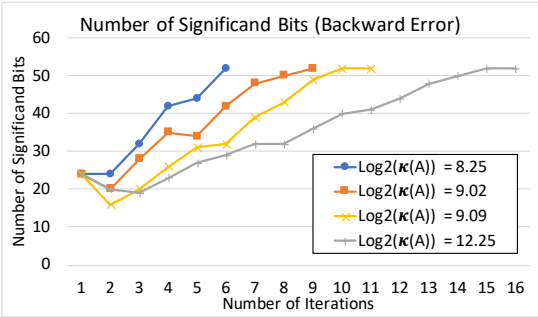

Figure 5: Step 2 significand bit-widths according to $\kappa(A)$s

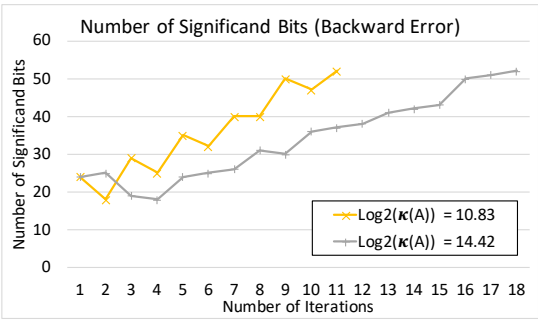

Figure 6: Step 2 significand bit-widths according to $\kappa(A)$s

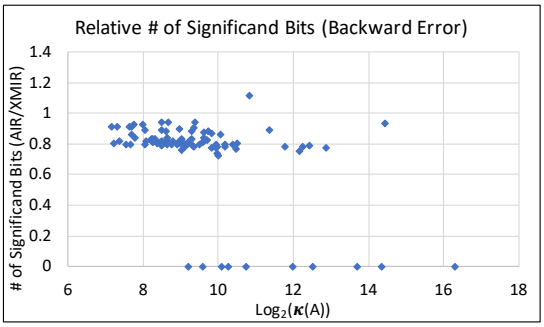

Figure 7: Ratios of significand costs of AIR to XMIR

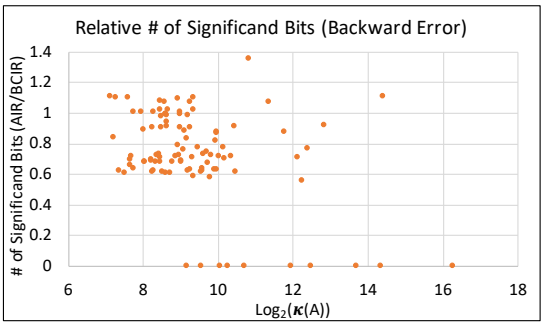

Figure 8: Ratio of significand cost of AIR to BCIR

shows the significand bits used per iteration for AIR for the two matrices for which AIR requires 4 additional iterations when comparing to XMIR. For the matrix of $\kappa(A) = 2^{10.83}$, the numbers of significand bits were oscillated, implying that the infinity norm convergence rates were not steady. For the matrix of $\kappa(A) = 2^{14.42}$, convergences might not occur in many places (i.e., It is highly possible that 1 bit increment occurs in AIR when the convergence does not occur.).

5.4. Significand cost reduction by AIR

We exclude the $O(n)$ adaptive precision determination computation from the theoretical evaluation for the significand cost reduction by AIR.

5.4.1. Significand cost reduction to XMIR

Fig. 7 describes the ratio of the required significand bit-widths of AIR to XMIR ($= TM_{AIR}/TM_{XMIR}$) based on Eq. (7) and (8). The x-axis represents the Log_2-based infinity norm condition numbers of the matrices and the y-axis represents the ratio of the total required significand bit-widths of AIR to XMIR. The ten ill-conditioned matrices for which either AIR or XMIR does not achieve the prescribed accuracy in 29 iterations are marked at 0 in the y-axis. Excluding the ten matrices, AIR shows less significand bits for all cases except one matrix of $\kappa(A) = 2^{10.8}$. For this case, AIR required four additional iterations over XMIR. AIR requires 0.83 significand cost on average with variance 0.003, compared to XMIR. This implies 1.20× speedup for $p = 1$ and 1.44× speedup for $p = 2$.

5.4.2. Significand cost reduction to BCIR

Fig. 8 describes the ratios of the required significand bit-widths of AIR to BCIR based on Eq. (7) and (9). The ten ill-conditioned matrices for which either AIR or BCIR does not achieve the prescribed accuracy are marked at 0 in the y-axis. Excluding the ten matrices, AIR shows the ratio of the significand costs to BCIR lower than '1' for most cases. AIR requires 0.81 significand cost ratio on average with variance 0.031 compared to BCIR. This implies 1.23× speedup for $p = 1$ and 1.51× speedup for $p = 2$.

5.4.3. Significand cost reduction to UIR

Fig. 9 shows the ratios of the required significand bits of XMIR, BCIR, and AIR to UIR respectively. AIR shows the lowest significand costs and most of the ratios of the significand costs of AIR to UIR are located below 1. On average, the required significand costs are 0.89 for AIR, 1.07 for XMIR, and 1.14 for BCIR, compared to UIR except the 10 matrices for which any of AIR, XMIR, and BCIR does not meet the prescribed accuracy. Notice that AIR brings the total significand cost reduction to UIR even for such small matrices, while two other algorithms require more significand costs. When the size of matrices becomes larger, XMIR and BCIR also can bring the total significand cost reduction to UIR as discussed in PP1, since the contribution of $O(n^3)$ matrix decomposition to the total significand cost becomes greater. We also perform the same experiments, but with 100 64×64 standard normally distributed matrices using $t_1 = 15$; we increase t_1 since the condition numbers of normally distributed matrices increase in proportion to the matrix size [29]. Fig. 10 shows the ratios of the significand costs of AIR, XMIR, and BCIR to UIR respectively. Overall, the ratios of the significand costs of AIR, XMIR and BCIR to UIR for $n = 64$ are distributed at lower than '1' for most cases. On average, the ratios of their significand costs to UIR are 0.67 for AIR, 0.76 for XMIR, and 0.83 for BCIR except the 11 matrices for which any of AIR, XMIR, and BCIR does

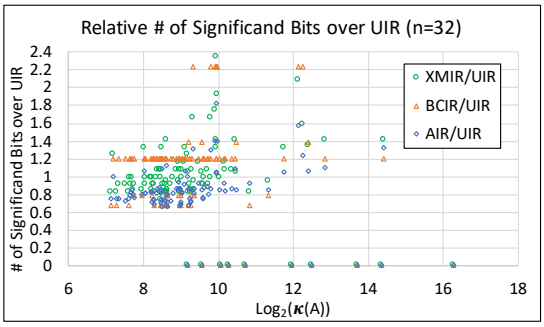

Figure 9: Ratios of significand costs of AIR, XMIR and BCIR to UIR

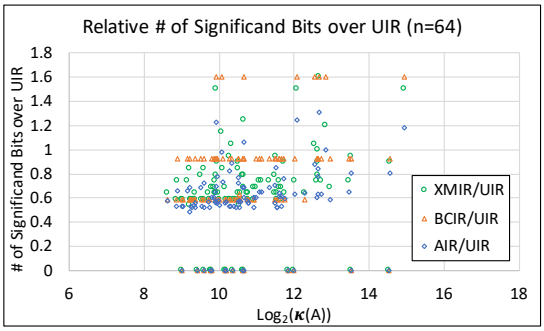

Figure 10: Ratios of significand costs of AIR, XMIR, and BCIR to UIR

not meet the prescribed accuracy. XMIR and BCIR generally show lower significand costs than UIR for standard normally distributed matrices of $n \geq 64$ and AIR minimizes the significand cost even further compared to XMIR and BCIR.

5.4.4. *Significand cost reduction according to matrix size*

Small size matrices (e.g., $n < 100$) might not be of interest to applications using instruction-driven architectures such as GPU or CPU, but can be of interest to some applications using FPGA or ASIC [31]. Therefore, we explore the ratios of the significand costs of AIR to XMIR and BCIR using more fictitious matrices having $n = \{2^7, 2^9, 2^{11}, 2^{13}\}$, while keeping the same numbers of iterations taken as Fig.s 3 and 4 and fixing $t_1 = 12$ (e.g., the condition numbers of the matrices are equivalent to the matrices in Fig.s 3 and 4.). Either diagonally dominant matrices or the matrices close to orthogonal matrices can have small condition numbers for large matrices. Notice that another iterative solver such as Jacobi method can be more efficient over iterative refinements for diagonally dominant matrices. Fig. 11 shows the ratios of significand costs of AIR to XMIR and BCIR according to $n = \{2^5, 2^7, 2^9, 2^{11}, 2^{13}\}$. The ratio of the significand costs of AIR to XMIR approaches '1' when the matrix size grows, due to the greater impact of $O(n^3)$ matrix decomposition on the significand cost. For $n = 8K$, the ratio reaches 0.996 over XMIR and 0.994 over BCIR.

5.4.5. *Significand cost reduction according to the levels of prescribed accuracy*

We pick a sample matrix of $n = 32$ with $\kappa(A) = 2^{12.25}$ in Fig. 5 to see the total significant cost by AIR according to the level of the prescribed accuracy. In this case, 41 significand bits have been cancelled out throughout 16 iterations, implying that

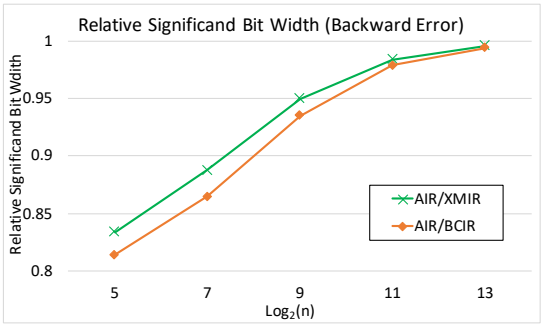

Figure 11: Ratios of significand costs of AIR to XMIR and BCIR according to matrix sizes

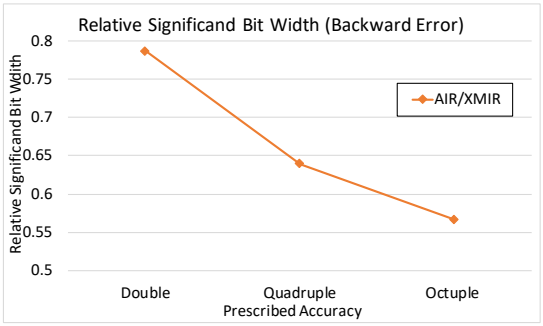

Figure 12: Relative significand bit-widths of AIR to XMIR according to the level of the prescribed accuracy

the average number of cancellation bits per iteration is approximately 2.6 bits. Fig. 12 shows the relative significand costs of AIR over XMIR when prescribed accuracy requires double precision ($=2^{-53}$), quadruple precision ($=2^{-113}$) and octuple precision accuracy ($=2^{-237}$). We used $(t_O - t_1)/2.6$ for the estimated number of iterations. For example, we used $(113 - 12)/2.6$ for the number of iterations required for quadruple precision accuracy to estimate the relative significand costs. The significand cost benefit of AIR over XMIR becomes greater when the prescribed accuracy becomes higher. The lower bound for the relative cost approaches 0.5 as proven in PP2.

5.5. *Increased number of PEs by reduced precision arithmetic on FPGAs*

This section explores the increased parallelism (i.e., p) for step 2 on a Xilinx XC6VSX475T FPGA exploitable by reduced precision arithmetic. We discuss our implementation first and explore the increased parallelism according to the increased number of the PEs for step 2 on the FPGA by employing reduced precision arithmetic later. Fig. 13 describes our implementation of single PE for step 2. One floating point multiplier and one adder are employed for a PE to perform a dot product and one subtraction. Once the pipeline of the adder is full with data, the reduction operation is initiated by feeding the output back to the input. When the last product from the multiplier is sent to the adder, partial sums are stored into a register file for further reduction to produce a scalar value. The register file size depends on the pipeline depth of the adder. The run time for the reduction depends on the adder pipeline depth. Each PE produces an element of a residual. Fig. 14 describes

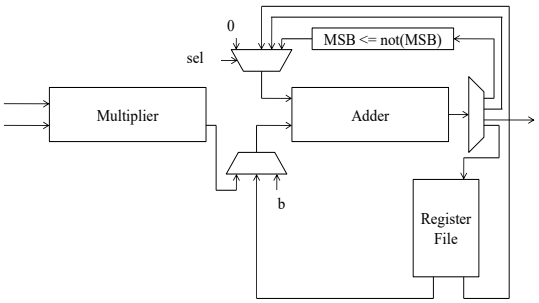

Figure 13: Processing element of step 2 on FPGAs

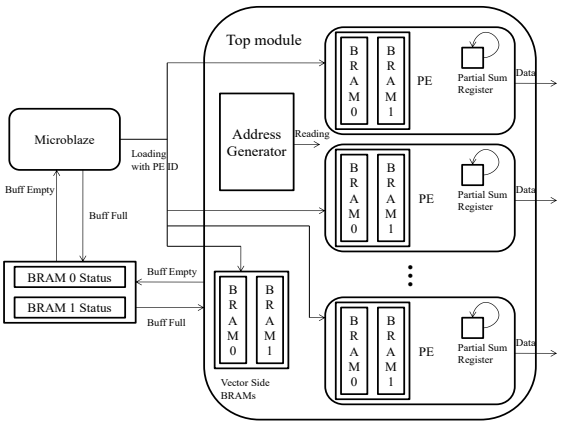

Figure 14: Multiple processing elements on FPGAs

our implementation of step 2 employing multiple PEs. Initially, partial elements of the matrix A and the vector $\tilde{\mathbf{x}}$ are fed into BRAM0. The BRAM0 status register is then set to '1', representing the BRAM0 is ready for the operations. Then, the next partial elements are fed into BRAM1, setting its status register. When the FPGA completely consumes the data from BRAM0, the FPGA resets the BRAM0 status register so that the Microblaze (i.e., host) can load the data into the BRAM0 again. While the Microblaze loads data into the BRAM0, the PE performs the dot product using the data in the BRAM1. Consequently, this method can help hide latency for the data transfer from the Microblaze to the FPGA to some extent. The Partial Sum Registers contain an element of the vector $\mathbf{b}$ initially, partial results of dot products later, and the elements of the residual finally. The Address Generator is used to read the data from the BRAMs inside the PEs.

Table 3 shows the variation of the numbers of the PEs according to precision on a Xilinx XC6VSX475T FPGA. The header uses DSPs for the required numbers of DSP 48 blocks out of 2,016, B for the required 36Kb BRAMs out of 1,064, Slices for the slices out of 74,400, MHz for the allowable clock frequencies from Place and Route in Xilinx ISE v13.2, and GFS for GFLOPS. The slices limit the design to employ 65-170 PEs according to various precision arithmetic. The limited number of PEs is calculated based on 75% of slice usage. We notice that $p \approx 1$ for FPGA in our experiments according to the reduced bit-width (including the exponent bits in the experiments).

Table 3: Number of PEs on XC6VSX475T according to precision arithmetic

Exp	Sig	DSPs	Slices	B	#PEs	MHz	GFS
8	23	680	67,294	342	170	80	27
11	38	530	69,591	321	106	79	17
11	52	1,079	66,945	336	83	82	14
15	63	1,040	68,619	330	65	78	10

6. Discussion

6.1. Impact of AIR according to problem size and prescribed accuracy

Since the factorization cost dominates the overall run time for large n, the ratio of t_O/t_1 is the upper bound for the speedup of AIR (or XMIR) over UIR on the condition $p = 1$ when $n \to \infty$. The ratio of the significand bit-width of AIR to XMIR increases in proportion to n as shown in Fig. 11 and decreases in proportion to the prescribed accuracy as shown in Fig. 12. AIR can improve the speedup further over XMIR when the prescribed accuracy becomes higher for large matrices. For example, for some applications requiring higher precision arithmetic [32], *software-emulated precision arithmetic* is required, resulting in significant slow-down. DynIR already improved $\sim 2\times$ speedups over XMIR in [18] for large dense matrices of $n = 16K$ by minimising the software emulated double-double precision arithmetic operations. DynIR associated with AIR can improve the speedups further by utilising the least sufficient software emulated arithmetic precision provided by AIR algorithm.

6.2. Impact of AIR according to condition numbers

Larger condition numbers require more significand bits for t_1, degrading the significand cost benefit by AIR (or XMIR) over UIR. For example, the significand cost benefits of AIR (or XMIR) over UIR can be decreased for large normally distributed matrices since the condition numbers of normally distributed matrices grow with the matrix size [11]. When the condition number becomes large, the significand cost benefit of AIR over UIR decreases, but the benefit of AIR over XMIR increases since the number of the required iterations increases unless the condition number exceeds the failure threshold. For example, Fig. 7 shows the negative slope of -0.0055 by linear regression, implying that the ratio of total significand bit-width required by AIR over XMIR is reduced in proportion to the condition numbers of the matrices.

6.3. Practical aspect and use of AIR

It will be more beneficial to use AIR for linear systems having multiple right-hand sides $\mathbf{b}$s, given a matrix A (i.e., $A\mathbf{x}_1 = \mathbf{b}_1$, $A\mathbf{x}_2 = \mathbf{b}_2$, ...) [33], since the matrix decomposition is performed only once in this case and the refinement procedure becomes more significant in terms of the total significand cost.

Fig. 15 shows suitability for the various iterative refinement algorithms according to the problem size and the level of the prescribed solution accuracy. When the prescribed accuracy becomes higher, AIR becomes more beneficial.

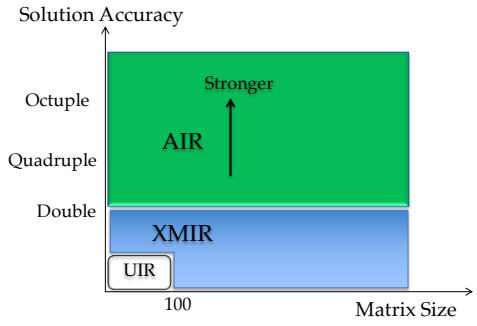

Figure 15: Various iterative refinements suitability

6.4. Limitation on this paper

Currently, this paper considers only significand bit-width to describe stability and performance of AIR. However, based on the IEEE 754 standard [34], the exponent bit-width should be considered along with significand bit-width for more practical performance analyses. Therefore, the numerical experimental results from this paper can be deviated to some extent from the numerical experiments using IEEE data formats.

7. Conclusion

This paper presents the first practical iterative refinement algorithm, AIR, that solves a linear system with the lowest significand cost compared to XMIR and BCIR by exploiting the dynamic information of the number of cancellation bits in the residual per iteration. AIR has been proven as a backward stable algorithm and can produce up to $(p+1)\times$ speedup over XMIR. Our software demonstration shows that AIR can solve linear systems with the standard normally distributed 32×32 matrices for double precision accuracy for backward error with 0.83 significand cost over XMIR and 0.81 significand cost over BCIR, implying 1.20× speedup over XMIR and 1.23× speedup over BCIR for $p = 1$ and 1.44× speedup over XMIR and 1.51× speedup over BCIR for $p = 2$. We are hoping that this work can contribute to arbitrary dynamic precision utilization for other application domain such as deep learning. The detailed implementation methodology for AIR using dynamic precision arithmetic units proposed in [22, 23] and/or variable significand bit length floating point arithmetic units proposed in [24] is our future work.

Appendix A. Proof for forward error for AIR

We prove Numerical Property 1 in this section. For the forward error proof, we first show that the approximation solution error in the i^{th} iteration, $\|\delta\mathbf{x}^{(i+1)}\|$, is equal or less than the approximation solution error in the $(i-1)^{th}$, $\|\delta\mathbf{x}^{(i+1)}\|$, by a factor of the forward convergence rate $\sigma_f^{(i)}$ in the i^{th} iteration.

We denote $\tilde{\mathbf{r}}^{(i)}$, $\tilde{\mathbf{x}}^{(i)}$, and $\tilde{\mathbf{z}}^{(i)}$ for the computed vectors at the i^{th} iteration and $\mathbf{r}^{*(i)}$ and $\mathbf{z}^{*(i)}$ for the vectors derived by exact arithmetic. Our proof is based on the proof used in [15]. The computed residual at the i^{th} iteration is:

$$\tilde{\mathbf{r}}^{(i)} = (I+\delta I^{(i)})(A\tilde{\mathbf{x}}^{(i)}-\mathbf{b}+\delta\mathbf{y}^{(i)}) = (I+\delta I^{(i)})(A\delta\mathbf{x}^{(i)}+\delta\mathbf{y}^{(i)}) \quad \text{(A.1)}$$

where I is an identity matrix, $\delta I^{(i)}$ is a diagonal matrix whose infinity norm is bounded to the machine epsilon for the precision applied for steps 2 and 4 at the i^{th} iteration (i.e., $\|\delta I^{(i)}\| \le \epsilon_2^{(i)}$), $\delta\mathbf{y}^{(i)}$ is a vector containing rounding errors from the matrix vector multiplication of $A\tilde{\mathbf{x}}^{(i)}$ and $\tilde{\mathbf{r}}^{(i)}$ is a computed residual. Notice that the truncation error experienced from step 2 to 3 is not considered, since there is no accuracy loss as discussed in Fig. 2. The norm bound for $\delta\mathbf{y}^{(i)}$ is represented:

$$\|\delta\mathbf{y}^{(i)}\| \le c_1^{(i)}\epsilon_2^{(i)}\|A\|\|\tilde{\mathbf{x}}^{(i)}\| \quad \text{(A.2)}$$

where $c_1^{(i)}$ is a small constant depending on the matrix size at the i^{th} iteration.

In step 3, the computed residual can be represented:

$$(A+\Delta A)(\mathbf{z}^{*(i)}+\delta\mathbf{z}^{(i)}) = \tilde{\mathbf{r}}^{(i)} \quad \text{(A.3)}$$

where $\mathbf{z}^{*(i)}$ is the exact solution vector in step 3 at the i^{th} iteration and ΔA is a matrix to make (A.3) hold. The $\mathbf{z}^{*(i)}$ is represented:

$$\mathbf{z}^{*(i)} = A^{-1}\tilde{\mathbf{r}}^{(i)} = \delta\mathbf{x}^{(i)} + A^{-1}\delta\mathbf{y}^{(i)} + A^{-1}\delta I^{(i)}(A\delta\mathbf{x}^{(i)}+\delta\mathbf{y}^{(i)}) \quad \text{(A.4)}$$

The norm bound for $\mathbf{z}^{*(i)}$ is represented:

$$\|\mathbf{z}^{*(i)}\| \le (1+\epsilon_2^{(i)}\kappa(A))\|\delta\mathbf{x}^{(i)}\| + (1+\epsilon_2^{(i)})\|A^{-1}\|\|\delta\mathbf{y}^{(i)}\| \quad \text{(A.5)}$$

The relative error using a LUPP approximator should be smaller than unity for the successful condition for the iterative refinement. Therefore, we describe the successful condition for the approximator as:

$$\|\delta\mathbf{z}^{(i)}\| \le q\|\mathbf{z}^{*(i)}\| \text{ and } q < 1 \quad \text{(A.6)}$$

Using (A.4), a computed solution in step 4 is represented:

$$\tilde{\mathbf{x}}^{(i+1)} = \mathbf{x} + \delta\mathbf{x}^{(i+1)} \quad \text{(A.7)}$$

where

$$\delta\mathbf{x}^{(i+1)} = -A^{-1}\delta\mathbf{y}^{(i)} - A^{-1}\delta I^{(i)}(A\delta\mathbf{x}^{(i)}+\delta\mathbf{y}^{(i)}) - \delta\mathbf{z}^{(i)} + \delta\mathbf{d}^{(i)} \quad \text{(A.8)}$$

The $\delta\mathbf{d}^{(i)}$ is a rounding error vector caused by floating point arithmetic for step 4 and the norm bound for $\delta\mathbf{d}^{(i)}$ is represented:

$$\|\delta\mathbf{d}^{(i)}\| \le \epsilon_{4(=2)}^{(i)}\|\mathbf{x}+\delta\mathbf{x}^{(i)} - (\mathbf{z}^{*(i)}+\delta\mathbf{z}^{(i)})\| \quad \text{(A.9)}$$

The norm bound for $\delta\mathbf{x}^{(i+1)}$ is represented:

$$\begin{aligned}\|\delta\mathbf{x}^{(i+1)}\| &\le \|A^{-1}\|\|\delta\mathbf{y}^{(i)}\| + \epsilon_2^{(i)}\|A^{-1}\|\|A\delta\mathbf{x}^{(i)}+\delta\mathbf{y}^{(i)}\| \\ &\quad + \|\delta\mathbf{z}^{(i)}\| + \|\delta\mathbf{d}^{(i)}\| \\ &\le (q + (1+(1+q)(1+\kappa(A)(1+c_1^{(i)})))\epsilon_2^{(i)})\|\delta\mathbf{x}^{(i)}\| \\ &\quad + (c_1^{(i)}\kappa(A)(1+q)+1)\epsilon_2^{(i)}\|\mathbf{x}\| + O(\epsilon^2).\end{aligned} \quad \text{(A.10)}$$

where $O(\epsilon^2)$ is a small number having a size of $\kappa(A)(\epsilon_2^{(i)})^2$ at most which is a negligible quantity for the proof. The detailed information of $O(\epsilon)$ is described in Lecture 12 in [12]. Therefore, the convergence rate for the solution at the i^{th} iteration is represented:

$$\|\delta\mathbf{x}^{(i+1)}\|/\|\mathbf{x}\| \le \sigma_f^{(i)}\|\delta\mathbf{x}^{(i)}\|/\|\mathbf{x}\| + p^{(i)} + O(\epsilon^2) \quad \text{(A.11)}$$

where $\sigma_f^{(i)} = q + (1 + (1 + q)(1 + \kappa(A)(1 + c_1^{(i)})))\epsilon_2^{(i)}$ and $p^{(i)} = (c_1^{(i)}\kappa(A)(1 + q) + 1)\epsilon_2^{(i)}$. Using a recursive relation, the (A.11) is represented:

$$\begin{aligned} &||\delta\mathbf{x}^{(i+1)}||/||\mathbf{x}|| \leq \\ &(\Pi_{k=1}^{i}\sigma_f^{(k)})q + \Sigma_{k=2}^{i}(\Pi_{l=k}^{i}\sigma_f^{(l)})p^{(k-1)} + p^{(i)} + O(\epsilon^2) \end{aligned} \tag{A.12}$$

If the required number of iteration m is sufficiently high so that the $(\Pi_{k=1}^{i}\sigma_f^{(k)})q$ approaches zero, the remaining term corresponds to the achievable accuracy as follows:

$$||\delta\mathbf{x}^{(i+1)}||/||\mathbf{x}|| \leq \Sigma_{k=2}^{i}(\Pi_{l=k}^{i}\sigma_f^{(l)})p^{(k-1)} + p^{(i)} + O(\epsilon^2) \tag{A.13}$$

By letting $\tilde{\kappa}(A) = \max_i(c_1^{(i)}\kappa(A)(1+q)+1)$ and $\sigma_{fMAX} = \max_i(\sigma_f^{(i)})$, the (A.13) is represented:

$$||\delta\mathbf{x}^{(i+1)}||/||\mathbf{x}|| \leq \tilde{\kappa}(A)\Sigma_{k=2}^{i}(\Pi_{l=k}^{i}\sigma_{fMAX})\epsilon_2^{(k-1)} + p^{(i)} + O(\epsilon^2) \tag{A.14}$$

We denote v for the number of iterations when the precision arithmetic for step 2 approaches the original precision arithmetic for the first time (i.e., $\epsilon_2^{(v)} = \epsilon_O$.) Then, the (A.14) at the v^{th} iteration is represented:

$$\begin{aligned} ||\delta\mathbf{x}^{(v+1)}||/||\mathbf{x}|| \leq \tilde{\kappa}(A)(\sigma_{fMAX}\epsilon_O + \sigma_{fMAX}^2\epsilon_2^{(v-1)} + \ldots\sigma_{fMAX}^{v-1}\epsilon_2^{(1)}) \\ +p^{(v)} + O(\epsilon^2) \end{aligned} \tag{A.15}$$

We denote m for the total number of iterations required to obtain the best accuracy. Then, using (A.15), the (A.14) at the m^{th} iteration is represented:

$$\begin{aligned} &||\delta\mathbf{x}^{(m+1)}||/||\mathbf{x}|| \leq \\ &\tilde{\kappa}(A)\sigma_{fMAX}(\epsilon_O + \sigma_{fMAX}^{m-v}\epsilon_2^{(1)})/(1 - \sigma_{fMAX}) + p^{(m)} + O(\epsilon^2) \end{aligned} \tag{A.16}$$

where

$$p^{(m)} = (c_1^{(m)}\kappa(A)(1+q)+1)\epsilon_2^{(m)} = (c_1^{(m)}\kappa(A)(1+q)+1)\epsilon_O \tag{A.17}$$

Assuming that $\sigma_{fMAX} < 1$ under the condition $\kappa(A)\epsilon_2^{(i)} \leq 1/2$, the achievable accuracy for forward error is represented as follows:

$$\begin{aligned} ||\delta\mathbf{x}^{(m+1)}||/||\mathbf{x}|| &\leq (\tilde{\kappa}(A)\sigma_{fMAX}/(1 - \sigma_{fMAX}))\epsilon_O + p^{(m)} + O(\epsilon^2) \\ &\leq (c_{1MAX}\sigma_{fMAX}/(1 - \sigma_{fMAX}))\kappa(A)\epsilon_O + p^{(m)} + O(\epsilon^2) \\ &= \mu_1\kappa(A)\epsilon_O + O(\epsilon^2) \end{aligned} \tag{A.18}$$

where $\mu_1 = c_{1MAX}\sigma_{fMAX}/(1-\sigma_{fMAX}) + c_1^{(m)}(1+q) + 1/\kappa(A)$. □

Appendix B. Proof for backward error for AIR

We prove Numerical Property 2 in this section. For the backward error proof, we first seek the norm bound of the difference from the two residuals at the i^{th} iteration between exact arithmetic and floating point arithmetic. Then, we will derive the backward error bound.

Using (A.7), the exact residual at the $(i + 1)^{\text{th}}$ iteration is represented:

$$\mathbf{r}^{*(i+1)} = A\delta\mathbf{x}^{(i+1)} = A\delta\mathbf{x}^{(i)} - \tilde{\mathbf{r}}^{(i)} - A\delta\mathbf{z}^{(i)} + A\delta\mathbf{d}^{(i)} \tag{B.1}$$

where $\mathbf{r}^{*(i+1)}$ is the residual computed with exact arithmetic at the $(i + 1)^{th}$ iteration. From (A.1), the difference between the exact residual and the computed residual at the i^{th} iteration is represented:

$$\tilde{\mathbf{r}}^{(i)} - \mathbf{r}^{*(i)} = \delta I^{(i)}(A\delta\mathbf{x}^{(i)} + \delta\mathbf{y}^{(i)}) + \delta\mathbf{y}^{(i)} \tag{B.2}$$

The norm bound of the difference is represented:

$$\begin{aligned} ||\tilde{\mathbf{r}}^{(i)} - \mathbf{r}^{*(i)}|| &\leq \epsilon_2^{(i)}(||A\delta\mathbf{x}^{(i)}|| + ||\delta\mathbf{y}^{(i)}||) + ||\delta\mathbf{y}^{(i)}|| \\ &\leq \epsilon_2^{(i)}||A||||\delta\mathbf{x}^{(i)}|| + c_1^{(i)}\epsilon_2^{(i)}||A||||\tilde{\mathbf{x}}^{(i)}|| + O(\epsilon^2) \end{aligned} \tag{B.3}$$

From (A.4) and (B.2), the norm bound for $\mathbf{z}^{*(i)}$ is represented in terms of residuals:

$$||\mathbf{z}^{*(i)}|| \leq ||\delta\mathbf{x}^{(i)}|| + ||A^{-1}||\,||\tilde{\mathbf{r}}^{(i)} - \mathbf{r}^{*(i)}|| \tag{B.4}$$

From (B.1), (B.2), (A.6), (A.9), and (B.3) in order, the norm bound of the residual is represented:

$$\begin{aligned} ||\mathbf{r}^{*(i+1)}|| &\leq (1 + \kappa(A)(q + (1 + q)\epsilon_2^{(i)}))||\mathbf{r}^{*(i)} - \tilde{\mathbf{r}}^{(i)}|| \\ &+(q + (2 + q)\epsilon_2^{(i)})||A||\,||\delta\mathbf{x}^{(i)}|| + \epsilon_2^{(i)}||A||\,||\mathbf{x}|| + O(\epsilon^2) \\ &\leq ((1 + c_1^{(i)})(1 + q\kappa(A))\epsilon_2^{(i)} + q + 2\epsilon_2^{(i)})||A||\,||\delta\mathbf{x}^{(i)}|| \\ &+(1 + c_1^{(i)}(1 + q\kappa(A)))\epsilon_2^{(i)}||A||\,||\mathbf{x}|| + O(\epsilon^2) \end{aligned} \tag{B.5}$$

Dividing $||A||\,||\mathbf{x}||$ at the both sides in (B.5) yields:

$$\begin{aligned} &||\mathbf{r}^{*(i+1)}||/||A||\,||\mathbf{x}|| \\ &\leq ((1 + c_1^{(i)})(1 + q\kappa(A))\epsilon_2^{(i)} + q + 2\epsilon_2^{(i)})||\delta\mathbf{x}^{(i)}||/||\mathbf{x}|| \\ &+(1 + c_1^{(i)}(1 + q\kappa(A)))\epsilon_2^{(i)} + O(\epsilon^2) \end{aligned} \tag{B.6}$$

At the $(m + 2)^{\text{th}}$ iteration, the norm bound for the computed residual is:

$$\begin{aligned} &||\mathbf{r}^{*(m+2)}||/(||A||\,||\mathbf{x}||) \\ &\leq ((1 + c_1^{(m+1)})(1 + q\kappa(A))\epsilon_O + q + 2\epsilon_O)||\delta\mathbf{x}^{(m+1)}||/||\mathbf{x}|| \\ &+(1 + c_1^{(m+1)}(1 + q\kappa(A)))\epsilon_O + O(\epsilon^2) \end{aligned} \tag{B.7}$$

Using (A.18), the (B.7) is represented:

$$\begin{aligned} &||\mathbf{r}^{*(m+2)}||/(||A||\,||\mathbf{x}||) \\ &\leq (\mu_1\kappa(A)((1 + c_1^{(m+1)})(1 + q\kappa(A))\epsilon_O + q + 2\epsilon_O) \\ &+(1 + c_1^{(m+1)}(1 + q\kappa(A))))\epsilon_O + O(\epsilon^2) = \hat{\mu}_2\epsilon_O + O(\epsilon^2) \end{aligned} \tag{B.8}$$

where
$\hat{\mu}_2 = \mu_1\kappa(A)((1 + c_1^{(m+1)})(1 + q\kappa(A))\epsilon_O + q + 2\epsilon_O) + (1 + c_1^{(m+1)}(1 + q\kappa(A)))$. Employing the norm inequalities of :

$$\begin{aligned} ||\mathbf{r}^{*(m+2)}||/(||A||\,||\tilde{\mathbf{x}}^{(m+2)}||) &\leq ||\mathbf{r}^{*(m+2)}||/||A||(||\mathbf{x}|| - ||\delta\mathbf{x}^{(m+2)}||) \\ &\leq ||\mathbf{r}^{*(m+2)}||/(||A||\,||\mathbf{x}||(1 - \mu_1\kappa(A)\epsilon_O)) \end{aligned} \tag{B.9}$$

completes the backward error proof:

$$\begin{aligned} &||\mathbf{r}^{*(m+2)}||/(||A||\,||\tilde{\mathbf{x}}^{(m+2)}||) \\ &= ||\mathbf{b} - A\tilde{\mathbf{x}}^{(m+2)}||/(||A||\,||\tilde{\mathbf{x}}^{(m+2)}||) \leq \mu_2\epsilon_O + O(\epsilon^2) \end{aligned} \tag{B.10}$$

where $\mu_2 = \hat{\mu}_2/(1 - \mu_1\kappa(A)\epsilon_O)$. □

Appendix C. Sketch of proof for the required number of iterations for AIR

We prove Numerical Property 3 in this section. For the proof, we first introduce a slack variable, Δq, to indicate the difference from the two forward error convergence rates between XMIR and AIR. Then, we prove that one additional iteration is required by AIR at most compared to XMIR if Δq is small enough (i.e., the two convergence rates between AIR and XMIR are equivalent each other.).

A well-conditioned matrix is defined in this paper if $q\kappa(A) \leq 10$ (or $\kappa(A) \leq \sqrt{10/\epsilon_1}$); From $q\kappa(A) \leq 10$, we can derive $\kappa(A) \leq 10/q_{MAX} \leq 10/q$, where $q_{MAX} = \epsilon_1\kappa(A)$ in this paper. From (A.12), we can describe the accuracy of AIR for forward error at the i^{th} iteration for well-conditioned matrices :

$$\begin{aligned}\|\delta\mathbf{x}^{(i+1)}\|/\|\mathbf{x}\| &\leq (\Pi_{k=1}^{i}\sigma_f^{(k)})q + \Sigma_{k=2}^{i}(\Pi_{l=k}^{i}\sigma_f^{(l)})p^{(k-1)} + p^{(i)} + O(\epsilon^2) \\ &\leq \sigma_{fMAX}^{i}q + p^{(i)}/(1-\sigma_{fMAX}) + O(\epsilon^2)\end{aligned} \tag{C.1}$$

If we let $\sigma_f^{MAX} = q + \Delta q$, $\Delta q/q$ is a very small quantity for well-condition matrices from (A.11):

$$\|\delta\mathbf{x}^{(i+1)}\|/\|\mathbf{x}\| \leq (q+\Delta q)^i q + p^{(i)}/(1-(q+\Delta q)) + O(\epsilon^2) \tag{C.2}$$

Next, the best possible accuracy of XMIR at the i^{th} iteration can be obtained approximately by replacing σ_f^{MAX} to q (i.e., the lowest value) from (C.1).

$$\|\delta\mathbf{x}^{(s+1)}\|/\|\mathbf{x}\| \lesssim q^{i+1} + p^{(i)}/(1-q) + O(\epsilon^2) \tag{C.3}$$

In order to achieve an equivalent accuracy with one additional iteration, the two conditions of $p^{(i+1)}/(1-(q+\Delta q))$ in (C.2) $\leq p^{(i)}/(1-q)$ in (C.3) and $(q+\Delta q)^{(i+1)}q$ in (C.2) $\leq q^{(i+1)}$ in (C.3) should be satisfied. The two conditions are valid with a small $\Delta q/q$. □

Appendix D. Proof for the maximum achievable speedup by AIR over XMIR

We prove Performance Property 2 in this section. For the proof, we first derive the summation rules by exploiting the steady increment of the significand bit-width per iteration according to the number of cancellation bits in step 2. Then, using the summation rules, we prove that the maximum speedup by AIR over XMIR approaches $(p+1)\times$ when the exact (i.e., infinite precision) solution accuracy is required (i.e., $\gamma(= t_1/t_O) \rightarrow 0$.). When $\gamma \rightarrow 0$, the run time of step 1 is theoretically negligible compared to the run time for the refinement procedure from step 2 to 4 based on Eq. (2). Therefore, in order to estimate the maximum speedup by AIR over XMIR, we only compare the refinement procedure time cost of AIR to XMIR when $\gamma \rightarrow 0$. The approximate time cost for the refinement procedure is:

$$T_{AIR-ref} = 2n^2\Sigma_{k=1}^{m_A}T(t_2^{(k)}) + 2n^2 m_A T(t_1) \tag{D.1}$$

where m_A is the required number of iterations for AIR. The time cost is $2n^2\Sigma_{k=1}^{m_A}T(t_2^{(k)})$ for step 2 and $2n^2 m_A T(t_1)$ for step 3. The proof does not consider the final iteration for the accuracy check, since the significand cost for one additional iteration for the accuracy check is negligible when $\gamma \rightarrow \infty$.

Appendix D.1. Linear speedup ($p = 1$)

Using the summation property and Eq. (2),

$$\Sigma_{k=1}^{m_A}T(t_2^{(k)}) = 2t_1 + \Sigma_{k=2}^{m_A}T(t_2^{(k)}) \approx a(m_A t_1 + m_A(m_A+1)t_q/2) \tag{D.2}$$

where t_q represents the approximate number of cancellation bits at step 2 per iteration. The significand bit-width for step 2 per iteration can be represented as $t_2^{(k)} = t_1 + kt_q$; every iteration, the number of cancellation bits, t_q, should be augmented in step 2 for AIR. In (D.2), we assume that $t_2^{(1)} = 2t_1 \approx t_1 + t_q$, since the number of iterations cannot be detected at the initial iteration. Therefore, the total run time for the refinement procedure for AIR is described :

$$T_{AIR-ref} \approx 2n^2 a(2m_A t_1 + m_A(m_A+1)t_q/2) \tag{D.3}$$

Using Eq. (3) for XMIR, the time cost for the refinement procedure for XMIR is:

$$T_{XMIR-ref} \approx m_X 2n^2 a(t_1 + t_O) \tag{D.4}$$

where m_X is the required number of iterations for XMIR. Using $t_O \approx m_A t_q + t_1$ (i.e., $t_q \approx (t_O - t_1)/m_A$), the achievable speedup by AIR over XMIR when $\gamma \rightarrow 0$ is :

$$\begin{aligned}S_{MAX\_A/X}^{(p=1)} &= \lim_{\gamma\rightarrow 0}(T_{XMIR}/T_{AIR}) \\ &= \lim_{\gamma\rightarrow 0}(T_{XMIR-ref}/T_{AIR-ref}) \approx 2m_X/(m_A+1)\end{aligned} \tag{D.5}$$

Using NP3 (i.e, $m_A \lesssim m_X + 1$), the maximum speedup by AIR over XMIR approaches $2\times$, when $p = 1$. □

Appendix D.2. Quadratic speedup ($p = 2$)

From Eq. (D.1), the total run time for the refinement procedure for AIR for $p = 2$ is described:

$$T_{AIR-ref} = 2n^2 a\Sigma_{k=1}^{m_A}(k\cdot t_q + t_1)^2 \tag{D.6}$$

Using the summation property of:

$$\Sigma_{k=1}^{m_A}k^2 = m_A(m_A+1)(2m_A+1)/6 \tag{D.7}$$

Eq.(D.6) is represented:

$$\begin{aligned}&T_{AIR-ref} \\ &= 2n^2 a(m_A(m_A+1)(2m_A+1)t_q^2/6 + m_A(m_A+1)t_q t_1 + m_A t_1^2)\end{aligned} \tag{D.8}$$

If $\gamma \rightarrow 0$ with the approximation of $t_q \approx (t_O - t_1)/m_A$, the total run time for the refinement procedure for AIR is represented :

$$\lim_{\gamma\rightarrow 0}T_{AIR-ref} \approx 2n^2 a(2m_A+1)t_O^2/6 \tag{D.9}$$

Using Eq. (3) for XMIR, the time cost for the refinement procedure for XMIR is:

$$T_{XMIR-ref} \approx m_X 2n^2 a(t_1 + t_O)^2 \tag{D.10}$$

Therefore, the speedup by AIR over XMIR can be represented:

$$S^{(p=2)}_{MAX\_A/X} = \lim_{\gamma\to 0}(T_{XMIR}/T_{AIR}) = \lim_{\gamma\to 0}(T_{XMIR-ref}/T_{AIR-ref}) \approx 6m_X/(2m_A+1) \tag{D.11}$$

Using NP3 (i.e, $m_A \lesssim m_X + 1$), the maximum speedup by AIR over XMIR approaches 3×, when $p = 2$. □

References

[1] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, Inc., New York, NY, USA, 2nd edn., ISBN 0195328485, 9780195328486, 2009.

[2] J. Langou, et al., Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy (Revisiting Iterative Refinement for Linear Systems), in: SC 2006 Conference, Proceedings of the ACM/IEEE, 2006.

[3] E. Carson, N. Higham, Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions, SIAM Journal on Scientific Computing 40 2, 2018, A817–A847.

[4] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep Learning with Limited Numerical Precision, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org, 1737–1746, 2015.

[5] J. Sun, G. D. Peterson, O. O. Storaasli, High-Performance Mixed-Precision Linear Solver for FPGAs, IEEE Transactions on Computers 57 (12) (2008) 1614–1623, doi:10.1109/TC.2008.89.

[6] J. Lee, G. D. Peterson, Iterative Refinement on FPGAs, in: 2011 Symposium on Application Accelerators in High-Performance Computing, ISSN 2166-5133, 8–13, doi:10.1109/SAAHPC.2011.19, 2011.

[7] A. Kiełbasiński, Iterative refinement for linear systems in variable precision arithmetic, BIT 21 (1981) 97–103.

[8] A. Smoktunowicz, J. Sokolnicka, Binary cascades iterative refinement in doubled-mantissa arithmetics, BIT Numerical Mathematics 24 (1) (1984) 123–127, ISSN 1572-9125, doi:10.1007/BF01934524.

[9] J. Lee, AIR: Adaptive Dynamic Precision Iterative Refinement, Ph.D. thesis, University of Tennessee, TN, USA, 2012.

[10] J. A. Goldstein, M. Levy, Linear Algebra and Quantum Chemistry, Am. Math. Monthly 98 (10) (1991) 710–718, ISSN 0002-9890, doi:10.2307/2324422.

[11] A. Edelman, Eigenvalues and condition numbers of random matrices, Ph.D. thesis, M.I.T., MA, USA, 1989.

[12] L. N. Trefethen, Numerical Linear Algebra, SIAM, 1998.

[13] J. Wilkinson, Rounding errors in algebraic processes, Prentice Hall, 1963.

[14] D. Goldberg, What Every Computer Scientist Should Know About Floating-point Arithmetic, ACM Comput. Surv. 23 (1) (1991) 5–48, ISSN 0360-0300, doi:10.1145/103162.103163.

[15] M. Jankowski, H. Woźniakowski, Iterative refinement implies numerical stability, BIT Numerical Mathematics 17 (3).

[16] C. Moler, Iterative Refinement in Floating Point, J. ACM 14 (2) (1967) 316–321, ISSN 0004-5411.

[17] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, E. J. Riedy, Error Bounds from Extra-precise Iterative Refinement, ACM Trans. Math. Softw. 32 (2) (2006) 325–351, ISSN 0098-3500, doi:10.1145/1141885.1141894.

[18] J. Lee, H. Vandierendonck, M. Arif, G. D. Peterson, D. S. Nikolopoulos, Energy-Efficient Iterative Refinement using Dynamic Precision, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 8 (4) (2018) 722–735, ISSN 2156-3357, doi:10.1109/JETCAS.2018.2850665.

[19] D. Tan, A. Danysh, M. Liebelt, Multiple-precision fixed-point vector multiply-accumulator using shared segmentation, in: Proceedings 2003 16th IEEE Symposium on Computer Arithmetic, ISSN 1063-6889, 12–19, doi:10.1109/ARITH.2003.1207655, 2003.

[20] A. Akkas, M. J. Schulte, A quadruple precision and dual double precision floating-point multiplier, in: Euromicro Symposium on Digital System Design, 76–81, doi:10.1109/DSD.2003.1231903, 2003.

[21] A. Isseven, A. Akkas, A Dual-Mode Quadruple Precision Floating-Point Divider, in: 2006 Fortieth Asilomar Conference on Signals, Systems and Computers, ISSN 1058-6393, 1697–1701, doi:10.1109/ACSSC.2006.355050, 2006.

[22] G. Liang, J. Lee, G. D. Peterson, ALU Architecture with Dynamic Precision Support, in: 2012 Symposium on Application Accelerators in High Performance Computing, ISSN 2166-515X, 26–33, doi:10.1109/SAAHPC.2012.29, 2012.

[23] G. Liang, Arithmetic Logic Unit Architectures With Dynamically Defined Precision, Ph.D. thesis, University of Tennessee, TN, USA, 2015.

[24] I. Tsiokanos, L. Mukhanov, G. Karakonstantis, Low-Power Variation-Aware Cores based on Dynamic Data-Dependent Bitwidth Truncation, in: 2019 Design, Automation Test in Europe Conference Exhibition (DATE), ISSN 1530-1591, 698–703, doi:10.23919/DATE.2019.8714942, 2019.

[25] S. Williams, A. Waterman, D. Patterson, Roofline: An Insightful Visual Performance Model for Multicore Architectures, Commun. ACM 52 (4) (2009) 65–76, ISSN 0001-0782, doi:10.1145/1498765.1498785.

[26] B. da Silva, A. Braeken, E. H. D'Hollander, A. Touhafi, Performance Modeling for FPGAs: Extending the Roofline Model with High-level Synthesis Tools, Int. J. Reconfig. Comput. 2013 (2013) 1–10, ISSN 1687-7195, doi:10.1155/2013/428078.

[27] M. Pahlavan Yali, FPGA-Roofline: An Insightful Model for FGPA-based Hardware Acceleration in Modern Embedded Systems, Master's thesis, Virginia Tech, VA, USA, 2015.

[28] J. Lee, Y. Bi, G. D. Peterson, R. J. Hinde, R. J. Harrison, HASPRNG: Hardware Accelerated Scalable Parallel Random Number Generators, Computer Physics Communications 180 (12) (2009) 2574 – 2581, ISSN 0010-4655, doi:https://doi.org/10.1016/j.cpc.2009.07.002.

[29] A. Edelman, Eigenvalues and condition numbers of random matrices, SIAM J. Matrix Anal. Appl. 9 (4) (1988) 543–560, 58854.

[30] N. J. Higham, Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics, 579525, 2002.

[31] U. I. Minhas, S. Bayliss, G. A. Constantinides, GPU vs FPGA: A Comparative Analysis for Non-standard Precision, in: D. Goehringer, M. D. Santambrogio, J. M. P. Cardoso, K. Bertels (Eds.), Reconfigurable Computing: Architectures, Tools, and Applications, Springer International Publishing, Cham, ISBN 978-3-319-05960-0, 298–305, 2014.

[32] D. Bailey, et al., High-precision computation: Mathematical physics and dynamics, Applied Mathematics and Computation 218 (20) (2012) 10106 – 10121, ISSN 0096-3003.

[33] M. Esghir, O. Ibrihich, S. Elgharbi, M. Essaouini, S. El Hajji, Solving large linear systems with multiple right-hand sides, in: 2017 International Conference on Engineering and Technology (ICET), 1–6, doi:10.1109/ICEngTechnol.2017.8308197, 2017.

[34] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019 (Revision of IEEE 754-2008) doi:10.1109/IEEESTD.2019.8766229.