Energy-Efficient Iterative Refinement using Dynamic Precision

JunKyu Lee*, Hans Vandierendonk*, Mahwish Arif*, Gregory D. Peterson†, Dimitrios S. Nikolopoulos*
*The Institute of Electronics, Communications and Information Technology (ECIT),
Queens University Belfast, Northern Ireland, UK
{junkyu.lee, h.vandierendonck, m.arif, d.nikolopoulos}@qub.ac.uk
†Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA
gdp@utk.edu

***Abstract*—Mixed precision is a promising approach to save energy in iterative refinement algorithms since it obtains speed-up without necessitating additional cores and parallelisation. However, conventional mixed precision methods utilise statically defined precision in a loop, thus hindering further speed-up and energy savings. We overcome this problem by proposing novel methods which allow iterative refinement to utilise variable precision arithmetic dynamically in a loop (i.e. a trans-precision approach). Our methods restructure a numeric algorithm dynamically according to runtime numeric behaviour and remove unnecessary accuracy checks. We implemented our methods by extending one conventional mixed precision iterative refinement algorithm on an Intel Xeon E5-2650 2GHz core with MKL 2017 and XBLAS 1.0. Our dynamic precision approach demonstrates 2.0–2.6× speed-up and 1.8–2.4× energy savings compared to mixed precision iterative refinement when double precision solution accuracy is required for forward error and with matrix dimensions ranging from 4K to 32K.**

***Index Terms*—transprecision, dynamic precision, dynamic algorithm, iterative refinement, energy savings.**

I. Introduction

MIXED precision arithmetic is a promising approach to save energy when solving linear systems of equations in the form of A**x**=**b** using iterative refinement. The reason is that mixed precision arithmetic can achieve substantial acceleration of the algorithms without necessitating additional cores for parallelisation. The idea behind mixed precision methods is to utilise lower precision arithmetic for the computationally intensive $O(n^3)$ task that generate LU factors, while attaining good solution accuracy through a higher precision O(n^2) refinement, where n is a matrix size [1]. The mixed precision method of [1] utilises dual precisions (e.g, single and double precision) to achieve speedup without losing accuracy for backward error (see Section II-B for further details). To achieve a working precision (i.e., the precision used for input data A and $\mathbf{b}$) accuracy for forward error, prior work proposed to use three levels of precision (e.g., half precision, single precision, and double precision) for iterative refinement [2]. However, most mixed precision methods (including iterative refinement algorithms using three levels of precision) employ statically defined precision in a loop, which hinders performance improvement.

Recent work has explored automated precision tuning tools [3], [4]. Unfortunately, these tools have not proven practical but several reasons. First, they customise precision tuning for a particular input set. The numerical accuracy of most scientific and machine learning applications depends highly on input data. For example, the accuracy of a linear system solver depends on a condition number of a matrix $\kappa(A)$ and most machine learning applications employ a regularisation parameter to minimise overfitting caused by tuning hyperparameters too much to a training set [5]. Therefore, it would be highly probable that the precision assigned from automated precision tuning tools may not function properly for different input sets. Second, tuned precision assignments do not consider runtime numerical behaviour. For example, it is not feasible to employ dynamic precision arithmetic in a loop with the tools and this limitation causes performance loss. The authors in [3] suggested that domain specific knowledge such as numeric properties of an algorithm might be necessary to deal with the limitations of automatic precision tuning tools.

All related work on mixed precision and automated precision tuning methods employs statically defined precision for statically defined algorithms. That is, the precision or levels of precision used by the algorithm are fixed at compile time and do not vary at runtime. However, many algorithms can be restructured at runtime to improve their performance. For example, conventional iterative refinement refines a computed solution by subtracting an error approximation per iteration. However, it can be adapted to refine both a solution and an error approximation according to runtime numerical behaviours. An alternative computing paradigm that exploits runtime numerical behaviour can optimise the performance of these algorithms and also maximise their potential energy savings.

In this paper we propose dynamic precision techniques which we refer to as *Transprecision Techniques (TTs)*. TTs allow a mixed precision iterative refinement algorithm to utilise dynamically varying precision in a loop (a capability we refer to as TT1), to restructure a numerical algorithm dynamically according to runtime numeric behaviour (a capability we refer to as TT2), and to remove unnecessary accuracy checks (a capability that we refer to as TT3). These capabilities

J. Lee, H. Vandierendonck, M. Arif, and D. S. Nikolopoulos are with Queens University Belfast, Northern Ireland, UK.

G. D. Peterson is with University of Tennessee, TN, USA.

Manuscript received January, 2018; revised .

combined achieve significant performance improvement and energy savings over standard mixed precision iterative refinement algorithms. TTs are precision utilisation techniques that aggressively exploit numerical properties of an algorithm. To validate the proposed TTs, we use a case study with a mixed precision iterative refinement algorithm employing LU decomposition with Partial Pivoting (LUPP) and producing double precision solution accuracy for forward error. To the best of our knowledge this is the first work that explores both dynamic precision arithmetic and dynamic algorithm optimisation to minimise runtime and maximise energy savings.

II. Mixed Precision Iterative Refinement

In this section, we describe a mixed precision iterative refinement algorithm and its basic numeric properties such as successful condition and achievable accuracy. A conventional iterative refinement algorithm to solve $A\mathbf{x} = \mathbf{b}$ is described in Algorithm I. We use ψ to indicate any approximator for a linear system such as a direct solver (e.g., LUPP, QR, and Cholesky), Conjugate Gradient (CG), or Generalized Minimal Residual (GMRES). We also use $\epsilon_{S,D,DD}$ for single, double and double-double precision machine epsilon respectively, $\varepsilon_{S,D,DD}$ for single, double and double-double precision arithmetic respectively and likewise, $\epsilon_{1,2,3,\psi}$ and $\varepsilon_{1,2,3,\psi}$ for a precision machine epsilon and a precision arithmetic applied for step 1, 2, 3 and generating a ψ respectively in this paper. For reference, the numeric analyses for mixed precision iterative refinement are described in Appendix A.

```
Algorithm I

for i = 1, 2, 3,..
  step 1: compute the residual :
      $\mathbf{r}^{(i)} = A\mathbf{x}^{(i)} - \mathbf{b}$ with $\varepsilon_1$
  step 2: seek the approx of the previous solution error :
      $A\mathbf{z}^{(i)} = \mathbf{r}^{(i)}$ using $\psi$ with $\varepsilon_2$
  step 3:  deduct the approximated error :
      $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \mathbf{z}^{(i)}$ with $\varepsilon_3$
```

In step 1, the residual $\mathbf{r}^{(i)}$ is sought as precisely as possible. In step 2, the approximation of the solution error is sought using ψ. In step 3, the approximated error is deducted. Since the approximated error contains the rounding error of the previous computed solution error and the error from the approximator ψ, the residual of the corrected solution is sought again at next step 1. The iterative procedures continues until the accuracy of solution meets prescribed accuracy. Assuming that ψ is a perfect solver and there are no rounding errors for all steps, just one iteration is required to find out the correct solution. The computed solution converges to an exact solution with a convergence rate (i.e., a relative accuracy quality of ψ). For example, if the relative error of a ψ approximator is smaller, the computed solution approaches the exact solution quicker.

We refer to Algorithm I as a mixed precision algorithm if ψ is generated by lower precision arithmetic than a working precision. For example, if A and $\mathbf{b}$ are double precision data, but single precision arithmetic is used for LUPP to generate a ψ, Algorithm I is a mixed precision iterative refinement algorithm. The computational complexity is $O(n^2)$ for steps 1 and 2, $O(n)$ for step 3, and $O(n^3)$ to generate ψ with LUPP. The mixed precision iterative refinement employing LUPP for a ψ minimises runtime by using lower precision arithmetic to generate ψ and higher precision arithmetic to refine the approximated solution.

A. Successful Condition

The mixed precision iterative refinement converges successfully once the relative error of ψ is less than 1 [6]. For example, the relative error of a LUPP approximator is in proportion to $\kappa(A)$ and the precision applied for matrix decomposition [7]. Therefore, the successful condition for the mixed precision iterative refinement employing a LUPP approximator requires the relative error of ψ in step 2 less than unity :

$$||\mathbf{z} - \tilde{\mathbf{z}}||/||\mathbf{z}|| \leq q_\psi := c_1\kappa(A)\epsilon_\psi < 1 \quad (1)$$

where $\tilde{\mathbf{z}}$ for the computed result of $\mathbf{z}$ and c_1 is a bounded constant depending on a matrix size, where $k = 1, 2, 3, ...$ and $\|\cdot\|$ is an infinity norm, which is the maximum absolute vector component value [7]. Based on Eq. (1), using a LUPP approximator for an ill-conditioned system requires higher precision arithmetic for the matrix decomposition to make the mixed precision iterative refinement converge successfully. Therefore, ε_ψ should be determined according to the condition number of a matrix. (i.e., $\varepsilon_\psi < 1/(c_1\kappa(A).)$)

B. Achievable Accuracies

There are two types of accuracies for linear system solvers: the accuracy for forward error and the accuracy for backward error [7]. Some applications, e.g. the GPS application [8] focus on forward error, while others such as linear regression (e.g., an interpolation problem) focus on backward error. The forward error indicates the error in the computed solution $\tilde{\mathbf{x}}$ caused by finite precision arithmetic (i.e., $||\mathbf{x} - \tilde{\mathbf{x}}||/||\mathbf{x}||$) while the backward error indicates the smallest η satisfying both $\eta \leq ||\Delta A||/||A||$ and $\eta \leq ||\delta\mathbf{b}||/||\mathbf{b}||$ to make $\tilde{\mathbf{x}} = (A + \Delta A)^{-1}(\mathbf{b} + \delta\mathbf{b})$ hold. (i.e., the computed solution $\tilde{\mathbf{x}}$ is the exact solution $\mathbf{y}$ for a perturbed system $(A + \Delta A)\mathbf{y} = \mathbf{b} + \delta\mathbf{b}$.)

An algorithm is called *backward stable* if it produces a small backward error for *any input data* (e.g., A and $\mathbf{b}$). The definition of "small" depends on circumstances of the problem that the algorithm solves. There is a rule of thumb for a backward stable algorithm [7]:

$$\textit{forward error} \lesssim \kappa(A) \cdot \textit{backward error} \quad (2)$$

Based on Eq. (2), backward stability is an important concept for numerical linear algebra, since it can tell if a computed solution is reliable for either forward or backward error. If an algorithm is backward stable, the algorithm is also "*forward stable*" because a forward error is bounded at most by a condition number times a backward error. If an algorithm produces a forward error as small as an backward stable algorithm, the algorithm is called forward stable. Therefore,

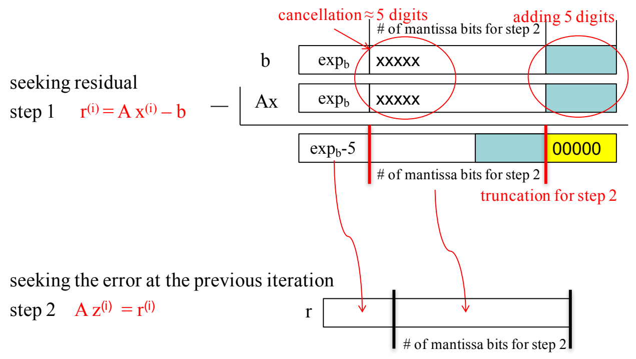

Fig. 1. Numeric Property of Cancellation Error

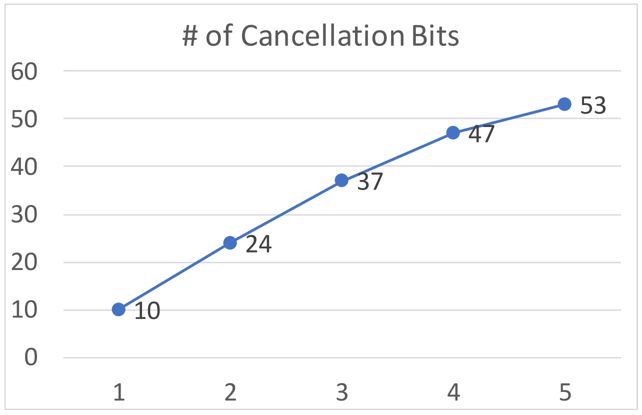

Fig. 2. Number of Bits for Cancellation Error for One Component of the Residual According to the Number of Iterations [10]

a backward stable algorithm always implies a forward stable algorithm, but not the other way round [7].

A conventional mixed precision iterative refinement is backward stable (conseqently, also forward stable) since it produces $c_2\kappa(A)\epsilon_1$ for forward error and $c_3\epsilon_1$ for backward error as long as the successful condition is met and $\epsilon_3 = \epsilon_1$ [6]. Therefore, satisfying the condition of Eq. (1), a mixed precision iterative refinement can achieve double precision solution accuracy for forward error by employing $\epsilon_1 \lesssim \epsilon_D/\kappa(A)$ and $\epsilon_3 = \epsilon_1$.

An approximated intermidiate accuracy can be measured either for forward error or backward error per iteration. An intermediate accuacy for forward error can be approximated by measuring $||\mathbf{z}^{(i)}||/||\mathbf{x}^{(i)}||$ right after step 2 in Algorithm I [9] and a backward error can be approximated by measuring $||\mathbf{r}^{(i)}||/(||A||\cdot||\mathbf{x}^{(i)}||+||\mathbf{b}||)$ right after step 1 [7].

III. Numerical Properties and Transprecision Techniques

Exploiting numerical properties aggressively in terms of precision utilisation can minimise overall runtime for solving linear systems with iterative refinement. In this section, we describe three numerical properties (NPs) exploitable by TTs. The three TTs are integrated into our baseline mixed precision iterative refinement in section IV. NP 1 was presented previously in [10] and in this work, we present two additional NPs to minimise runtime and energy consumption further.

A. Numeric Property 1 and Transprecision Technique 1

Iterative refinement minimises a solution error gradually per iteration in proportion to a convergence rate. The convergence rate is directly related to the increased number of cancellation bits per iteration as shown in Fig. 1. In Fig. 1, if we add the mantissa bits for step 1 as many as the number of cancellation bits, there does not exist truncation error, since the truncated bits are represented as all 0s after shift left operations according to the resultant exponent. Therefore, adding as many bits as the number of cancellation bits leaves the resultant residual value for step 2 unaffected by truncation after a shift left operation.

Numeric Property of Cancellation Error
NP1: Adding as many as cancellation bits per iteration in step 1 does not affect the quality of the resultant residual.

Figure 2 describes the numbers of cancellation bits according to the number of iterations i at the x axis for a random matrix of $n = 1K$. A residual component is used to measure the number of cancellation bits. The single precision arithmetic is used for step 2 (i.e., precision to generate ψ) and double-double precision arithmetic for steps 1 and 3. Please refer to Appendix B for the information of the measurements for the numbers of cancellation bits. The number of component wise cancellation bits increases in proportion to the number of iterations. At the 5^{th} iteration, the number of cancellation bits exceeds the mantissa width of double precision arithmetic. This breaks NP1 when double precision arithmetic is employed for step 1. The NP1 is preserved as long as the mantissa bit width of a precision arithmetic used for step 1 is equal or higher than the number of cancellation bits.

If NP1 is exploited for FPGAs, an arbitrary precision can be chosen every iteration according to the number of cancellation bits per iteration [10]. In [10], it was not recommended to support arbitrary precisions per iteration based on the numbers of component-wise cancellation bits, since arithmetic circuit supporting n types of precisions according to n components may cause overhead. Therefore, it was recommended to choose one representative precision arithmetic per iteration for FPGAs by utilising "the number of norm-wise cancellation bits (e.g., $\approx log_2(\|\mathbf{b}^{(i)}\|/\|\mathbf{r}^{(i)}\|)$)" instead of the number of component-wise cancellation bits (e.g., $\approx log_2(|b_j^{(i)}|/|r_j^{(i)}|)$).

Exploiting NP1 generates TT1 as follows:

TT1: Start with a lower precision arithmetic for step 1 and switch to a higher precision arithmetic when the convergence is saturated.

B. Numeric Property 2 and Transprecision Technique 2

The accuracy for $\mathbf{z}^{(i)}$ in step 2 depends on the quality of the approximator ψ (i.e., q_ψ) and can be improved by using an inner-loop iterative refinement. An iterative refinement with an inner loop is described in Algorithm II.

```
Algorithm II

for i = 1, 2, 3,..
 step 1: compute the residual :
      $\mathbf{r}^{(i)} = A\mathbf{x}^{(i)} - \mathbf{b}$
 step 2: seek the approx of previous solution error :
      $A\mathbf{z}^{(i)} = \mathbf{r}^{(i)}$ using $\psi$ ,
       for j = 1, 2, 3, ..
         $\mathbf{d}^{(1)} = \mathbf{z}^{(i)}$
         sub-step 1: $\mathbf{r}_{in}^{(j)} = A\mathbf{d}^{(j)} - \mathbf{r}^{(i)}$
         sub-step 2: $A\mathbf{z}_{in}^{(j)} = \mathbf{r}_{in}^{(j)}$ using $\psi$ ,
           if($\mathbf{z}_{in}^{(j)}$ is small enough) $\mathbf{z}^{(i)} = \mathbf{d}^{(j)}$; exit;
         sub-step 3: $\mathbf{d}^{(j+1)} = \mathbf{d}^{(j)} - \mathbf{z}_{in}^{(j)}$
 step 3: deduct the approximated error :
      $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \mathbf{z}^{(i)}$
```

The basic structure of Algorithm II is the same as Algorithm I, except Algorithm II refines the approximated error $\mathbf{z}$ prior to step 3. We notice a trade-off between the accuracy of $\mathbf{z}^{(i)}$ and overall runtime. For example, an inner loop refinement can improve the convergence rate of iterative refinement with an extra time cost of inner loop iterations. In order to minimise the overall runtime using an inner-loop refinement, we employ it when the following two conditions are met.

First, consider an inner loop refinement if the rounding error in step 1 $||\delta\mathbf{r}^{(i)}||$ is relatively small compared to the accuracy of ψ. In this case, a convergence rate mainly depends on the accuracy of ψ, so it would be worth improving the convergence rate by employing an inner loop refinement. In other words, it is not worth seeking an inaccurate residual accurately. As discussed in Eq. (6) in Appendix A, $||\delta\mathbf{r}^{(i)}||$ is proportional to $||\mathbf{r}^{(i)}||$ and ϵ_1. Therefore, it would be desirable to employ an inner loop refinement dynamically when $||\mathbf{r}^{(i)}||$ becomes small enough. For the detailed information of $||\delta\mathbf{r}^{(i)}||$, please refer to Appendix A.

Second, improving the convergence rate by using an inner loop is worth it if it contributes to minimising overall runtime. Employing an inner loop refinement minimises the number of outer-loop iterations by improving the convergence rate, but the time cost per outer loop iteration increases due to the additional time cost of inner-loop iterations. Please refer to Appendix C for the trade-off analyses between the improved convergence rate and extra time cost for inner-loop iterations. We propose NP2 exploitable by TT2 as follows:

Numeric Property of Residual Accuracy
NP2: Exact arithmetic in step 2 yields $\mathbf{z}^{(i)} = A^{-1}\tilde{\mathbf{r}}^{(i)} = A^{-1}(\mathbf{r}^{(i)} + \delta\mathbf{r}^{(i)})$. Therefore, $||A^{-1}\delta\mathbf{r}^{(i)}||$ is an irreducible error quantity in step 2.

Exploiting NP2 generates TT 2 as follows:

TT 2: Refine $\mathbf{z}^{(i)}$ with an inner-loop refinement using the least sufficient precision satisfying $\epsilon_{sub1} < \epsilon_2^2$ for sub-step 1 when $\epsilon_1^{(i)} << \epsilon_2$ and $T(\varepsilon_1^{(i)}) >> T(\varepsilon_{sub1})$, where $T(\varepsilon)$ is the theoretical time cost for an arithmetic operation depending on a precision arithmetic ε.

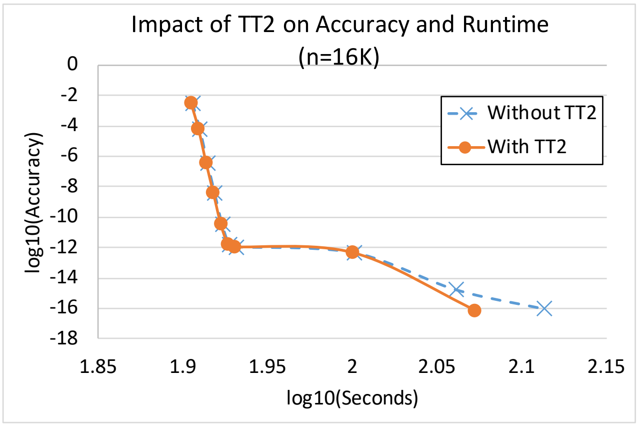

Fig. 3. Impact of Nested Loop Refinement on Accuracy and Runtime

Figure 3 represents the impact of nested loop refinement in terms of accuracy and runtime by TT2 for a matrix of $n = 16K$. The TT1 is applied in Fig 3. The dashed line represents the accuracies according to runtimes without TT2 and the solid line with TT2. Notice that the two iterative refinements are identical upto 100 seconds, thereafter TT2 becomes active for one of the two iterative refinements. The TT2 improves convergence rate by refining $\mathbf{z}^{(i)}$ with some minor time cost due to the nested loop refinement in the figure.

C. *Numeric Property 3 and Transprecision Technique 3*

We propose NP3 exploitable by TT3 as follows:

Numeric Property of Final Accuracy Guarantee
NP3: If a ψ is a backward stable algorithm and lets a mixed precision iterative refinement converge, the maximum allowable condition number is bounded to a positive constant times the reciprocal of the precision applied to generate ψ. (e.g., $\kappa(A) < (c_2\epsilon_\psi)^{-1}$: refer to Eq. (1))

NP3 can lead to guaranteed accuracy of the mixed precision iterative refinement at a certain iteration as long as successful convergence occurs during runtime. Therefore, NP3 can remove an unnecessary accuracy check for the mixed precision iterative refinement.

Exploiting NP3 generates TT3 as follows:

TT3: If ψ is a LUPP approximator using single precision for the matrix decomposition, $\epsilon_1^{(1 \text{ to } (s-1))} = \epsilon_D$, and $\epsilon_1^{(s)} = \epsilon_{DD}$ by TT1, and single precision accuracy of $\mathbf{z}^{(s)}$ is achieved through an inner-loop refinement by TT2 with $\epsilon_{sub1} = \epsilon_D$, then double precision solution accuracy for $\mathbf{x}$ is guaranteed in step 3 at the s^{th} iteration. Therefore, we can skip the final accuracy check if the above conditions are met. This can save the time cost of one double-double precision matrix-vector multiplication.

Please refer to Appendix D for numeric accuracy proof for TT3. Although we exemplify TT3 for a particular case using a LUPP approximator for a ψ, TT3 can be used for other backward stable ψs. Our proof employed NP3 twice at Eq. (16) and Eq. (18) in Appendix D, where $\kappa(A)$s appears. Therefore, NP3 can also be exploited where $\kappa(A)$s appears in any numeric analysis for iterative refinements satisfying Eq. (1).

IV. A Case Study

In this section, we discuss TTs with a case study of the mixed precision iterative refinement producing *double precision accuracy for forward error* with single precision LUPP approximator. The mixed precision iterative refinement used for the case study utilises precision arithmetic with $\epsilon_{1,3} = \epsilon_{DD}$ and $\epsilon_2 = \epsilon_S$ and is implemented on an Intel Xeon E5-2650 2GHz core with MKL 2017 and XBLAS 1.0 [11]. All iterative refinements are single thread implementations. TTs are integrated into the mixed precision iterative refinement as shown in Algorithm III (Transprecision iterative refinement).

Transprecision iterative refinement algorithm for double precision accuracy for forward error is described in Algorithm III. By TT1, double precision arithmetic is initially used for step 1 (i.e., $\epsilon_1 = \epsilon_D$ in Algorithm III) until convergence saturates (i.e., $||\mathbf{z}^{(i)}||/||\mathbf{z}^{(i-1)}|| > 1/2$). If convergence saturates and the solution achieves a desirable intermediate accuracy, the precision arithmetic for step 1 is switched to double-double precision arithmetic (i.e., $\epsilon_1 = \epsilon_{DD}$). If double-double arithmetic is used for step 1 and the latency gab between double-double precision arithemtic and sub-step1 arithmetic is larger than p, the inner-loop refinement is activated to refine the error from the previous solution until single precision accuracy is achieved. Finally, the refined error is deducted from the solution $\mathbf{x}^{(i)}$ and this update guarantees double precision solution accuracy by TT3. Therefore, the program terminates without final accuracy check.

We employed uniformly distributed dense matrices for tests and took averages of 10 test cases for both the runtime and the energy consumption measurements. The test matrices were generated using the drand48() functions in C. Therefore, the elements of the test matrices were uniformly distributed over the interval [0.0, 1.0). The distribution of $\kappa(A)$s for the standard normal distributed matrices were explored in [12]. Based on the distribution results, the distribution of the condition numbers of the uniformly distributed matrices for a large n was predicted in [12] :

$$\lim_{n\to\infty} P(\frac{2}{\sqrt{3}}\kappa(A)/n^{1.5} < x) = e^{-\frac{2}{x}(1+\frac{1}{x})} \tag{3}$$

For example, if $x = 1$, the probability of a scaled condition number (i.e., $\frac{2}{\sqrt{3}}\kappa(A)/n^{1.5}$) being less than 1 is around 2% and if $x = 100$, the probability of a scaled condition number being less than 100 is around 98%.

Algorithm III: Transprecision Iterative Refinement : Double Precision Accuracy for Forward Error

```
v = 0; //success exit from an inner loop
for i = 1, 2, 3, ...
 step 1: $\mathbf{r}^{(i)} = A\mathbf{x}^{(i)} - \mathbf{b}$        (TT1) $\epsilon_1 = \epsilon_D$ to $\epsilon_{DD}$
 step 2: $A\mathbf{z}^{(i)} = \mathbf{r}^{(i)}$ using $\psi$           $\epsilon_2 = \epsilon_S$

 (Opt) Accuracy Chk: if($||\mathbf{z}^{(i)}||/||\mathbf{x}^{(i)}|| < \epsilon_D$ and $i > 1$) exit(success);

 (TT2) if(T($\epsilon_1$)>p·T($\epsilon_{sub-step1}$) and $\epsilon_1 == \epsilon_{DD}$) {
 for j = 1, 2, 3, ...
   $\mathbf{d}^{(1)} = \mathbf{z}^{(i)}$
   sub-step 1: $\mathbf{r}_{in}^{(j)} = A\mathbf{d}^{(j)} - \mathbf{r}^{(i)}$       $\epsilon_{sub1} = \epsilon_D$
   sub-step 2: $A\mathbf{z}_{in}^{(j)} = \mathbf{r}_{in}^{(j)}$ using $\psi$      $\epsilon_{sub2} = \epsilon_S$
     if($||\mathbf{z}_{in}^{(j)}||/||\mathbf{d}^{(j)}|| < \epsilon_S$) { $\mathbf{z}^{(i)} = \mathbf{d}^{(j)}$; v=1; exit(for);}
   sub-step 3: $\mathbf{d}^{(j+1)} = \mathbf{d}^{(j)} - \mathbf{z}_{in}^{(j)}$       $\epsilon_{sub3} = \epsilon_D$
   }

   if($||\mathbf{z}^{(i)}||/||\mathbf{z}^{(i-1)}|| > 1/2$ and $i > 1$) {
     (TT1) if($||\mathbf{z}^{(i)}||/||\mathbf{x}^{(i)}|| < \epsilon_D/\epsilon_S$) $\epsilon_1 = \epsilon_{DD}$;
     else exit(failure); }

 step 3: $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \mathbf{z}^{(i)}$            $\epsilon_3 = \epsilon_{DD}$

 (TT3) if($\epsilon_1 == \epsilon_{DD}$ and $v == 1$) exit(success);
```

In this case study, we refer to an iterative refinement as Uni-precision Iterative Refinement (Uni-IR) if it employs double precision arithmetic (i.e., the same precision to the data) for a LUPP approximator and double-double precision for residual calculation, Mixed precision Iterative Refinement (Mixed-IR) if it employs single precision arithmetic for LUPP approximator and double-double precision for residual calculation, and Transprecision Iterative Refinement (Trans-IR) for a Mixed-IR integrated with TTs. Notice that Uni-IR employs double precision for step 2, since ψ for Uni-IR is generated by double precision matrix decomposition. Hence, Uni-IR requires a larger storage than either Mixed-IR or Trans-IR to store double precision L and U matrices instead of single precision. The p in Algorithm III is an empirical parameter considering the trade-off between an improved convergence rate by an inner loop and an increased time cost per outer loop iteration. Please refer to Appendix C for the analyses for the choice of p. We currently set p = 10. For example, $T(\epsilon_{DD}) \approx 15 \cdot T(\epsilon_D)$ for step 1 in our case study. It takes around 3.9 seconds for a double precision arithmetic step 1 and 60 seconds for a double-double precision arithmetic step 1 for $n = 32K$ (refer to Fig. 12.)

We measure the required number of iterations, accuracy, runtime and energy consumption for Uni-IR, Mixed-IR, and Trans-IR respectively under the same accuracy requirement of double precision accuracy for forward error when the matrix sizes range from 4K to 32K. The $\mathbf{x}^{(1)}$ in Algorithm III is a computed solution using a LUPP approximator ψ for $A\mathbf{x} = \mathbf{b}$.

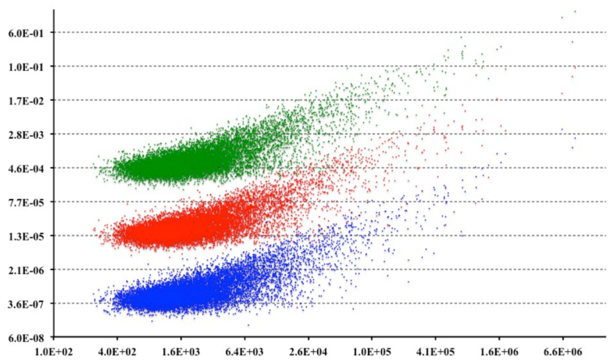

Fig. 4. Accuracies for ψ according to $\kappa(A)$ [10]

A. *Precision choices for ψ generation*

As discussed in section I, the accuracy of a linear solver ψ depends on $\kappa(A)$ and ϵ_ψ. The accuracies of LUPP ψs were explored in [10] as described in Fig. 4. The LUPP ψs were generated for 10,000 64x64 normal distributed matrices each with ϵ_ψ = 2^{-16}, 2^{-21} and 2^{-26}. Fig. 4 shows the relative accuracies of ψs (i.e., q_ψ in Eq (1)) according to various ϵ_ψs. The green, red and blue dots represent the accuracies of ψs for $\epsilon_\psi = 2^{-16}$, 2^{-21} and 2^{-26} respectively. The x axis represents the condition numbers of matrices and the y axis the accuracies of ϵ_ψs. The accuracies of ψs of $\epsilon_\psi = 2^{-26}$ is 2^5 times better than ψs of $\epsilon_\psi = 2^{-21}$. Based on Fig. 4, the accuracies have a linear relation with ϵ_ψs. The accuracies also linearly depend on the $\kappa(A)$s. A precision can be chosen as long as the relative accuracy is less than 1. For example, in Fig. 4, $\epsilon_\psi = 2^{-16}$ can be chosen for matrices of $\kappa(A) \leq 10^5$ for $7.7 \times 10^{-5} - 10^{-1}$ accuracies (i.e., convergence rates). An ϵ_ψ is generally recommended to be lower than a reciprocoal of condition number (i.e., $\epsilon_\psi < 1/\kappa(A)$) [1], [2].

B. *Number of Iterations*

The number of iterations taken for an iterative refinement directly depends on the convergence rate which mainly depends on the quality of ψ. (i.e., q_ψ in Eq. (4)) The LUPP approximator generated by double precision matrix decomposition has a much lower value for q_ψ (i.e., a better convergence rate) compared to the one with single precision matrix decomposition based on Eq. (1).

Figure 5 shows the number of iterations taken for the iterative refinements with respect to various matrix sizes. Notice that Trans-IR requires inner-loop iterations as shown in Fig. 6. As discussed, Uni-IR requires the least number of iterations. However, notice that it takes much longer to generate a ψ for Uni-IR than Mixed-IR (or Trans-IR) due to a higher precision arithmetic for $O(n^3)$ matrix decomposition. Trans-IR generally requires a few of additional outer-loop iterations compared to Mixed-IR (except $n = 32K$) due to supporting less number of bits than the number of cancellation bits in Fig. 1 when convergence becomes saturated and detecting convergence saturation prior to switching to a higher precision for step 1. Trans-IR also requires additional iterations for

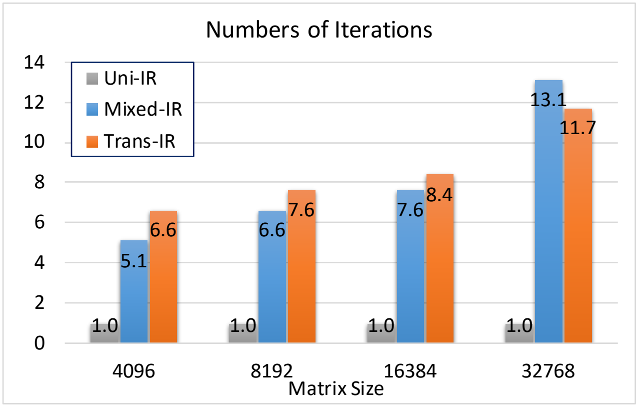

Fig. 5. Number of Out-loop Iterations according to Various Iterative Refinements

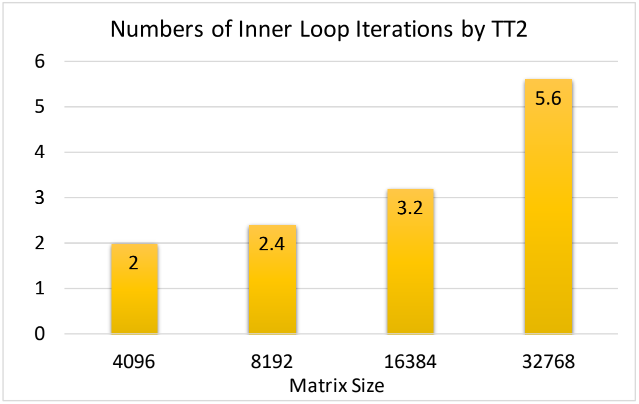

Fig. 6. Number of Nested-Loop Iterations for Trans-IR by TT2

an inner-loop refinement by TT2 as shown in Fig. 6. TT2 improves the convergence rate with an additional time cost of the inner loop. An improved convergence rate by TT2 can minimise the number of outer-loop iterations as shown in $n = 32K$ in Fig. 5. In this case study, TT2 restricts the number of double-double precision arithmetic step 1 to 1 with using TT3. Therefore, the overall runtime for a linear solver with Mixed-IR is significantly reduced by minimising software-emulated arithmetic with TTs as discussed in the next section.

C. *Runtime*

The overall runtime for solving a linear system consists of two parts: the runtime for generating an approximator ψ and the runtime taken for refinement. Figure 7 represents the runtimes for LUPP approximator generation according to various matrix sizes. Both Mixed-IR and Trans-IR minimize the runtimes to generate approximators by employing single precision arithmetic.

Figure 8 shows the overall runtimes (including LU) for the iterative refinements with respect to matrix sizes. First, Trans-IR always shows the shortest runtimes compared to the other two iterative refinements for all matrix sizes. Trans-IR also requires only once for double-double precision arithmetic step 1 as equal as Uni-IR, while the time cost for generating a ψ with single precision matrix decomposition for Trans-IR is

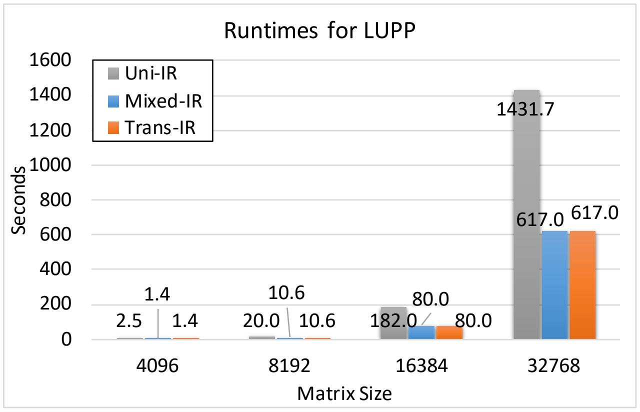

Fig. 7. Runtimes for LUPP Approximator Generation

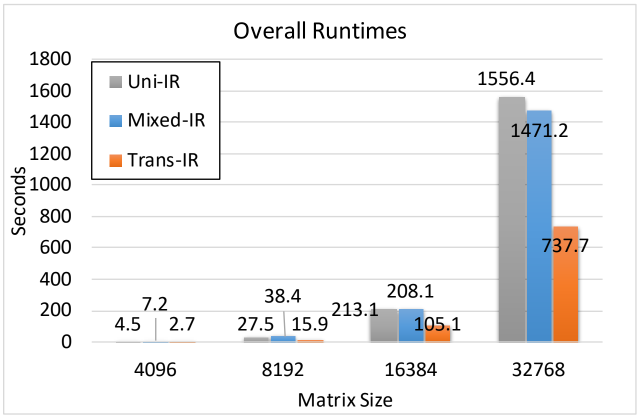

Fig. 8. Overall Runtimes for Various Iterative Refinements

much less than double precision matrix decomposition for Uni-IR. Second, Mixed-IR runtime becomes less than Uni-IR when N=16K thanks to the reduced runtime portion of refinement in proportion to a matrix size. When a matrix size increases, the runtimes for both Mixed-IR and Trans-IR approach to the runtime for generating a LUPP approximator, decreasing the runtime portion of refinement as long as successful condition of Eq. (1) is met.

D. *Refinement Portion of Overall Runtime*

As a matrix size increases, the time cost to generate a ψ becomes dominant, since it requires $O(n^3)$ computation while refinement requires $O(n^2)$. Figure 9 shows the refinement portions of the overall runtimes for the iterative refinements with respect to the matrix sizes. As discussed, the refinement runtime portions decrease for all iterative refinements in proportion to matrix size. This pattern appears since generating a ψ requires $O(n^3)$ computation while refinement requires $O(n^2)$. If generating a ψ requires less or equal to $O(n^2)$ computation (e.g, iterative methods such as GMRES and CG), this pattern is unlikely.

TTs are universally beneficial independent of a ψ, because TTs can minimize the refinement runtime portions given a ψ. In Fig. 9, Trans-IR shows less refinement runtime portions than Mixed-IR given the same ψ as Mixed-IR. It is the reduced

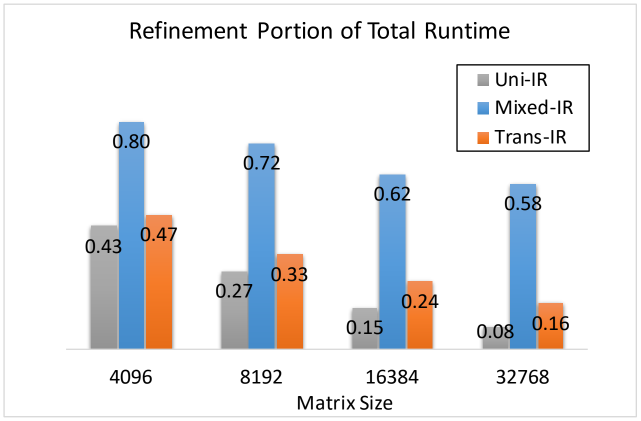

Fig. 9. Refinement Portion of Overall Runtime

runtime for refinement by TTs that produces the speed-ups over Mixed-IR. The software emulated precision arithmetic (double-double precision) for step 1 causes larger runtime portions for refinements for Mixed-IR in Fig. 9. Uni-IR has the least refinement runtime portions because Uni-IR employs a double precision LUPP approximator.

E. *Energy Consumption Estimations*

The minimized runtime by TTs brings further energy reduction over Mixed-IR. For energy consumption estimations, we used the ALEA tool employing constant power model which profiles with Intel Running Average Power Limit (RAPL) and then estimates total energy consumption for the processor [15], [16]. ALEA provides user interface for RAPL to measure fine-grained code blocks by energy profiling. To use ALEA for energy estimation, we first set up the environmental variables for ALEA such as the number of sockets and the number of cores per socket. In our setting, we use 2 for the number of sockets and 8 for the number cores per socket for an Intel E2650, even though we use 1 core out of 16 cores on an Intel E2650 for computation. Once the environmental variables are set, ALEA profiles power consumption for code blocks. Based on profiling information, ALEA builds energy estimation for each code block. We estimate energy consumption using ALEA by running the 10 test cases used for runtime measurements per each size and take the average of them

Figure 10 shows the energy consumption estimations for the iterative refinements measured by the ALEA tool. Energy savings are proportional to time savings. Almost all computing kernels occupying more than 99% of energy require the power ranged from 20 to 25 Watts. Therefore, the energy reduction mainly comes from the minimized runtime by TTs. As expected, *Trans-IR shows the least energy consumption compared to the two other iterative refinements for all matrix sizes thanks to the minimised runtimes by TTs.*

F. *Impact of Individual Transprecision Techniques on Accuracy, Runtime, Energy Consumption and Convergence Rate*

The accuracy of a computed solution becomes higher per iteration according to the convergence rate of a ψ. Figure

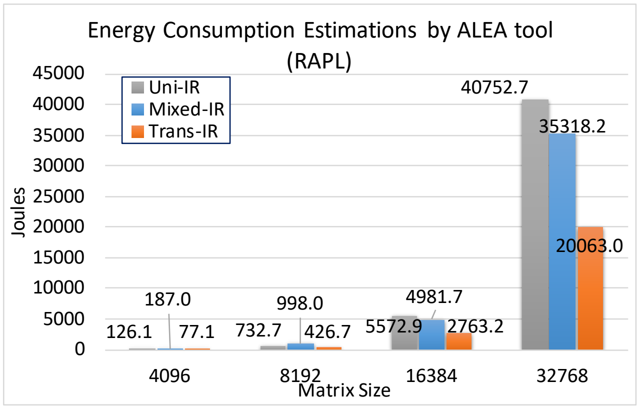

Fig. 10. Energy Consumption Estimations

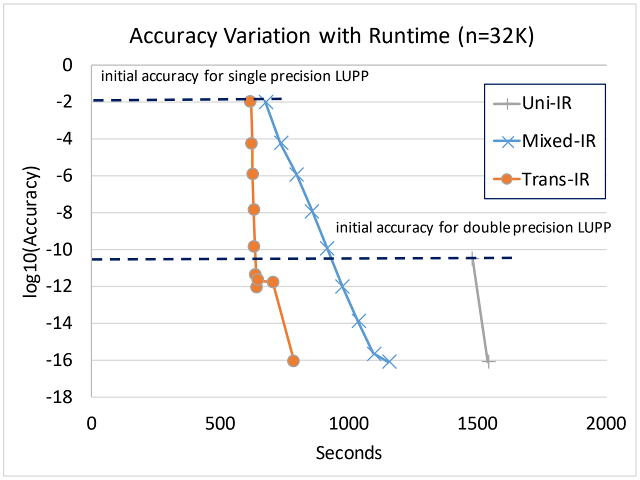

Fig. 11. Accuracy Variation with Runtime

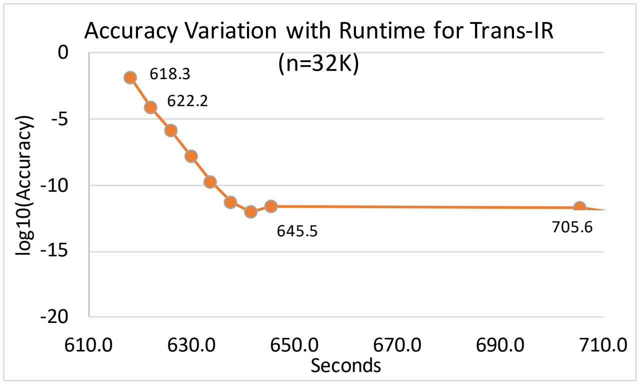

Fig. 12. Accuracy Variation with Runtime for Trans-IR

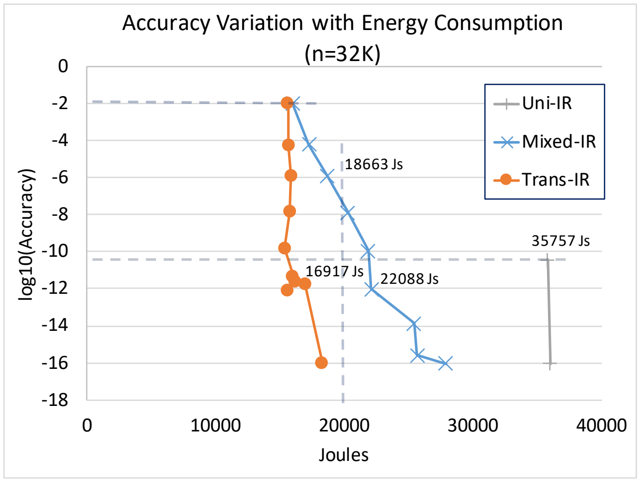

Fig. 13. Accuracy Variation with Energy Consumption

11 shows the accuracy variation with respect to the runtimes for the iterative refinements for a matrix of n=32K and the detailed information of Trans-IR is described in Figure 12. Figure 13 describes the accuracy variation according to energy consumption.

The horizontal axes represent the runtimes in Fig. 11 and energy consumption estimations in Fig. 13. The vertical axes represent the log_{10} based relative errors in the solutions for both Fig. 11 and Fig. 13. Each marker represents an accuracy of a solution per iteration and it is measured by calculating $||\mathbf{z}^{(i)}||/||\mathbf{x}^{(i)}||$ at the (Opt) Accuracy Chk in Algorithm III. Therefore, the runtime and the energy for generating a ψ for Mixed- and Trans-IR can be roughly estimated by the runtime and the energy at the top most mark of Trans-IR in Fig. 11 and 13. The runtime and the energy for generating a ψ for Uni-IR can be roughly estimated by the runtime at the top most mark of Uni-IR. Notice that the characteristic curves in Fig. 11 and 13 depend on a $\kappa(A)$. For example, for a smaller $\kappa(A)$, the accuracy gap between the two adjacent marks in the curve becomes larger, implying a better convergence rate.

TT1 allows Mixed-IR to achieve an intermediate accuracy faster. The “vertical” accuracy and energy variations for Trans-IR in Fig. 11 and 13 indicate minimised runtime and energy per iteration by TT1. In Fig. 12, double precision arithmetic is applied for steps 1 and 3 until the convergence rate is saturated at around 10^{-12} accuracy line. The saturation is detected at 645.5 seconds and double-double precision arithmetic is applied at next step 1 to improve a residual accuracy. The measured accuracies at 645.5 seconds and 705.6 seconds are equivalent, since the error vector (e.g., $||\mathbf{z}||$) sought from a highly accurate residual is not deducted yet at the measured point (e.g, (Opt) in Algorithm III). For around 10^{-12} accuracy, Trans-IR requires 705.6 while Mixed-IR 975.3 secs, deducting around 270 secs. Likewise, Trans-IR requires 16,917.2 Joules, while Mixed-IR 22,088.7 Joules, deducting around 5,172 Joules in Fig. 13.

TT2 initiates refinement of the error vector $\mathbf{z}$ at around 618.3 seconds in Fig. 12 to seek a high accuracy of the previous solution error. TT2 enables the accuracy to leap from 10^{-12} to 10^{-16} thanks to the improved convergence rate by refining an error vector $\mathbf{z}$ in Figs 11 and 13. As a result, TT2 saves around 120 seconds and around $2,980$ Joules (e.g., 1,490 Joules required per iteration on average) by removing the two double-double precision residual calculations compared Mixed-IR. Mixed-IR requires three further iterations from 10^{-12} accuracy, two for refining the solution and one for the final accuracy check.

TT3 can remove a double-double precision residual calculation for an unnecessary final accuracy check. In Fig. 11 and Fig. 13, the two double-double precision residual calculations appear for Trans-IR at "horizontal" runtime movement around the 10^{-12} accuracy line for a necessary residual calculation and at the accuracy variation from 10^{-12} to 10^{-16} due to an unnecessary accuracy check. TT3 affirms the double precision accuracy without an accuracy check and removes the time cost for the second double-double precision residual calculation. Consequently TT3 can save around 60 seconds and $1,490$ Joules (e.g., in Fig 12, the time cost for double-double precision arithmetic for step 1 is: $705.6 - 645.5 \approx 60$). The final accuracy of Trans-IR in Fig. 11 and Fig. 13 empirically supports the numeric proof for TT 3 in Section II-B. Therefore, the overall runtime of Mixed-IR is deducted by ~23% ($\approx 270/1156$) by TT1, ~10% by TT2, ~5% by TT3 for the matrix $n = 32K$ in Fig 11. Consequently, TTs bring ~38% deduction for overall runtime to Mixed-IR in this case study. Likewise, the overall energy consumption of Mixed-IR is deducted by ~19% ($\approx 5,172/27,890$) by TT1, ~11% by TT2, ~5% by TT3 for the matrix $n = 32K$ in Fig 13. TTs bring ~35% deduction for overall energy consumption to Mixed-IR in this case study.

The overall convergence rate mainly depends of q_ψ in Eq. (1). The convergence rate of either Trans-IR or Mixed-IR is around 10^{-2}, which also corresponds to the initial accuracy by the ψ (i.e., the first convergence rate). Based on Eq. (1), the convergence rate of Uni-IR should be higher by a factor of 10^{-8} ($\approx \frac{q_{Uni-IR}}{q_{Mixed-IR}}$) compared to either Mixed-IR or Trans-IR and this is shown by $\frac{\textit{initial accuracy for double precision LUPP}}{\textit{initial accuracy for single precision LUPP}}$ in the figure. Therefore, the convergence rate of Uni-IR is superior, since it directly depends on ϵ_ψ in Eq. (1) [14].

For a larger matrix size, the runtime gab between single and double precision LU factors generation will become larger in Fig. 11 and Fig. 13 due to $O(n^3)$ computation, implying overall runtime and energy of Mixed-IR less than Uni-IR. For a smaller matrix size, the gab becomes smaller and the overall runtime and energy of Uni-IR can be less than Mixed-IR. Trans-IR will show the least runtime and energy regardless of a matrix size, as long as the successful condition of Eq. (1) is met.

Given an energy budget, Trans-IR produces the best accuracy of the solution compared to other IRs. For example, given 18,633 Joules in Fig. 13, Trans-IR produces 10^{-16} solution accuracy while Mixed-IR produces 10^{-6} accuracy and Uni-IR does not produce any meaningful accuracy yet. Likewise, given a prescribed accuracy, Trans-IR requires the minimum energy budget. For example, for an initial accuracy for double precision LUPP ψ, Trans-IR requires 16,917 Joules, Mixed-IR 22,088 Joules, and Uni-IR 35,757 Joules.

G. Speed-ups and Energy Savings over Mixed Precision

We finally compare runtimes and energy consumption estimations of Trans-IR to Mixed-IR to see how much impact of TTs on speed-ups and energy savings for this case study.

We choose the mixed precision iterative refinement employing single precision arithmetic for step 2 and double-double precision for step 1 and 3 for our baseline. The reasons are followings. First, it is desirable to assign the precision for step 2 with the precision applied for generating a ψ considering the trade-off between numeric accuracies and runtime. A higher precision arithmetic can be applied for step 2, but it does not improve the numeric accuracies in practice [10], [14], [17], while requiring another storage to store higher precision LU factors. Therefore, the precision of step 2 is generally assigned with the precision for generating a LUPP approximator [1], [17]. Second, the precision for step 1 is mostly related to accuracy as mentioned in section II-B and the accuracy for forward error is bounded as $c_2\kappa(A)\epsilon_1$. Therefore, double-double precision arithmetic is applied for ϵ_1 to produce double precision solution accuracy. Finally, the precision for step 3 can be applied with double precision arithmetic, however the computational complexity for step 3 is $O(n)$, which is negligible compared to step 1 and 2 each having $O(n^2)$. Also, a higher precision arithmetic for step 3 can improve final solution accuracy [9].

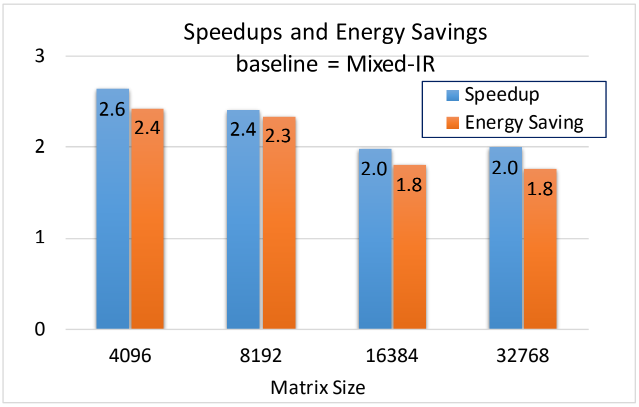

Fig. 14. Speed-ups and Energy Savings by Transprecison Techniques

Figure 14 shows the speed-ups and the energy savings over Mixed-IR by TTs and Figure 15 shows the percentages for the runtimes and the energy consumption estimations by TTs. TTs bring further 2.0-2.6× speedups (i.e., 38% to 51% of runtimes) and 1.8-2.4× energy savings (i.e., 41% to 57% of energy consumptions) to Mixed IR in the figures. In this paper TTs gave rise to significant speedups and energy savings by minimising software-emulated precision arithmetic operations. The impact of TTs is significant if they are used for some applications requiring software-emulated high precision arithmetic discussed in [18]. Notice that TTs are moreover promising when hardware provides multiple precisions. For example, in case a mixed precision iterative refinement employs a half precision arithmetic for ψ for an NVidia P100 GPU to produce single precision accuracy for forward error, TTs can significantly reduce double precision arithmetic step 1 operations for a low condition number matrix (i.e., $\kappa(A) \leq \epsilon_{half}^{-1}$). In this case, single precision arithmetic can be applied for early iterations for step 1 and switch to double precision by TT1. We plan to utilise MAGMA library kernels [19] for the implementation for future work.

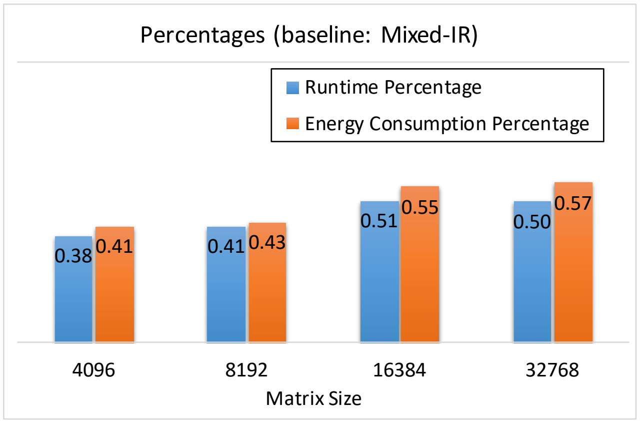

Fig. 15. Percentages for Runtimes and Energy Consumptions by Transprecison Techniques

V. Related Work

In this section we discuss related work for precision utilization to minimize runtime time and energy consumption.

A. Precision Utilization for Iterative Refinement

Linear solvers using iterative refinements are in majority of the applications utilising precisions [1], [2], [6], [10], [18], [20]–[22]. Wilkinson firstly proposed iterative refinement to improve a solution accuracy for forward error [17]. Wilkinson presented his correctness proof using fixed point arithmetic. Moler further investigated the iterative refinement using floating point arithmetic [14]. Since then, Jankowski and Wozniakowski proposed that any method to solve a system of equations (e.g., ψ) could be numerically stable and well behaved with an iterative refinement as long as the relative error in step 2 is less than unity and the system is not ill-conditioned [6]. Extending from the work of [6], Wozniakowski suggested employing an arbitrary lower precision for a matrix decomposition (i.e., computationally intensive task, $O(n^3)$) [20]. Based on his suggestion, Kielbasinski proposed employing an arbitrary lower precision for LU decomposition and intermediate arbitrary precisions for an iterative refinement in 1981 [20]. He named his algorithm Binary Cascade Iterative Refinement (BCIR). The BCIR algorithm has a unique structure unlike a conventional iterative refinement algorithm, but it is not practically feasible for real applications since a $\kappa(A)$ should be known prior to computation. In 2003, Geddes and Zeng pointed out that increasing a precision for steps 1 and 3 per iteration could achieve a prescribed accuracy effectively by exploiting fast hardware-supported precision arithmetic [22], but it does not provide with specific methods for the suggestion. Since then, Lanjou et al. proposed an iterative refinement employing single precision for matrix decomposition and double precision for steps 1 and 3 to minimize runtime to obtain double precision accuracy for backward error [1]. Sun et al. suggested employing an arbitrary lower precision for a matrix decomposition on FPGAs and a higher precision refinement on CPU [21]. In [21], Cray XD-1 reconfigurable computing platform was used for the implementation. Iterative refinement employing arbitrary precisions dynamically for steps 1 and 3 was explored on Xilinx XC6VSX475T FPGAs in [10]. In [10], NP1 was proposed to employ adaptive precisions dynamically for step 1 on a FPGA. Recently, iterative refinements utilising three types of precisions were explored in [2].

B. Automated Precision Tuning

Automated precision tuning for an application was investigated in [3], [4]. The algorithm of [3] adapts the delta debugging based search algorithm to seek 1-minimal test case (e.g., for 1-minimal test case, replacing any variable with a lower precision violates either accuracy constraint or performance constraint). Another automated precision tuning research was proposed in [4] to investigate precision tuning for a lower level implementation.

C. Discussions

The papers of [3], [4] discussed the limitation of the automated precision tools. First, the precisions are assigned statically only for a particular input set. Second, it is not feasible to employ variable precision arithmetic in a loop dynamically according to runtime numeric behaviours. Therefore, the automated precision tuning tools are currently not feasible to be used in practice due to the limited applicability and performance loss. In order to deal with such limitations, domain specific knowledge such as numerical linear algebra can be used to improve the performance of automated precision tuning tools [3]. We hope that this work can contribute to improving the performance of current automated precision tuning tools.

Most works related to utilising precision for iterative refinement discussed statically defined precisions in a loop. Even though we exemplify TTs utilising the three types of precisions for the case study, TTs can be expanded to multiple types of precisions to bring further speed-ups and energy savings.

VI. Conclusion

In this paper, we proposed TTs for iterative refinement which utilise variable precision arithmetic dynamically in a loop (TT1), restructure a numeric algorithm dynamically according to runtime numeric behaviour (TT2), and remove unnecessary accuracy checks (TT3) in order to achieve further speed-ups and energy savings over a conventional mixed precision iterative refinement. To validate our proposed TTs, we performed a case study with an iterative refinement employing a LUPP approximator for a ψ and producing a double precision solution accuracy for forward error. Through the case study, TTs brought further 2.0-2.6X speed-ups and 1.8-2.4X energy savings to conventional mixed precision iterative refinements for n = 4K-32K. The significant improvement on speed-ups and energy savings was possible by exploiting numeric properties of iterative refinement aggressively.

Regarding exploitation of this work, we expect that performance of automated precision tuning tools can be improved by exploiting domain specific knowledge with adapting this work. Future work includes developing TTs for parallel implementation on Multi cores and GPUs with runtime system supporting a dynamic precision programming model. To adapt this work for the trend towards large size matrices (e.g., sparse matrices), we will investigate another approximator for ψ such as either CG or GMRES for our future work.

In this paper, utilising dynamic precisions and dynamic algorithm brings further speed-ups to conventional mixed precision iterative refinements without increasing the number of cores. Therefore, moving from mixed precision paradigm to dynamic precision (including dynamic algorithm) is promising to maximize energy savings for linear system solvers.

Acknowledgement

This project has received funding by the European Commission Horizon 2020 research and innovation programme under grant agreement No. 732631(OPRECOMP). The authors also thank Dr. Lev Mukhanov for his support for energy estimations.

References

[1] 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.

[2] E. Carson and N. Higham, "Accelerating the solution of linear systems by iterative refinement in three precisions," Manchester Institute for Mathematical Sciences, The University of Manchester, UK, MIMS EPrint 2017.24, 2017.

[3] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2013, pp. 1–12.

[4] A. Angerd, E. Sintorn, and P. Stenström, "A framework for automated and controlled floating-point accuracy reduction in graphics applications on gpus," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 4, pp. 46:1–46:25, Dec. 2017. [Online]. Available: http://doi.acm.org/10.1145/3151032

[5] W. Liu *et al.*, *Kernel Adaptive Filtering: A Comprehensive Introduction*. Wiley Publishing, 2010.

[6] M. Jankowski and H. Woźniakowski, "Iterative refinement implies numerical stability," *BIT Numerical Mathematics*, vol. 17, no. 3, 1977.

[7] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[8] J. Bornholt, T. Mytkowicz, and K. S. McKinley, "Uncertain <T>: A first-order type for uncertain data," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 51–66. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541958

[9] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy, "Error bounds from extra-precise iterative refinement," *ACM Trans. Math. Softw.*, vol. 32, no. 2, pp. 325–351, Jun. 2006. [Online]. Available: http://doi.acm.org/10.1145/1141885.1141894

[10] J. Lee, "AIR: Adaptive dynamic precision iterative refinement," Ph.D. dissertation, University of Tennessee, TN, USA, 2012.

[11] X. Li *et al.*, "Design, implementation and testing of extended and mixed precision blas," *ACM Trans. Math. Softw.*, vol. 28, no. 2, pp. 152–205, 2002.

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

[13] L. N. Trefethen and R. S. Schreiber, "Average-case stability of gaussian elimination," *SIAM J. Matrix Anal. Appl.*, vol. 11, no. 3, pp. 335–360, May 1990. [Online]. Available: http://dx.doi.org/10.1137/0611023

[14] C. Moler, "Iterative refinement in floating point," *J. ACM*, vol. 14, no. 2, pp. 316–321, 1967.

[15] L. Mukhanov *et al.*, "ALEA: Fine-grain energy profiling with basic block sampling," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, 2015, pp. 87–98.

[16] ——, "ALEA: A fine-grained energy profiling tool," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 1, pp. 1:1–1:25, 2017.

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

[18] D. Bailey *et al.*, "High-precision computation: Mathematical physics and dynamics," *Applied Mathematics and Computation*, vol. 218, no. 20, pp. 10 106 – 10 121, 2012.

[19] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense linear algebra solvers for multicore with GPU accelerators," in *Proc. of the IEEE IPDPS'10*. Atlanta, GA: IEEE Computer Society, April 19-23 2010, pp. 1–8, DOI: 10.1109/IPDPSW.2010.5470941.

[20] A. Kielbasinski, "Iterative refinement for linear systems in variable precision arithmetic," *BIT*, vol. 21, pp. 97–103, 1981.

[21] J. Sun, G. D. Peterson, and O. O. Storaasli, "High-performance mixed-precision linear solver for fpgas," *IEEE Transactions on Computers*, vol. 57, no. 12, pp. 1614–1623, Dec 2008.

[22] K. O. Geddes and W. W. Zheng, "Exploiting fast hardware floating point in high precision computation," in *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC '03. New York, NY, USA: ACM, 2003, pp. 111–118. [Online]. Available: http://doi.acm.org/10.1145/860854.860886

[23] G. W. Stewart, *Matrix Algorithms*. Society for Industrial and Applied Mathematics, 2001.

Appendix A
Numeric Analysis for Each Step of Mixed Precision Iterative Refinement

In this section, we describe numeric analyses for a mixed precision iterative refinement of Algorithm I by utilising the work of [6]. We consider the mixed precision iterative refinement utilising *two levels of precision* in Algorithm I (e.g., lower precision arithmetic for step 2 and higher precision arithmetic for steps 1 and 3, $\varepsilon_3 = \varepsilon_1$) and producing a working precision accuracy for *forward error* for our baseline [9], [23]. We assume the following condition for an approximator ψ solving $A\mathbf{x} = \mathbf{b}$:

$$||\mathbf{z} - \tilde{\mathbf{z}}||/||\mathbf{z}|| \leq q_\psi \tag{4}$$

, where $||\cdot||$ represents an infinity norm.

In step 1, the computed residual can be represented as:

$$\tilde{\mathbf{r}}^{(i)} = \mathbf{r}^{(i)} + \delta\mathbf{r}^{(i)} = (I + \delta I^{(i)})[A\tilde{\mathbf{x}}^{(i)} + \delta\mathbf{w}^{(i)} - \mathbf{b}] \tag{5}$$

where I is an identity matrix, $\delta I^{(i)}$ is a truncation error diagonal matrix to make Eq. (5) hold and $\delta\mathbf{w}^{(i)}$ represents a rounding error vector caused by the matrix vector multiplication $A\tilde{\mathbf{x}}^{(i)}$ with a finite precision arithmetic ϵ_1. Therefore,

$$\delta\mathbf{r}^{(i)} = \delta\mathbf{w}^{(i)} + \delta I^{(i)}(\mathbf{r}^{(i)} + \delta\mathbf{w}^{(i)}) \tag{6}$$

where $||\delta I^{(i)}|| \leq \epsilon_2$ (e.g., truncation), $||\delta\mathbf{w}^{(i)}|| \leq c_1\epsilon_1||A||\cdot||\tilde{\mathbf{x}}^{(i)}||$, and c_k represents a bounded constant depending on a matrix size from this point forward ($k = 1, 2, 3, ...$).

In step 2, let us define a new residual of $\mathbf{z}$ as follows:

$$\mathbf{h}^{(i)} = A\tilde{\mathbf{z}}^{(i)} - \tilde{\mathbf{r}}^{(i)} \tag{7}$$

where $\tilde{\mathbf{z}}^{(i)}$ is a computed solution of $\mathbf{z}^{(i)}$ for $A\mathbf{z}^{(i)} = \tilde{\mathbf{r}}^{(i)}$ using an approximator ψ. (e.g., $\tilde{\mathbf{z}}^{(i)} = \mathbf{z}^{(i)} + \delta\mathbf{z}^{(i)}$) An approximator ψ satisfying Eq. (4) produces a norm bound of the error of $\mathbf{z}^{(i)}$ (i.e., $||\delta\mathbf{z}^{(i)}||$) as follows:

$$||\delta\mathbf{z}^{(i)}|| = ||A^{-1}\mathbf{h}^{(i)}|| = ||\tilde{\mathbf{z}}^{(i)} - A^{-1}\tilde{\mathbf{r}}^{(i)}|| \leq q_\psi||A^{-1}\tilde{\mathbf{r}}^{(i)}|| \tag{8}$$

In step 3, the approximate solution $\tilde{\mathbf{x}}^{(i)}$ is updated by subtracting the approximation of the error at the previous iteration. The updated solution $\tilde{\mathbf{x}}^{(i+1)}$ can be represented as follows:

$$\tilde{\mathbf{x}}^{(i+1)} = \tilde{\mathbf{x}}^{(i)} - \tilde{\mathbf{z}}^{(i)} - \boldsymbol{\delta}^{(i)} \tag{9}$$

where $||\boldsymbol{\delta}^{(i)}|| \leq \epsilon_3(||\tilde{\mathbf{x}}^{(i)}||+||\tilde{\mathbf{z}}^{(i)}||)$.

Appendix B
Error Bound for the Measurement for the Number of the Cancellation Bits

We used $floor(log_2(|b_j|/|r_j^{(i)}|))$ for the measurement for the number of the cancellation bits at the iteration i, where b_j is the j^{th} component for a vector $\mathbf{b}$ and $r_j^{(i)}$ is the j^{th} component for a vector $\mathbf{r}$ at the iteration i. The b_j and $r_j^{(i)}$ can be represented as follows: $b_j = 1.xx.. \times 2^{exp_b - bias}$ and $r_j^{(i)} = 1.yy.. \times 2^{exp_r - bias}$. The number of cancellation bits is represented as $exp_b - exp_r$ as described in Fig. 1.

$$\begin{aligned} log_2(|b_j|/|r_j^{(i)}|) &= log_2(|b_j|) - log_2(|r_j^{(i)}|) \\ &= exp_b - exp_r + log_2(1.xx../1.yy..) \end{aligned}$$

Therefore,
$log_2(|b_j|/|r_j^{(i)}|) - 1 < exp_b - exp_r < log_2(|b_j|/|r_j^{(i)}|) + 1$
Considering the number of cancellation is a integer,
$\lfloor log_2(|b_j|/|r_j^{(i)}|)\rfloor \leq exp_b - exp_r \leq \lfloor log_2(|b_j|/|r_j^{(i)}|)\rfloor + 1$
$exp_b - exp_r - 1 \leq \lfloor log_2(|b_j|/|r_j^{(i)}|)\rfloor \leq exp_b - exp_r$

Therefore, our measurement has 1 bit error at most for the number of cancellation bits.

Appendix C
Analyses for Inner Solver Activation

The main time cost occurs in steps 1 and 2 since steps 1 and 2 require $O(n^2)$ for each, while step 3 requires $O(n)$. For time cost analyses, we use $T(\varepsilon_k)$ for a theoretical time cost for an arithmetic operation depending on a precision, where $k = \{S(single), D(double), DD(double\text{-}double)\}$. For the mixed precision iterative refinement with $\varepsilon_{1,3} = \varepsilon_{DD}$ and $\varepsilon_2 = \varepsilon_S$, the matrix vector multiplication in step 1 requires $2n^2$ arithmetic operations, so $T_{step1} \approx 2n^2T(\varepsilon_{DD})$ for the time cost for step 1. Assuming that LUPP is employed for ψ and double precision arithmetic is employed for sub-step 1, the time cost for step 2 including the nested loop is approximately $T_{step2} \approx 2n^2T(\varepsilon_S) + T_{in}$, where $T_{in} = N_{in}(2n^2T(\varepsilon_D)(sub\text{-}step1) + 2n^2T(\varepsilon_S)(sub\text{-}step2))$ is a required time cost for an inner-loop refinement and N_{in} is a required number of the inner-loop iterations. The approximated overall time cost for the iterative refinement can be represented as follows :

$$\begin{aligned} &T_{iter\text{-}ref\text{-}without\text{-}in} \approx N_{out\text{-}without\text{-}in}(T_{step1} + T_{step2}) \\ &= 2n^2N_{out\text{-}without\text{-}in}(T(\varepsilon_{DD}) + T(\varepsilon_S)) \\ &T_{iter\text{-}ref\text{-}with\text{-}in} \approx N_{out\text{-}with\text{-}in}(T_{step1} + T_{step2}) \\ &= 2n^2N_{out\text{-}with\text{-}in}(T(\varepsilon_{DD}) + T(\varepsilon_S) + N_{in}(T(\varepsilon_D) + T(\varepsilon_S))) \end{aligned}$$

, where $N_{out\text{-}without\text{-}in}$ is a required number of iterations for out-loop refinement without employing an inner-loop and $N_{out\text{-}with\text{-}in}$ is a required number of iterations for out-loop refinement with employing an inner-loop. Notice that the common starting point for $N_{out\text{-}with\text{-}in}$ and $N_{out\text{-}without\text{-}in}$ is at the initial iteration considering TT2. With an improved convergence rate using an inner-loop refinement, $N_{out\text{-}with\text{-}in} \leq N_{out\text{-}without\text{-}in}$, while the time cost per out-loop iteration is increased. Therefore, it is worthy to employ an inner loop when

$$T_{iter\text{-}ref\text{-}with\text{-}in} \leq T_{iter\text{-}ref\text{-}without\text{-}in}. \tag{10}$$

The value of p in Algorithm III can be determined based on Eq. (10) as follows :

$$\begin{aligned} T_{iter\text{-}ref\text{-}with\text{-}in} &\leq T_{iter\text{-}ref\text{-}without\text{-}in} \\ 2n^2N_{out\text{-}with\text{-}in}(T(\varepsilon_{DD}) + T(\varepsilon_S) &+ N_{in}(T(\varepsilon_D) + T(\varepsilon_S))) \\ &\leq 2n^2N_{out\text{-}without\text{-}in}(T(\varepsilon_{DD}) + T(\varepsilon_S)) \\ \frac{N_{out\text{-}with\text{-}in}}{N_{out\text{-}without\text{-}in}}(1 + N_{in}&\frac{T(\varepsilon_D) + T(\varepsilon_S)}{T(\varepsilon_{DD}) + T(\varepsilon_S)}) \leq 1 \\ N_{in}\frac{T(\varepsilon_D) + T(\varepsilon_S)}{T(\varepsilon_{DD}) + T(\varepsilon_S)} &\leq \frac{N_{out\text{-}without\text{-}in}}{N_{out\text{-}with\text{-}in}} - 1 \\ \frac{T(\varepsilon_D)}{T(\varepsilon_{DD})} \approx \frac{T(\varepsilon_D) + T(\varepsilon_S)}{T(\varepsilon_{DD}) + T(\varepsilon_S)} &\leq N_{in}^{-1} \cdot (\frac{N_{out\text{-}without\text{-}in}}{N_{out\text{-}with\text{-}in}} - 1) \\ \frac{T(\varepsilon_{DD})}{T(\varepsilon_D)} &\gtrsim \frac{N_{in}N_{out\text{-}with\text{-}in}}{N_{out\text{-}without\text{-}in} - N_{out\text{-}with\text{-}in}} = p^* \end{aligned} \tag{11}$$

Based on Eq. (11), the optimal p, p^* is : $\frac{N_{in}N_{out\text{-}with\text{-}in}}{N_{out\text{-}without\text{-}in} - N_{out\text{-}with\text{-}in}}$. However, the variables contain some dynamic variables determined at the end of program, so it is hard to apply automated process for p choices. We recommend $p = [6, 10]$ based on our experiments. A p less than p^* makes the nested loop activation highly probable, however, the nested loop might degrade overall performance. A p higher than p^* makes the inner loop activation always bring performance improvement, but some of desirable activations might be killed. We chose $p = 10$ in this paper. Therefore, if $T(\varepsilon_{DD}) > 10 \cdot T(\varepsilon_D)$, the nested loop refinement will be operated. For example, in Fig. 3, applying TT2 reduces 2 double-double arithmetic out-loop iterations to 1 double-double arithmetic out-loop iteration. (i.e., $N_{out\text{-}with\text{-}in} = 1$ and $N_{out\text{-}without\text{-}in} = 2$.) If the number of iterations for nested loop N_{in} is less than 10, the nested loop refinement is beneficial when $p = 10$. For example, in Fig. 6 the averages of N_{in}s are less than 5. Therefore, p^* is around 6 and our choice for p is a bit strict so that the activation of inner loop refinement by TT2 always can bring performance improvement. For our case, if a machine has been changed and the latency gab $T(\varepsilon_1)/T(sub - step1)$ is still larger than p, the inner loop refinement will still be activated. If it is smaller than p, the nested loop will not be operated.

Appendix D
Proof for Numeric Accuracy using TT3

The proof for TT3 is as follows. When the convergence is saturated at the $(s-1)^{th}$ iteration in Algorithm III, the solution accuracy in step 3 using double precison arithemtic for step 1 is bounded as follows [6], [14] (refer to section II-B):

$$||\tilde{\mathbf{x}}^{(s)} - \mathbf{x}||/||\mathbf{x}|| = ||\delta\mathbf{x}^{(s)}||/||\mathbf{x}|| \leq c_3\kappa(A)\epsilon_D \tag{12}$$

The computed solution $\tilde{\mathbf{x}}^{(s)}$ becomes an input for double-double precision arithmetic for step 1 at the s^{th} iteration. The residual at the s^{th} iteration becomes [6] :

$$\tilde{\mathbf{r}}^{(s)} = (I + \delta I^{(s)})[A\tilde{\mathbf{x}}^{(s)} + \delta\mathbf{w}^{(s)} - \mathbf{b}] \tag{13}$$

where $||\delta \mathbf{w}^{(s)}|| \leq c_2||A||\cdot||\tilde{\mathbf{x}}^{(s)}||\epsilon_{DD}$ and $||\delta I^{(s)}|| \leq \epsilon_S$.

In step 2, the computed $\tilde{\mathbf{z}}^{(s)}$ satisfies :
$(A+\Delta A)\tilde{\mathbf{z}}^{(s)} = (A+\Delta A)(\mathbf{z}^{(s)}+\delta\mathbf{z}^{(s)}) = \tilde{\mathbf{r}}^{(s)}$. [14]
The $\mathbf{z}^{(s)}$ can be represented as follows using Eq. (5):

$$\begin{aligned} \mathbf{z}^{(s)} &= A^{-1}\tilde{\mathbf{r}}^{(s)} \\ &= (\delta\mathbf{x}^{(s)} + A^{-1}\delta\mathbf{w}^{(s)}) + A^{-1}\delta I^{(s)}(A\delta\mathbf{x}^{(s)}+\delta\mathbf{w}^{(s)}) \end{aligned} \tag{14}$$

and $||\delta\mathbf{z}^{(s)}|| \leq q_\psi||\mathbf{z}^{(s)}||$ using Eq. (4). The $\tilde{\mathbf{z}}^{(s)}$ becomes an input for the inner loop. The difference between $\delta\mathbf{x}^{(s)}$ and $\mathbf{z}^{(s)}$ is represented as follows:
$\mathbf{z}^{(s)} - \delta\mathbf{x}^{(s)} = A^{-1}\delta\mathbf{w}^{(s)} + A^{-1}\delta I^{(s)}(A\delta\mathbf{x}^{(s)}+\delta\mathbf{w}^{(s)})$
Therefore, the norm of the difference can be represented as follows :

$$\begin{aligned} ||\mathbf{z}^{(s)} - \delta\mathbf{x}^{(s)}|| \leq c_1\kappa(A)\epsilon_{DD}||\tilde{\mathbf{x}}^{(s)}|| + \kappa(A)\epsilon_S\epsilon_D||\delta\mathbf{x}^{(s)}|| \\ + c_1\kappa(A)\epsilon_S\epsilon_{DD}||\tilde{\mathbf{x}}^{(s)}|| \end{aligned} \tag{15}$$

Since ψ converges, we exploit NP3. (i.e., $\kappa(A) < 1/(c_2\epsilon_S)$) Therefore, Eq. (15) can be represented:

$$\begin{aligned} ||\mathbf{z}^{(s)} - \delta\mathbf{x}^{(s)}|| < c_1\epsilon_{DD}(1+1/(c_2\epsilon_S))||\tilde{\mathbf{x}}^{(s)}|| + (\epsilon_D/c_2)||\delta\mathbf{x}^{(s)}|| \\ \approx c_1\epsilon_{DD}/(c_2\epsilon_S)||\tilde{\mathbf{x}}^{(s)}|| + (\epsilon_D/c_2)||\delta\mathbf{x}^{(s)}|| \end{aligned} \tag{16}$$

We ignore the first value "1" in $(1+1/(c_2\epsilon_S))$ in Eq. (16) since $1 << (c_2\epsilon_S)^{-1}$. Based on Eq. (16), $||\mathbf{z}^{(s)}||$ can be bounded as:

$$||\mathbf{z}^{(s)}|| < ||\delta\mathbf{x}^{(s)}|| + c_1\epsilon_{DD}/(c_2\epsilon_S)||\tilde{\mathbf{x}}^{(s)}|| + (\epsilon_D/c_2)||\delta\mathbf{x}^{(s)}|| \tag{17}$$

In step 3 at the s^{th} iteration, the updated computed solution can be represented as follows :
$\tilde{\mathbf{x}}^{(s+1)} = \tilde{\mathbf{x}}^{(s)} - \tilde{\mathbf{z}}^{(s)} - \boldsymbol{\delta}^{(s)} = \mathbf{x} + \delta\mathbf{x}^{(s)} - \mathbf{z}^{(s)} - \delta\mathbf{z}^{(s)} - \boldsymbol{\delta}^{(s)}$
$= \mathbf{x} + \delta\mathbf{x}^{(s+1)}$, where $\delta\mathbf{x}^{(s+1)} = \delta\mathbf{x}^{(s)} - \mathbf{z}^{(s)} - \delta\mathbf{z}^{(s)} - \boldsymbol{\delta}^{(s)}$
and $||\boldsymbol{\delta}^{(s)}|| \leq ||\mathbf{x} + \delta\mathbf{x}^{(s)} - \mathbf{z}^{(s)} - \delta\mathbf{z}^{(s)}||\epsilon_{DD}$
$\leq ||\mathbf{x}||\epsilon_{DD} + ||\delta\mathbf{x}^{(s)} - \mathbf{z}^{(s)} - \delta\mathbf{z}^{(s)}||\epsilon_{DD}$.
Therefore, the norm bound for $\delta\mathbf{x}^{(s+1)}$ is :
$||\delta\mathbf{x}^{(s+1)}|| \leq ||\delta\mathbf{x}^{(s)} - \mathbf{z}^{(s)} - \delta\mathbf{z}^{(s)}|| + ||\boldsymbol{\delta}^{(s)}||$
$\leq ||\mathbf{x}||\epsilon_{DD} + (1+\epsilon_{DD})||\delta\mathbf{x}^{(s)} - \mathbf{z}^{(s)} - \delta\mathbf{z}^{(s)}||$
$\leq ||\mathbf{x}||\epsilon_{DD} + (1+\epsilon_{DD})||\delta\mathbf{x}^{(s)} - \mathbf{z}^{(s)}|| + (1+\epsilon_{DD})||\delta\mathbf{z}^{(s)}||$
$\approx ||\mathbf{x}||\epsilon_{DD} + ||\delta\mathbf{x}^{(s)} - \mathbf{z}^{(s)}|| + ||\delta\mathbf{z}^{(s)}||$
We ignore ϵ_{DD} in $(1+\epsilon_{DD})$ since $1 >> \epsilon_{DD}$. Employing the properties of $||\delta\mathbf{z}^{(s)}||/||\mathbf{z}^{(s)}|| \leq \epsilon_S$, Eq. (12), Eq. (16), and Eq. (17) yields:
$||\tilde{\mathbf{x}}^{(s+1)} - \mathbf{x}||$
$< ||\mathbf{x}||\epsilon_{DD} + c_1\epsilon_{DD}/(c_2\epsilon_S)||\tilde{\mathbf{x}}^{(s)}|| + (\epsilon_D/c_2)||\delta\mathbf{x}^{(s)}|| + \epsilon_S||\mathbf{z}^{(s)}||$
$< ||\mathbf{x}||\epsilon_{DD} + 2(c_1\epsilon_{DD}/(c_2\epsilon_S)||\tilde{\mathbf{x}}^{(s)}|| + (\epsilon_D/c_2)||\delta\mathbf{x}^{(s)}||) + \epsilon_S||\delta\mathbf{x}^{(s)}||$
$= ||\mathbf{x}||\epsilon_{DD} + 2c_5\epsilon_{DD}/\epsilon_S||\mathbf{x} + \delta\mathbf{x}^{(s)}|| + (\epsilon_S + 2\epsilon_D/c_2)||\delta\mathbf{x}^{(s)}||$
$\leq (\epsilon_{DD} + 2c_5\epsilon_{DD/S})||\mathbf{x}|| + (\epsilon_S + 2\epsilon_D/c_2 + 2c_5\epsilon_{DD/S})||\delta\mathbf{x}^{(s)}||$
$\leq (\epsilon_{DD} + 2c_5\epsilon_{DD/S})||\mathbf{x}|| + (\epsilon_S + 2\epsilon_D/c_2 + 2c_5\epsilon_{DD/S})c_3\kappa(A)\epsilon_D||\mathbf{x}||$
$= (\epsilon_{DD} + 2c_5\epsilon_{DD/S} + c_3\kappa(A)\epsilon_D(\epsilon_S + 2\epsilon_D/c_2 + 2c_5\epsilon_{DD/S}))||\mathbf{x}||$
where $\epsilon_{DD/SS} = \epsilon_{DD}/\epsilon_S$. Therefore, exploiting NP3 once again (i.e., $\kappa(A) < 1/(c_2\epsilon_S)$) proves that the accuracy of the solution at the s^{th} iteration in step 3 is subject to produce double precision accuracy for forward error:

$$\begin{aligned} &||\tilde{\mathbf{x}}^{(s+1)} - \mathbf{x}||/||\mathbf{x}|| \\ &< \epsilon_{DD} + 2c_5\epsilon_{DD/S} + (c_6 + 2c_7\epsilon_{D/S} + 2c_8\epsilon_{DD/SS})\epsilon_D \end{aligned} \tag{18}$$

We also verified that the final accuracy satisfies the condition $||\tilde{\mathbf{x}}^{(s+1)} - \mathbf{x}||/||\mathbf{x}|| < 10\epsilon_D$ for all test cases in our case study.