

A Parallel Mixed-Precision GMRES-IR Solver for Ill-Conditioned Equations in Device Simulation

Jiawen Cheng¹, Yuanyuan Yang², Ding Gong², and Wenjian Yu¹

¹Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China

²Peifeng Tunan Semiconductor Co. Ltd., Suzhou, China

Abstract—Efficient and reliable device simulation remains a critical challenge for modern electronic design automation (EDA), where ill-conditioned sparse linear equation systems are often solved. Traditional linear matrix solvers struggle to balance accuracy, performance, and scalability concurrently in the presence of ill-conditioning. In this work, we propose a parallel solver framework that integrates mixed-precision iterative refinement with GMRES algorithm and novel architecture-aware optimizations on modern CPUs. Our approach leverages vectorization, parallel scheduling, and memory hierarchy optimizations to accelerate Krylov subspace methods while preserving numerical robustness. Comprehensive evaluation on matrices arising from realistic device simulation problems demonstrates that our solver achieves $5.4\times$ speedup on average, compared to the high-precision direct solver baseline, while maintaining solution accuracy within given tolerances. Moreover, the proposed mixed-precision GMRES-IR solver attains further $3.3\times$ parallel speedup with 8 threads, demonstrating its parallel efficiency.

Index Terms—device simulation, mixed-precision solver, iterative refinement, sparse matrix solver, parallel computing.

I. INTRODUCTION

Semiconductor device simulation is a cornerstone of modern electronic design automation (EDA), enabling accurate modeling and optimization of nano-scale transistors and circuits. The discretizations of drift-diffusion or hydrodynamic partial differential equations (PDEs) in device modeling typically produce large-scale, sparse, and often ill-conditioned linear equation systems [1], [2], [3]. These ill-conditioned systems, if not solved to adequate precision, may lead to physically meaningless or unstable simulation results, posing a significant reliability issue in downstream EDA processes. As a result, the numerical stability and accuracy of linear solvers are essential to ensuring reliable results of device simulation.

Direct methods based on double-precision LU factorization have been widely adopted in device simulation due to their robustness, but they may still fail to deliver sufficient accuracy for highly ill-conditioned systems. A natural strategy to address this issue is to increase the precision of the LU factorization itself, e.g., using multiple-component precision [4] or quadruple precision. However, it is generally impractical for two main reasons: 1) high-precision floating-point operations are more computationally expensive than standard floating-point operations due to a lack of hardware arithmetic units [5], and 2) modern sparse direct solvers such as PARDISO [6] and SuperLU [7] heavily rely on highly optimized basic linear

algebra subprogram (BLAS) kernels, which are not available for non-standard precision formats.

An alternative is to retain the LU factorization in double precision and employ iterative refinement (IR) to progressively improve the solution accuracy via applying correction steps until convergence is reached [8], [9]. Classical IR corrects the solution by repeatedly solving residual equations, but its convergence can be limited when systems are extremely ill-conditioned. To address this, Krylov subspace methods such as generalized minimal residual (GMRES) method have been incorporated into the refinement process, forming GMRES-IR schemes that substantially enhance robustness of convergence [10]. Furthermore, several mixed-precision GMRES-IR frameworks were proposed, showing that carefully combining low, working, and high precisions can yield both efficiency and accuracy [11], [12], [13]. Nevertheless, few studies have systematically addressed the extremely ill-conditioned systems (condition number $\kappa(\mathbf{A}) > 10^{15}$) that frequently arise in device simulation, nor have they provided efficient GMRES-IR frameworks tailored to this setting.

In this work, we propose a parallel mixed-precision GMRES-IR solver specifically designed to address the challenges of extremely ill-conditioned linear systems in semiconductor device simulation. The main contributions of this paper can be summarized as follows.

- We present a plug-and-play mixed-precision GMRES-IR framework that can be coupled with any standard direct solver as a preconditioner, which achieves accuracy comparable to high-precision LU factorization at much lower cost.
- The Arnoldi process in GMRES is accelerated by reorganizing Krylov basis storage for SIMD-friendly multiple-component arithmetic. In addition, an approach based on nested dissection ordering is proposed to enhance the data locality of sparse matrix–vector multiplication (SpMV).
- A dynamic strategy is proposed for setting the inner GMRES tolerance, which can reduce redundant iterations while preserving convergence.
- We integrate load-aware balancing for parallel SpMV and parallelized sparse triangular solves (SpTRSVs) into the framework, yielding remarkable parallel speedups on benchmarks.

We evaluate the developed mixed-precision GMRES-IR solver on matrices arising from realistic device simulation problems. Compared with the baseline relying on high-precision LU

W.Yu is the corresponding author.

factorization, our approach achieves the same level of accuracy while delivering an average speedup of $5.4\times$. On the parallel side, our GMRES-IR implementation attains $3.3\times$ further speedup with 8 threads, demonstrating its parallel efficiency.

II. BACKGROUND

A. The GMRES-IR Algorithm

Iterative refinement (IR) is a classical technique for improving the accuracy of solutions to linear systems. Suppose we want to solve $\mathbf{Ax} = \mathbf{b}$ and already have an approximate solution $\mathbf{x}^{(0)}$ obtained from an LU factorization. Each refinement step k consists of three operations [8]:

- 1) Compute the residuals $\mathbf{r}^{(k)} = \mathbf{b}^{(k)} - \mathbf{Ax}^{(k)}$.
- 2) Solve the system $\mathbf{Ad}^{(k)} = \mathbf{r}^{(k)}$.
- 3) Apply the correction $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)}$.

Here, the residual $\mathbf{r}^{(k)}$ is evaluated in a higher precision to minimize round-off error. The correction $\mathbf{d}^{(k)}$ is computed by forward/backward substitution with the precomputed LU factors. This approach effectively compensates for rounding errors introduced during the initial factorization and converges to a highly accurate solution under appropriate conditions [14].

However, the correction step of classical IR relies entirely on triangular solves with the low-precision LU factors, so the attainable accuracy is limited when these factors are of poor quality. To address this limitation, Krylov subspace methods such as GMRES have been incorporated into the refinement loop, leading to GMRES-IR schemes [10]. In GMRES-IR, instead of triangular solves, the residual equation $\mathbf{Ad}^{(k)} = \mathbf{r}^{(k)}$ is solved by a preconditioned GMRES iteration, which provides more robust convergence behavior in the presence of high condition numbers. Theoretical analyses have established that GMRES-IR inherits the convergence guarantees of IR while extending its applicability to much more ill-conditioned problems. Recent advances further explore multi-precision variants such as three-precision GMRES-IR [11] and even five-precision GMRES-IR [13], showing that carefully assigning precisions to factorization, residual evaluation, and iterative corrections can balance efficiency and accuracy.

In this work, we adopt the GMRES-IR paradigm with a double-precision LU factorization as the preconditioner and employ multiple-component precision formats for the GMRES iteration. This combination enables us to reliably solve the extremely ill-conditioned systems that frequently arise in device simulation, where classical IR or standard double-precision GMRES would otherwise fail.

B. Multiple-Component Precision Formats

The demand for higher-than-double precision in numerical linear algebra often arises when solving ill-conditioned systems. While quadruple precision (IEEE binary128) is standardized, it is rarely practical because hardware support is absent on most architectures and software emulation is extremely slow [15]. As a result, the performance penalty of using binary128 or higher is prohibitive for large-scale sparse solvers.

A widely adopted alternative is multiple-component arithmetic, in which a real number is represented as the unevaluated

sum of several double-precision values. The simplest case is double-double (DD) format, representing a number as a tuple of two double-precision floating-point numbers (x_h, x_l) , where x_h stores the leading part and x_l the round-off correction. It can ensure about 106 bits of effective precision. Extensions to triple-double (TD) and quad-double (QD) use three or four components, respectively, achieving progressively higher accuracy [4].

The fundamental building blocks of multi-component arithmetic are error-free transformations such as TwoSum and TwoProd [16], [17], which allow the floating-point addition or multiplication into a rounded result and a residual error term. These operations make it possible to propagate and accumulate errors into the lower components, thereby extending the precision without requiring hardware support for wider floating-point formats.

As an example, the addition of two DD numbers (x_h, x_l) and (y_h, y_l) can be implemented as Alg. 1. Here, TwoSum(a, b) produces the floating-point representation of $a + b$ as s and an error term e such that $a + b = s + e$ exactly. The second error-free addition is implemented by QuickTwoSum, which is a more efficient variant of TwoSum that requires $|a| > |b|$.

This condition is satisfied because s stores the leading component of the result and dominates the smaller correction e' . Similarly, the multiplication of two DD numbers is summarized as Alg. 2. In this case, TwoProd(a, b) computes the floating-point representation of $a \times b$ as p and the error term e . On architectures supporting fused multiply-add (FMA), e can

Algorithm 1 Addition of two DD-precision numbers.

Input: DD-precision numbers (x_h, x_l) and (y_h, y_l) .

Output: Sum (s_h, s_l) .

- 1: **function** TWOSUM(a, b)
 - 2: $s = a + b$.
 - 3: $v = s - a$.
 - 4: $e = (a - (s - v)) + (b - v)$.
 - 5: **return** (s, e)
 - 6: **function** QUICKTWSUM(a, b) \triangleright requires $|a| > |b|$
 - 7: $s = a + b$.
 - 8: $e = b - (s - a)$.
 - 9: **return** (s, e)
 - 10: $(s, e) = \text{TWOSUM}(x_h, y_h)$.
 - 11: $e = e + (x_l + y_l)$.
 - 12: $(s_h, s_l) = \text{QUICKTWSUM}(s, e)$.
-

Algorithm 2 Multiplication of two DD-precision numbers.

Input: DD-precision numbers (x_h, x_l) and (y_h, y_l) .

Output: Product (p_h, p_l) .

- 1: **function** TWOPROD(a, b)
 - 2: $p = a \times b$.
 - 3: $e = \text{FMA}(a, b, -p)$.
 - 4: **return** (p, e)
 - 5: $(p, e) = \text{TWOPROD}(x_h, y_h)$.
 - 6: $e = e + (x_h \times y_l + y_h \times x_l)$.
 - 7: $(p_h, p_l) = \text{QUICKTWSUM}(p, e)$.
-

be computed efficiently by $\text{FMA}(a, b, -p)$, which computes $a \times b - p$ in a single operation with only one final rounding.

The DD arithmetic achieves accuracy comparable to IEEE binary128 but is substantially faster, as it relies solely on native double-precision instructions, which can be efficiently vectorized on modern architectures. In the proposed solver, multiple-component arithmetic serves as the working precision of the GMRES iteration.

III. PARALLEL MIXED-PRECISION GMRES-IR SOLVER

In this section we present our proposed parallel mixed-precision GMRES-IR solver. It is designed as a plug-and-play framework that combines any standard direct solver with a mixed-precision GMRES-IR iteration in any multiple-component arithmetic. In this paper, we adopt DD precision as the default working precision to illustrate the framework, owing to its balance between accuracy and computational cost.

The overall algorithm is summarized in Alg. 3. Lines 1-2 are a standard LU factorization. Steps that explicitly involve

Algorithm 3 Mixed-precision GMRES-IR algorithm.

Input: Matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, right-hand side $\mathbf{b} \in \mathbb{R}^n$, outer IR tolerance τ_{out} , inner GMRES tolerance τ_{in} .

Output: Solution \mathbf{x} .

```

1: Compute a double-precision LU factorization  $\mathbf{A} = \mathbf{LU}$ .
2: Solve  $\mathbf{x}^{(0)} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{b}$  in double precision.
3: for  $k = 0, 1, \dots$  do
4:    $\triangleright$  1. Compute the residuals
5:   Compute  $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$ .
6:   if  $\|\mathbf{r}^{(k)}\|_2 < \tau_{\text{out}}\|\mathbf{b}\|_2$  then
7:     break.
8:    $\triangleright$  2. Inner GMRES solving  $\mathbf{A}\mathbf{d}^{(k)} = \mathbf{r}^{(k)}$ 
9:    $\mathbf{v}^{(0)} = \mathbf{r}^{(k)} / \|\mathbf{r}^{(k)}\|_2$ .
10:  Initialize a vector  $\mathbf{s}$  with  $\mathbf{s}_0 = \|\mathbf{r}^{(k)}\|_2$ .
11:  Initialize an empty matrix  $\mathbf{H}$ .
12:  for  $j = 0, 1, \dots$  do
13:    Solve  $\mathbf{z} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{v}^{(j)}$  in double precision.
14:     $\mathbf{w} = \mathbf{Az}$ .  $\triangleright$  SpMV
15:    for  $i = 0, \dots, j$  do  $\triangleright$  MGS process
16:       $\mathbf{H}_{ij} = \langle \mathbf{w}, \mathbf{v}^{(i)} \rangle$ .
17:       $\mathbf{w} = \mathbf{w} - \mathbf{H}_{ij}\mathbf{v}^{(i)}$ .
18:       $\mathbf{H}_{j+1,j} = \|\mathbf{w}\|_2$ .
19:       $\mathbf{v}^{(j+1)} = \mathbf{w} / \mathbf{H}_{j+1,j}$ .
20:      for  $i = 0, \dots, j-1$  do
21:         $(\mathbf{H}_{ij}, \mathbf{H}_{i+1,j})^\top = \mathbf{J}_i(\mathbf{H}_{ij}, \mathbf{H}_{i+1,j})^\top$ .
22:        Compute a Givens rotation  $\mathbf{J}_j$  on  $(\mathbf{H}_{jj}, \mathbf{H}_{j+1,j})^\top$ .
23:         $(\mathbf{H}_{jj}, \mathbf{H}_{j+1,j})^\top = \mathbf{J}_j(\mathbf{H}_{jj}, \mathbf{H}_{j+1,j})^\top$ .
24:         $(\mathbf{s}_j, \mathbf{s}_{j+1})^\top = \mathbf{J}_j(\mathbf{s}_j, 0)^\top$ .
25:        if  $|\mathbf{s}_{j+1}| < \tau_{\text{in}}\|\mathbf{r}^{(k)}\|_2$  then
26:          break.
27:       $\mathbf{H}$  = the upper triangular part of  $\mathbf{H}_{0:j+1,0:j+1}$ .
28:      Solve  $\mathbf{y} = \widetilde{\mathbf{H}}^{-1}\mathbf{s}$ .
29:       $\mathbf{v} = \sum_{0 \leq i \leq j+1} \mathbf{y}_i \mathbf{v}^{(i)}$ .
30:      Solve  $\mathbf{d}^{(k)} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{v}$  in double precision.
31:       $\triangleright$  3. Apply the correction
32:       $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)}$ .

```

the LU factors (factorization in line 1 and triangular solves in lines 2, 13, and 30) are executed in double precision. All other operations (residual computation, Krylov basis construction, and orthogonalization) are performed in DD precision. The outer refinement loop computes high-precision residuals to guide convergence, while the inner loop invokes GMRES preconditioned by the LU factors to approximately solve the correction equation $\mathbf{A}\mathbf{d}^{(k)} = \mathbf{r}^{(k)}$. The Arnoldi process is implemented with modified Gram–Schmidt (MGS) orthogonalization (lines 15–17), stabilized by Givens rotations (lines 20–24) for ensuring numerical robustness.

In addition to this basic workflow, we incorporate several optimizations tailored for efficiency, which will be presented in the following subsections in turn. Specifically, we restructure the storage of Krylov basis vectors to improve SIMD utilization of multiple-component arithmetic, exploit nested dissection (ND) ordering to improve the memory locality of SpMV, and propose a dynamic strategy for adjusting the inner GMRES tolerance to reduce redundant iterations. To scale the solver on multi-core architectures, we parallelize the major kernel SpMV through load-aware balancing and SpTRSV by utilizing the block structure induced by ND, enabling consistent speedups in the GMRES-IR workflow. Together, these techniques provide a robust and efficient realization of parallel mixed-precision GMRES-IR for extremely ill-conditioned systems.

A. Optimized Data Layout of Krylov Basis Vectors

A key performance bottleneck in GMRES lies in the orthogonalization of the Krylov basis vectors. In a naive array-of-structures (AoS) layout, each element in a basis vector is stored as a pair (x_h, x_l) . While straightforward, this arrangement forces SIMD units to stride irregularly through memory to access the high and low parts of consecutive elements, leading to poor vectorization efficiency and increased cache pressure.

To address this, we adopt a **structure-of-arrays** (SoA) layout as illustrated in Fig. 1, where $\mathbf{v}_{i,h}^{(j)}/\mathbf{v}_{i,l}^{(j)}$ denotes the high/low component of the i -th element in the j -th basis vector. In this scheme, all high components of the vector are stored contiguously, followed by all low components. This reorganization enables contiguous memory access for each component array, allowing the hardware to fully utilize SIMD units when applying BLAS-like kernels such as dot products.

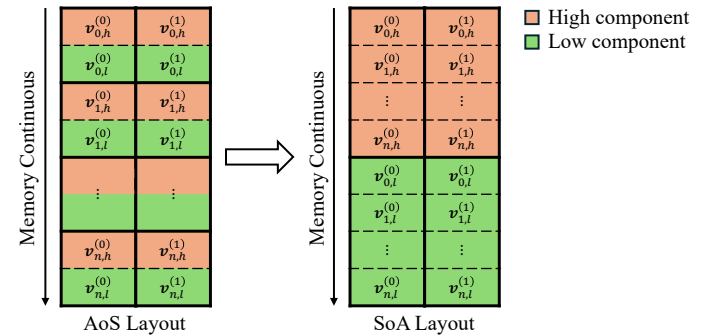


Fig. 1. AoS and SoA layouts of DD-precision Krylov basis vectors. SoA improves vectorization and simplifies precision conversion.

Algorithm 4 Dot product of two DD-precision vectors under the SoA layout using AVX-512.

Input: DD-precision vectors x and y of length n (stored as a double-precision vector of length $2n$).

Output: Dot product (p_h, p_l) .

```

1: ▷  $s_h, s_l, x_h, x_l, y_h, y_l, p, e, s, e'$  are AVX-512 registers <
2:  $s_h = \_mm512\_set1\_pd(0)$ .
3:  $s_l = \_mm512\_set1\_pd(0)$ .
4: for  $i = 0, 8, 16, \dots, n - 8$  do
5:   ▷  $(p, e) = (x_h, x_l) \times (y_h, y_l)$  <
6:    $x_h = \_mm512\_loadu\_pd(x_{i:i+7})$ .
7:    $x_l = \_mm512\_loadu\_pd(x_{i+n:i+n+7})$ .
8:    $y_h = \_mm512\_loadu\_pd(y_{i:i+7})$ .
9:    $y_l = \_mm512\_loadu\_pd(y_{i+n:i+n+7})$ .
10:   $(p, e) = \text{AVX512TwoProd}(x_h, y_h)$ .
    $e = \_mm512\_add\_pd(e,$ 
11:    $\_mm512\_add\_pd(\_mm512\_mul\_pd(x_h, y_l),$ 
    $\_mm512\_mul\_pd(y_h, x_l))$ .
12:   $(p, e) = \text{AVX512QuickTwoSum}(p, e)$ .
13:  ▷  $(s_h, s_l) = (s_h, s_l) + (p, e)$  <
14:   $(s, e') = \text{AVX512TwoSum}(s_h, p)$ .
15:   $e' = \_mm512\_add\_pd(e', \_mm512\_add\_pd(s_l, e))$ .
16:   $(s_h, s_l) = \text{AVX512QuickTwoSum}(s, e')$ .
17:  $h, l =$  two double-precision vectors of length 8.
18:  $\_mm512\_store\_pd(h, s_h)$ .
19:  $\_mm512\_store\_pd(l, s_l)$ .
20:  $(p_h, p_l) = \sum_{0 \leq i \leq 7} (h_i, l_i)$ . < DD addition

```

As a concrete example, Alg. 4 illustrates the implementation of a DD-precision dot product under the SoA layout using AVX-512. With high and low components stored contiguously, each `_mm512_loadu_pd` instruction fetches eight consecutive values without strided access, eliminating the overhead of gather operations. The computation relies on vectorized error-free transformations (`AVX512TwoSum`, `AVX512QuickTwoSum`, and `AVX512TwoProd`), which directly extend their corresponding scalar versions primitives to SIMD lanes. Finally, a reduction combined with a scalar DD addition produces the global dot product (lines 17-20).

An additional benefit of SoA layout arises from the frequent type conversions between double and DD within the GMRES-IR workflow. With the SoA layout, conversion becomes trivial: promoting a double vector to DD requires simply setting the low component array to zero, and demoting a DD-precision vector back to double requires reading only the high component array. These conversions involve no element-wise unpacking and thus incur negligible overhead.

B. Enhancing SpMV Locality via Nested Dissection Ordering

Within GMRES-IR, sparse matrix-vector multiplication (SpMV) is invoked repeatedly during the Arnoldi process. Its performance is typically limited by the irregular fetches of vector entries. In device simulation, the matrix A often exhibits irregular nonzero distributions, leading to frequent cache misses and degraded memory efficiency.

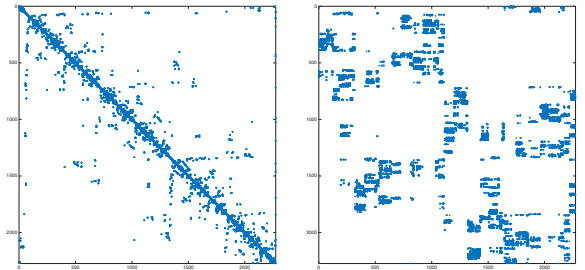
To mitigate this, we exploit **nested dissection** (ND) ordering to improve the memory locality of SpMV. Specifically, instead of directly solving $Ax = b$, we reformulate the problem as

$$AP^T y = b, \quad x = P^T y, \quad (1)$$

where P is the permutation matrix produced by ND. As illustrated in Fig. 2, the reordered matrix AP^T exhibits much more regular block structures compared to the original A , which reduces irregular vector accesses and enhances cache reuse during SpMV.

An alternative strategy would be to form PAP^T , i.e., permuting both rows and columns. However, it is inefficient for matrices stored in the compressed sparse row (CSR) format, which is a widely used format in device simulation. This would require permuting the value array of A whenever the matrix value changes, which is particularly costly since in device simulation one often solves a sequence of systems with the same sparsity but varying values. By storing only AP^T , the column index array can be modified only once in preprocessing, while the value array remains unpermuted in subsequent solves, thus avoiding redundant data movement.

It is important to note that the factorization is performed on PAP^T to minimize fill-ins, but without explicit row permutations. In practice, storing AP^T is sufficient. When doing factorization, accessing the i -th row of PAP^T is simply equivalent to accessing the p_i -th row of AP^T , where p is the permutation vector corresponding to P .



(a) Original A

(b) Reordered A

Fig. 2. Sparsity pattern of a matrix A and the reordered matrix AP^T using nested dissection. The reordering clusters non-zeros into block structures, improving memory locality for SpMV.

C. A Dynamic Strategy for Setting GMRES Tolerance

In the GMRES-IR framework, the setting of inner tolerance τ_{in} has a significant impact on the total number of GMRES iterations. A straightforward approach is to fix a strict tolerance τ_{in} across all steps. However, this choice is suboptimal for two reasons:

- 1) In early IR iterations, the outer iterate is still inaccurate, hence the correction must be relatively accurate and a smaller GMRES tolerance is desirable.
- 2) In late IR iterations, the outer iterate is already accurate, so a fixed tight tolerance wastes Krylov work and runtime.

To improve efficiency, we propose a **dynamic GMRES tolerance** strategy that adapts the stopping criterion of the inner

GMRES to the progress of the outer IR loop. Specifically, at the k -th refinement step, the inner GMRES solves $\mathbf{A}\mathbf{d}^{(k)} = \mathbf{r}^{(k)}$ with a tolerance τ_{in} , which satisfies that

$$\frac{\|\mathbf{r}^{(k)} - \mathbf{A}\mathbf{d}^{(k)}\|_2}{\|\mathbf{r}^{(k)}\|_2} < \tau_{\text{in}}. \quad (2)$$

Then the updated solution becomes $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)}$ and the outer residual satisfies

$$\begin{aligned} \frac{\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k+1)}\|_2}{\|\mathbf{b}\|_2} &= \frac{\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} - \mathbf{A}\mathbf{d}^{(k)}\|_2}{\|\mathbf{b}\|_2} \\ &= \frac{\|\mathbf{r}^{(k)} - \mathbf{A}\mathbf{d}^{(k)}\|_2}{\|\mathbf{b}\|_2} < \tau_{\text{in}} \frac{\|\mathbf{r}^{(k)}\|_2}{\|\mathbf{b}\|_2}. \end{aligned} \quad (3)$$

To ensure convergence of the outer refinement, (3) should be not greater than the outer tolerance τ_{out} . Therefore, it is sufficient to keep

$$\tau_{\text{in}} \leq \tau_{\text{out}} \frac{\|\mathbf{b}\|_2}{\|\mathbf{r}^{(k)}\|_2}. \quad (4)$$

Motivated by this bound, we set the dynamic inner tolerance

$$\tau_{\text{in}} = \min(\tau_{\text{max}}, c \cdot \tau_{\text{out}} \frac{\|\mathbf{b}\|_2}{\|\mathbf{r}^{(k)}\|_2}) \quad (5)$$

with a safety factor $c \in (0, 1]$ and an upper bound τ_{max} to avoid overly loose solves. This strategy yields stricter inner solves when $\|\mathbf{r}^{(k)}\|$ is large in early IR and progressively relaxes the effort as $\|\mathbf{r}^{(k)}\|$ shrinks in late IR, thereby reducing total Krylov iterations without compromising the convergence of the outer refinement.

D. Parallelization Techniques

While the optimizations in Sections III-A to III-C significantly improve the efficiency of the serial solver, large-scale device-simulation problems demand effective parallelization to fully exploit modern multi-core architectures. In our framework, parallelism is primarily introduced into the two dominant computational kernels SpMV and sparse triangular solves (SpTRSV).

Efficient parallelization of SpMV is often hindered by irregular sparsity patterns that cause thread imbalance when rows are assigned uniformly. To mitigate this issue, we partition the workload based on the number of non-zeros per row. Specifically, the rows are grouped into T contiguous blocks, where T is the number of threads. Each block is assigned to a thread and contains approximately $\text{nnz}(\mathbf{A})/T$ non-zeros. This ensures that every thread processes a balanced amount of work while maintaining CSR's contiguous row access.

For SpTRSV, we adopt a coarse-grained parallelization strategy derived from the block structure induced by ND ordering. Since the LU factorization is actually performed on the matrix reordered by ND, the LU factors naturally decompose into independent subdomains and a separator. Fig. 3 illustrates the forward solve $\mathbf{L}\mathbf{x} = \mathbf{b}$ in the case of a one-level ND reordering. The lower triangular factor \mathbf{L} is partitioned into two independent subdomain blocks \mathbf{L}_0 and \mathbf{L}_1 , the separator coupling block $\tilde{\mathbf{L}}$, and a separator block \mathbf{L}_s . The solution \mathbf{x}

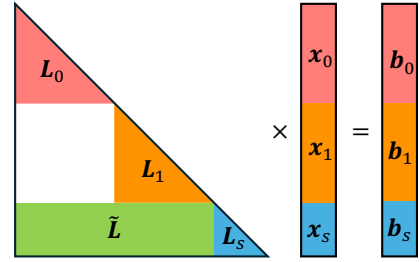


Fig. 3. One-level ND decomposition of the triangular system $\mathbf{L}\mathbf{x} = \mathbf{b}$.

and right-hand side (RHS) \mathbf{b} are partitioned accordingly as $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_s)^\top$ and $(\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_s)^\top$.

In this formulation, the systems $\mathbf{L}\mathbf{x}_0 = \mathbf{b}_0$ and $\mathbf{L}\mathbf{x}_1 = \mathbf{b}_1$ are completely independent and can be solved concurrently with 2 threads. Once \mathbf{x}_0 and \mathbf{x}_1 are obtained, their contributions are propagated into the separator equations, forming $\mathbf{L}_s\mathbf{x}_s = \mathbf{b}_s - \tilde{\mathbf{L}}(\mathbf{x}_0, \mathbf{x}_1)^\top$. Though it is solved sequentially, the constructing of the RHS is an SpMV, which can be computed in parallel efficiently with the proposed load-aware balancing strategy. When the number of threads T is greater than 2, a $\log_2 T$ -level ND ordering can yield T independent subdomains. Therefore, the proposed approach can be easily extended to accommodate more threads.

For the backward substitution phase, the dependency order is reversed. The separator equations must be solved first, followed by the subdomain systems in parallel.

IV. EXPERIMENTAL RESULTS

We have implemented a parallel GMRES-IR solver in C++ with OpenMP based on the proposed algorithms and conducted several numerical experiments to validate its performance. For LU factorization, the Eigen library [18] is chosen because it provides a templated interface that allows both standard and multiple-component precision types. For the dynamic strategy for setting GMRES tolerance, the safety factor c is set to 0.1 and τ_{max} is set to 10^{-4} . To compute the ND ordering, the widely used library METIS [19] is adopted. The baseline for comparison is a direct solver using DD-precision LU factorization, which is also implemented using the Eigen library.

The test data consists of 10 matrices extracted from realistic semiconductor device simulation problems from our industrial partner, covering condition numbers from $2.5\text{E}18$ up to $1.5\text{E}57$. To obtain reference solutions, we set the exact solution vector \mathbf{x}^* to all ones and construct the RHS as $\mathbf{b} = \mathbf{A}\mathbf{x}^*$. These matrices are representative of the highly ill-conditioned systems encountered in practical workflows and provide a challenging benchmark for both accuracy and efficiency. All experiments are conducted on a Linux server equipped with Xeon Gold 6230R CPUs and 128 GB memory.

A. Results of Serial Computing

We begin by evaluating the accuracy of direct solutions obtained with a double-precision LU factorization. Table I summarizes the information of the test matrices, which are sorted by their condition numbers. The table also reports the relative residuals $r = \frac{\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2}{\|\mathbf{b}\|_2}$, the relative errors $e = \frac{\|\mathbf{x} - \mathbf{x}^*\|_\infty}{\|\mathbf{x}^*\|_\infty}$, and

TABLE I
CHARACTERISTICS OF TEST MATRICES AND RESULTS OF
DOUBLE-PRECISION LU FACTORIZATION

Matrix	N	$\frac{nnz(\mathbf{A})}{N}$	$\kappa(\mathbf{A})$	Double-Precision LU		
				r	e	$T(s)$
SEU_HM62V8100	3.7E5	14.5	2.5E18	3.7E-15	8.4E-01	18.2
SuperJunction_IGBT_1	4.6E3	15.6	2.7E22	4.2E-09	3.1E-02	0.01
SuperJunction_IGBT_2	5.5E3	15.8	5.0E22	4.2E-09	2.2E-02	0.02
FinFET_LDMOS_gts	1.1E5	29.4	2.9E23	7.1E-10	1.1E+00	4.09
SIC_TrenchMOS	5.7E3	15.3	1.2E25	2.5E-16	4.4E-01	0.01
SIC_VDMOS_TID	1.8E4	11.1	3.9E35	6.3E-02	7.9E+05	0.07
SIC_Diode	2.3E3	16.3	1.6E55	1.4E-11	5.8E+02	0.01
SIC_nIGBT	1.0E4	12.3	1.7E56	2.5E-09	9.9E+01	0.04
SIC_piIGBT	1.0E4	16.0	1.2E57	2.4E-10	2.9E+01	0.03
SIC_TrenchMOS	5.2E3	15.4	1.5E57	5.0E-10	1.1E-02	0.01

N , $\frac{nnz(\mathbf{A})}{N}$, $\kappa(\mathbf{A})$ denote the size, the average number of non-zeros per row, and the condition number of the test matrix, respectively.

the runtimes T for solving. Although the residuals are small in most cases, the relative errors are unacceptably large, in some cases exceeding 10^5 . This implies that the double-precision LU factorization fails to produce physically meaningful solutions.

Next, we compare our mixed-precision GMRES-IR solver with a direct DD-precision LU factorization baseline. The results are summarized in Table II. For fair comparison, the reported runtime of DD-precision LU factorization corresponds to the sum of the factorization time and the forward/backward substitutions, whereas the runtime of GMRES-IR includes the double-precision LU factorization time and the iterative refinement phase. From the table we can see, both approaches achieve essentially the same level of accuracy. The relative residuals remain on the order of 10^{-25} to 10^{-31} and the relative errors are consistently below 10^{-10} , confirming that the proposed framework preserves the accuracy of high-precision direct methods. However, the runtime differences are substantial. Across all test matrices, GMRES-IR converges in no more than 25 iterations and is on average $5.4\times$ faster than the baseline, with individual speedups ranging from $2.8\times$ to $14.1\times$. These results demonstrate that our solver achieves the accuracy of DD-precision LU factorization at only a fraction of the computational cost.

TABLE II
ACCURACY AND RUNTIME COMPARISON BETWEEN DD-PRECISION LU
FACTORIZATION AND MIXED-PRECISION GMRES-IR

Matrix	DD-Precision LU				GMRES-IR (Ours)			
	r	e	$T(s)$	Iter	r	e	$T(s)$	Sp
SEU_HM62V8100	3.1E-31	9.0E-16	264	25	2.6E-31	1.1E-15	28.0	9.4
SuperJunction_IGBT_1	2.9E-25	8.5E-19	0.10	2	2.3E-25	8.6E-19	0.02	4.9
SuperJunction_IGBT_2	3.0E-25	1.6E-18	0.11	2	1.8E-25	1.1E-18	0.02	4.6
FinFET_LDMOS_gts	5.8E-26	1.1E-16	67.2	2	1.7E-26	5.4E-17	4.77	14.1
SIC_TrenchMOS	2.7E-32	4.9E-17	0.10	2	1.2E-32	5.9E-18	0.02	4.0
SIC_VDMOS_TID	5.3E-18	9.8E-11	0.59	2	2.1E-29	2.7E-11	0.09	6.1
SIC_Diode	1.1E-27	6.6E-14	0.04	2	5.0E-28	2.2E-14	0.01	4.3
SIC_nIGBT	6.4E-27	2.8E-16	0.30	9	7.8E-27	3.8E-17	0.07	4.0
SIC_piIGBT	2.3E-26	1.0E-13	0.30	21	3.8E-27	3.0E-14	0.11	2.8
SIC_TrenchMOS	4.9E-26	1.2E-19	0.14	2	8.0E-27	2.8E-20	0.02	6.4
Average								5.4

Iter denotes the total number of GMRES iterations. Sp denotes the speedup ratio of our GMRES-IR solver against the DD-precision LU factorization.

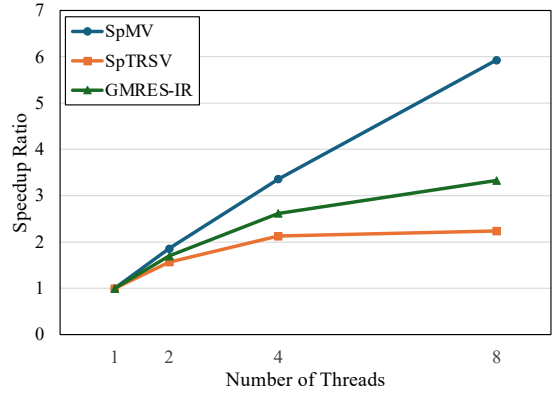


Fig. 4. Average speedup of SpMV, SpTRSV, and GMRES-IR across 10 device simulation test cases with increasing number of threads.

B. Results of Parallel Computing

We next evaluate the parallel scalability of our solver on multi-core processors. Since efficient parallel implementations of sparse LU factorization are already well studied in existing direct solvers, we do not focus on it in this work. Instead, we examine the scalability of GMRES-IR, which is the main target of our parallelization effort. Fig. 4 reports the average speedup across all 10 test cases as the number of threads increases, for two critical kernels (SpMV and SpTRSV), as well as the GMRES-IR excluding the LU factorization time.

The results show that SpMV scales nearly linearly, reaching a speedup of $5.9\times$ with 8 threads, consistent with the effectiveness of our load-aware balancing strategy. In contrast, SpTRSV exhibits more limited parallelism due to its inherent data dependencies, with a maximum speedup of $2.2\times$. As a consequence, the end-to-end GMRES-IR solver achieves a parallel speedup of $3.3\times$ on 8 threads. These results demonstrate that while SpTRSV remains a scalability bottleneck, the proposed optimization skills enable GMRES-IR to be benefited significantly from shared-memory parallelism in practice.

V. CONCLUSION

In this paper, we propose a parallel mixed-precision GMRES-IR solver for extremely ill-conditioned systems in device simulation. By combining a double-precision LU preconditioner with multiple-component precision GMRES iterations, and incorporating novel optimization skills such as SIMD-friendly basis reordering, nested dissection ordering, and a dynamic strategy for setting inner tolerance, the solver achieves high accuracy at reduced computational cost. Parallelization of SpMV and SpTRSV further improves the efficiency remarkably. Experimental results show that the proposed solver achieves $5.4\times$ speedup on average, over DD-precision LU factorization in serial computing and further $3.3\times$ parallel speedup on 8 threads. These results demonstrate that the proposed framework provides an accurate, efficient, and plug-and-play solution for the challenging device-simulation problems.

REFERENCES

- [1] Q. Fan, P. A. Forsyth, J. R. F. McMacken, and W.-P. Tang, "Performance issues for iterative solvers in device simulation," *SIAM J. Sci. Comput.*, vol. 17, no. 1, pp. 100–117, 1996.

- [2] O. Schenk, M. Hagemann, and S. Rollin, "Recent advances in sparse linear solver technology for semiconductor device simulation matrices," in *SISPAD'03*, 2003, pp. 103–108.
- [3] D.-W. Wang, Q. Zhang, H. Wan, and W.-S. Zhao, "Finite element approach based numerical framework for device simulator," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 44, no. 8, pp. 3197–3207, 2025.
- [4] Y. Hida, X. Li, and D. Bailey, "Algorithms for quad-double precision floating point arithmetic," in *ARITH'01*, 2001, pp. 155–162.
- [5] Y. Dou, Y. Lei, G. Wu, S. Guo, J. Zhou, and L. Shen, "FPGA accelerating double/quad-double high precision floating-point applications for ExaScale computing," in *ICS'10*, New York, NY, USA, 2010, p. 325–336.
- [6] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with PARDISO," *Future Generation Computer Systems*, vol. 20, no. 3, pp. 475–487, 2004.
- [7] X. S. Li, "An overview of SuperLU: Algorithms, implementation, and user interface," *ACM Trans. Math. Softw.*, vol. 31, no. 3, p. 302–325, 2005.
- [8] C. B. Moler, "Iterative refinement in floating point," *J. ACM*, vol. 14, no. 2, p. 316–321, 1967.
- [9] Å. Björck, "Iterative refinement of linear least squares solutions I," *BIT Numerical Mathematics*, vol. 7, no. 7, pp. 257–278, 1979.
- [10] E. Carson and N. J. Higham, "A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems," *SIAM J. Sci. Comput.*, vol. 39, no. 6, pp. A2834–A2856, 2017.
- [11] —, "Accelerating the solution of linear systems by iterative refinement in three precisions," *SIAM J. Sci. Comput.*, vol. 40, no. 2, pp. A817–A847, 2018.
- [12] T. Ina, Y. Idomura, T. Imamura, S. Yamashita, and N. Onodera, "Iterative methods with mixed-precision preconditioning for ill-conditioned linear systems in multiphase CFD simulations," in *ScalA'21*, 2021, pp. 1–8.
- [13] P. Amestoy, A. Buttari, N. J. Higham, J.-Y. L'Excellent, T. Mary, and B. Vieublé, "Five-precision GMRES-based iterative refinement," *SIAM J. Matrix Anal. Appl.*, vol. 45, no. 1, pp. 529–552, 2024.
- [14] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. SIAM, 2023.
- [15] D. Mukunoki, K. Ozaki, T. Ogita, and T. Imamura, "Accurate matrix multiplication on binary128 format accelerated by Ozaki scheme," in *ICPP'21*, 2021.
- [16] T. J. Dekker, "A floating-point technique for extending the available precision," *Numerische Mathematik*, vol. 18, pp. 224–242, 1971.
- [17] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms, Volume 2*. Addison Wesley, 1981.
- [18] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," <http://eigen.tuxfamily.org>, 2010.
- [19] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Scientific Comput.*, vol. 20, no. 1, pp. 359–392, 1998.