Beyond Moore's Law: Solving Linear Systems with Integer Arithmetic

AFAE talk on: "An Integer Arithmetic-Based Sparse Linear Solver" by Iwashita et al.

Giulia Lionetti   Alberto Taddei

Abstract

Abstract. As conventional semiconductor scaling declines, new computing architectures are being explored and may favor integer arithmetic over complex floating-point units. This paper introduces **int-GMRES**, a GMRES variant whose main iteration kernels operate in fixed-point/integer arithmetic. Embedded within an iterative refinement framework, the approach achieves convergence behavior comparable to standard double-precision GMRES on a set of sparse test problems, especially when preconditioning is used to control numerical range and overflow risk.

1. The Problem

The Hardware Motivation: For decades, scientific computing has relied on Floating-Point (FP) arithmetic. FP is convenient because its exponent provides a large *dynamic range* (it can represent very large and very small magnitudes without manual rescaling). However, the end of Moore's Law is driving interest in alternative devices (e.g., SFQ circuits). In early-stage technologies, FP units can be prohibitively expensive in terms of power and complexity, while integer arithmetic is comparatively efficient.

The Numerical Challenge: The goal is to solve the sparse linear system:

$$Ax = b \tag{1}$$

using *almost exclusively* integer arithmetic. Unlike FP, fixed-point/integer representations have a **fixed range**. This introduces two critical failure modes for iterative solvers such as GMRES:

1. **Overflow:** intermediate values (especially dot products, norms, and scaled updates) can exceed the word length (e.g., 64-bit), producing invalid results.
2. **Precision loss:** to prevent overflow, operands are shifted right (division by 2^{β}), which discards low-order bits and increases quantization error.

2. Why is this Interesting?

This work reframes precision as an **algorithmic responsibility** rather than a purely hardware feature: the solver must explicitly manage scaling and range.

It asks: *Can advanced sparse linear algebra be made viable on hardware where floating-point is absent or too costly?* A positive answer would open the door to scientific computing on ultra-low-power systems or architectures with limited arithmetic capabilities.

3. State of the Art (Pre-Paper)

Prior to this work, the dominant paradigm was **mixed-precision** computing. Well-known approaches (e.g., Göddeke, Anzt, Haidar) perform most work in low-precision floating-point (e.g., FP16/FP32) and recover accuracy via higher-precision correction.

The Gap: Most prior methods still assume that some floating-point arithmetic is available for the inner kernels. This paper targets a more radical constraint: **the main GMRES kernels (matvecs, dot products, norms, rotations) must run in fixed-point/integer arithmetic**, with floating-point used only around the loop (residual computation, scaling, and final update).

4. The "Big Idea": A 3-Layer Architecture

To make integer arithmetic viable for GMRES, the authors propose a layered strategy that controls range and progressively injects accuracy.

Layer 1: Iterative Refinement (Outer Loop)

Instead of solving $Ax = b$ in a single run, the method refines an approximation through successive correction solves. Each refinement computes a residual in FP, scales it, and calls the integer-based solver to obtain a correction:

$$x_{\text{final}} = \tilde{x}^{(1)} + \tilde{x}^{(2)} + \cdots + \tilde{x}^{(k)}. \tag{2}$$

This keeps the integer solver focused on *scaled* subproblems and limits the magnitude of intermediates.

Layer 2: Matrix Decomposition

The coefficient matrix is represented as a sum of integer matrices with power-of-two weights:

$$A \approx \bar{A}_0 + 2^{-\bar{\alpha}_1}\bar{A}_1 + \cdots + 2^{-\bar{\alpha}_p}\bar{A}_p. \tag{3}$$

Early refinements can use only the dominant term ($\bar{A}_0$), adding smaller contributions later to improve accuracy without immediately increasing overflow risk.

Layer 3: int-GMRES (Inner Kernel)

Within each refinement, GMRES(m) is executed using fixed-point numbers $Q_{dm.df}$ for vectors and scalars (with *df* fractional bits), while the sparse matrices are stored as integers. The crucial ingredient is operation-specific bit shifting to trade overflow safety against accuracy.

5. The Algorithm & Technical Solution

The core innovation is the **operand shift strategy**. In fixed-point arithmetic, multiplying two int64 values can exceed the 64-bit range. To avoid overflow, operands may be shifted right before multiplication, and the result is shifted to preserve the chosen number of fractional bits *df*.

The implementation exploits GMRES structure to reduce accuracy-sacrificing shifts:

- **Normalized Krylov vectors:** since $\|\bar{v}_i\| \approx 1$, many leading bits are zero, lowering overflow risk in dot products and updates.
- **Givens rotations:** rotation coefficients satisfy $|\sin|, |\cos| \leq 1$, so multiplications involving them are naturally bounded.

Algorithm 1 int-GMRES(m) inside one refinement step (high-level)

```
1: Inputs: integer matrices $\bar{A}_\ell$, shifts $\bar{\alpha}_\ell$, $df$, FP right-hand side $b^{(k)}$, FP initial guess $x^{(k)}$
2: Output: FP correction $x^{(k)}$
3: $r_0 \leftarrow b^{(k)} - A^{(k)}x^{(k)}$                      ▷ (FP) residual
4: $v_1 \leftarrow r_0/\|r_0\|$                                    ▷ (FP) normalize
5: $\bar{v}_1 \leftarrow$ CastToFixedPoint$(v_1, df)$               ▷ (INT)
6: for $j = 1 \to m$ do
7:     $\bar{w} \leftarrow \bar{A}^{(k)}\bar{v}_j$       ▷ (INT) matvec, uses $\bar{A}_0 + \sum 2^{-\bar{\alpha}_\ell}\bar{A}_\ell$
8:     for $i = 1 \to j$ do
9:         $\bar{h}_{i,j} \leftarrow (\bar{w}, \bar{v}_i)$      ▷ (INT) dot product (with shifts if needed)
10:        $\bar{w} \leftarrow \bar{w} - \bar{h}_{i,j}\bar{v}_i$    ▷ (INT) orthogonalize
11:    end for
12:    $\bar{h}_{j+1,j} \leftarrow \|\bar{w}\|$                   ▷ (INT) norm
13:    $\bar{v}_{j+1} \leftarrow \bar{w}/\bar{h}_{j+1,j}$          ▷ (INT) division
14:    ApplyGivensRotations$(\bar{H})$                             ▷ (INT)
15: end for
16: $y \leftarrow \arg\min \| \|r_0\|e_1 - H_m y\|$               ▷ (FP) small least-squares
17: $x^{(k)} \leftarrow x^{(k)} + \sum_{i=1}^{m} y_i v_i$         ▷ (FP) update
```

6. The Role of Preconditioning

The paper identifies **ILU(0) (Incomplete LU)** preconditioning as a critical enabler.

Normally, preconditioning accelerates convergence. Here, it also plays a structural role: **overflow risk reduction**. By applying M^{-1} (with $M \approx A$), the effective operator $M^{-1}A$ is better behaved, so intermediate vectors and dot products tend to have smaller magnitudes.

- **No preconditioning:** larger intermediate magnitudes require aggressive operand shifts (e.g., $\beta = 16$), which discards low-order bits and degrades accuracy.
- **With ILU:** intermediate magnitudes are reduced, allowing most shifts to be set to $\beta = 0$ (as reported in the paper), preserving substantially more fixed-point information.

7. Experimental Results

The method was tested on matrices from the *SuiteSparse Matrix Collection*. The target relative residual was 10^{-8} (measured in double precision).

Case 1: Without Preconditioning. Table 1 shows that int-GMRES often matches FP64 GMRES, but can require more iterations on harder instances due to accuracy loss from shifting.

Table 1: Iterations: No Preconditioning ($m = 30$)

Dataset	Double (FP64)	int-GMRES	Diff
atmosmodj	2,100	2,100	0%
atmosmodl	420	420	0%
cage14	30	60	+100%
wang3	510	**630**	+24%

Case 2: With ILU Preconditioning. Table 2 highlights that ILU makes the integer-based solver closely track FP64 convergence.

Table 2: Iterations: With ILU Preconditioning ($m = 30$)

Dataset	Double (FP64)	int-GMRES	Diff
atmosmodj	300	300	0%
atmosmodl	120	120	0%
cage14	30	60	+1 restart
wang3	120	120	0%

8. Conclusion and Discussion

This work demonstrates that **GMRES can be executed with integer/fixed-point kernels** when embedded in iterative refinement, achieving convergence comparable to double-precision GMRES on the tested problems.

Key points:

- **Feasibility:** Krylov solvers can operate on integer-centric hardware when accuracy is recovered through an outer refinement loop.
- **Replacing the exponent:** floating-point range handling is replaced by explicit scaling and operation-specific bit shifting.
- **Preconditioning matters twice:** it improves convergence and reduces overflow risk, allowing milder (or zero) shifts and higher effective accuracy.