Integer Arithmetic GMRES: Theoretical Reference

Slide-by-Slide Concept Mapping

Giulia Lionetti Alberto Taddei

AFAE – January 2026

Purpose

This document maps each slide to its theoretical foundations, providing formulas and explanations to prepare for technical questions during the presentation.

1 Slides 3–5: Motivation – Why Integer Arithmetic?

Slide Content

- Novel hardware (SFQ circuits, neuromorphic) may only support integer arithmetic
- Integer operations: ~ 5 pJ vs. FP operations: ~ 50 pJ
- Trade-off table: FP has automatic range, integer has fixed range
- Question: Can we do *real* scientific computing without floating-point?

Theoretical Foundation

Floating-Point Representation:

$$x_{\mathrm{FP}} = \pm m \times 2^e$$

where m is the mantissa and e is the exponent. The exponent provides *automatic dynamic range.*

Fixed-Point Representation:

$$x_{\mathrm{fixed}} = \pm \frac{I}{2^f}$$

where I is an integer and f is the number of fractional bits. Format notation: $\mathrm{Q}_{m.f}$ where m is integer bits, f is fractional bits.

Representable Range:

$$\begin{aligned} \text{Min value:} &\quad 2^{-f} \\ \text{Max value:} &\quad 2^m - 2^{-f} \\ \text{Quantization step:} &\quad \Delta = 2^{-f} \end{aligned}$$

Key Limitation: Without the exponent, numbers outside $[2^{-f}, 2^m]$ cause overflow or underflow. The algorithm must manage range explicitly.

Potential Questions

Q1: Why is overflow more problematic in integer arithmetic than floating-point?
A: In FP, the exponent adjusts automatically to accommodate large values. In fixed-point, values exceeding 2^m wrap around or saturate, causing catastrophic errors.
Q2: What's the energy advantage quantitatively?
A: $\sim 10\times$ lower energy per operation. This comes from simpler hardware: no exponent alignment, normalization, or special case handling (NaN, Inf).
Q3: Can you give the exact precision of FP64 vs. your fixed-point format?
A: FP64 has $p = 53$ bits of precision (unit roundoff $u = 2^{-53} \approx 10^{-16}$). Our fixed-point uses $f = 30$ fractional bits (quantization step $2^{-30} \approx 10^{-9}$), with 34 integer bits for a total word length of 64 bits.

2 Slides 7–8: Layer 1 – Iterative Refinement

Slide Content

- Compute residual in FP, scale for integer solver, accumulate corrections
- Residual $b' = b - Ax$ shrinks each iteration $\Rightarrow$ scale by $\gamma = 1/\max|b'_i|$
- Result: integer solver always sees magnitudes near 1

Theoretical Foundation

Classical Iterative Refinement:
Given approximate solution $x^{(k)}$ to $Ax = b$:

$$\begin{aligned} r^{(k)} &= b - Ax^{(k)} && \text{(residual)} \\ Ad^{(k)} &= r^{(k)} && \text{(solve for correction)} \\ x^{(k+1)} &= x^{(k)} + d^{(k)} && \text{(update)} \end{aligned}$$

Forward Error Bound: If residual is computed in precision u_r and correction in precision u_f:

$$\frac{\|x^{(k+1)} - x\|}{\|x\|} \lesssim \kappa(A)u_f + \frac{u_r}{u_f}$$

where $\kappa(A) = \|A\|\|A^{-1}\|$ is the condition number.
Modified Refinement for Integer Arithmetic:
Scale the residual before passing to integer solver:

$$\begin{aligned} r^{(k)} &= b - Ax^{(k)} && \text{(compute in FP64)} \\ \gamma^{(k)} &= \frac{1}{\max_i |r_i^{(k)}|} && \text{(scaling factor)} \\ \tilde{b}^{(k)} &= \gamma^{(k)} r^{(k)} && \text{(scaled residual, } \|\tilde{b}^{(k)}\|_\infty = 1\text{)} \\ A\tilde{d}^{(k)} &= \tilde{b}^{(k)} && \text{(solve in integer arithmetic)} \\ x^{(k+1)} &= x^{(k)} + \frac{1}{\gamma^{(k)}}\tilde{d}^{(k)} && \text{(update in FP64)} \end{aligned}$$

Key Property: By construction, $\|\tilde{b}^{(k)}\|_\infty = 1$, ensuring all components are in $[-1, 1]$. This prevents overflow in the integer solver and maximizes utilization of the fixed-point range.

Potential Questions

Q1: Why compute the residual in FP64 instead of integer arithmetic?
A: The residual $r = b - Ax$ involves subtracting nearly equal large numbers (catastrophic cancellation). FP64's 53-bit precision preserves more significant digits than our 30-bit fractional part.
Q2: What if the residual components have very different magnitudes?
A: Scaling by $\max_i |r_i|$ normalizes to $[-1, 1]$, but doesn't equalize component magnitudes. Small components may still be quantized aggressively. This is acceptable because the residual shrinks geometrically, so even with quantization, convergence continues.
Q3: Why does this provide "automatic range control"?
A: As iterations progress, $\|r^{(k)}\| \to 0$ at rate $\approx \rho^k$ where $\rho < 1$ depends on $\kappa(A)$. Scaling ensures each integer solve operates on $O(1)$ values regardless of iteration number, automatically adapting the range.
Q4: How many refinement iterations are needed?
A: Typically 2–5 iterations suffice. Each iteration approximately squares the error (quadratic convergence when $\kappa(A)u_f \ll 1$).

3 Slide 9: Layer 2 – Matrix Decomposition

Slide Content

$$A = \bar{A}_0 + 2^{-\alpha_1}\bar{A}_1 + 2^{-\alpha_2}\bar{A}_2 + \cdots$$

where $\bar{A}_i$ are integer matrices and $2^{-\alpha_i}$ are power-of-2 weights.

- $\bar{A}_0$: most significant (coarse approximation)
- $\bar{A}_1, \bar{A}_2, \ldots$: fine details
- Power-of-2 scaling = cheap bit shifts

Theoretical Foundation

Decomposition Construction:
Given FP matrix A with entries $a_{ij} \in \mathbb{R}$:

1. Find global scale: $s = \max_{ij} |a_{ij}|$
2. Normalize: $A' = A/s$, so $\|A'\|_\infty = 1$
3. Represent each a'_{ij} in binary:

$$a'_{ij} = \sum_{k=0}^{\infty} b_k(i,j) \cdot 2^{-k}$$

   where $b_k(i,j) \in \{0, \pm 1\}$
4. Group bits into blocks:

$$\bar{A}_0 = \{b_0(i,j), \ldots, b_{\alpha_1 - 1}(i,j)\} \qquad \text{(most significant } \alpha_1 \text{ bits)}$$
$$\bar{A}_1 = \{b_{\alpha_1}(i,j), \ldots, b_{\alpha_2 - 1}(i,j)\} \qquad \text{(next } \alpha_2 - \alpha_1 \text{ bits)}$$

Matrix-Vector Product:

$$Ax = s\left(\bar{A}_0 x + 2^{-\alpha_1}\bar{A}_1 x + 2^{-\alpha_2}\bar{A}_2 x + \cdots\right)$$

In practice, use only first K terms:

$$Ax \approx s \sum_{i=0}^{K-1} 2^{-\alpha_i}\bar{A}_i x$$

Computational Cost: Each $\bar{A}_i x$ is integer matrix-vector product. Multiplication by $2^{-\alpha_i}$ is a right bit shift by α_i positions (essentially free).
Approximation Error: If using K terms where $\alpha_K = K \cdot b$ (uniform bit allocation):

$$\|A - A_K\| \leq s \cdot 2^{-Kb}$$

Choosing $K \cdot b \geq 53$ ensures FP64-level accuracy.

Potential Questions

Q1: Why is power-of-2 scaling important?
A: Multiplication/division by 2^k is implemented as a bit shift, which is vastly cheaper than general multiplication. In hardware, shifts may complete in 1 cycle vs. 3–10 cycles for multiplication.
Q2: How do you choose the α_i values?
A: Typically uniform: $\alpha_i = i \cdot b$ where $b = 8$ or $b = 16$. This gives regular structure. Adaptive schemes could allocate more bits where matrix entries have larger magnitude variation.
Q3: Can you progressively add terms during refinement?
A: Yes! Early refinement iterations have large residuals, so low accuracy suffices. Use only $\bar{A}_0$ initially. As $\|r^{(k)}\|$ decreases, add $\bar{A}_1, \bar{A}_2, \ldots$ to maintain progress. This is "progressive accuracy."
Q4: What about sparse matrices?
A: Each $\bar{A}_i$ has the same sparsity pattern as A. The decomposition doesn't create fill-in. Storage is $(K + 1) \times \text{nnz}(A)$ integers.

4 Slides 10–12: Layer 3 – int-GMRES

Slide Content

- Standard GMRES structure: Arnoldi orthogonalization, Givens rotations
- All inner kernels use fixed-point arithmetic: matvec, dots, norms, Givens
- Problem: Dot products and norms overflow easily
- Solution: "Smart shifting" exploits GMRES invariants

Theoretical Foundation

Standard GMRES Algorithm:
To solve $Ax = b$:

1. Compute initial residual: $r_0 = b - Ax_0$, $v_1 = r_0/\|r_0\|$
2. For $j = 1, 2, \ldots, m$:
   (a) $w = Av_j$
   (b) Orthogonalize: for $i = 1, \ldots, j$:
   $$h_{i,j} = \langle w, v_i \rangle$$
   $$w \leftarrow w - h_{i,j} v_i$$
   (c) Normalize: $h_{j+1,j} = \|w\|$, $v_{j+1} = w/h_{j+1,j}$
   (d) Apply Givens rotations to H to maintain upper triangular form
3. Solve least squares problem: $\min_y \|He_1\|r_0\| - y\|$
4. Update: $x = x_0 + V_m y$

Fixed-Point Operations:
In format $\mathrm{Q}_{34.30}$ (34 integer bits, 30 fractional bits, total 64 bits):
Dot Product:

$$\langle w, v \rangle = \sum_{i=1}^{n} w_i v_i$$

Each product $w_i v_i$ produces a 64-bit result. Without shifting, the sum overflows. With shifting:

$$w_i v_i \approx \left\lfloor \frac{w_i}{2^{\beta_1}} \cdot \frac{v_i}{2^{\beta_2}} \right\rfloor \cdot 2^{-(30-\beta_1-\beta_2)}$$

Norm:

$$\|w\| = \sqrt{\sum_{i=1}^{n} w_i^2}$$

Similar overflow risk in computing w_i^2 and summing.
The Shifting Trade-off:
More shift (β large):

- Safer: reduces overflow probability
- Less accurate: effective quantization becomes $2^{-30+\beta}$ instead of 2^{-30}

Less shift (β small):

- Higher precision maintained
- Risk of overflow

Error Accumulation:
Each inner product has error bounded by:

$$|\langle w, v \rangle_{\text{computed}} - \langle w, v \rangle_{\text{exact}}| \lesssim n \cdot 2^{-30+\beta} \cdot \|w\|\|v\|$$

In GMRES, this affects orthogonality: $\langle v_i, v_j \rangle_{\text{computed}} \neq \delta_{ij}$ (Kronecker delta). Loss of orthogonality degrades convergence.

Potential Questions

Q1: Why do dot products overflow in fixed-point but not FP?
A: In FP, each $w_i v_i$ is rounded to FP format with exponent adjustment, keeping magnitude reasonable. In fixed-point, $w_i v_i$ grows like $|w_i||v_i|$, and summing n such terms can exceed the integer bit capacity if w, v are not carefully scaled.
Q2: What is the "smart shifting" strategy?
A: Exploit GMRES structure:

- Krylov vectors: $\|v_j\| = 1$ by construction $\Rightarrow$ leading bits are often zero $\Rightarrow$ β can be small
- Givens coefficients: $c^2 + s^2 = 1 \Rightarrow |c|, |s| \leq 1 \Rightarrow$ no overflow
- Dot products $\langle v_i, v_j \rangle$ with $\|v_i\| = \|v_j\| = 1$ satisfy $|\langle v_i, v_j \rangle| \leq 1 \Rightarrow$ mostly safe

Q3: How much precision is lost compared to FP64 GMRES?
A: FP64 GMRES has effective precision $\sim 10^{-16}$ in inner products. Fixed-point with $f = 30$ and minimal shifting ($\beta \approx$ 0–4) achieves $\sim 10^{-9}$ to 10^{-8}. This is partially compensated by the outer iterative refinement loop.
Q4: Could you use multiple precision levels within GMRES?
A: Yes! The paper uses uniform precision, but one could:

- Accumulate dot products in higher precision ($64 \to 128$ bits)
- Compute Givens rotations in FP32
- Keep Krylov vectors in fixed-point

This is a natural extension and future work direction.

5 Slide 13–14: Algorithm Pseudocode

Slide Content

`int-GMRES` algorithm showing:

- Line 1: r_0, v_1 computed in FP
- Line 2: Cast to fixed-point
- Lines 3–11: Integer loop (matvec, dots, norms, Givens)
- Line 12: Solve LS and update in FP

Theoretical Foundation

Hybrid Precision Strategy:
Use FP64 where essential:

1. Initial residual $r_0 = b - Ax_0$ (catastrophic cancellation risk)
2. Least squares solve for y (small system, $m \times m$ with $m \sim 20$–50)
3. Solution update $x \leftarrow x + Vy$ (accumulating small corrections)

Use integer arithmetic where dominant:

1. Matrix-vector products Av_j ($\sim 90\%$ of flops for sparse A)
2. Arnoldi orthogonalization (dot products, saxpy)
3. Givens rotations

Why This Works:
The integer operations dominate computational cost but don't require highest precision because:

- They operate on $O(1)$ scaled values (no cancellation)
- Errors accumulate slowly due to structure (normalized vectors)
- Outer refinement corrects any degradation

The FP operations are cheap (small m) but critical for accuracy:

- Residual needs high precision to detect convergence
- LS solve determines correction direction
- Update must not lose digits in accumulation

Convergence Criterion:
Stop refinement when:

$$\frac{\|r^{(k)}\|_2}{\|b\|_2} < \tau$$

where $\tau = 10^{-8}$ (target tolerance).
Stop inner GMRES when:

$$\|Hy - e_1\beta\| < \tau_{\text{inner}}\|\tilde{b}\|$$

where $\tau_{\text{inner}} \sim 10^{-2}$–$10^{-4}$ (much looser, since refinement corrects).

Potential Questions

Q1: Why cast to fixed-point after computing v_1?
A: The normalization $v_1 = r_0/\|r_0\|$ in FP ensures $\|v_1\| = 1$ exactly (within FP precision). Casting introduces quantization error:

$$\bar{v}_1 = \text{cast}(v_1) = v_1 + \delta, \quad \|\delta\| \lesssim \sqrt{n} \cdot 2^{-30}$$

This is acceptable because GMRES is stable under small perturbations.
Q2: What prevents the integer loop from accumulating unbounded error?
A: Two factors:

1. GMRES has inherent stability: orthogonalization projects out previous error directions
2. Iteration count m is limited (typically $m = 20$–50), so error growth is $O(m \cdot 2^{-30+\beta})$, which remains manageable

Q3: How expensive is the cast operation?
A: Casting FP64 to fixed-point $Q_{34.30}$:

$$x_{\text{FP64}} = m \times 2^e \qquad \text{(mantissa } m\text{, exponent } e\text{)}$$
$$x_{\text{fixed}} = \lfloor m \times 2^{e+30} \rfloor \qquad \text{(shift mantissa to align fractional part)}$$

This is a shift and truncate, very cheap (few cycles).
Q4: Could the entire algorithm run in integer, including residual and LS?
A: Theoretically yes, but:

- Residual would require very wide integers (128+ bits) to avoid cancellation errors
- LS solve on H (upper triangular) involves divisions, which are expensive in integer and lose precision

The hybrid approach is pragmatic: use FP where it's clearly superior, integer where it dominates cost.

6 Slide 15: Preconditioning

Slide Content

- Standard reason: faster convergence
- Integer arithmetic reason: reduces overflow risk
- Better conditioning $\Rightarrow$ smaller values $\Rightarrow$ fewer shifts $\Rightarrow$ higher precision
- Use ILU(0) in integer arithmetic

Theoretical Foundation

Preconditioned System:
Instead of solving $Ax = b$, solve:

$$M^{-1}Ax = M^{-1}b$$

where $M \approx A$ is easy to invert.
Effect on Condition Number:

$$\kappa(M^{-1}A) \ll \kappa(A)$$

This reduces GMRES iterations: $m_{\text{precond}} \ll m_{\text{unprecond}}$.
Effect on Value Magnitudes:
Consider $Ax = b$ where A is ill-conditioned. During GMRES, Krylov vectors span:

$$\mathcal{K}_m = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}$$

If $\|A\| \gg 1$, then $\|A^k r_0\|$ grows rapidly, forcing aggressive shifting to prevent overflow. With preconditioning, $\|M^{-1}A\| \approx O(1)$, so Krylov vectors remain $O(1)$ in magnitude.
ILU(0) Preconditioner:
Incomplete LU factorization with zero fill-in:

$$M = LU \quad \text{with sparsity pattern}(L) = \text{sparsity pattern}(A_{\text{lower}})$$

For integer implementation:

1. Compute ILU(0) in FP64 during setup
2. Represent L, U using matrix decomposition:

$$L = \bar{L}_0 + 2^{-\alpha_1}\bar{L}_1 + \cdots$$

3. Forward/backward solves use integer operations

Cost Analysis:
Setup: $O(\text{nnz}(A))$ FP64 operations (done once)
Per iteration: $2 \times O(\text{nnz}(L) + \text{nnz}(U))$ integer operations (forward + backward solve)
The setup cost is amortized over multiple solves (important for parameter studies, time-stepping).

Potential Questions

Q1: Why does preconditioning reduce overflow in integer arithmetic specifically?
A: Overflow occurs when intermediate values exceed 2^{34} (integer bit capacity). Without preconditioning, $\|A^k r_0\|$ can grow exponentially if $\rho(A) > 1$ (spectral radius). Preconditioning clusters eigenvalues near 1, preventing this growth.
Q2: Can you quantify the improvement?
A: For a matrix with $\kappa(A) = 10^6$:

- Unpreconditioned: May need $\beta \sim$ 10–15 shift to prevent overflow $\Rightarrow$ effective precision $\sim 10^{-5}$
- ILU(0) preconditioned with $\kappa(M^{-1}A) = 10^2$: Can use $\beta \sim$ 0–4 $\Rightarrow$ effective precision $\sim 10^{-8}$

Q3: Why not use a more sophisticated preconditioner?
A: ILU(0) is a proof-of-concept. More sophisticated options:

- ILU(k) with $k > 0$: More fill-in, better conditioning, but harder to implement in integer
- Multigrid: Excellent for PDEs, requires coarse grid operators (complex in integer)
- Incomplete Cholesky for SPD systems

The paper demonstrates feasibility; production code would optimize preconditioner choice.
Q4: Does preconditioning change the solution?
A: No. Mathematically, $M^{-1}Ax = M^{-1}b \iff Ax = b$. Numerically, the computed solution may differ slightly due to rounding, but converges to the same result within tolerance.

7 Slides 16–20: Experimental Results

Slide Content

- 10 sparse matrices from SuiteSparse, target tolerance 10^{-8}
- Fixed-point: WL $= 64$, $d_f = 30$
- Without preconditioning: mixed results (1.0×–2.0× iterations)
- With ILU(0): most cases identical to double GMRES
- Convergence curves show int-GMRES tracks double closely (with preconditioning)

Theoretical Foundation

Iteration Count Analysis:
Let m_d = iterations for double GMRES, m_i = iterations for int-GMRES.
Without Preconditioning:

$$\frac{m_i}{m_d} = 1.0 \text{ to } 2.0$$

The variation depends on matrix properties:

- Well-conditioned matrices: $\frac{m_i}{m_d} \approx 1$ (fixed-point precision sufficient)
- Moderately ill-conditioned: $\frac{m_i}{m_d} > 1$ (precision loss slows convergence)

With ILU(0):

$$\frac{m_i}{m_d} \approx 1.0$$

The preconditioner eliminates the precision gap by improving conditioning.
Convergence Rate Theory:
GMRES convergence for a system with condition number κ satisfies:

$$\frac{\|r_m\|}{\|r_0\|} \lesssim \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m$$

With rounding errors of magnitude ϵ:

$$\frac{\|r_m\|}{\|r_0\|} \lesssim \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m + m\epsilon\kappa$$

The second term (error accumulation) dominates when $m\epsilon\kappa \sim 1$, i.e., when:

$$m \sim \frac{1}{\epsilon\kappa}$$

For fixed-point with $\epsilon \sim 2^{-30+\beta}$ and $\kappa = 10^3$:

$$m \lesssim \frac{10^{9-\beta}}{10^3} = 10^{6-\beta}$$

This explains why very long iteration sequences can stagnate without preconditioning.
Backward Error:
The computed solution $\hat{x}$ satisfies:

$$(A + \Delta A)\hat{x} = b$$

where:

$$\frac{\|\Delta A\|}{\|A\|} \lesssim m \cdot 2^{-30+\beta} \cdot \kappa(A)$$

This bounds how close we are to solving a nearby system.

Potential Questions

Q1: Why do some matrices show identical convergence without preconditioning?
A: These matrices (e.g., atmosmodj) are naturally well-conditioned or well-scaled. Their Krylov vectors remain $O(1)$, allowing minimal shifting ($\beta \approx 0$), so fixed-point precision $\sim 10^{-9}$ suffices for the same convergence as FP64.
Q2: For wang3, why does slowdown occur without preconditioning?
A: This matrix has worse conditioning or scaling. Without preconditioning, overflow prevention requires $\beta \sim 6$–8, reducing effective precision to $\sim 10^{-7}$. The accumulated errors slow convergence by $\sim 24\%$.
Q3: How do you measure convergence in practice?
A: After each refinement iteration k:

1. Compute residual in FP64: $r^{(k)} = b - Ax^{(k)}$
2. Check: $\|r^{(k)}\|_2/\|b\|_2 < 10^{-8}$?
3. If yes, stop. If no, continue.

This ensures convergence is measured accurately despite integer solver imprecision.
Q4: What about final solution accuracy?
A: The paper reports relative residual, not forward error $\|x_{\text{computed}} - x_{\text{exact}}\|$. For well-conditioned systems:

$$\frac{\|x_{\text{computed}} - x_{\text{exact}}\|}{\|x_{\text{exact}}\|} \lesssim \kappa(A)\frac{\|r\|}{\|b\|}$$

So achieving $\|r\|/\|b\| < 10^{-8}$ gives solution accuracy $\sim 10^{-8}\kappa(A)$. For $\kappa \sim 10^3$, this is $\sim 10^{-5}$ forward error.

8 Slide 21: What They Contributed

Slide Content

1. Working integer-GMRES implementation
2. Iterative refinement framework for integer arithmetic
3. Operation-specific shift strategy exploiting GMRES invariants
4. Empirical evidence: ILU preconditioning is essential

Main insight: Precision management moves from hardware to algorithm.

Theoretical Foundation

Algorithmic Precision Management:
Traditional (FP): Hardware handles range via exponent
Novel (integer): Algorithm handles range via three mechanisms:

1. *Outer loop scaling:* Iterative refinement normalizes residuals to $O(1)$
2. *Representation:* Matrix decomposition adapts precision progressively
3. *Inner loop shifting:* Smart bit shifts exploit structural properties

Comparison to Mixed-Precision FP:
Mixed-precision iterative refinement (Göddeke 2007, Carson & Higham 2018):

- Factorize in FP32 ($u_f = 2^{-24}$)
- Compute residual in FP64 ($u_r = 2^{-53}$)
- Update in FP64 ($u = 2^{-53}$)

Achieves FP64 accuracy with $\sim 75\%$ cost of full FP64 (since factorization dominates).
This paper:

- Factorize in integer ($u_f \sim 2^{-30+\beta}$)
- Compute residual in FP64 ($u_r = 2^{-53}$)
- Update in FP64 ($u = 2^{-53}$)

Achieves FP64 accuracy with $\sim 10\times$ lower energy (potential, pending hardware measurements).
Key Theoretical Advance:
Extends iterative refinement theory to non-FP arithmetic:
Classical result: If $u_f < 1/\kappa(A)$, refinement converges.
Integer extension: If effective precision $2^{-f+\beta}$ and $\kappa(M^{-1}A) \cdot 2^{-f+\beta} < 1$, then preconditioned integer refinement converges.
This shows preconditioning isn't just beneficial, it's *necessary* for convergence in integer arithmetic.

Potential Questions

Q1: Is this the first integer solver?
A: No. Integer linear algebra has been studied for:

- Cryptography (exact arithmetic modulo p)
- Computational geometry (avoiding FP errors)
- Embedded systems (low-power, no FPU)

But this is the first *Krylov subspace method* (GMRES) working entirely in integer arithmetic within the inner loop.
Q2: Why is GMRES harder than direct methods in integer?
A: Direct methods (Gaussian elimination) have $O(n^3)$ cost but fixed structure: predictable intermediate value ranges. Krylov methods are $O(n \cdot \mathrm{nnz}(A) \cdot m)$ but adaptive: the Krylov subspace depends on the input, making range prediction harder.
Q3: What about conjugate gradient (CG) instead of GMRES?
A: CG (for symmetric positive definite systems) would be simpler:

- No Arnoldi orthogonalization (cheaper)
- Three-term recurrence instead of full orthogonalization
- But requires SPD matrix (restrictive)

GMRES is more general (any nonsingular A), so the paper demonstrates the harder case.
Q4: Could you combine this with other mixed-precision strategies?
A: Absolutely. Natural extensions:

- Use bfloat16 instead of integer for some operations (if hardware supports)
- Use integer for matvec, FP16 for Arnoldi, FP32 for Givens
- Adaptive precision: start with low precision, increase as residual shrinks

The three-layer architecture provides a framework for exploring this design space.

9 Slide 22: Limitations

Slide Content

- Only tested on moderately conditioned problems
- Several tuning parameters (d_f, shifts, decomposition depth)
- No actual performance/energy measurements
- Theoretical convergence guarantees unclear

But: demonstrates feasibility.

Theoretical Foundation

Open Theoretical Questions:

1. **Convergence rate bound:** What is the guaranteed convergence rate as a function of $\kappa(M^{-1}A)$, d_f, β?

   Classical GMRES: $\|r_m\|/\|r_0\| \lesssim 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m$

   Integer GMRES: Need to characterize how fixed-point errors affect this bound.

2. **Optimal shifting strategy:** The paper uses heuristic shifts. Can we derive optimal β_i minimizing error accumulation subject to overflow probability $< \delta$?

3. **Preconditioning sufficiency:** What condition number is "good enough"? Empirically, $\kappa(M^{-1}A) \sim 10^2$–$10^3$ works. Can we prove a threshold?

4. **Stagnation detection:** How to detect when fixed-point precision limits further progress? Need criteria to switch from integer to FP solver mid-refinement.

Practical Challenges:

1. **Parameter tuning:** Users must choose:

   - Word length WL and fractional bits d_f
   - Decomposition depth K and bit allocation α_i
   - Shift amounts β for each operation type
   - Restart parameter m for GMRES

   Current work provides guidelines, but not automatic tuning.

2. **Matrix-dependent behavior:** Some matrices converge identically to FP64, others require 2× iterations. Need a priori prediction: which matrices are suitable for integer arithmetic?

3. **Performance modeling:** Energy savings depend on target hardware:

   - Standard CPUs: integer ops $\sim 2\times$ faster than FP (modest gain)
   - FPGAs/ASICs: integer ops $\sim 10\times$ faster (significant gain)
   - Neuromorphic chips: integer only (enabling technology)

   Need actual implementations to validate claimed benefits.

Extension Opportunities:

1. **Other Krylov methods:** BiCGStab, QMR, etc.

2. **Eigenvalue problems:** Arnoldi for eigenvalues

3. **Nonlinear systems:** Newton-Krylov methods

4. **Time integration:** Implicit ODE/PDE solvers

Potential Questions

Q1: Why only moderately conditioned problems?
A: For $\kappa(A) \sim 10^6$ or higher:

- Even with ILU(0), $\kappa(M^{-1}A) \sim 10^4$
- Fixed-point precision $2^{-30} \sim 10^{-9}$ becomes insufficient
- Need $\kappa \cdot \epsilon \ll 1$, so $\kappa \lesssim 10^5$ for $\epsilon = 10^{-9}$

For highly ill-conditioned problems, need either wider word length ($WL = 128$) or better preconditioners.
Q2: How sensitive is performance to parameter choices?
A: Moderately sensitive:

- d_f too small: quantization errors dominate
- d_f too large: integer bits insufficient, overflow risk
- β too large: precision loss
- β too small: overflow

Sweet spot: $d_f = 28$–32, $\beta = 0$–4 for $WL = 64$ with preconditioning.
Q3: Could you measure performance on current hardware?
A: Partially. Standard CPUs have both FP and integer units, so integer isn't dramatically faster (maybe 1.5×–2×). The real test requires:

- FPGA implementation (can optimize integer datapath)
- ASIC design (custom integer units)
- SFQ or neuromorphic hardware (future technology)

Q4: What prevents production use today?
A: Three barriers:

1. Lack of hardware showing significant speedup (chicken-and-egg problem)
2. Parameter tuning complexity (need automated tools)
3. Limited testing (only 10 matrices, no comparison to other solvers)

This paper is a proof-of-concept, not production-ready software.

10 Key Formulas Summary

Essential Formulas for Questions

1. Fixed-Point Representation:

$$x = \pm\frac{I}{2^f}, \quad I \in \mathbb{Z}, \quad \text{range: } [2^{-f}, 2^m - 2^{-f}]$$

2. Quantization Error:

$$|x_{\text{quant}} - x_{\text{exact}}| \leq 2^{-f}/2$$

3. Iterative Refinement:

$$r^{(k)} = b - Ax^{(k)}, \quad \gamma^{(k)} = 1/\max|r_i^{(k)}|, \quad x^{(k+1)} = x^{(k)} + \frac{1}{\gamma^{(k)}}d^{(k)}$$

4. Matrix Decomposition:

$$A = \sum_{i=0}^{K-1} 2^{-\alpha_i}\bar{A}_i, \quad \bar{A}_i \text{ integers}$$

5. Fixed-Point Product with Shift:

$$t_r = \lfloor (t_1 \gg \beta_1) \cdot (t_2 \gg \beta_2) \rfloor \gg (d_f - \beta_1 - \beta_2)$$

6. GMRES Convergence:

$$\frac{\|r_m\|}{\|r_0\|} \lesssim 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m + m\epsilon\kappa$$

where ϵ is effective precision ($2^{-30+\beta}$ for integer).

7. Condition Number Effect:

$$\frac{\|\Delta x\|}{\|x\|} \lesssim \kappa(A)\frac{\|\Delta b\|}{\|b\|}$$

8. Overflow Prevention Condition: For dot product $\sum w_i v_i$ with $\|w\|, \|v\| \leq 1$: Choose β such that $n \cdot 2^{2(d_f-\beta)} < 2^m$ (integer capacity).