Karlsruhe Institute of Technology

Accelerating GMRES with mixed precision for high-performance GPUs

José I. Aliaga, Hartwig Anzt, Thomas Grützmacher, Enrique S. Quintana-Ortí, and Andrés E. Tomás
Steinbuch Centre for Computing (SCC)

EXASCALE COMPUTING PROJECT

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

Idea of CB-GMRES

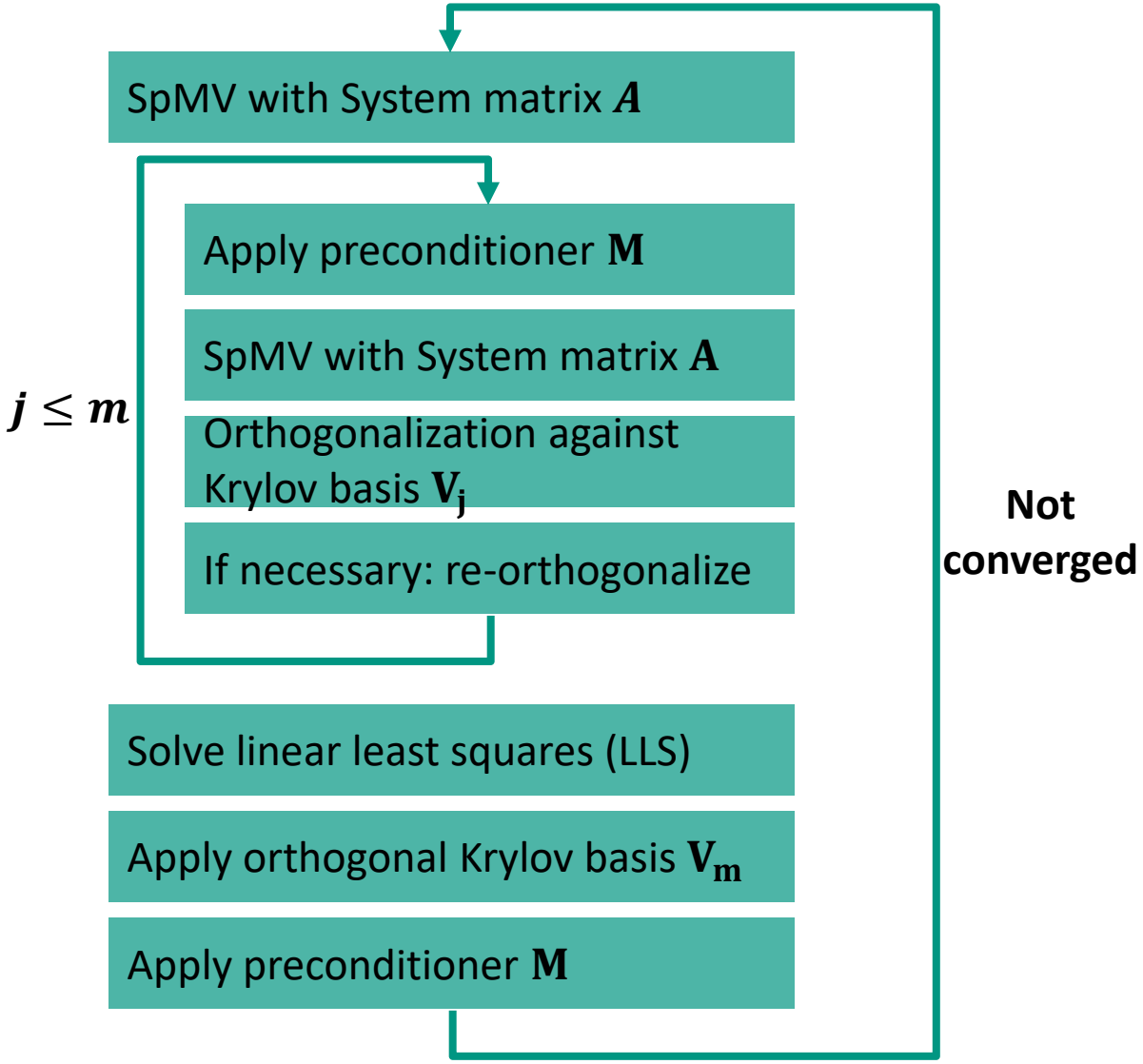
1. Compute $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, and $v := r_0/\beta$. Set $V_1 = [v]$
2. **for** $j := 1, 2, \ldots, m$
3. Compute $w := A(M^{-1}v)$
4. $\omega := \|w\|_2$
5. Orthogonalize $h_{1:j,j} := V_j^T w$, $w := w - V_j h_{1:j,j}$
6. $h_{j+1,j} := \|w\|_2$
7. **if** $(h_{j+1,j} < \eta\omega)$ **then**
8. Re-orthogonalize $u := V_j^T w$, $w := w - V_j u$
9. $h_{1:j,j} := h_{1:j,j} + u$
10. $h_{j+1,j} := \|w\|_2$
11. **endif**
12. **if** $(h_{j+1,j} = 0)$ **or** $(h_{j+1,j} < \eta\omega)$ **then** set $m := j$ and **go to step 17**, **endif**
13. $v := w/h_{j+1,j}$
14. Set $V_{j+1} := [V_j,\ v]$
15. **endfor**
16. Define the $(m+1) \times m$ Hessenberg matrix $\bar{H}_m = (h_{ij})_{1 \le i \le m+1, 1 \le j \le m}$
17. Compute y_m the minimizer of $\|\beta e_1 - \bar{H}_m y\|_2$ and $x_m := x_0 + M^{-1}(V_m y_m)$
18. **if** satisfied **then Stop**, **else** set $x_0 := x_m$ and **go to step 1**, **endif**

Restarted GMRES algorithm with CGS-re-orthogonalization

Idea of CB-GMRES

1. Compute $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, and $v := r_0/\beta$. Set $V_1 = [v]$
2. **for** $j := 1, 2, \ldots, m$
3. Compute $w := A(M^{-1}v)$
4. $\omega := \|w\|_2$
5. Orthogonalize $h_{1:j,j} := V_j^T w$, $w := w - V_j h_{1:j,j}$
6. $h_{j+1,j} := \|w\|_2$
7. **if** $(h_{j+1,j} < \eta\omega)$ **then**
8. Re-orthogonalize $u := V_j^T w$, $w := w - V_j u$
9. $h_{1:j,j} := h_{1:j,j} + u$
10. $h_{j+1,j} := \|w\|_2$
11. **endif**
12. **if** $(h_{j+1,j} = 0)$ **or** $(h_{j+1,j} < \eta\omega)$ **then** set $m := j$ and **go to step 17**, **endif**
13. $v := w/h_{j+1,j}$
14. Set $V_{j+1} := [V_j,\ v]$
15. **endfor**
16. Define the $(m+1) \times m$ Hessenberg matrix $\bar{H}_m = (h_{ij})_{1 \le i \le m+1, 1 \le j \le m}$
17. Compute y_m the minimizer of $\|\beta e_1 - \bar{H}_m y\|_2$ and $x_m := x_0 + M^{-1}(V_m y_m)$
18. **if** satisfied **then Stop**, **else** set $x_0 := x_m$ and **go to step 1**, **endif**

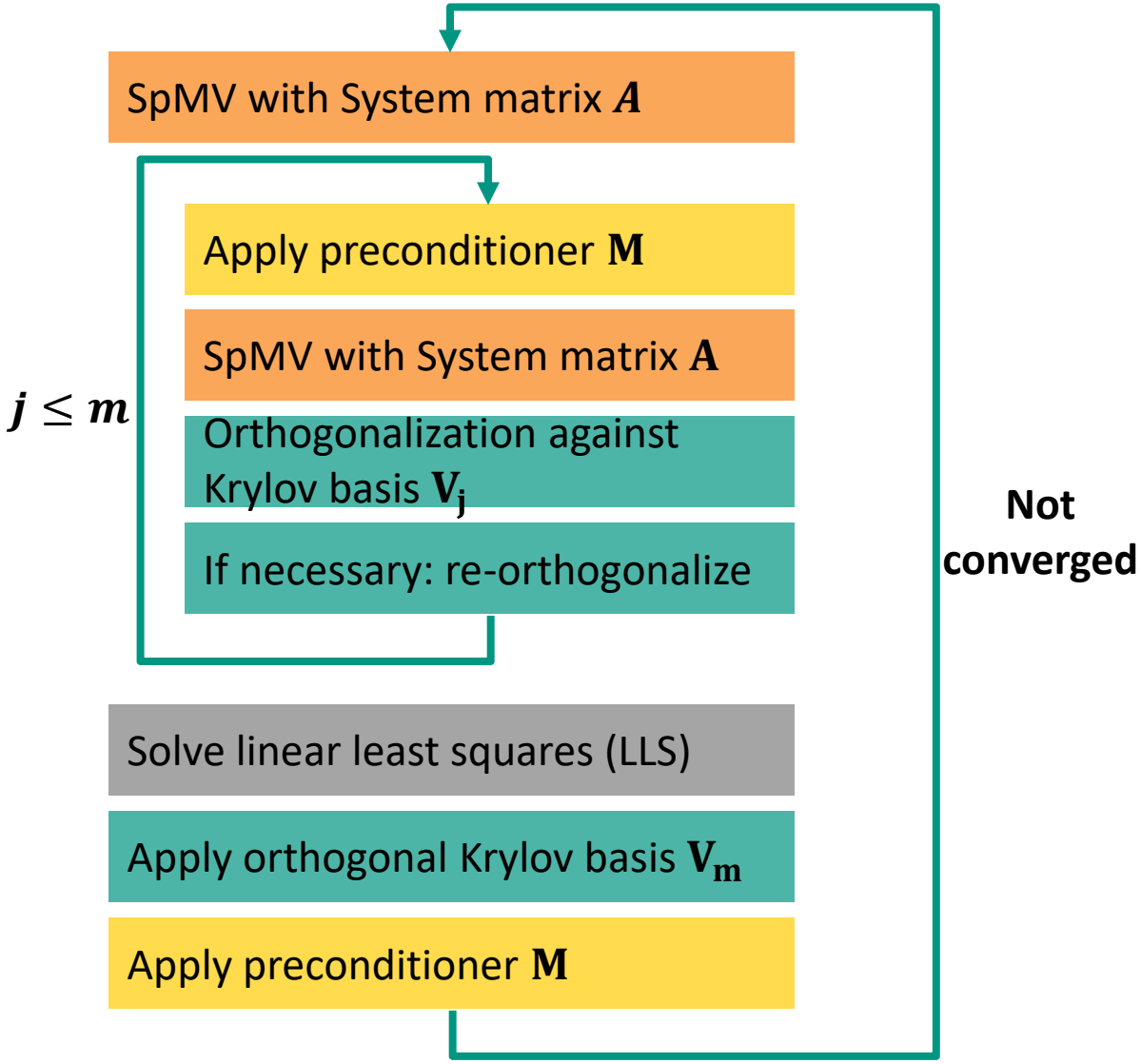
Restarted GMRES algorithm with CGS-re-orthogonalization



Idea of CB-GMRES

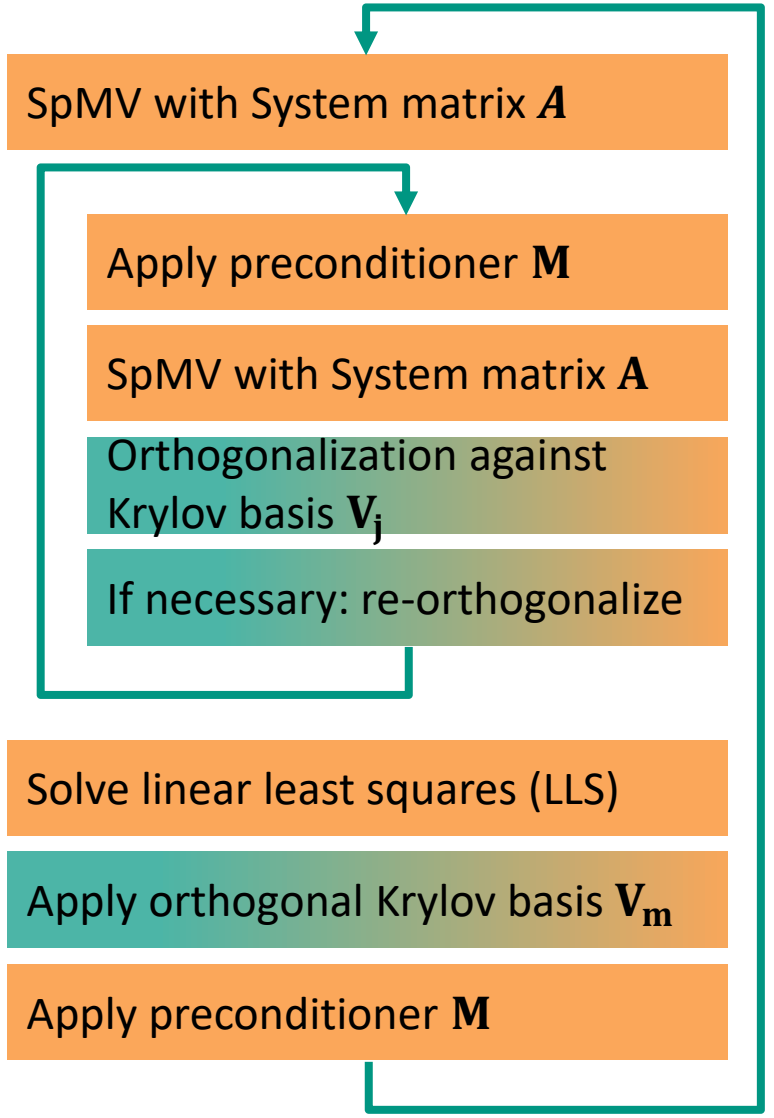
1. Compute $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, and $v := r_0/\beta$. Set $V_1 = [v]$
2. **for** $j := 1, 2, \ldots, m$
3. Compute $w := A(M^{-1}v)$
4. $\omega := \|w\|_2$
5. Orthogonalize $h_{1:j,j} := V_j^T w$, $w := w - V_j h_{1:j,j}$
6. $h_{j+1,j} := \|w\|_2$
7. **if** $(h_{j+1,j} < \eta\omega)$ **then**
8. Re-orthogonalize $u := V_j^T w$, $w := w - V_j u$
9. $h_{1:j,j} := h_{1:j,j} + u$
10. $h_{j+1,j} := \|w\|_2$
11. **endif**
12. **if** $(h_{j+1,j} = 0)$ **or** $(h_{j+1,j} < \eta\omega)$ **then** set $m := j$ and **go to step 17**, **endif**
13. $v := w/h_{j+1,j}$
14. Set $V_{j+1} := [V_j,\ v]$
15. **endfor**
16. Define the $(m+1) \times m$ Hessenberg matrix $\bar{H}_m = (h_{ij})_{1 \le i \le m+1, 1 \le j \le m}$
17. Compute y_m the minimizer of $\|\beta e_1 - \bar{H}_m y\|_2$ and $x_m := x_0 + M^{-1}(V_m y_m)$
18. **if** satisfied **then Stop**, **else** set $x_0 := x_m$ and **go to step 1**, **endif**

Restarted GMRES algorithm with CGS-re-orthogonalization



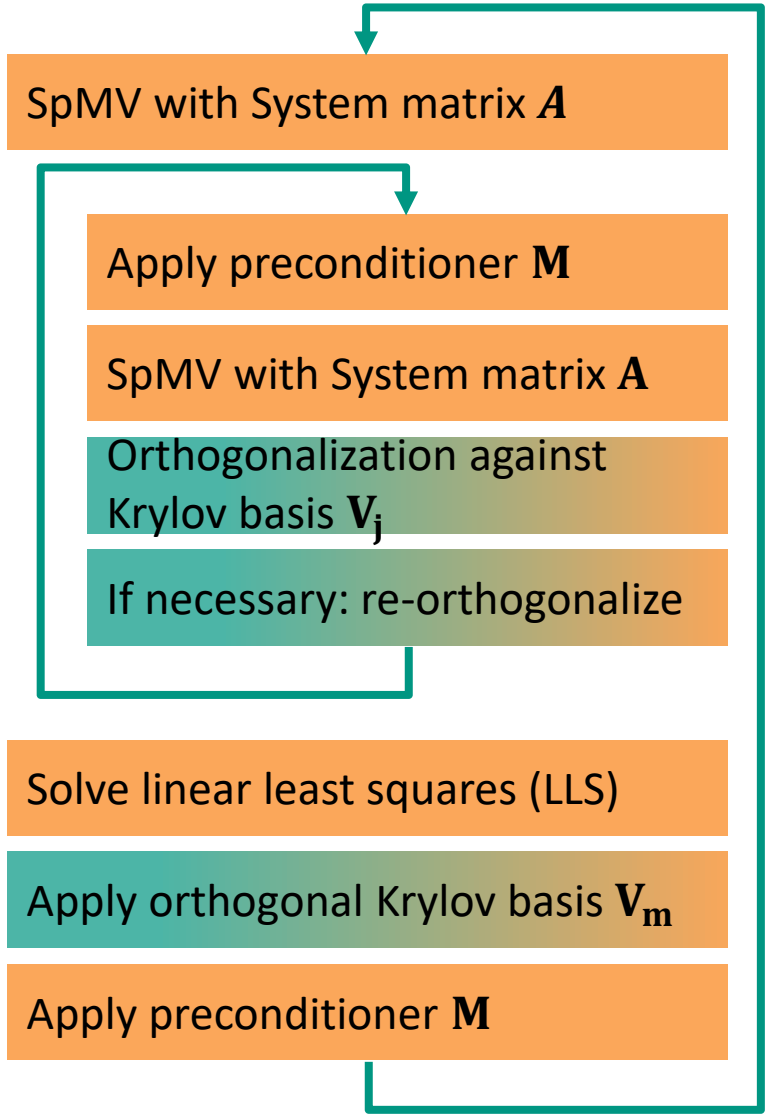
Idea of CB-GMRES

- **Decouple** memory precision from arithmetic precision
- Use IEEE Double precision for all arithmetic operations
- Store Krylov search directions in lower precision



Idea of CB-GMRES

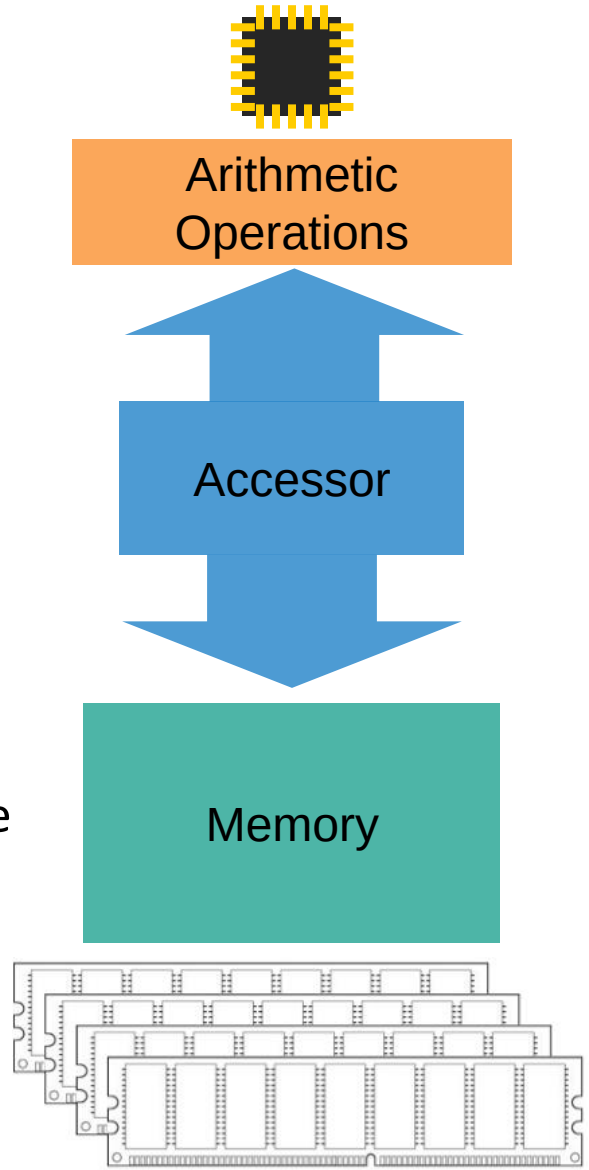
- **Decouple** memory precision from arithmetic precision
- Use IEEE Double precision for all arithmetic operations
- Store Krylov search directions in lower precision
  - *Search directions are no longer DP-orthogonal*
  - *"Does not matter":*
    the solution approximation is generated in the perturbed subspace;
  - *We should not see major convergence degradation*
  - *In the worst case we need a few more search directions (more iterations)*



The Accessor

Insert layer between arithmetic precision and memory precision:
The Accessor

- Hides the actual storage precision from the implementation
  - ➔ Allows changing of memory precision without modifying the algorithm
- Uses static polymorphism (templates) to avoid runtime overhead
- Converts memory to arithmetic precision and back on the fly (in registers)
  - ➔ Imposes a small overhead that is neglectable because we are memory bound

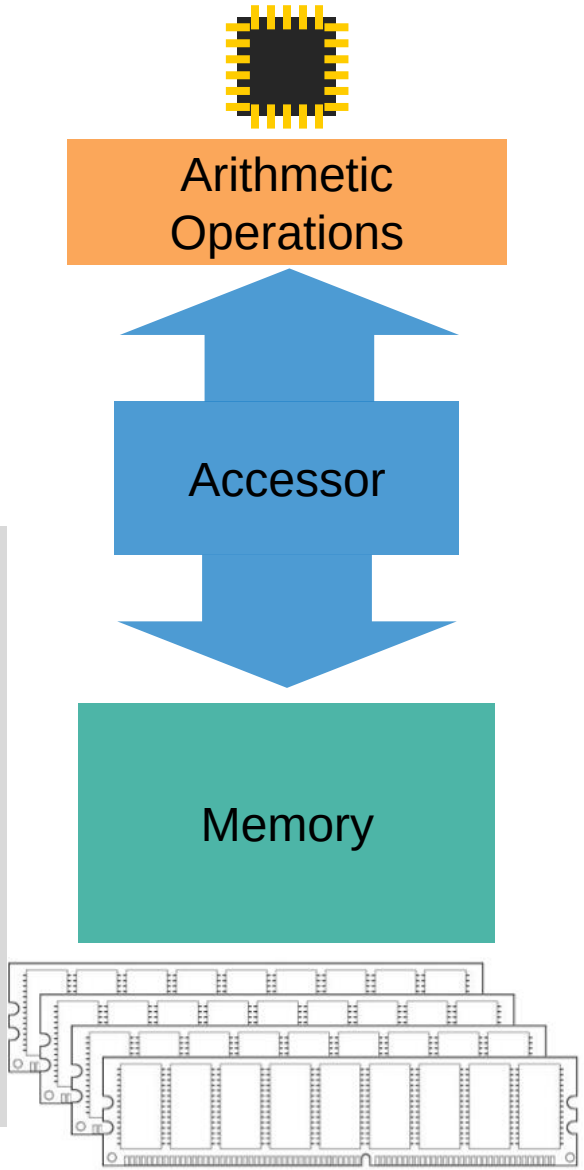


The Accessor

Insert layer between arithmetic precision and memory precision:
The Accessor

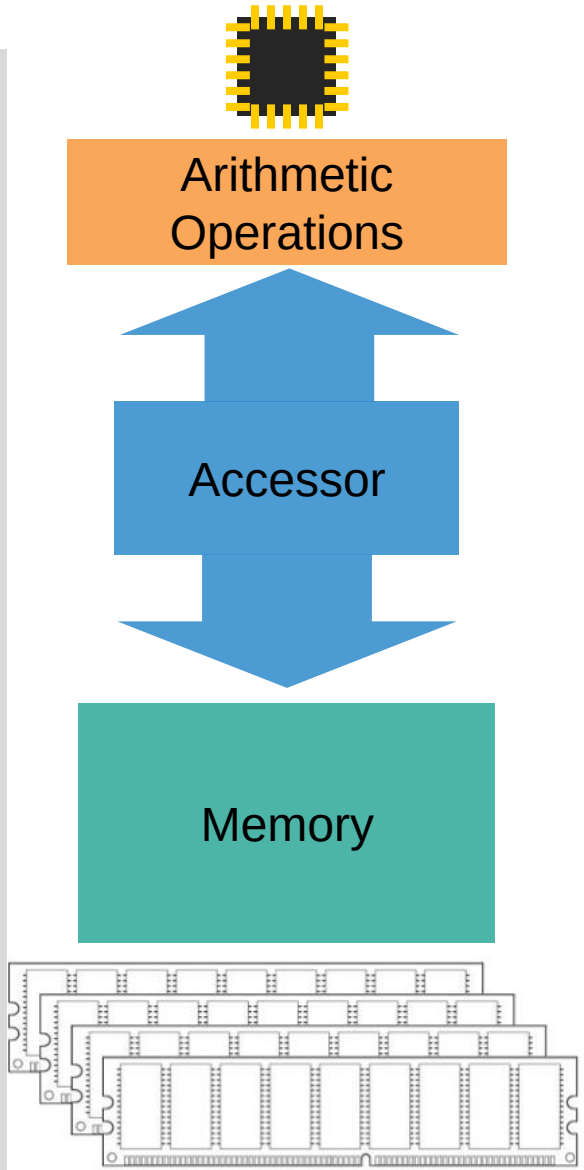
- Hides the actual storage precision from the implementation
  ➔ Allows changing of memory precision without modifying the algorithm

```
template <typename AccessorA, typename AccessorB,
    typename AccessorC>
void vector_add(AccessorA a, AccessorB b,
    AccessorC result)
{

    for (std::size_t i = 0; i < a.length(0); ++i) {
        result(i) = a(i) + b(i);
    }
}
```



The Accessor

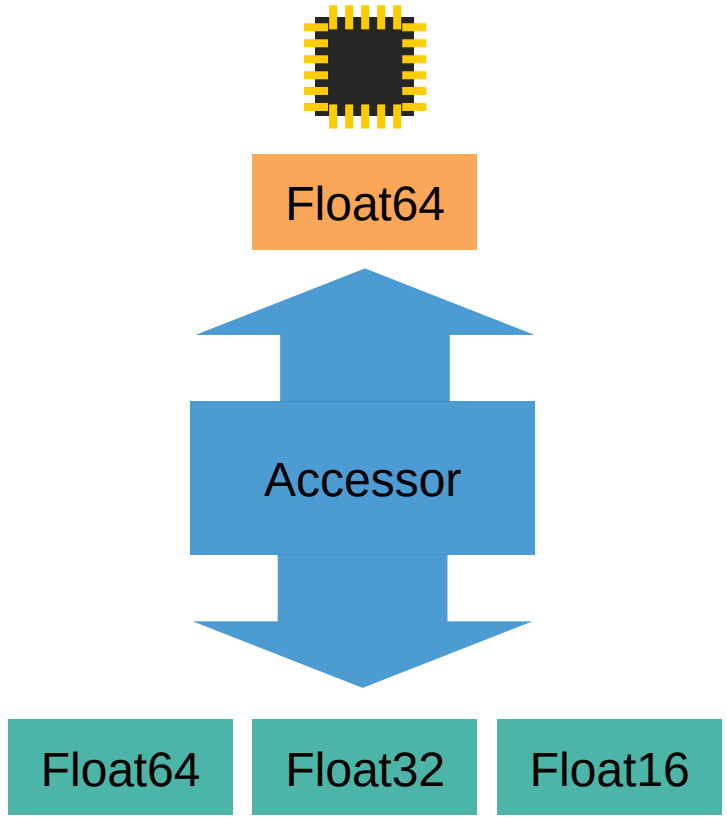
```
template <typename AccessorA, typename AccessorB,
    typename AccessorC>
void vector_add(gko::acc::range<AccessorA> a,
    gko::acc::range<AccessorB> b,
    gko::acc::range<AccessorC> result)
{
    static_assert(decltype(a)::dimensionality == 1
        && decltype(a)::dimensionality
            == decltype(b)::dimensionality
        && decltype(a)::dimensionality
            == decltype(result)::dimensionality,
        "Dimensionalities must match!");
    assert(a.length(0) == b.length(0)
        && a.length(0) == result.length(0));

    for (std::size_t i = 0; i < a.length(0); ++i) {
        result(i) = a(i) + b(i);
    }
}
```

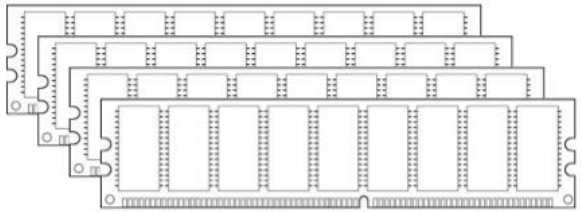


Memory compressions used in CB-GMRES

Floating-point ↔ Float64

- Use regular cast function to convert between floating point values
- Overhead: Cast function





Memory compressions used in CB-GMRES

Floating-point ↔ Float64

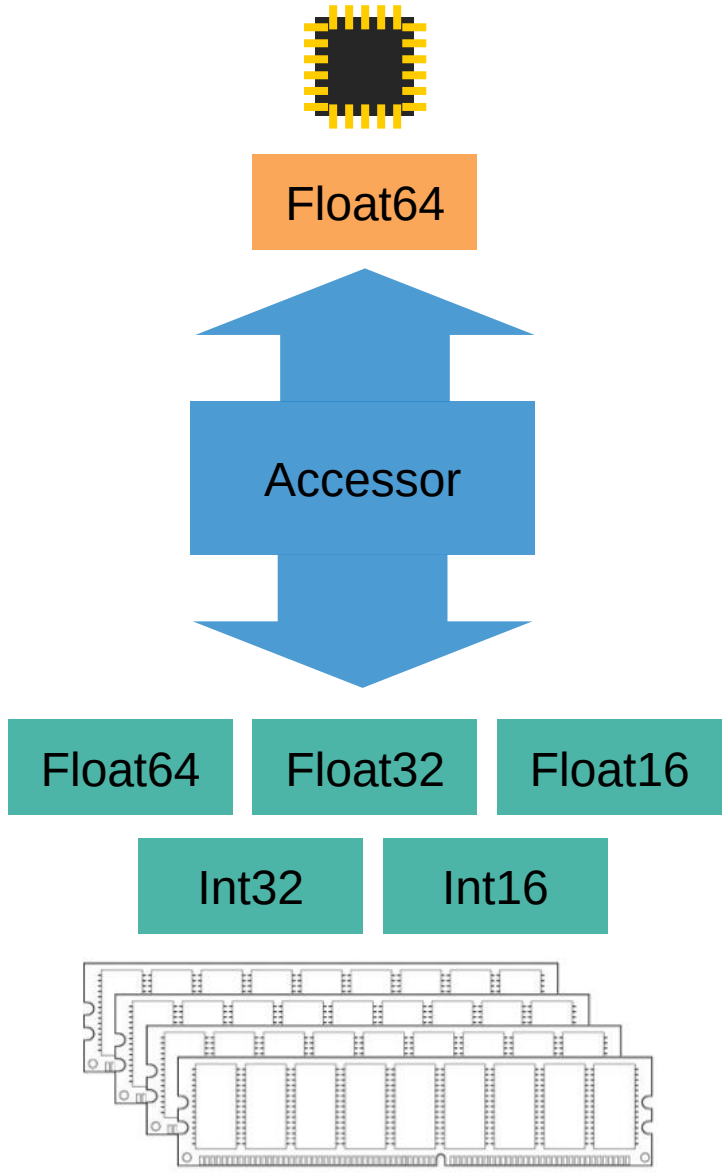
- Use regular cast function to convert between floating point values
- Overhead: Cast function

Integer ↔ Float64

- Since Krylov vectors are normalized, each value ∈ $[-1,1]$
- Each vector has its own scalar value defined as:

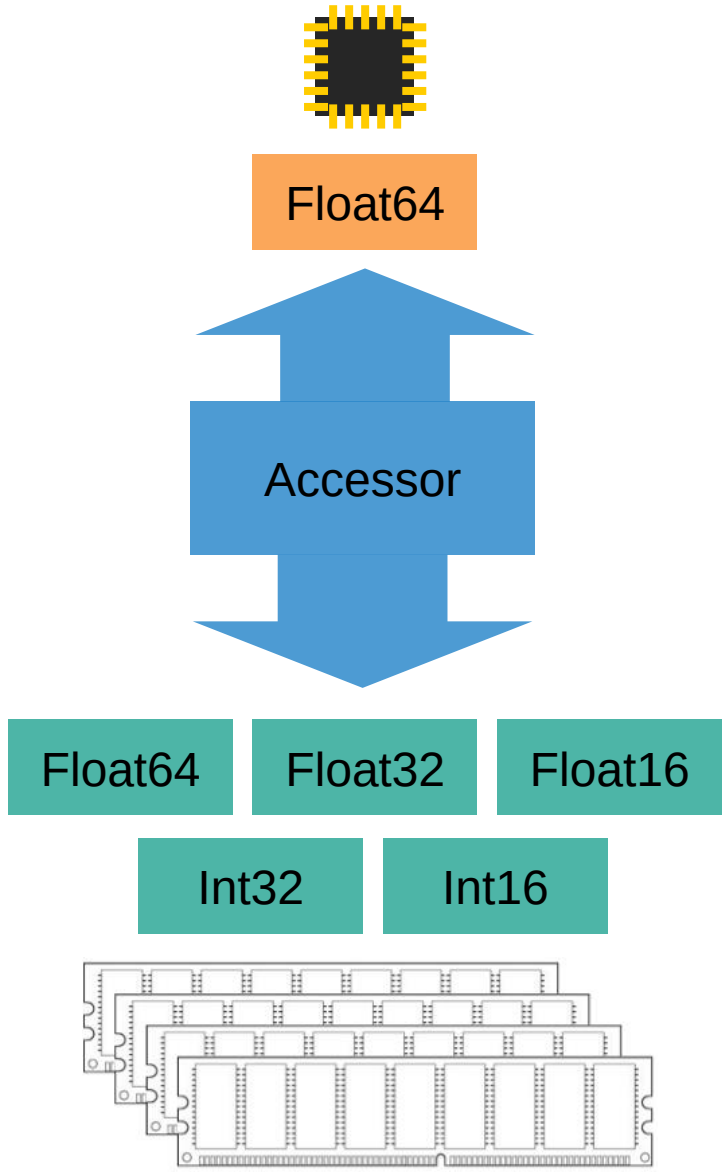
$$\sigma_j = \frac{\|v_j\|_\infty / \|v_j\|_2}{\max\_\text{int}}$$

- Overhead
  - Compute $\|v_j\|_\infty$ and store σ_j
  - For each read & store, scalar must be multiplied
  - For each Krylov vector, additional scalar must be read

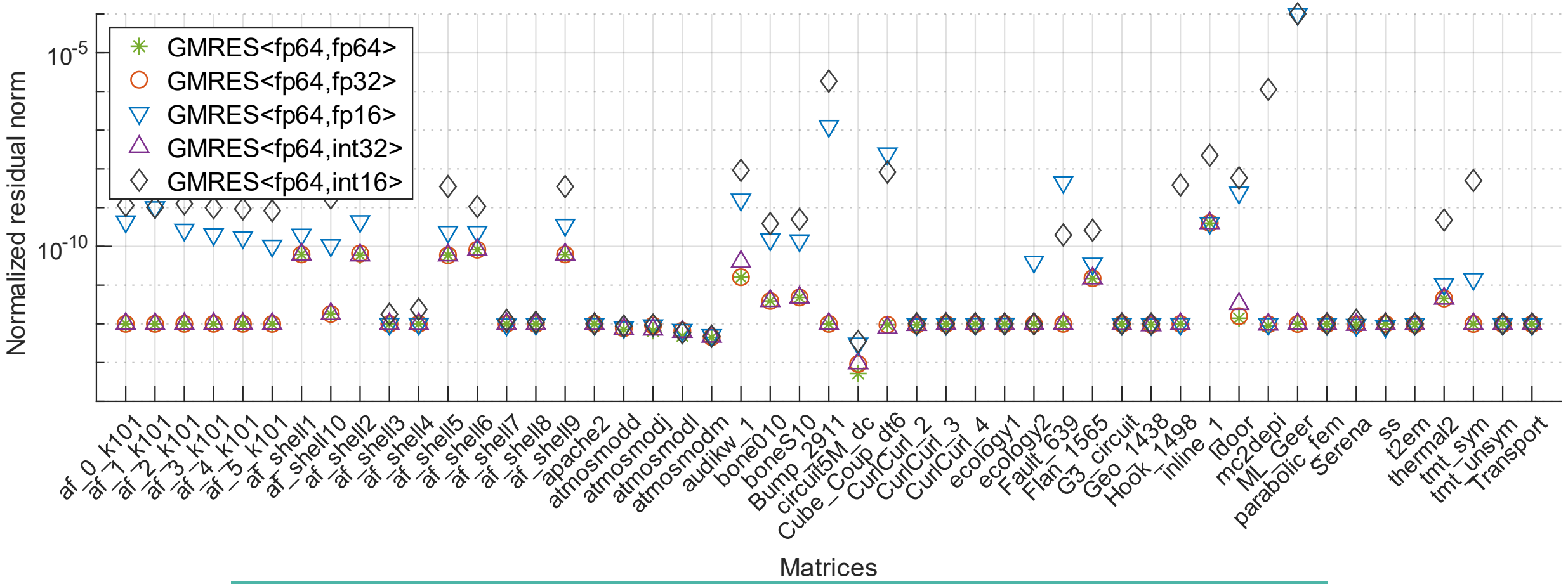


The Setup for CB-GMRES

- Stopping criterion: $\|A \cdot x\| < 10^{-12} \cdot \|b\|$
  or 50 000 iterations
- Preconditioner: Jacobi (chosen to not dominate runtime)
- Initial guess: $\vec{0}$; restart iteration = 100
- 49 matrices from SuiteSparse Matrix Collection
- Krylov basis have their memory precision decoupled from the arithmetic precision
  - Notation: **GMRES<fp64,fp32>**
  - All other parts of GMRES (the matrix, preconditioner, RHS, …) use float64 as memory and arithmetic precision
- Implemented in Ginkgo (https://ginkgo-project.github.io/)
- Benchmarks run on a Summit node with a NVIDIA V100

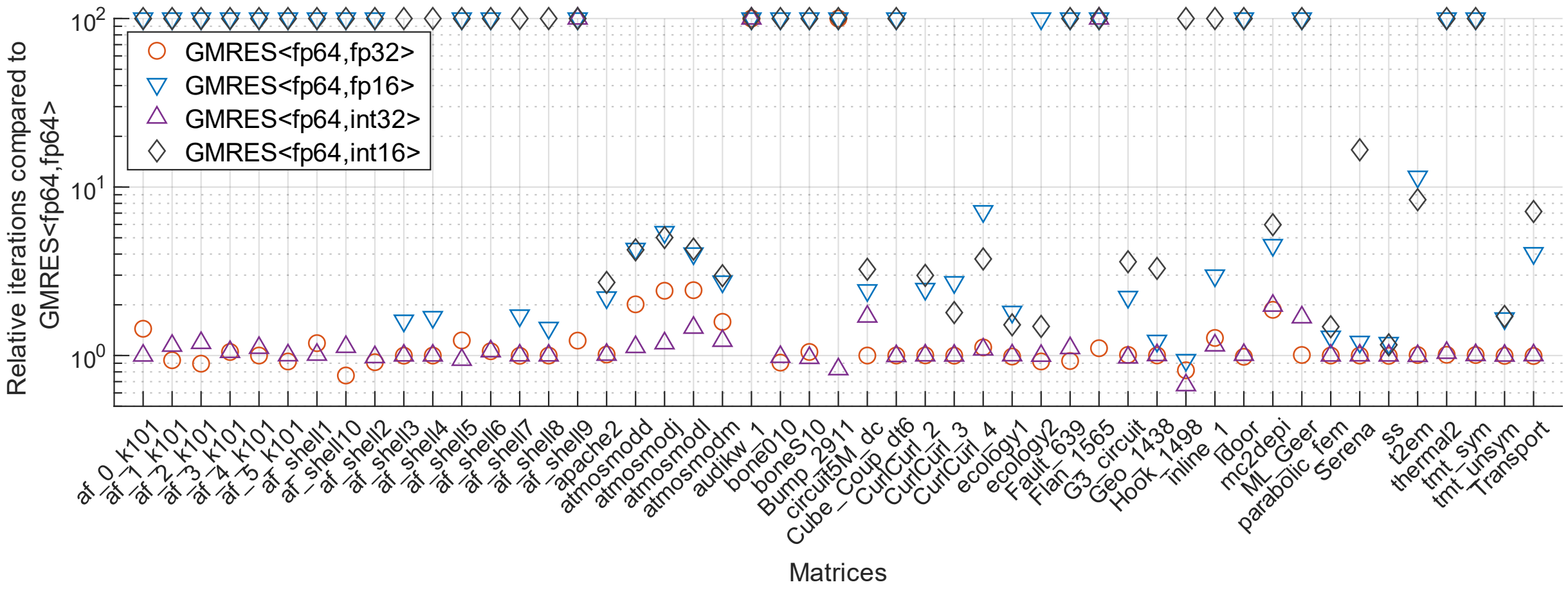


Correctness check with the normalized residual norm

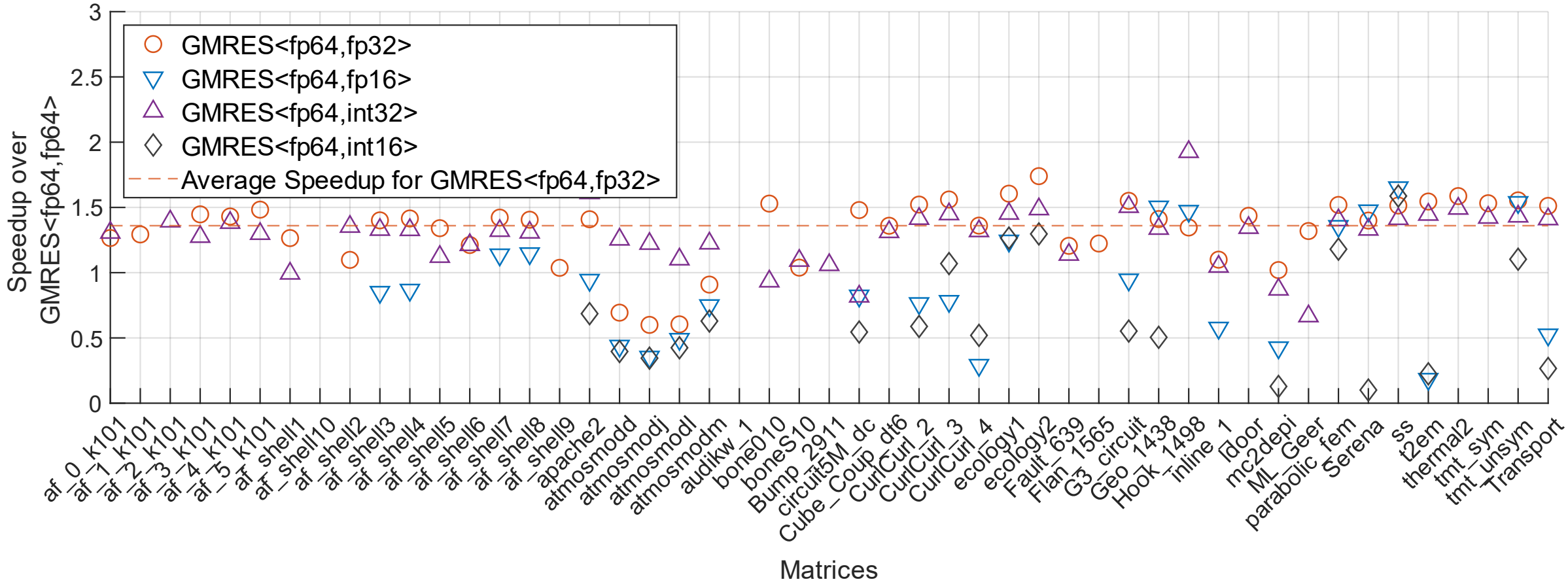


Note: GMRES<fp32,fp32> only reaches a normalized residual norm of $\sim 10^{-7}$

Iteration count comparison



CB-GMRES Speedup



Summary and Outlook

- Only the memory precision of the Krylov basis was changed
- Average Speedup of **1.36** for float32 and **1.32** for int32 with almost no iteration overhead
  ➔ technique is useful to tackle the memory wall
- Float16 (half) and int16 are outperformed by float32 since they fail to preserve the convergence characteristics
- CB-GMRES solver available here: https://ginkgo-project.github.io/

- Use other compression techniques for the Krylov basis
- Take advantage of the accessor in other fields (e.g. preconditioner)

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration and the Helmholtz Impuls und VernetzungsfondVH-NG-1241

EXASCALE COMPUTING PROJECT

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES