Task-Based Polar Decomposition Using SLATE on Massively Parallel Systems with Hardware Accelerators

Dalal Sukkari, **Mark Gates**, Mohammed Al Farhan, Hartwig Anzt, Jack Dongarra

Innovative Computing Laboratory, University of Tennessee, Knoxville

ScalAH 2023 Workshop

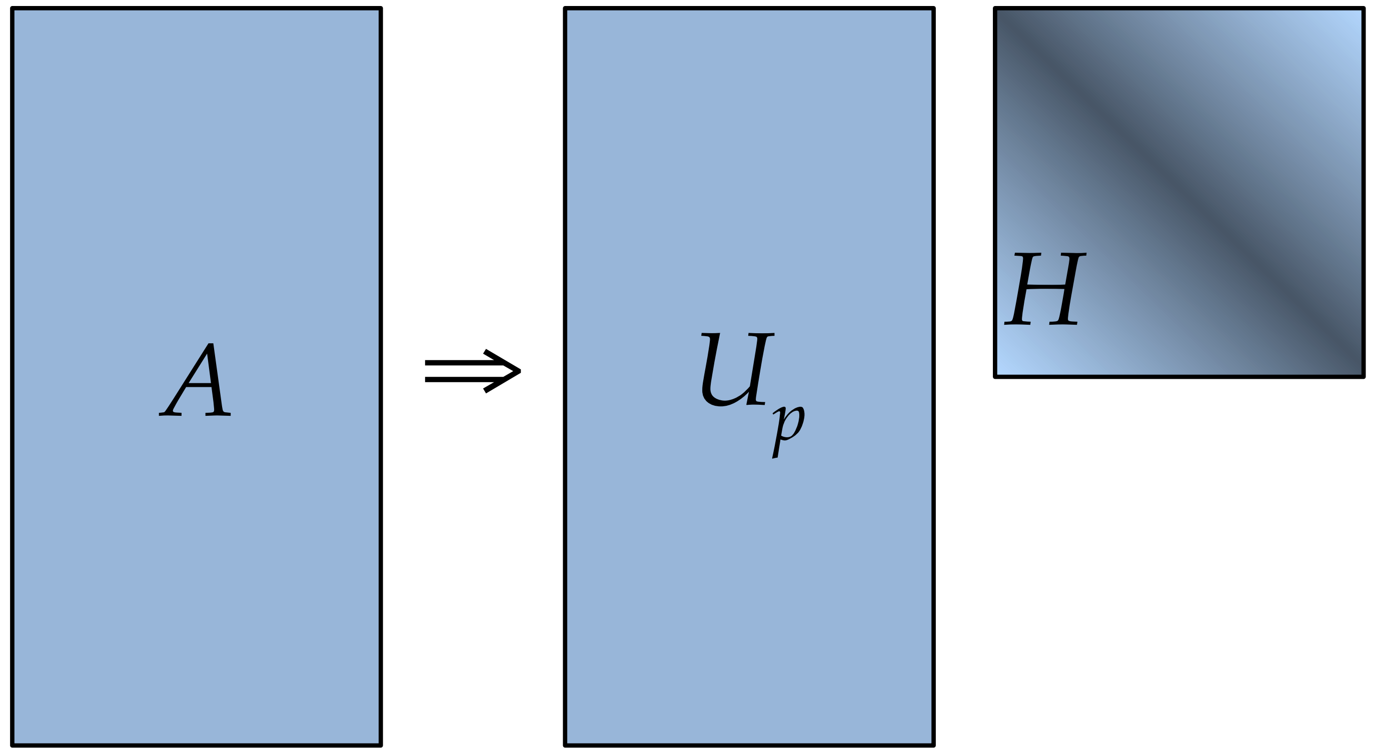




Polar Decomposition (PD)

- Factor matrix $A \in \mathbb{C}^{m \times n}$ $(m \geq n)$ into

$$A = U_p H$$



- Unitary matrix U_p
- Hermitian positive semi-definite matrix $H = \sqrt{A^H A}$,

$$HH = A^H A$$

- Useful in aerospace computations, factor analysis, computing SVD and pseudo-inverse

Motivation: Subaru telescope

- 8.2-meter telescope of the National Astronomical Observatory of Japan, located at Mauna Kea, Hawaii
- Adaptive Optics to measure and correct for atmospheric turbulence using pseudo-inverse in real time
- Used MAGMA-based Polar Decomposition (QDWH) SVD for pseudo-inverse, 2018

Polar Decomposition to SVD

- Compute $A = U_p H$
- Substitute Hermitian eigenvalues $H = V \Lambda V^H$
- Yields SVD: $A = U_p H = (U_p V) \Lambda V^H = U \Lambda V^H$
- See also High-Performance SVD Partial Spectrum Computation, Keyes et al., Thu 11 a.m.
- Or in reverse, compute PD from SVD:

$$A = U \Sigma V^H = U(V^H V) \Sigma V^H = (U V^H)(V \Sigma V^H) = U_p H$$

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Factorizations

- LU: $A = P^T LU,$ triangular L and U
- Cholesky: $A = LL^H,$ Hermitian positive definite A, triangular L
- QR: $A = QR,$ unitary Q, triangular R
- Eigenvalue: $A = V \Lambda V^{-1},$ diagonal Λ
- $A = V \Lambda V^H,$ Hermitian A, unitary V, real diagonal Λ
- SVD: $A = U \Sigma V^H,$ unitary U and V, real diagonal Σ
- Polar: $A = U_p H,$ unitary U_p, Hermitian semi-definite H

INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

History of Polar Decomposition

- 1902 — Autonne. Definition of polar decomposition.
- 1986 — Higham. Scaled Newton's Method.
- 1990 — Gander. Halley's Iteration.
- 1994 — Higham and Papadimitriou. Matrix inverse dynamically weight Halley (DWH).
- **2010 — Nakatsukasa et. al. Inverse-free QR-based DW Halley (QDWH).**
- 2013 — Nakatsukasa and Higham. QDWH Eig, SVD.
- 2016 — Sukkari, Ltaief, Keyes. Block algorithm, GPUs (**MAGMA**).
- 2016 — Sukkari, Ltaief, Keyes. ScaLAPACK, distributed memory (**POLAR**, Cray LibSci).
- 2017 — Sukkari, Ltaief, Faverge, Keyes. Task-based, shared memory (**Chameleon**).
- 2018 — Ltaief, Sukkari, Guyon, Keyes. QDWH SVD.
- 2019 — Sukkari, Ltaief, Keyes, Faverge. Task-based, distributed memory (**DPLASMA**).
- 2023 — Sukkari et al. Distributed, task-based, GPUs (**SLATE**).
- 2023 — Keyes, Ltaief, Nakatsukasa, Sukkari. QDWH Partial SVD. (Thu 11 a.m.)

Methods

- **Newton's Method**

$$X_0 = A,$$
$$X_{k+1} = \frac{1}{2}\left(X_k + X_k^{-H}\right) \qquad X_k \to U_p$$

  - Quadratic convergence, but initially slow
  - Explicit inverse, stability issues

- **Scaled Newton's Method**

$$X_{k+1} = \frac{1}{2}\left(z_k X_k + (z_k X_k)^{-H}\right)$$

  - Improves convergence

Higham. SIAM, 1986.

Methods

- **Halley Iteration**

$$X_0 = A,$$

$$X_{k+1} = X_k \left(3I + X_k^H X_k\right) \left(I + 3X_k^H X_k\right)^{-1}$$

  - Cubic convergence, but initially slow; explicit inverse

- **Dynamically Weighted Halley (DWH) Iteration**

$$X_0 = A / \left\| A \right\|_2,$$

$$X_{k+1} = X_k \left(a_k I + b_k X_k^H X_k\right) \left(I + c_k X_k^H X_k\right)^{-1}$$

  - Improves convergence: 6 iterations for cond(A) $= 10^{16}$

Gander. SIAM, 1990.

Yuji Nakatsukasa, Zhaojun Bai, and François Gygi. SIAM, 2010.

Methods

- Goal: implement DWH without inverse

$$X_{k+1} = X_k \left(a_k I + b_k X_k^H X_k\right) \left(I + c_k X_k^H X_k\right)^{-1}$$

- **QR-based Dynamically Weighted Halley (QDWH) Iteration**

$$\begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R$$ QR factorization

$$X_{k+1} = \alpha Q_1 Q_2^H + \beta X_k,$$ matrix multiply (gemm)

$$\alpha = \frac{1}{\sqrt{c_k}} \left(a_k - \frac{b_k}{c_k} \right),$$

$$\beta = \frac{b_k}{c_k}$$

Yuji Nakatsukasa, Zhaojun Bai, and François Gygi. SIAM, 2010.

Methods

- Goal: implement DWH as X_k becomes well-conditioned

$$X_{k+1} = X_k \left(a_k I + b_k X_k^H X_k\right) \left(I + c_k X_k^H X_k\right)^{-1}$$

- **Cholesky-based Dynamically Weighted Halley Iteration**

$$Z_k = I + c_k X_k^H X_k,$$ matrix multiply (herk)

$$L_k = \mathrm{chol}(Z_k),$$ Cholesky factorization

$$X_{k+1} = \left(a_k - \frac{b_k}{c_k}\right) X_k L_k^{-1} L_k^{-H} + \frac{b_k}{c_k} X_k$$ Cholesky solve, add

Yuji Nakatsukasa and Nicholas J. Higham. SIAM, 2013.

Algorithm

function polar_qdwh(input A; output U_p, H)

X_k = A / norm2est(A)

QR factor of X_k

$smin$ = norm1(R) / cond1est(R) / sqrt(n) // $smin$ is lower bound of $\sigma_{\min}$ of X_k

until convergence: $|smin - 1| < \text{tol}$

update weights $smin, a_k, b_k, c_k$

if ill-conditioned: $c_k > 100$

do QR-based iteration

else

do Cholesky-based iteration

$U_p = X_k$

$H = U_p^H A$ matrix multiply: gemm, herkx (cuBLAS, rocBLAS), or gemmt (MKL)

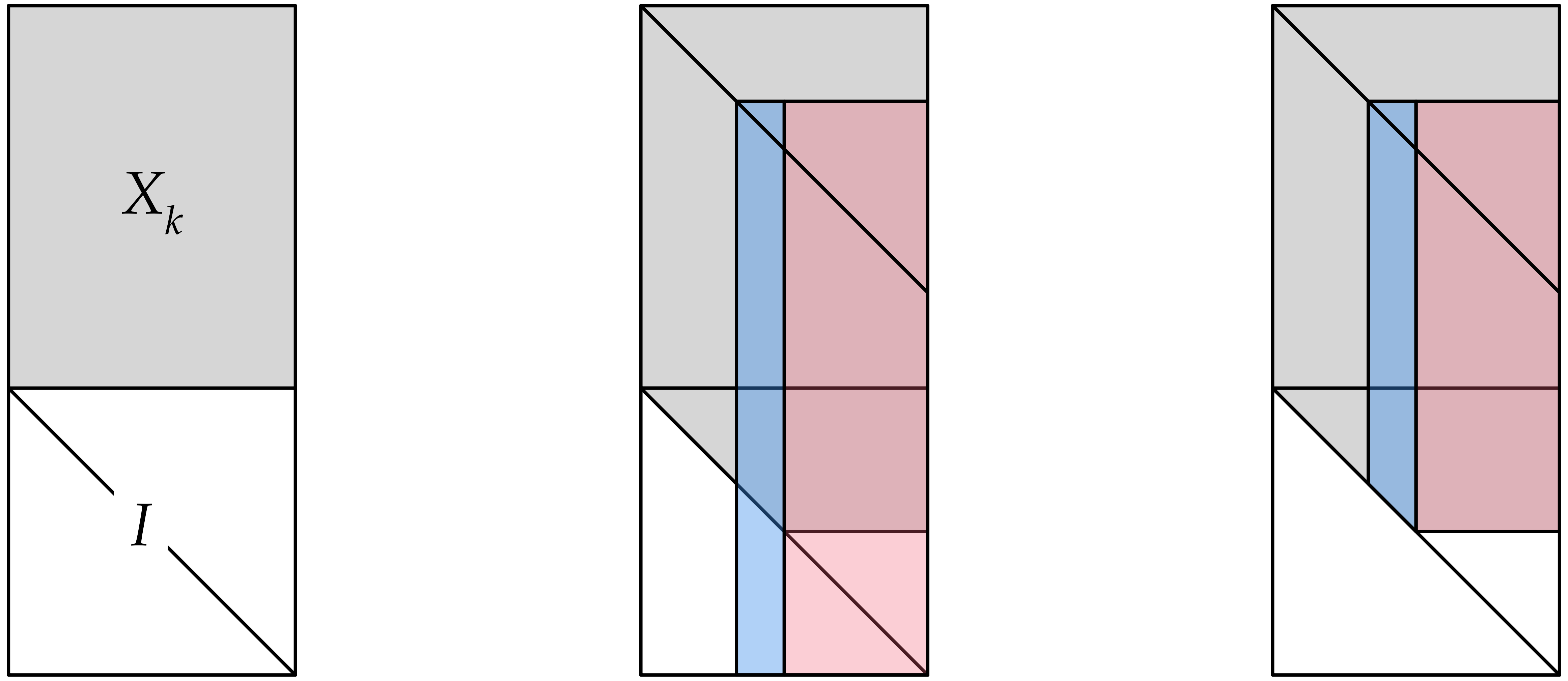
QR optimization

- QR of dense matrix on top of diagonal (identity)

$$\begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R$$

QR factorization

- Modify standard QR so panels go only to diagonal of lower block



SLATE: Software for Linear Algebra Targeting Exascale

- **Distributed, GPU-accelerated, dense linear algebra library**
  - BLAS
  - Linear systems
  - Least squares
  - Eigenvalues, SVD, Polar

- **Built on BLAS++ and LAPACK++ portability layer**
  - Wrappers around CPU and GPU BLAS & LAPACK
- **Modern replacement for ScaLAPACK**
  - C++ templates, MPI, OpenMP tasks

SLATE Coverage

Basic linear algebra $(C = AB, \ldots)$

	ScaLAPACK	SLATE
Level 1 PBLAS	✔	✘
Level 2 PBLAS	✔	✔ Level 2+ optimizations
Level 3 PBLAS	✔	✔
Auxiliary routines (add, set, scale, ...)	✔	✔
Matrix norms	✔	✔
Test matrix generation	✔	✔

Least squares $(Ax \cong b)$

	ScaLAPACK	SLATE
QR	✔	✔
Cholesky QR	✘	✔
LQ	✔	✔
Least squares solver	✔	✔
PAQR (pivoting avoiding)	✘	✔ dev branch

Linear systems $(Ax = b)$

	ScaLAPACK	SLATE
LU (partial pivoting, threshold)	✔	✔
CALU (tournament pivoting)	✘	✔
LU, band (pp)	✔	✔
LU (non-pivoting)	✘	✔
Cholesky	✔	✔
Cholesky, band	✔	✔
Symmetric Indefinite (block Aasen)	✘	✔ CPU only
Mixed precision (single-double)	✘	✔
Inverses (LU, Cholesky)	✔	✔
Condition estimate	✔	✔

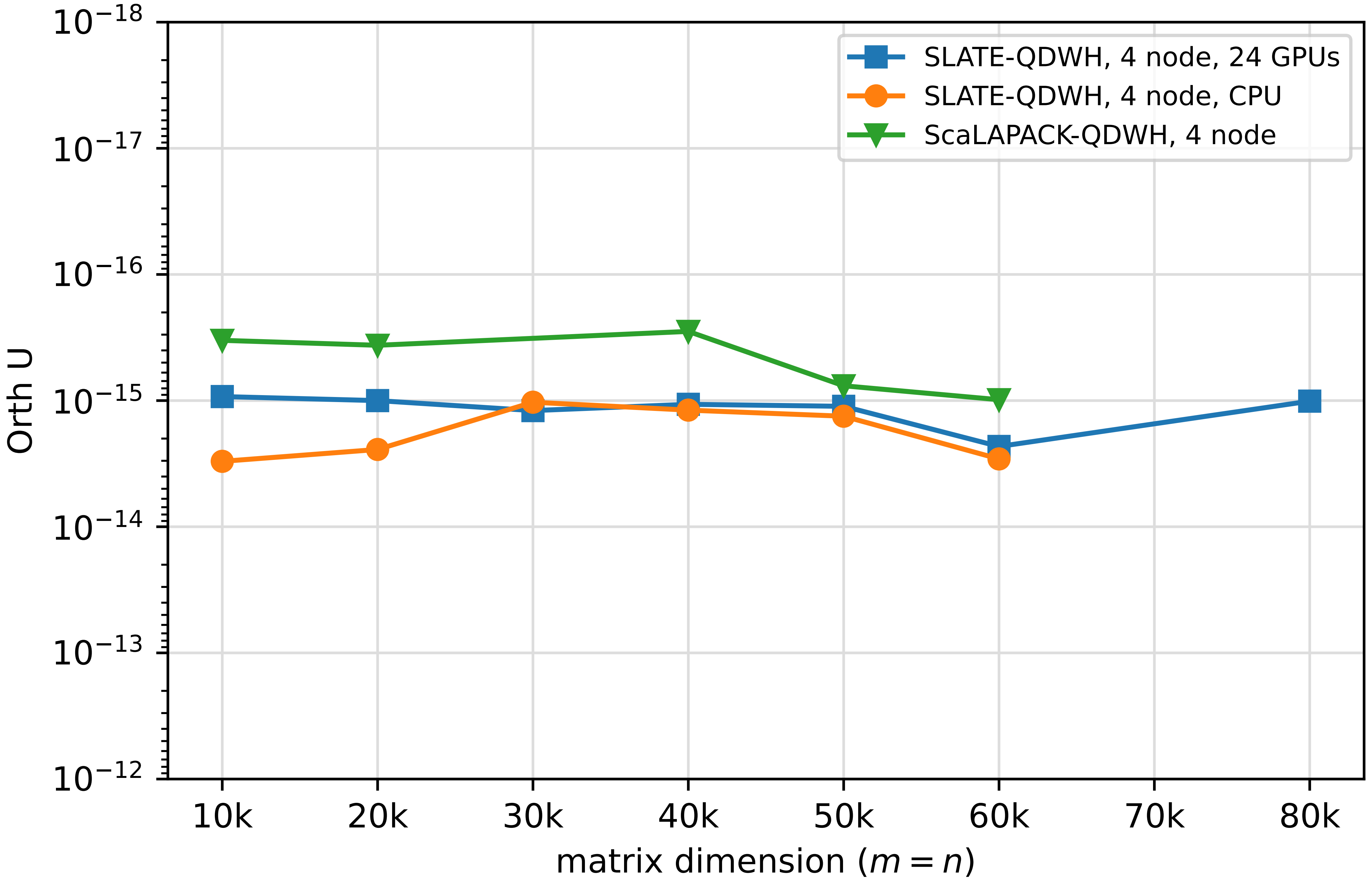
SVD, Eig, PD $(A = U\Sigma V^H, Ax = \lambda x, A = UH)$

	ScaLAPACK	SLATE
Singular value decomposition (SVD)	✔	✔ values & vectors
Hermitian eigenvalue	✔	✔ values & vectors
Generalized Hermitian eigenvalue	✔	✔ values & vectors
Polar decomposition (QDWH)	✘	✔ dev branch
LOBPCG	✘	✔ dev branch
Non-symmetric eigenvalue	pieces	✘ future
Complex–symmetric eigenvalue	✘	✘ future

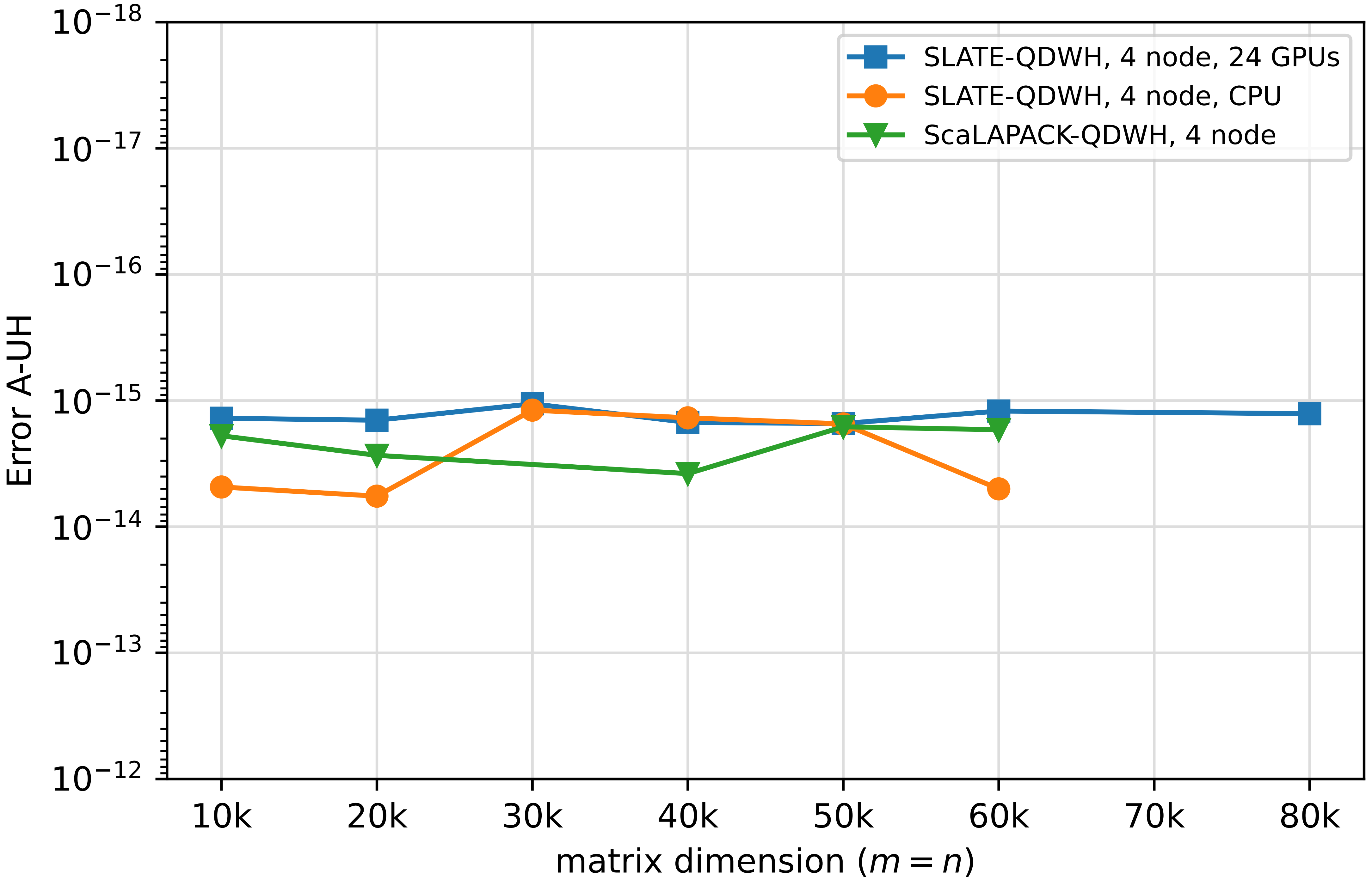
Unless noted, all routines are GPU-accelerated

Results

- Orthogonality of U_p

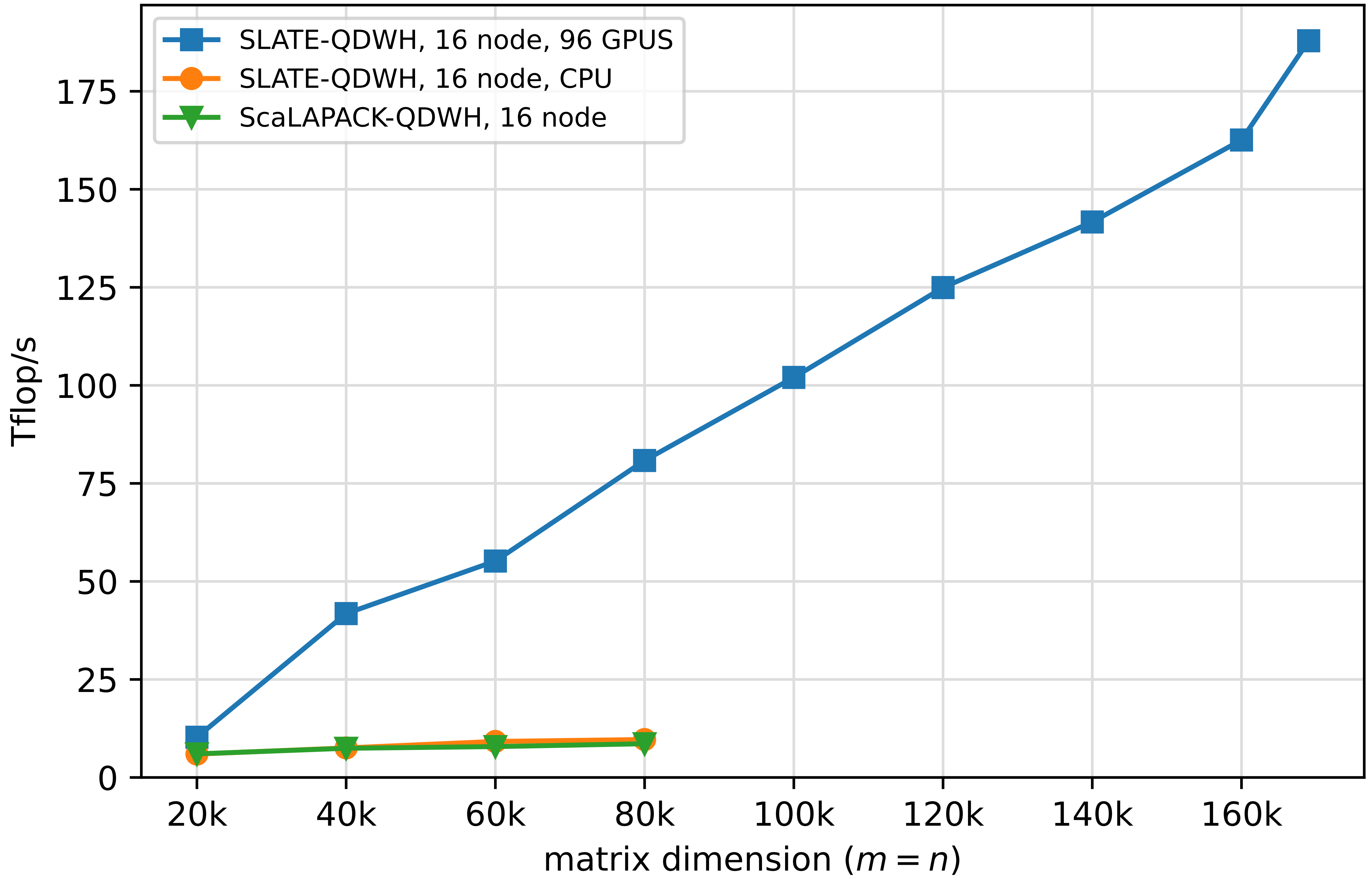


- Backward Error, $A - U_p\, H$



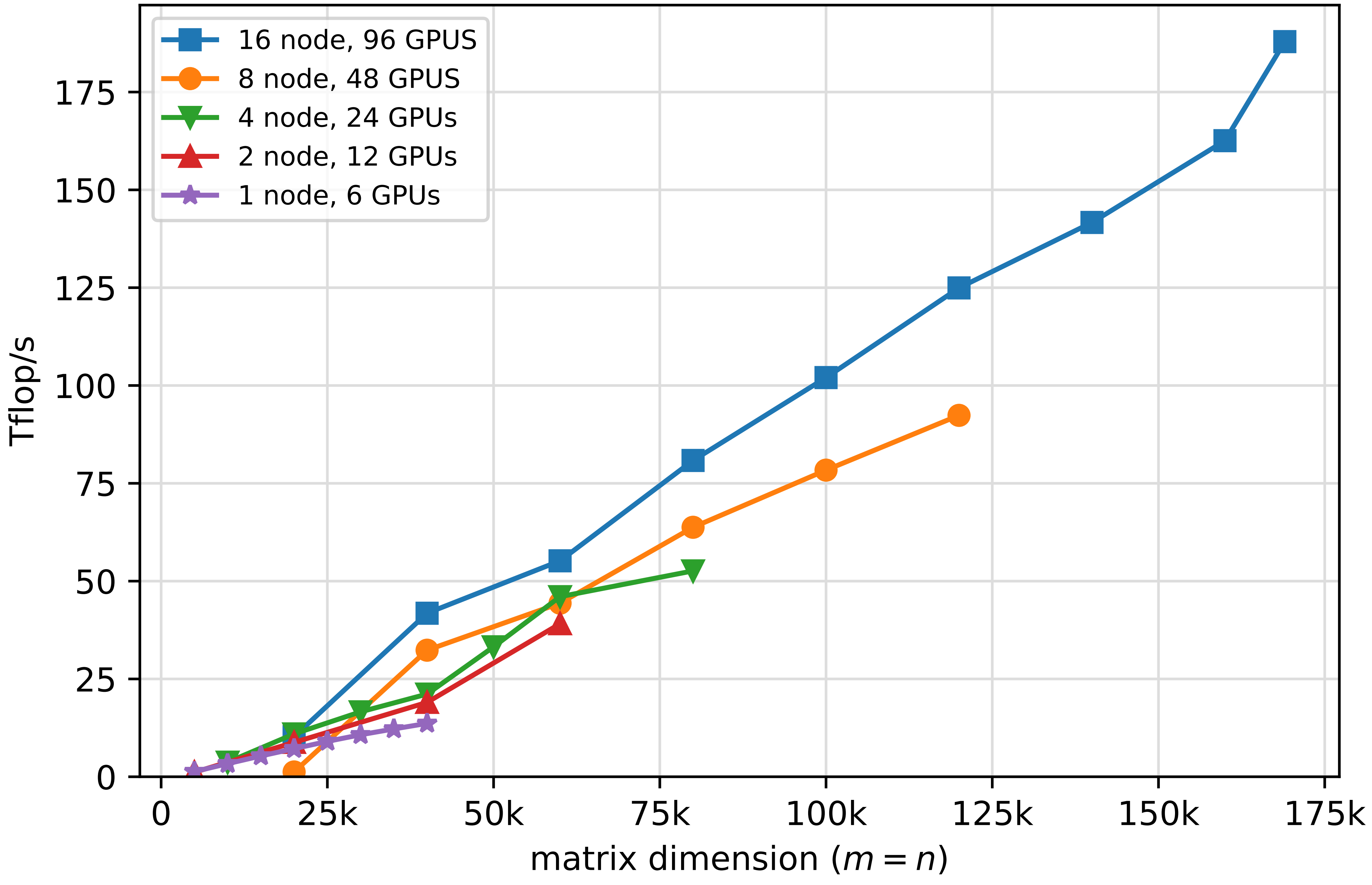
Results on Summit

- 16 node, SLATE GPU, SLATE CPU, ScaLAPACK CPU



Up to 18x speedup using GPUs

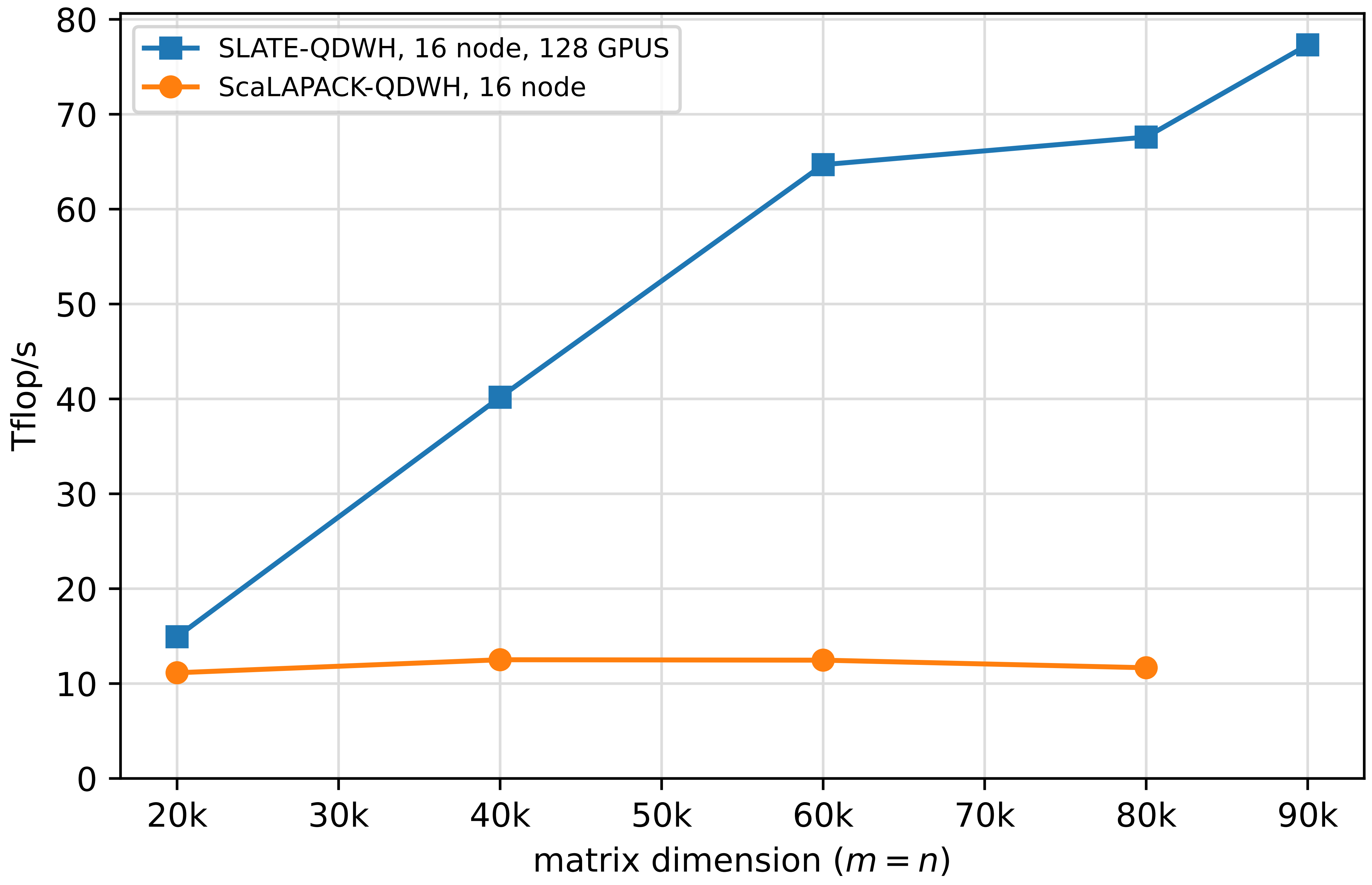
- 1 to 16 nodes, SLATE GPU



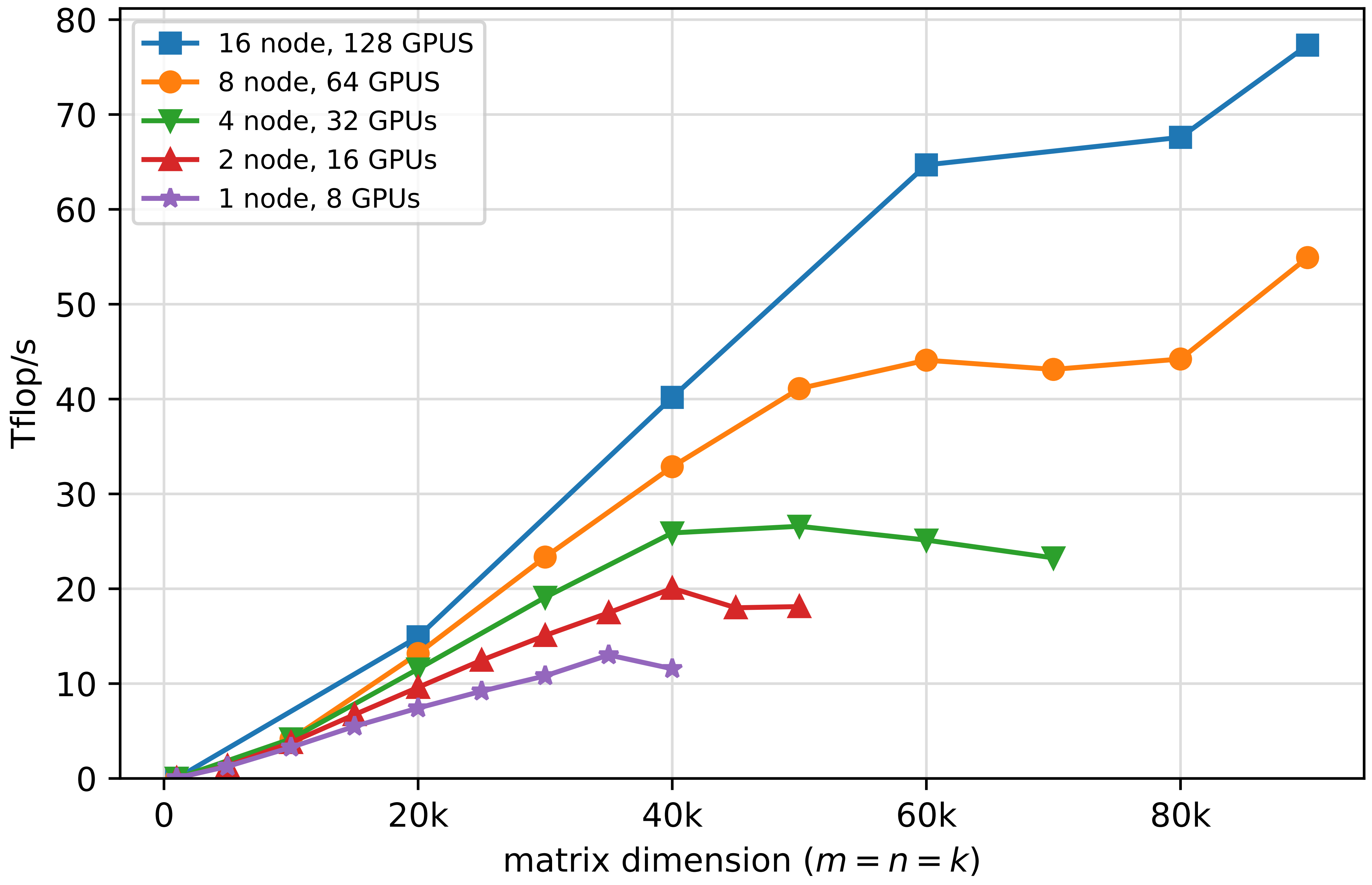
2x 22-core IBM POWER 9 CPU
+ 6 NVIDIA V100 GPUs per node

Results on Frontier

- 16 node, SLATE GPU, ScaLAPACK CPU



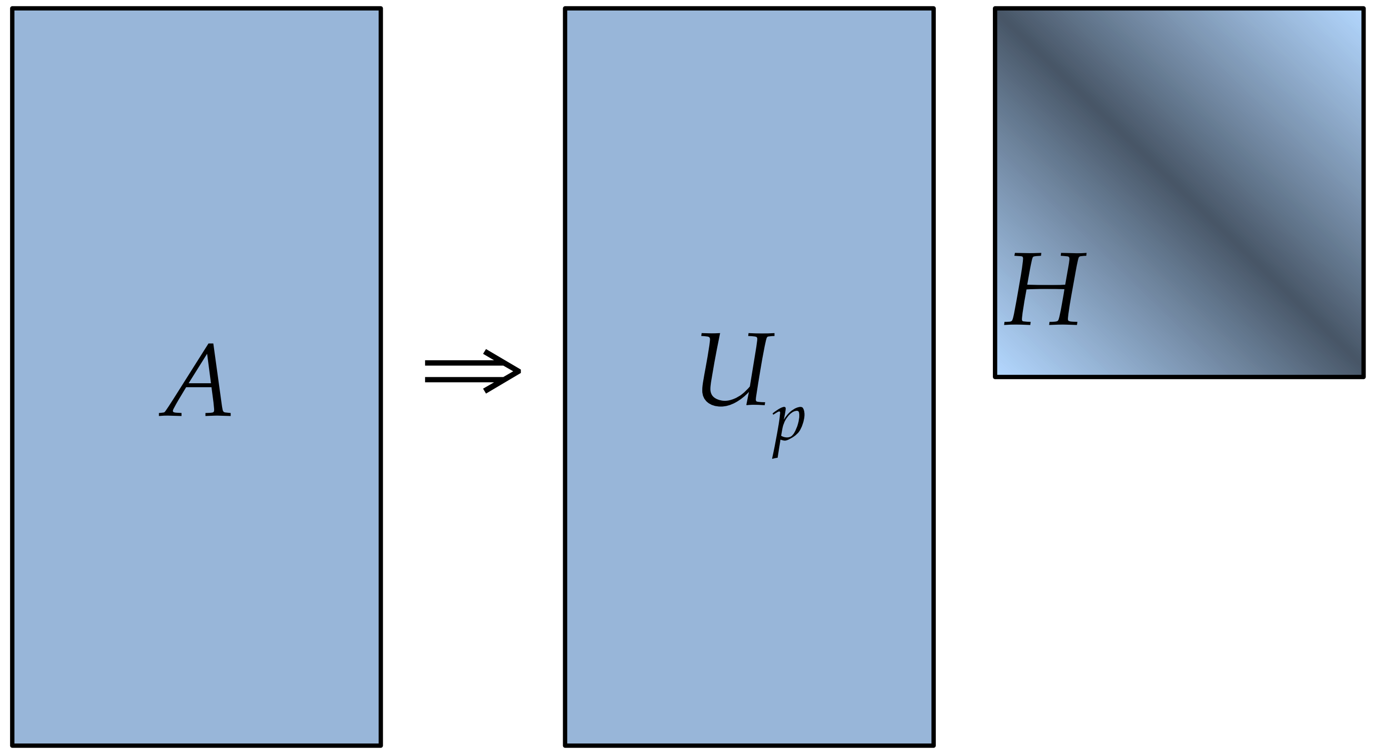
- 1 to 16 nodes, SLATE GPU



64-core AMD 3rd gen EPYC CPU
+ 8 AMD MI250X GPU GCDs per node

Summary

- Polar Decomposition using QR-based Dynamic Weighted Halley Iteration (QDWH)



- With SLATE, get CPU and GPU-accelerated, distributed implementation
- 18x speedup over ScaLAPACK CPU-based version in POLAR / Cray LibSci

Links and Acknowledgements

- SLATE, BLAS++, and LAPACK++ libraries
    - https://github.com/icl-utk-edu/slate/
    - https://github.com/icl-utk-edu/blaspp/
    - https://github.com/icl-utk-edu/lapackpp/

ECP
EXASCALE COMPUTING PROJECT

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.

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

ICL

INNOVATIVE
COMPUTING LABORATORY

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE