QR-Ω: Communication Avoiding 2.5D Householder QR

Felipe Tomé [1,3], Rabab Alomairy[2, 3], Alan Edelman[3], Hermes Senger[1,4]

[1]Federal University of São Carlos (UFSCar), Brazil, [2]King Abdullah University of Science and Technology (KAUST), KSA, [3]Massachusetts Institute of Technology (MIT), USA, [4]Universidade de São Paulo (USP), Brazil

Motivation

Krylov methods repeatedly orthogonalize newly generated basis vectors or blocks of vectors [1,2]; electronic-structure calculations orthonormalize large sets of wavefunctions [3]; and tensor-network methods repeatedly restore orthogonal representations [4]. Orthogonalization has also emerged in large-scale machine learning, where recent matrix-based optimizers perform it as part of the training step [5]. Householder QR provides a numerically robust orthogonalization and is also a core building block in least-squares problems, eigensolvers, and randomized numerical linear algebra.

Objective. Adapt the memory–communication tradeoff within and across GPUs while keeping Householder transformations and their order.

From hierarchical QR to QR-Ω

HQR [6] specifies an **elimination list**: which tiles interact and when their updates become ready. QR-Ω keeps that list and specifies how **each matrix product** runs across the GPU hierarchy.

$$Q = I - VTV^{\mathsf{T}}, \qquad W = V^{\mathsf{T}}X, \quad Z = T^{\mathsf{T}}W,$$
$$D = VZ, \qquad X \leftarrow X - D.$$

- **Split**: partition row, column, and contraction indices among concurrent peers, using a carrier (p_i, p_j, c).
- **Replicate**: place the operand blocks needed by multiple peers in the appropriate memory domain.
- **Combine**: sum partial products over disjoint contraction ranges in a fixed order; commit each owner output once.
- **Pipeline**: overlap column strips through W, Z, and D; prioritize the strip that releases the next panel.

The additive product reduction is distinct from the Householder QR operations that merge triangular factors.

Why 2.5D: replication within a memory budget

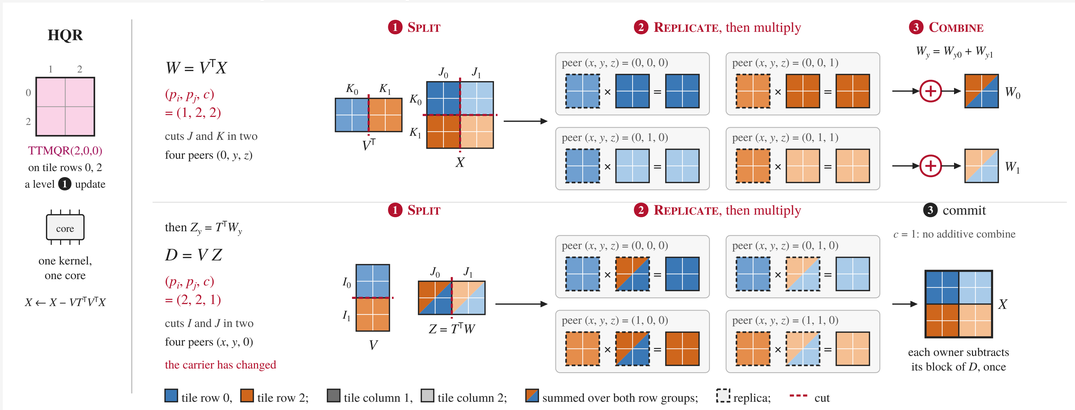
A full 3D decomposition fixes replication through the processor grid. The 2.5D carrier instead adjusts c to the capacity of each memory domain. Smaller c keeps most of that reuse at a fraction of the storage of full 3D replication, where registers and shared memory are tightest.

Across GPUs, only the small W partials are reduced; T and Z are replicated and each GPU updates its own rows of X, so the full trailing matrix is never replicated.

Experimental setup

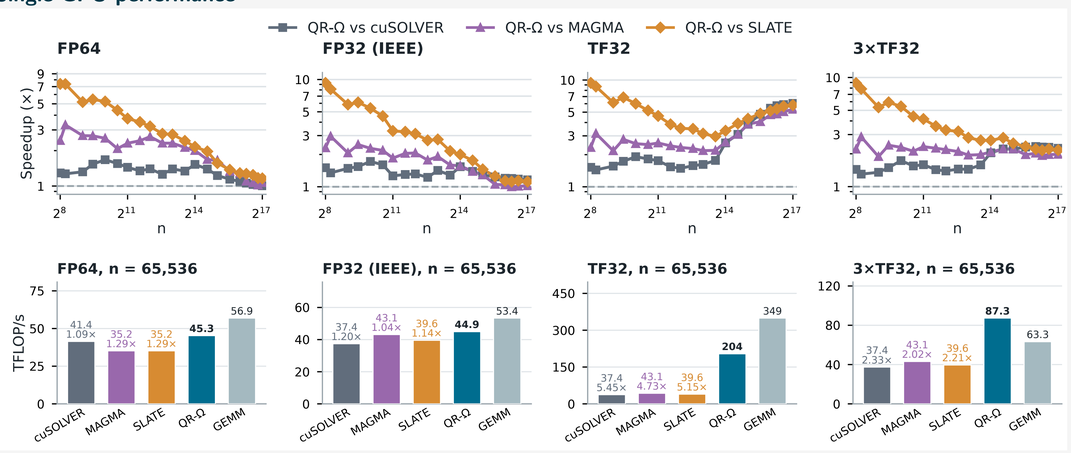
Hardware: NVIDIA H200, 141 GB HBM; CUDA 13.0.2.
References: cuSOLVER 12.0.4.66, MAGMA 2.10.0, SLATE 2025.05.28, tuned per size and precision.
Timing: medians after warm-up; device-ready input to completed factorization; allocation and validation excluded.
Modes: FP64, IEEE FP32, TF32, and 3×TF32. Tensor modes are compared with reference FP32.

One elimination, different product decompositions



Example: W uses $(1, 2, 2)$, with disjoint contraction partials; D uses $(2, 2, 1)$, with one update per owner. Carriers can differ by product and memory level.

Single-GPU performance



Top: speedup over each tuned reference, $n = 256$–$131{,}072$. Bottom: throughput at $n = 65{,}536$. GEMM is a measured same-size comparison, not a hardware ceiling. Tensor-mode comparisons use reference FP32.

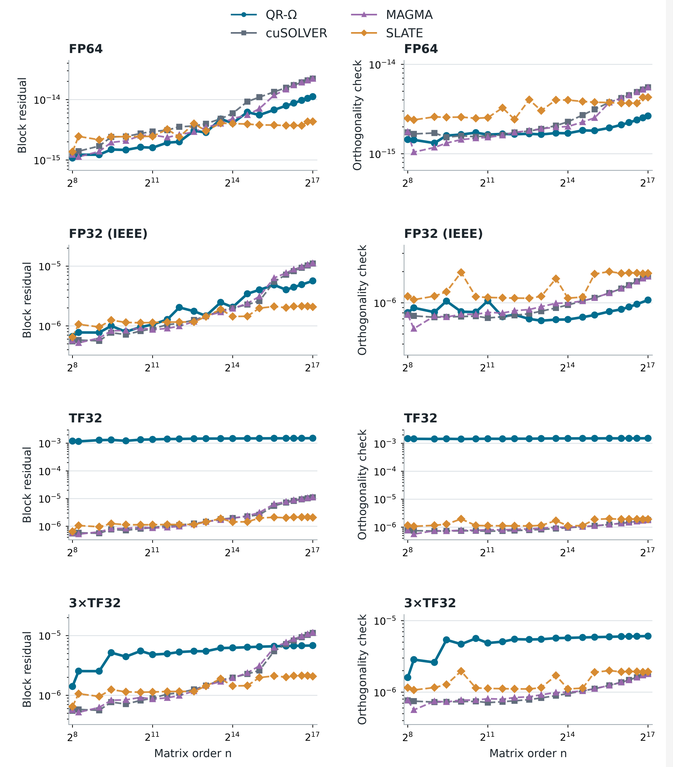
Results at $n = 2{,}048$ on one H200

Mode	Time (s)	TFLOP/s	vs fastest ref.	Residual
FP64	0.00676	1.69	1.44×	1.6×10^{-15}
FP32	0.00704	1.63	1.26×	1.1×10^{-6}
TF32	0.00507	2.26	1.76×	1.4×10^{-3}
3×TF32	0.00560	2.04	1.59×	4.8×10^{-6}

Fastest reference per mode: cuSOLVER in all four tiers. FP64: 1.44/2.31/3.75× vs. cuSOLVER/MAGMA/SLATE; FP32: 1.26/1.85/3.31×; tensor tiers compared against FP32 references.

[1] Y. Saad and M. H. Schultz. *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems.* SIAM J. Sci. Stat. Comput. 7(3) (1986), 856–869. doi:10.1137/0907058.
[2] I. Yamazaki, A. J. Higgins, E. G. Boman, and D. B. Szyld. *Two-stage block orthogonalization to improve performance of s-step GMRES.* Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS) (2024), 26–37. doi:10.1109/IPDPS57955.2024.00012.
[3] S. Das, P. Motamarri, V. Subramanian, D. M. Rogers, and V. Gavini. *DFT-FE 1.0: A massively parallel hybrid CPU-GPU density functional theory code using finite-element discretization.* Comput. Phys. Commun. 280 (2022), 108473. doi:10.1016/j.cpc.2022.108473.
[4] U. Schollwöck. *The density-matrix renormalization group in the age of matrix product states.* Ann. Phys. 326(1) (2011), 96–192. doi:10.1016/j.aop.2010.09.012.
[5] M. Khona et al. *SOAP, Muon, and Beyond: Pushing LLM Pretraining Scales.* arXiv preprint arXiv:2607.20548 (2026). doi:10.48550/arXiv.2607.20548.
[6] J. Dongarra et al. *Hierarchical QR factorization algorithms for multi-core clusters.* Parallel Computing 39 (2013), 212–232. doi:10.1016/j.parco.2013.01.003.

Numerical validation



Left: maximum relative column-block reconstruction residual, covering all columns. Right: $\|Q(Q^{\mathsf{T}}X) - X\|_F / \|X\|_F$.

Conclusion

QR-Ω extends hierarchical Householder QR by fixing how each update runs at every level of the memory hierarchy. On the H200, controlled replication and pipelining move QR toward matrix-product throughput while keeping FP64 and FP32 errors small. Tensor modes trade accuracy for speed.

Future work

The single-node multi-GPU version of this schedule is implemented and tested. Next: measure QR-Ω where QR sits on the critical path (tensor-network contractions, pre-training of LLMs, and physical wave simulation codes), then validate distributed multi-node scaling.

Code, data, and reproduction

Single- and multi-GPU sources, measurements, plot scripts, and reproduction commands.

`github.com/NextLinearAlgebra/qr-omega`