Tall Skinny QR decomposition

Pudit Tempattarachoke and Chutchatut Sutichavengkul

1 Summary

We plan to efficiently solve overdetermined linear systems, which have more constraints than variables, by leveraging parallelism. Our approach focuses on computing the TSQR (Tall-and-Skinny QR) factorization of the system matrix, using a thread pool on a shared-memory architecture. We plan to use openMP to schedule the computation. Our web project is avaiable here `https://pudit.github.io/TSQR/`.

2 Background

Overdetermined systems arise in many applications such as computer graphics, data fitting, and scientific computing. Standard QR factorization can be inefficient for tall-and-skinny matrices due to their large number of rows. TSQR addresses this by breaking the factorization into smaller, parallelizable tasks. On shared-memory systems, employing a thread pool allows these tasks to execute concurrently with minimal thread management overhead, enabling scalable and high-performance solutions.

3 Challenge

A key challenge of this project is reducing idle time by overlapping computation across the TSQR reduction tree. Computing portions of the Q factor while other threads perform QR on intermediate R blocks requires careful scheduling, and splitting the matrix efficiently to avoid load imbalance is nontrivial.

Designing a parallel strategy that minimizes downtime is difficult because TSQR has irregular dependencies and multi-stage communication. Tasks depend on intermediate R factors, causing potential idle threads, while communication latency and synchronization further complicate scheduling. Achieving high parallel efficiency requires careful coordination of computation, data movement, and resource allocation.

4 Resources

The TSQR algorithm [1], which is optimized for solving overdetermined systems and improves upon the standard QR factorization [2], will be developed from scratch without relying on external libraries. Development and testing will primarily take place on the GHC machines, targeting both CUDA-based GPU parallelism and OpenMPI-based distributed-memory parallelism. After the implementation is completed, performance and scalability experiments will be conducted on PSC to evaluate the algorithm's strength and behavior under large-scale workloads

5 Goals and Deliverables

5.1 Plan to Achieve

The primary goal of this project is to implement a full Tall-and-Skinny QR (TSQR) solver in C++ using both CUDA and OpenMPI, developed entirely from scratch without external linear algebra libraries. At minimum, the project will:

- Implement the TSQR reduction tree and reconstruct the global R factor in parallel.
- Generate correct results for a range of tall-and-skinny matrices.
- Measure performance and scalability on both CUDA and MPI backends, including timing breakdowns and communication cost analysis.
- Produce speedup and scalability graphs comparing single-threaded QR with parallel TSQR.

Completing these items ensures a functional, parallel TSQR implementation and provides enough results for a strong final evaluation.

5.2 Hope to Achieve

If progress is ahead of schedule, the project will extend TSQR with more advanced deliverables:

- Implement the parallel reconstruction of the global Q factor using the TSQR tree.
- Compare performance with NVIDIA's cuSolver QR routines and analyze when TSQR outperforms standard GPU QR.
- Conduct a deeper study of performance bottlenecks and extend the analysis by comparing TSQR with alternative large-scale strategies, such as MapReduce-style distributed QR or direct-method solvers. This comparison will help identify when communication-avoiding algorithms provide

the greatest advantage and clarify the limits of TSQR on modern parallel platforms.

These "stretch goals" allow a deeper understanding of TSQR's communication-avoiding behavior and its benefits on modern parallel systems.

6 Platform Choice

C++ is used because it offers efficient memory control and strong support for both CUDA and MPI, making it well-suited for implementing TSQR from the ground up. GHC machines provide accessible GPU and MPI resources for development and testing, while PSC systems offer larger-scale parallel hardware to evaluate performance at high core counts. These platforms match the communication-avoiding nature of TSQR and allow us to test how the algorithm benefits from modern parallel architectures.

7 Schedule

Date	**Milestones**
Week 1 (Nov 17–23)	Literature survey; implement baseline QR solvers for TSQR
Week 2 (Nov 24–30)	Add parallelism to TSQR; construct and combine local R factors
Week 3 (Dec 1–7)	Debugging, performance testing, and analysis
Week 4 (Dec 8–15)	Construct global Q; finalize experiments and write report

Table 1: Project Timeline and Milestones

References

[1] James Demmel et al. "Communication-optimal parallel and sequential QR and LU factorizations". In: *SIAM Journal on Scientific Computing* 34.1 (2012), A206–A239.

[2] Justin Solomon. *Numerical Algorithms: Methods for Computer Vision, Machine Learning, and Graphics*. USA: A. K. Peters, Ltd., 2015. ISBN: 1482251884.