Title: Parallel Eigensolver for Tridiagonal Symmetric Matrices

Objective

The objective of this challenge is to develop a parallel algorithm to compute the eigenvalues and eigenvectors of a symmetric tridiagonal matrix, optimized for execution on a GPU. The focus is on leveraging GPU architecture to accelerate computations using either the implicitly shifted QR algorithm (`steqr`), the divide-and-conquer method (`stedc`), or both, depending on their suitability for parallelism.

Background

Symmetric eigenvalue problems, expressed as $Ax = \lambda x$, where A is a symmetric matrix, λ is an eigenvalue, and x is an eigenvector, are ubiquitous in scientific computing. Solving these problems efficiently for large matrices is computationally intensive and typically involves three key steps:

1. **Tridiagonalization**:
   Transform the symmetric matrix A into a symmetric tridiagonal matrix T using orthogonal similarity transformations, $T = Q^T A Q$, where Q is an orthogonal matrix. This step is commonly performed using Householder reflections, reducing A to a form with non-zero elements only on the main diagonal and the adjacent sub- and super-diagonals.
2. **Eigenvalue Computation**:
   Compute the eigenvalues and eigenvectors of the tridiagonal matrix T. This is the focus of the challenge, as it is a critical bottleneck for large-scale problems. Two primary methods are considered: `steqr` and `stedc`.
3. **Back Transformation**:
   Map the eigenvalues and eigenvectors of T back to those of A using the orthogonal matrix Q. This step completes the solution for the original matrix.

This challenge targets the second step, leveraging GPU parallelism to accelerate the computation of eigenvalues and eigenvectors for symmetric tridiagonal matrices.

Implicitly Shifted QR Algorithm (`steqr`)

Overview

The implicitly shifted QR algorithm (`steqr`) is an iterative method that computes all eigenvalues and eigenvectors of a symmetric tridiagonal matrix by repeatedly applying QR factorizations with shifts. It is robust and widely implemented in libraries like LAPACK and cuSOLVER due to its reliability and numerical stability.

The algorithm iteratively transforms the tridiagonal matrix T into a diagonal matrix, revealing the eigenvalues on the diagonal, while accumulating orthogonal transformations to compute the eigenvectors. The steps are:

1. **Shift Selection**:
   Choose a shift μ, typically an approximation of an eigenvalue, to accelerate convergence. Common choices include the bottom-right element of T or a Rayleigh quotient derived from the current matrix.
2. **QR Factorization**:
   Compute the QR factorization of the shifted matrix $T - \mu I = QR$, where Q is orthogonal and R is upper triangular. This factorization is done implicitly using Givens rotations, which preserve

the tridiagonal structure. The process of 'chasing the bulge' refers to eliminating off-diagonal elements (introduced by the shift) as they are moved down the matrix, restoring tridiagonality.

3. **Matrix Update**:
   Update the matrix as $T' = RQ + \mu I$. This new matrix T' remains symmetric and tridiagonal and serves as the input for the next iteration.
4. **Convergence**:
   Repeat the process until T becomes diagonal, with the diagonal elements being the eigenvalues. The eigenvectors are obtained by accumulating the Q matrices from each iteration into a single orthogonal matrix.

Parallelization Potential

Although `steqr` is inherently iterative, parallelism can be exploited within each QR factorization step. Specifically:

- **Givens Rotations**: These can be computed and applied in parallel across multiple threads.
- **Matrix Updates**: The application of Q and R to update T can leverage GPU memory coalescing and thread blocks. cuSOLVER's implementation demonstrates this, but scalability may be limited for very large matrices due to the sequential nature of iterations.

Complexity

- **Eigenvalues and Eigenvectors**: $O(n^2)$, where n is the matrix size. Each iteration requires $O(n)$ operations, and convergence typically takes $O(n)$ iterations.

Divide-and-Conquer Method (`stedc`)

Overview

The divide-and-conquer method (`stedc`) is a recursive algorithm used to compute eigenvalues and eigenvectors of tridiagonal matrices. By splitting the matrix into smaller subproblems, solving them independently, and recombining the results, this method becomes highly parallelizable, making it particularly well-suited for GPU architectures. Its recursive nature allows for efficient use of parallel computing resources, enabling faster solutions for large-scale eigenvalue problems.

The divide-and-conquer algorithm follows a series of steps to compute eigenvalues and eigenvectors of a tridiagonal matrix. The key steps are as follows:

1. **Matrix Splitting**:
   The tridiagonal matrix T is split into two smaller submatrices T_1 and T_2 at a chosen point (typically the middle). This introduces a rank-one modification in the form of:

$$T = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + \rho u u^T$$

   where u is a vector with non-zero elements only at the splitting points, and ρ is the off-diagonal element at the split. This modification is critical for simplifying the problem into two smaller subproblems.
2. **Recursive Solution**:
   The eigenvalues and eigenvectors of the submatrices T_1 and T_2 are computed recursively. For sufficiently small submatrices (e.g., 2x2 matrices), direct methods such as the QR algorithm or closed-form solutions are used.

3. **Combination via Secular Equation**:
   The solutions from T_1 and T_2 are combined by solving the secular equation:

$$1 + \rho \sum_i \frac{z_i^2}{\lambda - d_i} = 0$$

   where d_i are the eigenvalues of the submatrices, z_i are the components of the eigenvectors adjusted by u, and λ represents the eigenvalues of the original matrix T.

4. **Eigenvector Construction**:
   The eigenvectors of T are constructed by combining the eigenvectors of T_1 and T_2, adjusting them based on the solutions of the secular equation and the rank-one update. This provides the final eigenvectors for the matrix T.

Parallelization Potential

The `stedc` method excels in parallel environments:

- **Independent Subproblems**: T_1 and T_2 can be solved concurrently on separate GPU threads or blocks.
- **Recursive Parallelism**: Each recursive level can spawn additional parallel tasks.
- **Secular Equation**: While inherently sequential, this step can be optimized by distributing the summation across threads, though it remains a bottleneck for very large matrices. This makes `stedc` highly scalable on GPUs, especially for large n.

Complexity

- **Eigenvalue Computation**: The complexity of computing eigenvalues using the divide-and-conquer strategy is $O(n \log n)$, where n is the dimension of the matrix. This results from the recursive splitting of the problem into smaller submatrices.
- **Eigenvector Computation**: The complexity of computing the eigenvectors is $O(n^2)$, as this step involves combining the results of all recursive levels and adjusting eigenvectors at each stage.

Challenge Tasks

Literature Review

- Conduct an extensive review of existing algorithms for symmetric tridiagonal eigenvalue problems, with a particular focus on the `steqr` and `stedc` methods.
- Provide a detailed summary of their mathematical foundations, computational complexities, and strategies for parallelization.
- Highlight key advancements in the implementation of these algorithms on GPUs, drawing from recent literature.

Algorithm Development

- Implement a GPU-optimized version of:
  - `steqr`, focusing on parallelizing the QR factorization steps.
  - `stedc`, utilizing recursive parallelism and exploiting the independence of subproblems.

    - (Optional) A hybrid approach that combines the strengths of both `steqr` and `stedc` to improve performance.
- Optimize for GPU-specific features, including:
    - **Memory Coalescing**: Ensure efficient memory access patterns to maximize bandwidth.
    - **Thread Management**: Minimize thread divergence and maximize thread utilization for better parallel efficiency.

Performance Evaluation

- Evaluate the performance across a variety of matrix sizes (e.g., $n = 100, 1000, 10000$) and conditions (e.g., well-conditioned vs. ill-conditioned matrices).
- Analyze the following aspects:
    - **Execution Time**: Compare runtime for different methods across various matrix sizes.
    - **Scalability**: Assess how performance improves as more GPU resources are utilized.
    - **Accuracy**: Compare results with known eigenvalues and eigenvectors to ensure correctness.

Optimality Conditions

- Investigate the conditions under which each method excels:
    - **Matrix Size**: Determine the circumstances in which `stedc` outperforms `steqr` and vice versa.
    - **GPU Architecture**: Explore the impact of thread count, memory bandwidth, and other architectural factors on performance.
    - **Problem Characteristics**: Evaluate the influence of matrix condition number and eigenvalue distribution on the choice of method.
- Provide evidence-based guidelines for selecting the optimal method based on empirical data.

Deliverables

Report

A comprehensive report covering the following:

- A detailed literature review of `steqr` and `stedc`, including their mathematical foundations, computational complexities, and suitability for parallel GPU implementations.
- A clear description of the algorithm(s) implemented, including pseudocode, optimization strategies, and performance considerations.
- A complexity analysis along with a detailed performance comparison to cuSOLVER's `steqr` solver (optional).
- A discussion on optimality conditions, identifying when each method excels and the associated trade-offs.

Code (Optional but recommended)

- A well-documented GPU implementation in CUDA (or similar), including the source code for `steqr`, `stedc`, or hybrid approaches.
- Include scripts for benchmarking, performance evaluation, and testing to validate the implementation against cuSOLVER's `steqr` solver.

References

1) Cuppen, J. J. M. (1981). "A divide and conquer method for the symmetric tridiagonal eigenproblem." *Numerische Mathematik*, 36(2), 177-195.
2) Golub, G. H., & Van Loan, C. F. (2013). *Matrix Computations* (4th ed.). Johns Hopkins University Press.
3) Dongarra, J. J., Sorensen, D. C., & Bunch, J. R. (1985). "A parallel algorithm for the symmetric tridiagonal eigenvalue problem." *SIAM Journal on Scientific and Statistical Computing*, 6(1), 222-232.
4) Hernández-Rubio, E., Estrella-Cruz, A., Meneses-Viveros, A., Rivera-Rivera, J. A., Barbosa-Santillán, L. I., & Chapa-Vergara, S. V. (2024). "Symmetric Tridiagonal Eigenvalue Solver Across CPU Graphics Processing Unit (GPU) Nodes." *Applied Sciences*, 14(22), 10716.
5) Tomov, S., Dongarra, J., & Baboulin, M. (2010). "Towards dense linear algebra on graphics hardware." *International Conference on Supercomputing*, 1-10.
6) Haidar, A., Dongarra, J., Solca, R., Tomov, S., Schulthess, T. C., & Yamazaki, I. (2012). "MAGMA: A new generation of linear algebra libraries for GPU and multicore architectures." *International Conference on Supercomputing*, 1-10.
7) Ralha, R. (2001). "An Efficient Parallel Algorithm for the Symmetric Tridiagonal Eigenvalue Problem." *Journal of Computational and Applied Mathematics*, 131(1-2), 1-12.
8) Demmel, J. W., & Veselić, K. (1989). "Jacobi's method is more accurate than QR." *SIAM Journal on Matrix Analysis and Applications*, 10(4), 512-534.
9) Chang, L.-W., Stratton, J. A., Kim, H.-S., & Hwu, W.-M. W. (2012). "A scalable, numerically stable, high-performance tridiagonal solver using GPUs." *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*, Article 27, 11 pages.
10) Dongarra, J., & Sorensen, D. C. (2002). "Computational Algorithms for Symmetric Tridiagonal Eigenvalue Problems." *Journal of Computational and Applied Mathematics*, 138(1), 219-237.
11) Badia, J. M., Movilla, J. L., Climente, J. I., Castillo, M., Marqués, J. M., Mayo, R., & Quintana-Ortí, E. S. (2015). "A Blocked QR-Decomposition for the Symmetric Tridiagonal Eigenvalue Problem on GPUs." *Procedia Computer Science*, 51, 2682-2686.