15-618 Final Project Proposal

Parallelized Multiple Signal Classification (MUSIC) Algorithm

https://15618.bearcmu.com

Bear Xiong (yuxuanxi) and Luojia Hu (luojiah)

Summary

We plan to develop a parallel implementation of the **MUSIC (MUltiple SIgnal Classification)** algorithm. The key steps require computing a covariance matrix and perform eigenvalue decomposition (EVD) on it. Specifically the EVD step is challenging to parallelize, so we will mainly be focusing on it. We will also explore some approximate EVD methods since a few The target system is a multicore CPU (using C++/OpenMP for parallelism and possibly BLAS/LAPACK libraries). Our goal is to demonstrate meaningful speedups over a baseline, without significantly degrading the accuracy of the algorithm's Direction of Arrival (DOA) estimations.

Background

Multiple Signal Classification (MUSIC) algorithm is a high-resolution spectral estimation method used for Direction of Arrival (DOA) estimation or frequency estimation with multiple receivers. It's widely applied in radar, sonar and wireless communications.

Shortly speaking, MUSIC takes in data from an array of receivers, distinguishes the spatial characteristics of noise and signals, and then identifies those directions that are "least likely to be noise" in the parameter space (such as angle or frequency) to estimate the source of the signal.

The algorithm works by:

1. Constructing a covariance matrix from sensor array data.
2. Performing eigenvalue decomposition (EVD) to separate signal and noise subspaces.
3. Computing a pseudospectrum over a parameter space (e.g., angles) to locate signal sources.

Because the covariance matrix is $N \times N$ (where N is the number of antennas or sensors) and eigenvalue decomposition can cost $O(N^3)$, this step often dominates runtime. The rest—forming the covariance matrix, constructing the pseudo-spectrum, searching for peaks—can be parallelized more straightforwardly. Our project will attempt to reduce the runtime of EVD by using approximate techniques such as randomized SVD or Lanczos-based approaches, which can often provide good-enough accuracy for DOA estimation. In other words, the final angular estimates can tolerate slight numerical differences, which is analogous to approximate parallelization strategies we have used in prior assignments (e.g., the VLSI wire routing assignment).

Challenge

Why it's difficult to parallelize:

- **Eigenvalue Decomposition**: Traditional EVD has complex data dependencies. Even with parallel BLAS/LAPACK, scaling beyond a certain point is non-trivial because of synchronization and communication overheads in each iteration of the factorization process.
- **Approximate Methods**: Introducing approximate methods means that while we can gain parallel performance, we must manage trade-offs between approximation error and DOA accuracy. Determining how to distribute computations (e.g., random projections, partial reorthogonalization) to different threads or across different computing nodes also presents challenges.

What we hope to learn: • How to design an approximate parallel algorithm that is both faster and sufficiently accurate for applications like MUSIC. • How to effectively balance concurrency and synchronization, ensuring that approximate parallel EVD is robust enough to provide reliable DOA estimates.

Works Cited

Goals and Deliverables

Plan to Achieve

1. **Baseline Implementation**: Implement or reuse a standard serial MUSIC library with exact eigenvalue decomposition (e.g., via LAPACK). Measure runtime and accuracy for different array sizes N.
2. **Parallel Approximate Eigen Vector Decomposition (EVD)**

- Implement a parallel EVD algorithms.
- Implement an approximate EVD routine using random projection or Lanczos methods.
- Explore using GPU for matrix operations (especially the covariance calculation and pseudospectrum).

3. Evaluation: compare runtime, speedup and DOA accuracy between approximate EVD and exact EVD.

Resources

The resource we need for this project is the GHC/PSC machine cluster, with OpenMP/MPI/CUDA installed.

Schedule

4/8 - 4/14: Basic implementation of B+ tree, supporting insertion, deletion, and search in single thread, write milestone report due on 4/16 4/15 - 4/21: Write milestone report due on 4/16, implement lock free B+ tree according to the literature, generate test cases 4/22 - 4/28: Performance testing of the lock free B+ tree, further optimization if needed 4/29 - 5/5: Compare performance of the lock free implementation with other implementations, write final report