# 15-618 Final Project Milestone Report

# Parallelized Multiple Signal Classification (MUSIC) Algorithm

## Current Progress

We have successfully implemented the serial version of MUSIC and integrated with the ESPARGOS dataset. We verified the correctness of our serial implementation. We identified two methods to parallelize the MUSIC algorithm. The first one is to parallelize between time steps of captures. As there are not dependency between different time steps, this way is considered to be too obvious for this homework. So our parallelism will primarily focus on the second method, in which processors collaborate to compute one time step faster.

Currently, we are exploring **parallelizing the eigenvalue decomposition process** (which is the longest time-consuming computation step) with OpenMP, but encountered a disappointing speedup.

## Preliminary Results

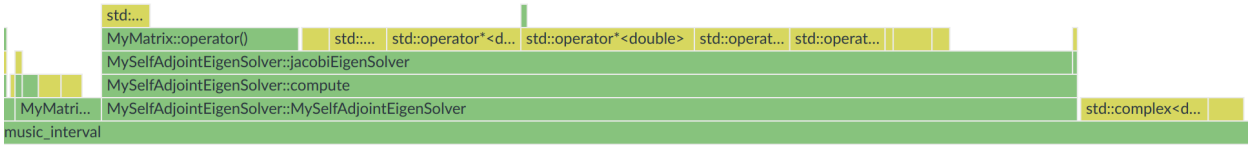


Figure 1: Flame Graph of Serial MUSIC Algorithm

We have implemented the serial MUSIC algorithm as our baseline. We also profiled this implementation to get a flame graph shown in Figure 1. From Figure 1 we can easily spot the performance hotspot of **eigenvalue solving**. Therefore we are first try to parallelize this part of the whole MUSIC algorithm.

For the current parallel implementation, specifically for the Jacobi method eigenvalue decomposition, we found little speedup on multiple threads (about 1.65x speedup on 8 threads), mainly because of the limited parallel region and small matrix size.

Below is our execution time on a dataset with 89083 records, each with 16x16 antenna sensor data.

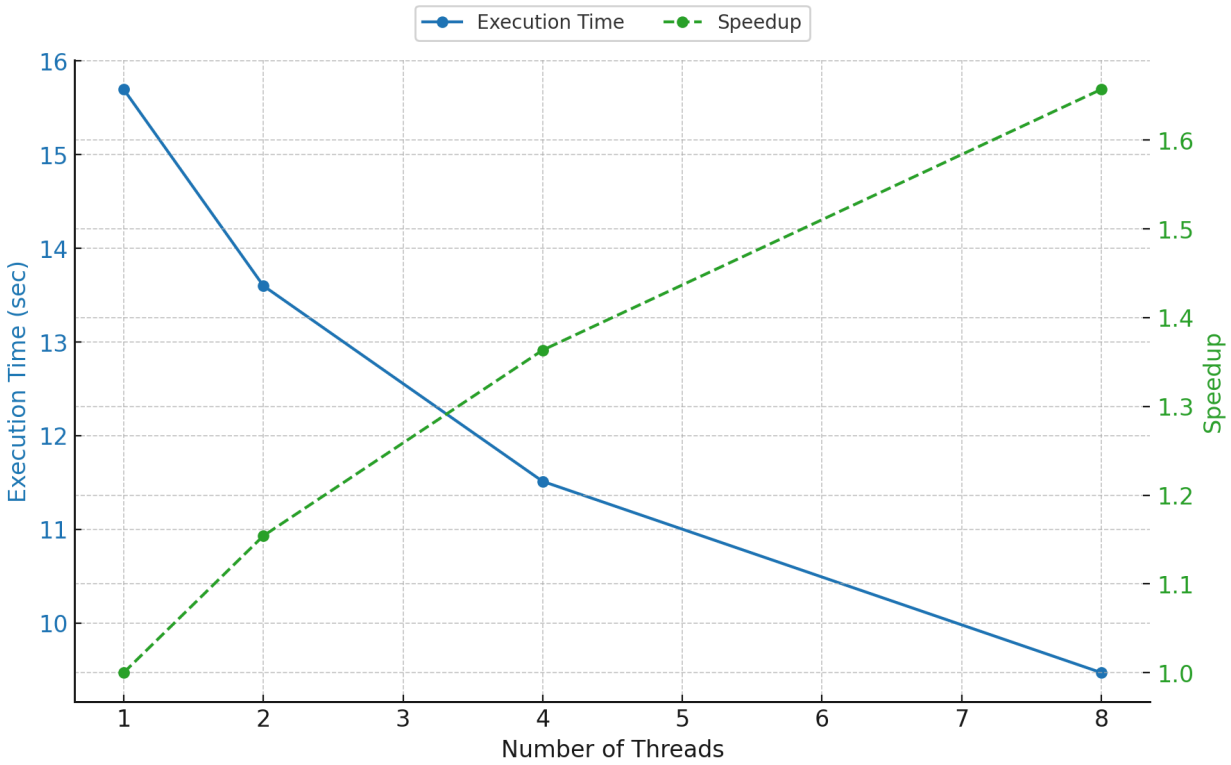


Figure 2: Execution Time and Speedups for Current OpenMP Implementation

## Issues and Concerns

We acknowledge that parallelizing across time steps is relatively straightforward to implement (because there's almost no data dependencies), but we believe this level of parallelism is too obvious and cannot fully meet the assignment requirements. Whereas within each time step, the performance hotspot lies in the eigenvalue algorithm for Hermitian matrices, which inherently lacks straightforward parallelization opportunities as we discussed in previous proposal.

Furthermore, due to the nature of the MUSIC algorithm, the eigenvalue computation operates on matrices whose dimensions correspond to the number of antennas. However, we observe that, in the ESPARGOS dataset, the maximum number of phase-synchronized antennas is only 16. This implies that, even if we optimize the parallel algorithm well, the overhead of thread scheduling could far outweigh the computational benefits.

Currently, we artificially increase the total amount of computation by augmenting the data (i.e., duplicating existing antenna data, we are using 256 antennas currently), but this approach feels rather contrived. Moreover, real-world MUSIC systems do not require such an excessive number of antennas. We recognize that the MUSIC algorithm itself was designed for real-time processing. **Therefore, we are now tend to**

**switch to focus on large Hermitian matrices eigenvalue decomposition, or a completely new topic.**

## Difference with Expectations

Week 1 goal has completed.

- We have "**implemented serial MUSIC algorithm as baseline**", and study eigenvalue decomposition algorithms.
- We have "**established performance metrics and validation strategy**": We run our implementation on the https://espargos.net/datasets/data/espargos-0001/ [espargos-0001-diagonal1 dataset] and make sure our handwritten matrix results is consistent with what the Eigen library calculates.

Week 2 goal has partially completed but we are encountering difficulties, which will be described below.

- We have "**implemented OpenMP parallelization for convariance matrix computation**". This step mainly requires multipling a matrix with its adjoint, and dividing it element-by-element by the interval length. Parallelizing this step is relatively easy.
- **Begin OpenMP implementation for eigenvalue decomposition**: We are currently parallelizing only a portion of this step, specifically:
  - During processing of a specific pair of indices (p,q), after the algorithm computes rotation parameters and saves current rows p and q to a temporary storage, it updates the elements in row p, q and their symmetric column entries for all indices $k$ that are not o or q. We can parallelize on $k$.
  - After the main iterative process, each eigenvector can be normalized in parallel.

  The above technique only gives a very limited speedup (about 1.65x on 8 threads) because some steps cannot or will be very hard to be parallelized:
  - The outer sweep (the main iterative loop) involves repeatedly applying rotations until the matrix becomes nearly diagonal. This cannot be parallelized because the next status of the matrix depends highly on the previous one.
  - In addition, we found it hard to explore different (p,q) pairs that don't share rows or columns in parallel, because these two rotations would interfere strongly with each other.

  The inability to parallelize all sections is part of the reason why our speedup is disappointing.
- ~~**Start CUDA implementation for covariance matrix computation**~~ (haven't done): We didn't do this because our profiling results in Figure 1 shows this step only occupies a small portion of the computation time. Parallelizing it wouldn't improve performance a lot.

Week 3 and 4 goals

- In our proposal we mentioned **approximate computing**, because it would potentially eliminate data dependencies and make more portions parallelizable. However we haven't found any ideas on how to do it yet.

## Updated Goals & Schedule

Our updated schedule & goals for the upcoming two weeks is as follows:

- This week (4.15 - 4.19): Both of us: schedule extra meetings with professors to decide if we continue to work on this topic;

  Bear: write CUDA version of current OpenMP Eigenvalue decomposition and performance test it;

  Luojia: explore if there are more use cases for eigenvalue decomposition using high-dimensional matrices and decide if we switch the topic to eigenvalue decomposition for Hermitian matrices.
- 4.20 - 4.23: If continuing on this topic, explore if it's possible to make more parallelism potentially at the cost of degraded accuracy (like what we did in VLSI wire routing).
- 4.23 - 4.26: Measure results, profile the code, find hotspots to apply further optimization, and repeat this process.
- 4.27 - 4.28: Finalize all implementations, optimizations and performance data, and prepare final report and presentation materials.

## Poster Session

If we continue to go with this topic, we may arrange the layout of our poster in the way below:

- Algorithm Introduction: we will use about one third of the poster layout for introducing the background and basic process.
- Parallelization Implementation: we will describe our parallelism attempts especially on the eigenvalue decomposition step. We will describe what part of the algorithm can be parallelized, and how we tried to parallelize challenging portions.
- Performance Results: we will dedicate one third of the poster to showcase the improvement and reasonings for non-perfect speedups.