# Project Proposal for DS295 - Parallel Programming Parallelizing the Expectation Maximization Algorithm for Gaussian Mixture Models

Ayush Raina

March 3, 2025

## 1 Introduction and Motivation

The Expectation-Maximization (EM) algorithm is widely used for training Gaussian Mixture Models (GMMs) in clustering, density estimation, and anomaly detection. However, its iterative nature and high computational cost make it inefficient for large datasets on traditional CPUs.

Since key operations in EM, such as computing responsibilities in the E-step and updating parameters in the M-step, can be parallelized, GPU acceleration using CUDA offers a promising solution. By leveraging thousands of GPU cores, we can significantly reduce execution time and scale GMM training to larger datasets.

This project proposes a CUDA-based parallel implementation of EM for GMMs. We will explore optimization strategies, memory access patterns, and benchmark performance improvements over CPU implementations to highlight the benefits of GPU acceleration.

## 2 Related Work

The Expectation-Maximization (EM) algorithm for Gaussian Mixture Models (GMMs) has been extensively studied due to its applications in clustering, density estimation, and pattern recognition. Several works have proposed optimizations to improve its efficiency.

Parallel implementations of EM have been explored using multi-core CPUs and distributed computing frameworks. For example, OpenMP-based parallelization has been used to accelerate the E-step and M-step, reducing runtime on shared-memory architectures. Additionally, GPU implementations leveraging CUDA have demonstrated significant speedups by parallelizing matrix operations and probability computations.

Despite these advancements, optimizing memory access patterns, efficient workload distribution, and minimizing synchronization overhead remain open challenges. This project builds upon existing GPU-based approaches and aims to further improve performance through optimized CUDA kernels and memory management techniques.

## 3 Proposed Methodology

Our approach focuses on accelerating the Expectation-Maximization (EM) algorithm for Gaussian Mixture Models (GMMs) using CUDA for GPU-based parallelization. The primary objective is to efficiently compute the E-step and M-step by leveraging parallelism, particularly through data parallelism and optimized summation using shared memory parallelism.

### 3.1 Expectation Step (E-Step) Parallelization

In the E-step, we compute the posterior probability (responsibility) of each data point belonging to a Gaussian component:

$$r_{ik} = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} \tag{1}$$

where $r_{ik}$ is the responsibility of component $k$ for data point $x_i$, $\pi_k$ is the mixture weight, and $\mathcal{N}(x_i|\mu_k, \Sigma_k)$ is the Gaussian probability density function.

To parallelize this step, we assign each data point to a GPU thread, enabling concurrent computation of Gaussian densities and responsibilities. This ensures efficient execution, especially for large datasets.

## 3.2 Maximization Step (M-Step) Parallelization

In the M-step, we update the parameters of each Gaussian component based on the computed responsibilities:

$$N_k = \sum_{i=1}^{N} r_{ik}, \quad \text{(effective number of points assigned to component } k\text{)} \tag{2}$$

$$\pi_k = \frac{N_k}{N}, \quad \text{(updated mixture weight for component } k\text{)} \tag{3}$$

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} r_{ik} x_i, \quad \text{(updated mean for component } k\text{)} \tag{4}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} r_{ik} (x_i - \mu_k)(x_i - \mu_k)^T, \quad \text{(updated covariance matrix for component } k\text{)} \tag{5}$$

where:

- $N$ is the total number of data points.
- $K$ is the total number of Gaussian components.
- $x_i \in \mathbb{R}^d$ represents the $i$-th data point in $d$-dimensional space.
- $r_{ik}$ is the responsibility of component $k$ for data point $x_i$, i.e., the probability that $x_i$ was generated by Gaussian $k$.
- $N_k$ represents the effective number of data points assigned to component $k$.
- $\pi_k$ is the updated mixture weight for component $k$.
- $\mu_k \in \mathbb{R}^d$ is the updated mean of component $k$.
- $\Sigma_k \in \mathbb{R}^{d \times d}$ is the updated covariance matrix of component $k$.

To efficiently compute summations like $N_k$, $\mu_k$, and $\Sigma_k$, we will use **shared memory parallelism** to accelerate reduction operations:

- Each CUDA block loads a subset of data into **shared memory**, allowing fast local summation.
- Within each block, **parallel reduction** is performed in shared memory to compute partial sums.
- A final global sum is computed by aggregating the partial results from all blocks.

Using shared memory ensures **low-latency** summations and reduces **global memory accesses**, leading to a significant speedup in the M-step.

## 3.3 Memory Optimization and Kernel Design

Efficient memory access patterns are crucial for high performance. We employ:

- **Shared Memory:** Reduces global memory access latency during summation.
- **Coalesced Access:** Ensures memory reads and writes are aligned for optimal throughput.
- **Efficient Kernel Launch Configurations:** Choosing appropriate thread and block configurations to balance computational load.

## 3.4 Comparison with CPU Implementation

To assess the benefits of GPU acceleration, we implement a baseline CPU version of the EM algorithm. We compare:

- Execution time for different dataset sizes.
- Speedup achieved with CUDA parallelization.
- Numerical stability and convergence behavior.

This methodology ensures efficient parallel execution while maintaining numerical stability and accuracy. The optimized implementation will be tested on large datasets to measure its scalability and real-world applicability.

## 4 Experimental Plan

To validate our approach, we will follow a structured plan:

### 4.1 Implementation Plan

- Develop a baseline sequential implementation of EM for GMM.
- Implement CUDA-based parallelization for the E-step using parallel reduction.
- Optimize the M-step using shared memory for efficient parameter updates.

### 4.2 Datasets and Evaluation

- We will test our implementation on synthetic datasets and real-world benchmark datasets.
- Metrics for evaluation will include execution time, speedup factor, and convergence behavior.

## 5 Conclusion and Expected Outcomes

This project aims to accelerate the Expectation-Maximization algorithm for Gaussian Mixture Models using CUDA. By leveraging parallel reduction in the E-step and shared memory parallelism in the M-step, we expect significant speedup over CPU-based implementations.

We anticipate the following outcomes:

- Faster execution times compared to sequential implementations, especially for large datasets.
- Efficient memory utilization and optimized parallel computations using CUDA.

## 6 References

1. N. S. L. P. Kumar, S. Satoor and I. Buck, ”Fast Parallel Expectation Maximization for Gaussian Mixture Models on GPUs Using CUDA,” 2009 11th IEEE International Conference on High Performance Computing and Communications, Seoul, Korea (South), 2009, pp. 103-109, doi: 10.1109/HPCC.2009.45.
2. Azizi, Ilia. 2023. “Parallelizing Expectation Maximization (EM).”