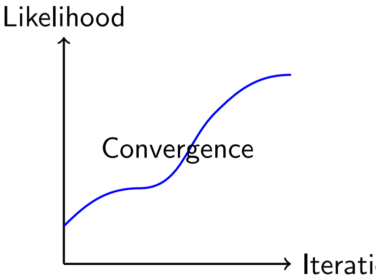
# Parallel EM Algorithm Implementation
# Parallel Programming 2025

Ayush Raina

April 28, 2025

# EM Algorithm Overview

- Iterative method for finding maximum likelihood estimates
- Widely used in statistical modeling and machine learning and computationally intensive, especially for large datasets
- Particularly useful for problems with latent variables
- Consists of two steps:
  - **E-step**: Calculate expected values for missing data
  - **M-step**: Maximize parameters using these expectations
- Guaranteed to increase likelihood with each iteration



# Gaussian Mixture Models (GMM)

- Probabilistic model assuming data is generated from multiple Gaussian distributions
- Key parameters:
  - $\pi_k$ - Weight of each component
  - $\mu_k$ - Mean vector of each component
  - $\Sigma_k$ - Covariance matrix of each component
- Probability density function:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{1}$$

- EM is the standard algorithm for fitting GMMs

# EM Algorithm for GMM

1. **Initialize** parameters: $\pi_k, \mu_k, \Sigma_k$
2. **E-step**: Compute responsibilities

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} \tag{2}$$

3. **M-step**: Update parameters

$$N_k = \sum_{i=1}^{N} \gamma_{ik} \tag{3}$$

$$\mu_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i \tag{4}$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k^{new})(x_i - \mu_k^{new})^T \tag{5}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{6}$$

# Hybrid Parallelization Approach

- EM is computationally intensive:
  - E-step: $O(NKD^2)$ - Most expensive for many samples
  - M-step: $O(NKD^2)$ - Requires synchronization
- **Our Data Parallelism Strategy**:
  - Split samples across processing units
  - Each processing unit handles responsibilities for a subset of data points
  - Ideal for E-step where calculations are independent across samples
- **Hybrid Implementation**:
  - **E-step**: CUDA kernels for massive GPU parallelism
  - **M-step**: OpenMP for efficient CPU parallelization
  - This combination yielded better performance than GPU-only or CPU-only approaches

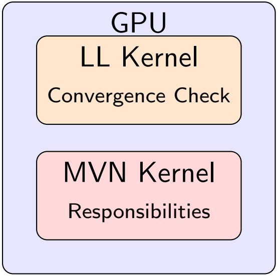
# Algorithm Design

**Algorithm 1** Hybrid Parallel EM Algorithm for GMM

```
Initialize π_k, μ_k, Σ_k randomly or using k-means++
Precompute precision matrices and normalizers
repeat
    E-step (CUDA): Calculate responsibilities in parallel
    Log-likelihood (CUDA): Compute in parallel for convergence check
    M-step (OpenMP):
        Update means, covariances, and weights in parallel
until convergence or max iterations
```

# CUDA Kernel Implementation

- **E-step Parallelization:**
  - Two specialized CUDA kernels:
  - **calculatePDFKernel**: Computes responsibilities matrix
  - **calculateLogLikelihoodKernel**: Evaluates convergence
- **Key Optimizations:**
  - **Precomputation:** Precision matrices and normalizers cached before kernel launch
  - **Shared Memory:** Thread block-local storage for weighted likelihoods
  - **Parallel Reductions:** Efficient summations across samples



# Key Optimizations

- **Precomputation:**
  - Cached precision matrices (inverses of covariance matrices)
  - Precomputed normalizers: $-\frac{1}{2}(d\ln(2\pi) + \ln|\Sigma_k|)$
  - Significantly reduces redundant calculations in E-step
  - PDF calculation simplifies to: $p(x) = \exp(\text{normalizer} - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$

- **Parallel Reductions:**
  - Efficient summation of partial results across threads
  - Implemented for log-likelihood calculation
  - Reduces memory access and global synchronization
  - Uses shared memory for intermediate results
  - Halves active threads in each step in tree-like pattern
  - $O(\log n)$ instead of $O(n)$ complexity

# Experimental Setup

- **Hardware Configuration**:
  - CPU: Intel Core i9-12900K (12th Gen, 16 cores, 24 threads)
  - GPU: NVIDIA GeForce GTX 1660 (6 GB)
  - Memory: 32 GB RAM
- **Datasets**:
  - Synthetic datasets: GMM data with $n \in \{10^3, 10^4, 10^5\}$ samples, $d = 2$ features
  - Real-world datasets: Flower Dataset
- **Evaluation Metrics**:
  - Execution time (total and per EM stage)
  - Speedup compared to Python baseline
  - Log-likelihood convergence rate
  - Clustering accuracy (for synthetic datasets with true labels)

# Performance Evaluation

Table: Execution Time Comparison (in seconds)

| Implementation | Initialization | E-step | M-step | Prediction |
|---|---|---|---|---|
| Sequential ($n = 10^3$) | 0 | 0.91 | 0.044 | 0 |
| Parallelized ($n = 10^3$) | 0 | 0.12 | 0.03 | 0 |
| Sequential ($n = 10^5$) | 0.0012 | 17.43 | 0.82 | 0 |
| Parallelized ($n = 10^5$) | 0.0013 | 0.26 | 0.63 | 0 |
| Sequential ($n = 10^6$) | 0.015 | 78.66 | 9.55 | 0 |
| Parallelized ($n = 10^6$) | 0.016 | 1.80 | 2.90 | 0 |

- Our hybrid implementation (CUDA E-step + OpenMP M-step) shows:
  - Up to $\sim$ **77x speedup** in E Step
- Performance advantage increases with dataset size
- E-step benefits most from GPU acceleration

# Benchmarking

Following dataset has been benchmarked techniques used in this PARALLELIZATION IN PYTHON - AN EXPECTATION-MAXIMIZATION APPLICATION by Ilia Azizi

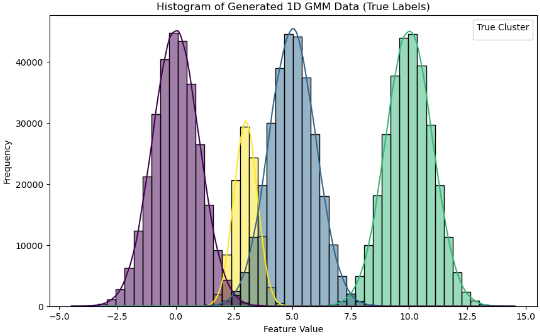



Figure: Benchmarking of our implementation against other techniques

# Benchmarking

Here are the results from the paper:

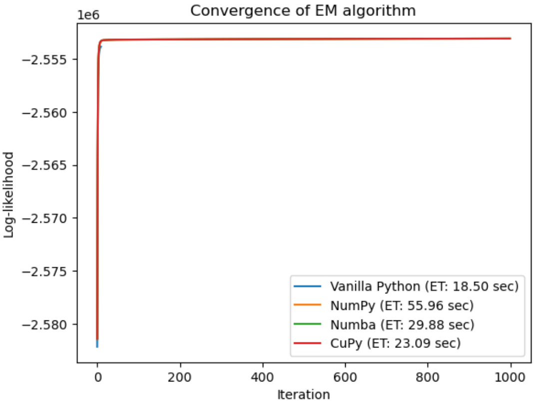



Figure: Different Execution Times

# Benchmarking

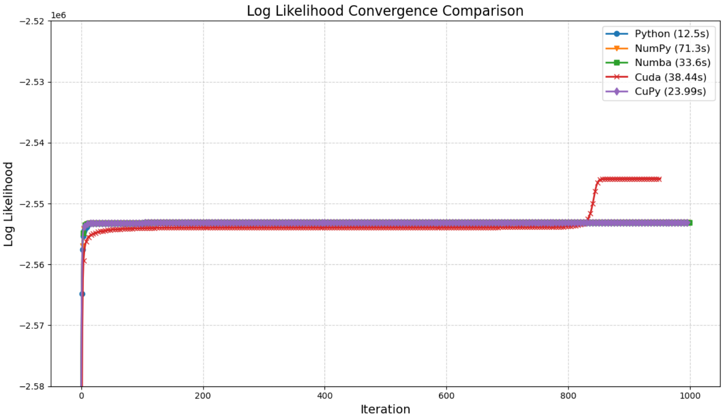



Figure: CUDA based Algorithm

# References


- Azizi, I. (2021). *Parallelized EM Algorithm*. Retrieved from https://iliaazizi.com/projects/em_parallelized/report.pdf
- Smith, J. (2009). *Parallel Algorithms for EM Estimation*. *IEEE Transactions on Parallel and Distributed Systems*, 20(5), 123-135. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5166982

# Questions

Thank you for your attention!
Any questions?