Parallel Implementation of Expectation Maximization algorithm with K-Means clustering initialization in MPI

Abinaya K[1], Ritika Ratnu[2]
Department of Computation and Data Sciences
Indian Institute of Science, Bangalore, India
[1]abinayak@iisc.ac.in, [2]ritikaratnu@iisc.ac.in

***Abstract*—With growing interests in modelling and analysis of data and the avilability of large data, it becomes time consuming to deal with sequential Machine Learning algorithms. Gaussian Mixture Models (GMM) are popular probabilistic models that assume data points are generated from finite number of Gaussian densities. Expectation Maximization(EM) algorithm is often used in the Maximum Likelihood(ML) estimation of GMM parameters. Given the importance of EM algorithm and the embrassingly parallel steps involved in the iterations of EM algorithm motivates the need for parallelization of EM algorithm. Our results show good speed ups for varying input sizes.**

Keywords: Expectation Maximization, K-Means Clustering

I. Introduction

Finite Mixture models are widely used for modelling and analysis of data. ML estimation of parameters of such models is often done using EM algorithm. One popular mixture model is GMM with the mixture components as Gaussian. The time taken for convergence of EM algorithm depends on its initializations. Also, the algorithm might converge to a local maxima of log likelihood function due to skewed initializations. Hence, clustering is used to give EM algorithm good initializations so that EM converges faster. The objective of the project is to parallelize EM algorithm with K-Means initialization in distributed memory platforms using MPI.

II. Related Work

Several works were done in parallel implementation of K-Means clustering and EM algorithm in different plaforms - OpenMP, MPI and CUDA. Some relevant works are discussed here. In the MPI implementation of KMeans by Bhimani et al. in [1], Master Slave model is followed, where cluster assignment is done by slaves in parallel and the slaves send labels of data points it owns to the master for computation of new means. In our work, we use Master Slave paradigm similar to [1]. Our approach would differ from [1] as follows. Instead of sending the whole label array of cluster assignments, the slaves can compute local sum and local count for the datapoints and send them to the Master process. This way, we exploit more parallelism. Each process would only send 2K data to the Master process instead of N, thereby reducing the load of computations by the Master process and also the bandwidth of communication is reduced to 2K, irrespective of the total number of datapoints. This idea is inspired from [2] in the way the number of datapoints in each cluster being computed in GPU. Lee et al. have parallelized EM algorithm in GPUs in [3]. The approach followed in [3] is to divide the computations with respect to K components across K threads. The naive translation of this approach to MPI would be to divide computations for K components across K processes. Our approach divides data between processes.

III. Methodology

EM Algorithm is an iterative algorithm used for estimating mixture of densities. The probability density function of mixture density of K components is given by

$$f(x/\theta) = \sum_{k=1}^{K} \lambda_k f_k(x/\theta_k) \quad , \sum_{k=1}^{K} \lambda_k = 1 \tag{1}$$

where θ denotes the parameters of mixture density function, λ_k denotes the mixing coefficient of k^{th} component and θ_k denotes the parameters of k^{th} component in mixture density. GMM is a mixture density model, where the components of the mixture are Gaussian i.e, $f_k \sim N(\mu_k, \Sigma_k)$. Here μ_k and Σ_k represents the mean and covariance of k^{th} Gaussian density. For a D dimensional input, mean μ_k is a vector of size $D * 1$ and covariance Σ_k is a matrix of size $D * D$ and λ_k is a scalar.

$$f_k(x) = \frac{1}{\sqrt{|\Sigma_k|(2\pi)^D}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)} \tag{2}$$

The parameter vector θ for a Gaussian Mixture Model of K components is $\theta = (\lambda_k, \mu_k, \Sigma_k), k = 1, ..., K$. EM Algorithm estimates the parameters of GMM by maximizing its log likelihood. In each iteration of the algorithm, there are two steps as follows.

- E Step - Given the current value of parameters, responsibilities are estimated.

$$\gamma_{ij}^{(k)} = \frac{\lambda_j^{(k)} f_j(x_i/\theta_j^{(k)})}{\sum_{j=1}^{K} \lambda_j^{(k)} f_j(x_i/\theta_j^{(k)})} \tag{3}$$

, $i = 1, ..., N$ and $j = 1, ..., K$, N is the number of points. The superscript (k) indicates k^{th} iteration. γ is a

matrix of size $N * K$. γ_{ij} is the probability that the i^{th} data point belongs to j^{th} mixture component.

- M Step - Given the responsibility values obtained with the current value of parameters, estimate the new value of parameters.

$$\lambda_j^{(k+1)} = \frac{1}{N}\sum_{i=1}^{N}\gamma_{ij}^{(k)} \tag{4}$$

$$\mu_j^{(k+1)} = \frac{\sum_{i=1}^{N}\gamma_{ij}^{(k)}x_i}{\sum_{i=1}^{N}\gamma_{ij}^{(k)}} \tag{5}$$

$$\Sigma_j^{(k+1)} = \frac{\sum_{i=1}^{N}\gamma_{ij}^{(k)}(x_i-\mu_j^{(k)})^2}{\sum_{i=1}^{N}\gamma_{ij}^{(k)}} \tag{6}$$

The convergence of this algorithm is highly sensitive to initialization. A good initialization of means can be the output cluster centroids from any clustering technique. K-Means clustering is a simple unsupervised learning algorithm for clustering data into groups. We use K-Means clustering with Euclidean distance as the measure of distance to initialize the means of GMM in EM algorithm.

A. Sequential Implementation

Sequential implementation of EM algorithm with K-Means initialization involves the following three steps in sequence.

1 Load Data
2 Use K-Means custering algorithm to get good initialization for EM
3 Use EM algorithm to estimate the parameters of Gaussian Mixture Density

Algorithm 1 K-Means Clsutering Algorithm

Inputs: Samples $X_1, X_2, ..., X_N$ and K
Initialize $\mu_j^{(0)}$, $j = 1, ..., K$ arbitrarily, $k \leftarrow 0$
while Convergence Not reached **do**
  for $i \leftarrow 0$ to $N-1$ **do**
    $L_i = \underset{j}{\operatorname{argmin}}||X_i - \mu_j^{(k)}||_2$
  end for
  for $j \leftarrow 1$ to K **do**
    $\mu_j^{(k+1)} = \frac{1}{N_j}\sum_{i=0}^{N-1} 1_{L_i==j} X_i$
  end for
  $k \leftarrow k+1$
end while
Outputs: Centroids μ_j, $j = 1, ..., K$ and Label L

Algorithm 1 shows the steps involved in K-Means clustering. To start with, K centroids are chosen randomly. In each iteration, the elements are assigned to the closest cluster. The centroids are then recomputed as the mean of datapoints assigned to it. These two steps are repeated iteratively. Convergence is achieved when no element moves between clusters in successive iterations. Since our aim is to get good initialization for means from K-Means clustering and not necessarily the accurate clustering results, our criteria for termination of algorithm is either if the max iterations is reached or if convergence is acheived before max iterations is reached (i.e, no points change cluster). For faster convergence of EM algorithm, K-Means clustering is used to initialize the means of Gaussians.

The output centroids from K-Means can be used to initialize the means for EM algorithm. Variance of the K components in the mixture density is initialized as the sample variance of elements assigned to each cluster. The mixing coefficients λ is initialized as the proportion of points assigned to each cluster.

Algorithm 2 shows the steps involved in EM algorithm. Given the initial values of the parameters (means, covariances, mixing coefficients), the algorithm alternates between E and M steps iteratively. Convergence is acheived when the relative change in parameter values is less than a threshold.

Algorithm 2 EM Algorithm

Inputs: $X_1, X_2, ..., X_N$, K, $\lambda^{(0)}, \mu^{(0)}, \Sigma^{(0)}$ and tolerance
$k \leftarrow 0$
while Convergence Not reached **do**
  for $i \leftarrow 0$ to $N-1$ **do**
    for $j \leftarrow 1$ to K **do**
      $\gamma_{ij}^{(k)} = \frac{\lambda_j^{(k)} f_j(x_i/\theta_j^{(k)})}{\sum_{j=1}^{K}\lambda_j^{(k)} f_j(x_i/\theta_j^{(k)})}$
    end for
  end for
  for $j \leftarrow 1$ to K **do**
    $\lambda_j^{(k+1)} = \frac{1}{N}\sum_{i=1}^{N}\gamma_{ij}^{(k)}$
    $\mu_j^{(k+1)} = \frac{\sum_{i=1}^{N}\gamma_{ij}^{(k)}x_i}{\sum_{i=1}^{N}\gamma_{ij}^{(k)}}$
    $\Sigma_j^{(k+1)} = \frac{\sum_{i=1}^{N}\gamma_{ij}^{(k)}(x_i-\mu_j^{(k)})^2}{\sum_{i=1}^{N}\gamma_{ij}^{(k)}}$
  end for
  $k \leftarrow k+1$
end while
Outputs: $\lambda^{(k)}, \mu^{(k)}, \Sigma^{(k)}$

B. Parallel Implementation

Parallel implementation of our work involves the same three steps of data loading, K-Means initialization and EM algorithm in sequential fashion.

The input data of N elements is partitioned to P processes using 1D Block partitioning. Each process is given $\frac{N}{P}$ elements each of D dimension. The data is read parallely by P processes using MPI Parallel IO.

In our parallel implementation of K-Means clustering, master slave paradigm is followed. The Master process initializes the means randomly and broadcasts the initial means using MPI_Bcast. First for loop of the algorithm 1 i.e., assignment of each element to clusters is independent of any other element. All processes computes cluster assignment for the elements they own. The next step of recomputing the means involves division of sum of points assigned to each cluster by the number of points assigned to each cluster. Each process computes the local sum and local count of elements assigned

to each cluster for the elements it owns. Master process collects the local sum and local counts from all processes using MPI_Reduce with a MPI_Sum operation. Master process then computes new means and broadcasts them to all processes. Each process maintains a flag of whether any of the points it own moved between clusters. These local flags are brought to Master process with MPI_Reduce and if globally no points moved between clusters or if max iterations is reached, Master process broadcasts the flag showing convergence is acheived and all the process stop their iterations.

Communications are involved in each iteration for broadcating the means - K*D elements for D dimensional data. For the reduction operation, each process sends K*D elements for local sum and K elements for the local count.

Initial covariance needs to be computed as the sample covariance of the elements assigned to each cluster. i.e., Covariance matrix for j^{th} component is initialized as below.

$$\Sigma_j = \frac{1}{N_j} \sum_{X_i \in C_j} (X_i - \mu_j)^2$$

where N_j is the number of points assigned to the j^{th} cluster and C_j is the set of all elements assigned to j^{th} cluster. Each process computes the local sum and local count for the elements it owns. Master process collects local sum and local counts using MPI_Reduce with MPI_Sum operation. Master process then computes the covariance of each cluster.

Mixing coefficients are initialized as the proportion of points assigned to each cluster as below.

$$\lambda_j = \frac{N_j}{N}$$

where N_j is the number of points assigned to j^{th} cluster and N is the total number of points. Master already has the number of points assigned to each cluster used in computing covariance and hence computes the initial coeffitients λ without any additional communication.

Master process has the initial values of parameters for EM algorithm. The rest of this section explains the parallelization of EM algorithm. Similar to K-Means parallelization, Master Slave paradigm is followed for parallelization of EM algorithm. Master process broadcasts the initial values of parameters to all processes. Given the current value of parameters, the computation of γ for different elements are independent. Hence each process computes γ_{ij} for $i = 1, 2, ..., \frac{N}{P}$, $j = 1, ..., K$. Computation of γ_{ij} involves evalauting the probability of X_i belonging to $j^{(th)}$ cluster. Evaluation of this probability as shown in equation 2, involves computing $(X_i - \mu_j)^T \Sigma_j^{-1} (X_i - \mu_j)$ and the determinant of Σ_j, where X_i and μ_j are D dimensional vector and Σ_j is a matrix of dimension $D * D$. This needs computation of inverse of $D * D$ matrix. Lapack package is used for computation of inverse and determinant of $D * D$ matrix. In M step, given the current value of parameters and the γ values, new values of parameters need to be computed as given by equations 4, 5 and 6. Computation of all parameters involves summation over all elements $i = 1, ..., N$ and computation with respect to one element is independent of other. Each process computes local values of numerator for the $\frac{N}{P}$ elements it own. Master process collects local numerator values using MPI_Reduce with MPI_Sum operation. Master process then computes the updated value of parameters and broadcasts them to all processes for use in next iteration. Master process computes the relative change in parameter values and based on whether or not the relative change is less than a tolerance value, Master process broadcast a flag (1 if convergence acheived, 0 if convergence is not acheived). Based on the value of the flag received, the other process decide to continue with the iterations or to come out of iterations.

The amount of communication involved in broadcasting the parameters is $K + (K * D) + (K * D * D)$ where K for mixing coefficients λ, $K * D$ for the K means and $K * D * D$ for K covariance matrices. The amount of data sent by each process in MPI_Reduce in M Step comes out to be $K + (K * D) + (K * D * D)$.

IV. Experiments and Results

The performance of parallelized algorithm is evaluated using the speed up obtained. Speedup in parallelization is calculated as

$$Speed\ \ up = \frac{T_{sequential}}{T_{parallel}}$$

where $T_{sequential}$ and $T_{parallel}$ are the sequential and parallel execution times.

In order to observe the effect of number of elements on the speed up, below experiment is done. The number of clusters is fixed to $K = 2$ and the dimensionof data is set to $D = 2$. The number of elements is varied and the speed up of the parallel algorithm is observed for varying number of processes. Figure 1 shows the speedup performance of parallel algorithm for varying input sizse. The x axis is shown in log scale for better readability. The curves in different colors are for different number of processes.

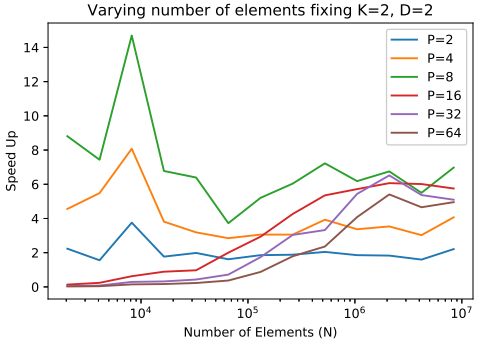

Fig. 1. Performance of parallelized algorithm for varying input size and number of processes

We observe that for smaller input size, the speed up increases with increasing number of processes from 2 to 8. However for number of processes greater than 8, the speedup

is found to saturate. For larger number of processes ($P > 8$), the speed up is found to increase with increasing input size.

Figure 2 shows the execution times in seconds for two different parameter setting. The first subfigure shows the execution time with $N = 2048$ where the second is with $N = 524,288$. With lesser input size the time taken for K-Means initialization is comparable to that of EM algorithm, while for greater input size, the time taken for K-Means is almost negligible compared to EM algorithm. It is also observed that the number of iterations for convergence of EM algorithm decreases with K-Means initialization rather than a random initialization. This shows that using K-Means for initialization doesn't add much to the execution time, but results in faster convergence.

```
Number of Elements = 2048, D = 2, K = 2
Number of processors=1
Time taken for file reading : 0.000564
Time taken for kmeans : 0.016567
Time taken for EM : 0.130554
Time taken for kmeans + EM: 0.147120
Total time taken : 0.147684
```

```
Number of elements = 524288, K = 2, D = 4
Number of processors=1
Time taken for file reading : 0.016117
Time taken for kmeans : 0.550115
Time taken for EM : 90.391741
Time taken for kmeans + EM: 90.941856
Total time taken : 90.957973
```

Fig. 2. Execution times of Sequential Implementation

In order to observe the performance of parallelized algorithm with respect to the number of mixtre components K, the below experiment is done. Fixing the input size constant, the number of mixture components is varied. Figure 3 shows the plot of execution times for sequential and parallel implementation for varying choice of K values.

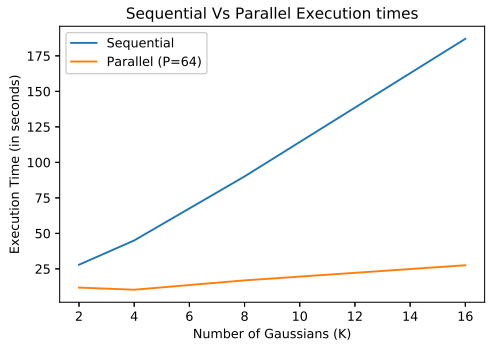

Fig. 3. Execution Times of sequential, parallel algo for varying K

We observe that with increase in the value of K, the execution time increases for both sequential and parallel algorithms. This is expected since the amount of computation increases with increasing K value. Also, the rate of increase in execution time for increasing value of K for parallel algorithm is lesser than that of sequential algorithm, since the computations are for $\frac{N}{P}$ elements in case of parallel while in sequential algorithm, the computations are for N elements.

V. Experimental comparision with existing strategies

Figure 4 compares the speed up of K-Means with our approach (computing local sum and local counts in slaves) and the base paper strategy (labels of all elements are brought to master process). For $K = 2$ and $D = 2$, figure shows the plot of speed up for varying input sizes for different number of processes. The earlier plots in this report are for EM with K-Means while this plot shows speed up with K-Means to illustrate the performance of our approach. We observe that speed up of our approach is greater than that of base paper.

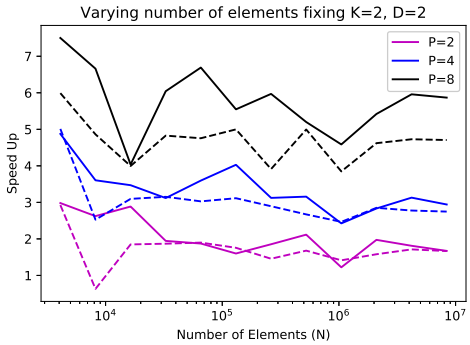

Fig. 4. Speed up K-Means clustering with base paper strategy (solid lines - our approach, dashed lines - base paper strategy)

VI. Conclusion

Parallelized version of EM algorithm with K-Means algorithms is found to give good speed up performance. With smaller number of elements, speed up increases with increasing number of processes till $P = 8$, for $P > 8$, speed up drops. For larger number of processes ($P > 8$), the speed up is found to increase with increasing input size. The rate of increase in execution time for increasing value of K for parallel algorithm is lesser than that of sequential algorithm. K-Means initialization is obseved to result in faster convergence of EM Algorithm. Our approach for K-Means parallelization is found to show better speed up than the base paper [1] approach.

VII. Acknowledgement

We extend our sincere thanks to TA Abhishek for quick turnaround in installing Lapack in turing cluster.

References

[1] J. Bhimani, M. Leeser, and N. Mi, "Accelerating k-means clustering with parallel implementations and gpu computing," in *High Performance Extreme Computing Conference (HPEC), 2015 IEEE*. IEEE, 2015, pp. 1–6.

[2] M. Baydoun, M. Dawi, and H. Ghaziri, "Enhanced parallel implementation of the k-means clustering algorithm," in *Advances in Computational Tools for Engineering Applications (ACTEA), 2016 3rd International Conference on*. IEEE, 2016, pp. 7–11.

[3] S. X. Lee, K. L. Lee, and G. J. McLachlan, "A simple multithreaded implementation of the em algorithm for mixture models," *arXiv preprint arXiv:1606.02054*, 2016.