(ACM SRC Poster) PEAK: Parallel EM Algorithm using Kd-tree

Laleh Aghababaie Beni and Aparna Chandramowlishwaran (Advisor)
University of California, Irvine
laghabab, amowli@uci.edu

***Abstract*—The data mining community voted Expectation Maximization (EM) algorithm as one of the top ten algorithms having the most impact on data mining research [5]. EM is a popular iterative algorithm for learning mixture models with applications in various areas from computer vision, astronomy, to signal processing. We present a new *high-performance parallel* algorithm on multicore systems that impacts all stages of EM. We use tree data structures and user-controlled approximations to reduce the asymptotic runtime complexity of EM with significant performance improvements. PEAK utilizes the same tree and algorithmic framework for all the stages of EM.**

Experimental results show that our parallel algorithm significantly outperforms the state-of-the-art algorithms and libraries on all dataset configurations (varying number of points, dimensionality of the dataset, and number of mixtures). Looking forward, we identify approaches to extend this idea to a larger scale of similar problems.

I. Introduction

EM is popular and used in a wide variety of applications of mixture models. In the area of speech recognition, Gaussian Mixture Model (GMM) based speaker model is used to present acoustic classes. In data stream clustering, EM is used to learn parameters of mixture models while the data streams are generated by GMMs. In the field of computer vision, researches have applied EM to several problems from image motion and image segmentation to visual learning.

We focus on the classical EM algorithm which is an iterative algorithm. It includes two main steps: (1) Expectation or **E-step**, and (2) Maximization or **M-step**. After each iteration, the algorithm measures the objective function, **log-likelihood**, to maximize the likelihood of data under the model parameters. The EM algorithm is proved to monotonically converge to some local maximum of the likelihood; therefore, one can check the correctness of the algorithm by observing whether or not the log-likelihood function increases after each parameter update. The computation terminates when the model parameters have converged. We use the EM algorithm for GMMs because the Gaussian mixtures have been shown to have a smooth approximation to the sample distribution in many applications [2].

Overall, we claim **two main contributions**. The first is a fast algorithm for EM. We observe that $\approx 99\%$ of the total time in an EM algorithm is spent in two stages namely, E-step, and log-likelihood calculation. We present a tree-based approximation algorithm for both stages that extends previously proposed algorithms for just E-step [3], [4].

Secondly, we present the first extensive performance study of EM on multicore architectures. This includes optimizations and parallelization such as numerical optimizations and Cilk-based parallelization. We compare our parallel EM against state-of-the-art libraries in different programming languages such as Python, Java, and C++ on real world and synthetic datasets. This lays the foundation for future ultra-scalable libraries and domain-specific compilers on current and emerging high-end systems that can process massive datasets.

II. Fast EM Algorithm

Moore [3] presented a space-partitioning tree-based algorithm to reduce the complexity of the E-step. We extend this idea and propose a fast algorithm for estimating both the E-step and log-likelihood computation.

Kd-trees are commonly used in data analysis and mining applications. This data structure allows us to efficiently compute the minimum and maximum node-to-point distances during evaluation without accessing the actual points in each node. Algorithm 1 shows the pseudo-code of tree traversal given three inputs, a node of the reference tree, $\mathcal{N}_r$, a query point, q, and a *rules* closure, $\mathcal{R}$. We perform four operations:

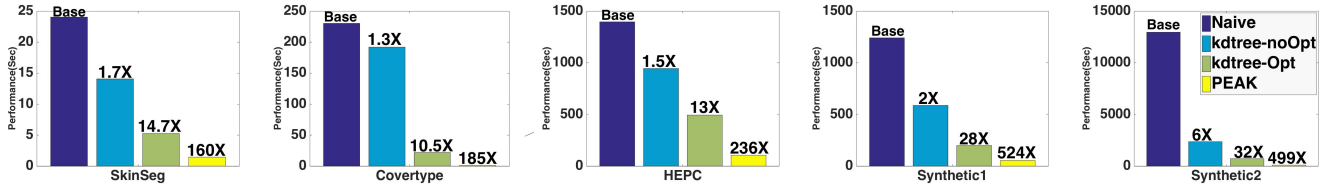
- **BaseCase(node, point)** implements the direct point-to-point computation.
- **Prune(node, point)** checks to see if the node can be *approximated* based on criteria defined by the rules closure.
- **CentroidCase(node, point)** returns the *approximation* of the pruned node.
- **VisitOrder(node, point)** returns the order in which the children of the node should be traversed.

Algorithm 1 TreeTraversal($\mathcal{N}_r$, q)

Input: reference node $\mathcal{N}_r$, query point q, rules closure $\mathcal{R}$.

```
if R.Prune(N_r, q) then
    return R.CentroidCase(N_r, q)
if N_r is leaf then
    return R.BaseCase(N_r, q)
else
    C_r = R.VisitOrder(N_r, q)
    for all N_rc ∈ C_r do
        TreeTraversal(N_rc, q)
```

We use the same tree-traversal template in Algorithm 1 for both E-step and log-likelihood computations albeit with



(a) (b) (c) (d) (e)

Fig. 1. Performance comparison: Naive vs kd-tree vs kd-tree-opt vs PEAK implementations for five datasets outlined in Table I.

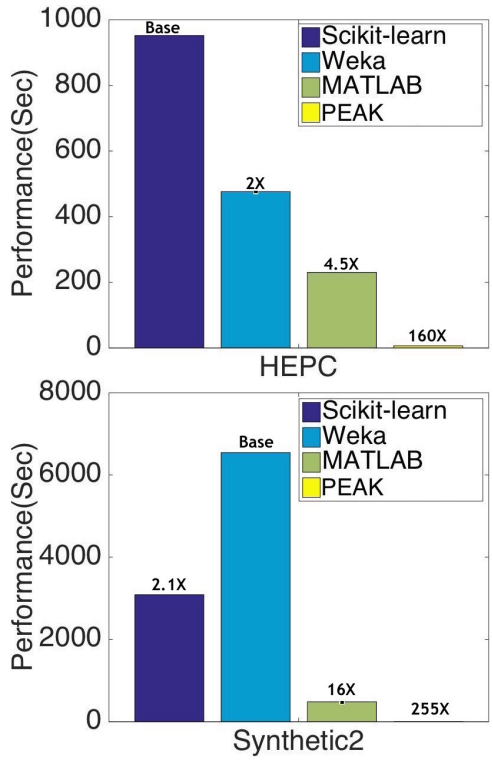


Fig. 2. Comparison against other state-of-the-art libraries for two data sets HPEC and Synthetic2.

a different base case, centroid case, and pruning conditions that is specific to each stage.

III. Optimizations and Parallelization

We apply numerical optimizations which require exploiting domain-specific knowledge, parallelize the tree traversal, and tune for algorithmic parameters such as leaf size.

Two main computationally intensive calculations in EM are computing the responsibility (in E-step) and log-likelihood equation; together, they constitute a significant portion of the overall runtime. This is primarily attributed to the Gaussian estimation which we optimize by a combination of Cholesky decomposition and forward substitution. After optimizing the EM algorithm, we parallelize it using Cilk to exploit modern multicore architectures. We parallelize the tree-traversal in Algorithm 1 by employing task-parallelism to recursively spawn threads using **cilk_spawn** at each level of the tree until the number of tasks spawned is on the order of the cores on the target machine in both E-step and Log-likelihood.

We present results on three real world datasets from UCI machine learning repository [1] as well as two synthetically generated datasets characterized in Table I. We evaluate our implementations on *Zelda*, a dual-socket 8-core Intel Xeon E5-2630 v3 processor (Haswell-EP). We use Intel C++ complier (icpc version 15.0.2) with C++11 feature support.

TABLE I
Summary of the benchmark datasets.

Datasets	N	dim	$clusters$
SkinSeg	245057	3	2
Covertype	581012	5	15
HEPC	2075259	4	8
Synthetic2	1000000	2	64
Synthetic1	1000000	4	4

We present results for (a) naive EM, (b) kd-tree EM without optimizations, and (c) kd-tree EM with optimizations and parallelization which helps distinguish the algorithmic improvements from optimization and parallelization. Figure 1 shows $1.3 - 6\times$ speedup from the kd-tree algorithm and $94 - 267\times$ from optimization and parallelization. We observe $10-18\times$ parallel scaling on 32 threads of Zelda. Cumulatively, we observed significant speedups from $160 - 524\times$.

Finally, we compare PEAK against state-of-the-art libraries in different programming languages from Python to C++ on real world and synthetic datasets. These libraries include Scikit-learn, WEKA toolkit, and MATLAB. PEAK significantly outperforms these libraries by $160 - 255\times$ as seen in Figure 2 reinforcing the need for a new software infrastructure targeted towards current and future architectures.

References

[1] A. Frank, A. Asuncion, et al. UCI machine learning repository. 2010.

[2] N. Kumar, S. Satoor, and I. Buck. Fast parallel expectation maximization for Gaussian mixture models on GPUs using CUDA. In *High Performance Computing and Communications, 2009. HPCC'09. 11th IEEE International Conference on*, pages 103–109. IEEE, 2009.

[3] A. W. Moore. Very fast EM-based mixture model clustering using multiresolution KD-trees. *Advances in Neural information processing systems*, pages 543–549, 1999.

[4] S.-K. Ng and G. J. McLachlan. Speeding up the EM algorithm for mixture model-based segmentation of magnetic resonance images. *Pattern Recognition*, 37(8):1573–1589, 2004.

[5] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.