The document discusses the development of a parallel EM (Expectation Maximization) algorithm using a kd-tree approach. The main contributions include:

**Motivation & Contributions**

- **Why Expectation Maximization?**
  - One of the top ten algorithms having the most impact on data mining
  - Popular iterative algorithm for learning mixture models
  - Apps: computer vision, machine learning, astronomy, and signal processing

- **How did we improve it?**
  - 99% of total time of EM is spent in two stages namely E-step and Log-likelihood
  - We present a tree-based approximation algorithm for both of these stages
  - We introduce a tree-based approximation algorithm for computing the Log-likelihood
  - We present the first extensive performance study that includes various optimizations and parallelization

**EM algorithm**

- Gaussian Mixture Models (GMM): $p(x|\theta) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)$
- EM is a popular algorithm for fitting GMM parameters
- **Expectation Maximization (EM)**
  - Initialize parameters
  - Repeat until convergence of Log-likelihood:
    - E-step
    - M-step
    - Log-likelihood
- **EM using kd-tree**
  - We use kd-tree based algorithm to approximate and discard regions of space to reduce the asymptotic complexity of EM
  - We use kd-tree based algorithm to approximate and discard regions of space to reduce the asymptotic complexity of EM
  - Traversing the kd-tree for EM includes 4 operations: BaseCase, Prune, CentroidCase, VisitOrder
  - BaseCase: Direct point-to-point distance computation
  - Prune: Check to see if the node can be approximated
  - CentroidCase: Centroid approximation of the pruned node
  - VisitOrder: Order of tree traversal

**TreeTraversal($N_v$, q)**

- Input: reference node $N_v$, query point q, rules closure R
- if R.Prune($N_v$, q) then return R.CentroidCase($N_v$, q)
- if $N_v$ is leaf then return R.BaseCase($N_v$, q)
- else $C_v = R.VisitOrder(N_v, q)$
- for all $N_v \in C_v$ do TreeTraversal($N_v$, q)

**Comparison with other Libraries**

- We compared our algorithm against three state-of-the-art libraries:
  - **Weka**: Waikato environment for data mining written in Java
  - **SciKit-learn**: Python module built on top of numPy and sciPy
  - **MATLAB**: Uses C in the backend

**PEAK**

- We present a new high-performance parallel algorithm on multicore systems for EM:
  - Tree-based approximation algorithm for E-step (previous work)
  - Tree-based approximation algorithm for computing the Log-likelihood (our contribution)
  - Optimizations and multicore parallelization (our contribution)

**Log-likelihood Algorithm**

- Initializing the Log-likelihood
- BaseCase: Computing the Log-likelihood only for points in the leaf
- Prune: Check the below criteria: $\l (r_{\text{max}} - r_{\text{min}}) < \beta r_{\text{total}}$ ($i = 1,...,K$)
- CentroidCase: Log-likelihood computation for center of hyper-rectangle

**Parallelization**

- Parallelizing the post-order tree traversal for both stages
- Using Cilk work-stealing scheduler
- Task-level parallelism (cilk_spawn)

**Optimization**

- Compiler optimizations
- Loop fusion, inlining
- Numerical optimizations
- Cholesky decomposition
- Forward substitution

**Performance Results**

- **Summary of benchmark datasets**
- **Speedup over the naive baseline code for three datasets**
- **Parallel scaling of PEAK**
  - *System spec*: Dual-socket Intel Xeon E5-2630 v3 processor (Haswell). Each socket has 8 cores, total of 16 cores (with Intel C++ compiler).

**Conclusion & Future Work**

- We introduced a parallel EM algorithm using the same tree for all the stages
- Our result shows up to 500x speedups on real world and synthetic datasets.
- We will extend this idea to larger classes of similar machine learning algorithms such as nearest neighbors, density estimation, and range search.
- We are currently building a N-body DSL and code generator that will provide a high-level interface with high performance on target platforms for domain scientists.

**References**