**Summary:** A New Scalable Parallel DBSCAN Algorithm Using the Disjoint-Set Data Structure, by Md. Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, Alok Choudhary

**Main Point:** The authors parallelize the DBSCAN clustering algorithm by using a disjoint-set data structure. The disjoint-set data structure allows their algorithm to break the sequential data access pattern inherent to the original DBSCAN algorithm.

The disjoint-set data structure is used to keep track of clusters found thus far in a density-based algorithm. Disjoint set implements two operations – union, which merges two sets together, and find, which returns a value that represents a unique cluster in the data set. To implement a disjoint set, the authors begin by representing each point $x$ as a tree, where the parent $p(x)$ of a single-node tree is just the node itself, i.e. $p(x) = x$ for all points initially. A root of a tree is defined as a point $y$ for which $p(y) = y$, thus all points are roots initially. To build up clusters, two trees are merged by the union operation when it is found that a point in tree $t_i$ is density-reachable from a node in tree $t_j$. Suppose $r_i$ and $r_j$ are the roots of trees $t_i$ and $t_j$, respectively. If it is found that tree $t_i$ should be merged with tree $t_j$, it suffices to change $p(r_i)$ to $r_j$ or to change $p(r_j)$ to $r_i$. To enforce consistency, the authors adopt the rule that a root with lower index will always change its parent pointer to be the root with the higher index. Thus, if $i < j$, then $p(r_i)$ becomes $r_j$ in the union operation.

Parallel algorithms are presented for DBSCAN for both the shared memory and distributed memory models. In the shared memory setting, the original dataset $X$ is partitioned into $p$ parts $X_1, X_2, \ldots, X_p$ corresponding to $p$ threads. Next, the $p$ threads compute a local clustering independently. In other words, each thread $i$ executes the DBSCAN algorithm on its set $X_i$ of data. Since this is a shared memory model, all threads will have read access to the entire set $X$ to find the neighbors of a point $x_i$ in the DBSCAN algorithm. Each thread continues to explore the points in its partition of the data until it has explored the neighborhood of each point in the dataset. The output is a set of clusters, along with a set of pairs of points that require the clustering results of a different thread in order to perform a union operation. This set of pairs is called $Y$ and is processed in a second stage, again in parallel. $Y$ is again divided into $p$ partitions $Y_1, \ldots, Y_p$ amongst $p$ threads. Using a locking mechanism, each thread attempts to perform a union of the clusters containing one of the two points in a pair $(x, x') \in Y$. If the union was successful, the pair $(x, x')$ is removed from $Y_t$. Otherwise, the union is attempted again. This process continues until $Y$ is empty, resulting in the final output of clusters.

In the distributed setting, a similar approach is used, except that now each processor $P_i$ can only read data that is stored locally. Thus, after partitioning the data in into $p$ parts among $p$ processors, an additional call is needed in order to read the neighborhood of each point $x$. Each processor must make a call to the `GetRemoteNeighborhood` function for each point $x$ that is being
processed. The \texttt{GetRemoteNeighborhood}(x) function requires communication among the \( p \) processors in order to pass all points in \( \mathcal{X} \) that are within an \( \epsilon \)-neighborhood of \( x \) to the processor containing the partition \( \mathcal{X}_j \) that \( x \) belongs to. Then, it can proceed in the same way as the shared memory implementation, storing pairs that require the knowledge of cluster membership of a point belonging to a different processor. Some extra care is required when performing the parallel unions, because each processor only sees the “local” root of the tree representing the cluster that a point belongs to, so message passing is used to recursively find the parent of a node on different processors until it is found that the parent is a root.

In the sequential, shared memory, and distributed memory experimental evaluation of the disjoint-set based DBSCAN algorithm, a \( kd \)-tree is constructed for use by DBSCAN in order to ensure that the \( \epsilon \)-neighborhood search of a data point takes \( O(\log n) \) time. Experimental results show that both the shared memory and distributed memory implementations achieve significant speedup over both the sequential disjoint-set based implementation and the original sequential DBSCAN implementation. Both synthetic and real datasets where used, with varying numbers of data points (8k to 115,000k) and varying dimensionalities (from 2 to 20). For the shared memory implementation, speedups of up to a factor of 25.97 where observed on 40 cores, and in the distributed memory setting, 8,192 processors achieve a speedup of up to 5,765. When compared to the master-slave parallel implementation of DBSCAN, the disjoint-set implementation is also shown to be significantly more efficient due to the parallel union implementations.