Data Reduction and Partitioning in an Extreme Scale GPU-Based Clustering Algorithm

Benjamin Welton
University of Wisconsin - Madison
Madison, WI
welton@cs.wisc.edu

Barton P. Miller
University of Wisconsin - Madison
Madison, WI
bart@cs.wisc.edu

ABSTRACT

The emergence of leadership-class systems with GPU-equipped nodes has the potential to vastly increase the performance of existing distributed applications. An increasing number of applications that are converted to run on these systems are reliant on algorithms that perform computations on spatial data. Algorithms that operate on spatial data, such as density-based clustering algorithms, present unique challenges in data partitioning and result aggregation when porting to extreme scale environments. These algorithms require that spatially-near data points are partitioned to the same node and that the output from individual nodes needs to be aggregated to ensure that relationships between partition boundaries are discovered. In the development of an extreme scale density-based clustering algorithm, called Mr. Scan, we leveraged the increased computational power provided by GPUs to overcome these challenges. Three main techniques allowed Mr. Scan to cluster 6.5 billion points from the Twitter dataset using 8,192 GPU nodes on Cray Titan in 7.5 minutes: (1) a data partitioner scheme that ensures correctness by using shadow regions to selectively duplicate computation between nodes to detect clusters that lie in-between partition boundaries, (2) a dense box algorithm that reduces the computational complexity of clustering dense spatial regions, and (3) a new tree based method for merging results with spatial dependencies that requires only a fraction of the intermediate results to produce an accurate final result. The pure computational power of the GPUs allowed us to effectively summarize the initial data, making the scalable use of these techniques possible.

ACM Reference Format:
Benjamin Welton and Barton P. Miller. 2017. Data Reduction and Partitioning in an Extreme Scale GPU-Based Clustering Algorithm. In *Proceedings of , , ,* 6 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern leadership-class supercomputers tie together a diverse collection of resources, such as CPUs, GPUs, and high speed interconnects. To get maximum performance on leadership class machines, an application must efficiently use (and balance the use of) these resources. In the development of an extreme scale clustering algorithm, an implementation of the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [7] algorithm called Mr. Scan [18] [17], we found that for effective utilization of resources on leadership-class machines was difficult to achieve for algorithms dealing with spatial data. Data partitioning, GPU load imbalance, and data aggregation were major limiting factors that prevented the scale-up of Mr. Scan to multi-billion point datasets.

At their core, these factors were a result of over utilization of a system resource during some phase of execution. The challenge we faced was how to balance utilization of system resources within each phase of Mr. Scan. Data aggregation and reduction limits resulted in over utilization of network resources. We developed techniques that trade compute time for smaller data sizes to eliminate network over-utilization. GPU load imbalance was resolved by slightly increasing the computational complexity of partitioning to significantly reduce GPU use in the clustering phase of Mr. Scan. In this paper, we describe the issues that we faced and the techniques that we used to ultimately overcome these challenges to scale Mr. Scan to cluster 6.5 billion points using 8,192 GPU nodes. The specific techniques we describe, though tuned for clustering algorithms, may be generalizable to other algorithms performing decompositions and aggregations on spatial data.

Clustering is the act of classifying data points, where data points that are considered similar are contained in the same cluster and dissimilar points are in different clusters. Clustering helps researchers and data analysts gain insight into their data, e.g., identifying and tracking objects such as gamma-ray bursts in sky observation data [6], monitoring the growth and decline of forests in the United States [14] and identifying performance bottlenecks in large-scale parallel applications [8]. We focus on a type of clustering algorithm called density-based clustering, which classifies points into clusters based on the density of the region surrounding the point. Density-based clustering detects the number of clusters in a dataset without prior knowledge and is able to find clusters with non-convex shapes.

The ability to cluster billions of data points with DBSCAN can only be realized if the key obstacles to scaling DBSCAN are overcome: distributing data advantageously, load balancing, and cluster merging. The running time of DBSCAN increases as a function of spatial density of the input data points, which causes a load imbalance when compute nodes contain regions of varying density. We modify DBSCAN to find the most dense regions and infer their membership in a cluster without evaluating the points inside these dense regions. Results from DBSCAN compute nodes must be merged accurately without requiring the entirety of each cluster. Mr. Scan leverages a programming paradigm that organizes processes into a multi-level tree with an arbitrary topology to resolve

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

, ,
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

this issue. In this multi-level tree paradigm, DBSCAN calculations are done on the GPU leaf nodes and these results are combined on non-leaf nodes. A new data reduction algorithm requiring where only a small, bounded number of representative points per cluster is needed to accurately merge clusters using a multi-level tree topology. Finally, data must be distributed in a manner that balances DBSCAN's clustering operation and the overhead of merging clusters. We achieve this with a heuristic that spatially decomposes the data into partitions to balance the merge overhead. Each partition contains roughly equal point counts to aid in balancing DBSCAN clustering time.

In Section 2 we describe the DBSCAN algorithm and discuss other methods that attempt to parallelize DBSCAN. Section 3 describes a high level overview of the Mr. Scan algorithm. Section 4 describes the techniques we used to overcome the issues of data partitioning, GPU load imbalance, and data aggregation. Section 5 presents and discusses the scaling results on a dataset from Twitter and Sloan Digital Sky Survey [1]. Finally, in Section 6 we discuss future work on GPU-based aggregations.

2 BACKGROUND AND RELATED WORK

Due to DBSCAN's popularity among density-based clustering algorithms, optimization and parallelization of the algorithm has been widely studied [3]. We first explain the DBSCAN algorithm in detail, then present previous parallelization efforts that are most significant to the parallelization style of Mr. Scan along with the most scalable algorithms.

2.1 The DBSCAN Clustering Algorithm

DBSCAN clusters data points by density. Its notion of density comes from its two parameters known as *Eps* and *MinPts*. DBSCAN operates by finding the *Eps-neighborhood* of each point. The *Eps*-neighborhood of a point p is the set of points that are located within *Eps* distance of p. The point p is considered a core point if there are at least *MinPts* points in its *Eps*-neighborhood. All other points are classified as *non-core* points. Non-core points can have two distinctions: a *border point* or a *noise point*. A border point is a non-core point that contains at least one core point in its *Eps*-neighborhood, whereas a noise point does not.

A cluster is formed by the set of core and border points reachable from a particular core point. Once an unvisited core point is found, it is considered a new cluster along with its *Eps*-neighborhood. This cluster is expanded by finding the *Eps*-neighborhood of each point classified in the cluster until all points that are reachable from the first core point are found. For this reason, DBSCAN's clustering results can vary slightly if the order in which *Eps*-neighborhoods are discovered is changed. Figure 1 shows an example of the DBSCAN clustering process.

The performance of the DBSCAN algorithm varies greatly based on the presence (or lack thereof) of a spatial index. DBSCAN without a spatial index is $O(n^2)$ in time complexity. This is due to not limiting the amount of points compared by the distance function. Without a spatial index all points in the dataset must be compared with each other to determine which points are core. A spatial index however reduces the number of points which must be compared by limiting the search to a smaller subset of points that are in the region of the point being queried. The average case complexity improves to $O(n \log n)$ by use of a spatial index (e.g., R*-tree or KD-tree).

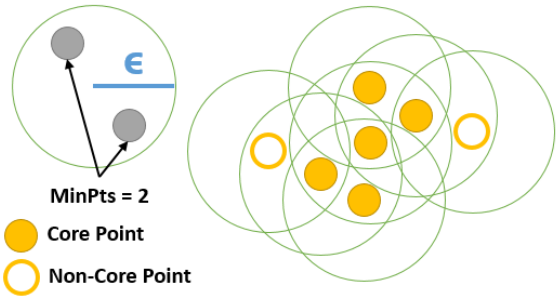

Figure 1: DBSCAN clustering example showing classification of *core* and *non-core* points in a dataset.

2.2 Past Optimizations of DBSCAN

DBSCAN has been parallelized by multiple past projects. One of the first was PDBSCAN [19]. This algorithm used a distributed R*-tree to partition the dataset among many compute nodes. Distributed R*-trees partition data but they replicate the entire index on each node. If a neighborhood query included an area of the dataset that resides on different node, the node that started the query must send a message to obtain the data. This algorithm showed linear speedup up to 8 nodes, but the amount of messages sent grew super-linearly in most cases, which hampered its scalability. Another algorithm, DBDC [11], assumes that the dataset to cluster is already distributed among the compute nodes. DBDC pioneered the idea of using many slave nodes to cluster a portion of the dataset and merging the final result at a master node, and also the idea of sending a smaller number of points to represent the locally found clusters to increase scalability. This technique scaled linearly up to 30 nodes, but the manner in which representative points were picked decreased the quality of the clustering output when compared to traditional DBSCAN, and the assumption of already distributed data further degraded quality.

There have been two Map/Reduce implementations of DBSCAN, MR-DBSCAN [10] and DBSCAN-MR [5]. MR-DBSCAN was able to cluster 1.9 billion points of 2D taxi-cab traces in approximately 5,800 seconds. However, the authors preprocessed the data prior to running DBSCAN to reduce the negative effects of high-density regions and did not account for this preprocessing time in their results (they did not measure end-to-end time). Also, the parameters for MR-DBSCAN's runs were chosen solely for speed and not for quality of the data analysis [9]. Aside from these issues, neither of the Map/Reduce implementations showed near-linear speedup nor the ability to scale weakly and only demonstrated their algorithms on up to 12 multi-core nodes.

Recently, a distributed heuristic based approach for approximating DBSCAN has been developed called Paridcle [15]. Paridcle uses a density based sampling approach to dramatically improve performance by limiting the number of points that need to be processed

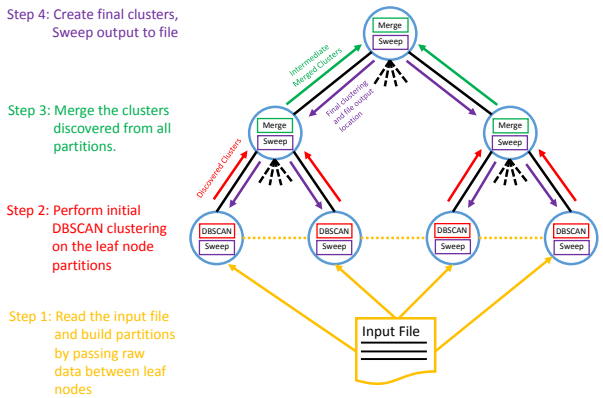

Figure 2: Overview of the Mr. Scan algorithm

by DBSCAN. Unlike previous DBSCAN implementations that attempted a density based sampling approach, Paridcle is able to maintain a high quality for output by carefully constructing the sample for which to perform DBSCAN on. The authors show that Paridcle is capable of near linear speed up, showing a performance of 3917x while using 4096 cores. While Paridcle shows good performance, the usage of an approximate DBSCAN algorithm differs from the parallel DBSCAN approach of Mr. Scan where no quality reducing approximations are used.

Several algorithms attempted to improve the single-core performance of DBSCAN. TI-DBSCAN [12] uses the triangle inequality. The input dataset is sorted to determine a point's *Eps*-Neighborhood, which is similar to the way our GPU implementation of the algorithm uses its KD-tree. Another version of DBSCAN [13] attempts to remove core points early from the DBSCAN calculation. This idea is similar to Mr. Scan's dense box optimization, but their method appears that it would change the result of DBSCAN significantly, even though the authors do not comment on this effect in the paper. In comparison, Mr. Scan's dense region calculation has an extremely small impact on quality when compared to traditional DBSCAN.

3 OVERVIEW OF THE MR. SCAN ALGORITHM

Mr. Scan is a hybrid/hybrid implementation of the DBSCAN algorithm with four phases: partition, cluster, merge, and sweep. Mr. Scan starts with a single input file on a parallel file system and writes as output a file of the points included in a cluster and their cluster IDs as output. The input points are contained in a single binary or text file. Each input point has a unique ID number, coordinates, and an optional weight that can be used for analysis of the clustered output. Figure 2 gives an overview of the Mr. Scan algorithm. All four phases of the Mr. Scan algorithm take place using the same tree layout of processes and the layout of the tree (number of leaf nodes and fanout) is user definable.

In the partition phase, the input file is read by the leaf nodes of the partitioner. The partitioner is responsible for creating one partition per clustering process (one partition per leaf node) from a given input file. The input file can contain billions of points and reach sizes up to 300 GB, so the partitioner is distributed using MRNet [16] to parallelize this step. Each worker process of the partitioner moves the completed partitions (via message passing) to the node responsible for processing that partition. The cluster phase begins after all leaf nodes have received the completed partitions they are responsible for processing. Each Mr. Scan leaf process clusters its assigned partition using our GPU version of DBSCAN and picks a small, constant set of points to represent each cluster. The representative points are sent to the intermediate processes to start the merge phase where the clusters are progressively merged by each level of intermediate processes until they reach the root. The root performs the final merge and assigns a global ID to each cluster. Mr. Scan then starts the sweep phase, and sends the global cluster IDs down the tree, where each point is identified with its correct global cluster ID and written to the output file in parallel by the leaf processes.

4 ROADBLOCKS TO EXTREME SCALE

The core challenge we faced when scaling Mr. Scan was how to reduce the amount and type of resources used in each of its phases. In particular, the imbalance of resource usage in the cluster and merge phases of Mr. Scan (shown in Figure 2) originally made scaling beyond a small number of nodes impractical.

In the clustering phase, there was a large difference in compute time between nodes that were processing that same number of input points due to large variations in point density. At a small scale of 16 nodes, the difference in time between the fastest and slowest nodes in computing the initial clusters was greater than 25 minutes on a run with a total time of 40 minutes. As we scaled the up, this gap widened, resulting in Mr. Scan being unable to run effectively on a large set of nodes. As we discovered, DBSCAN was strongly influenced by point density. DBSCAN has a complexity of $O(n^2)$, where n is the number of points when clustering dense regions of data, but an $O(nlog(n))$ complexity when clustering sparse data regions. Resolving the imbalance issue requires that we address the problem of density.

The merge phase required sending a significant fraction of the dataset up the tree to correctly identify and merge all clusters across partitions. In our first attempt at creating Mr. Scan, we required that all points be available for aggregation to ensure accurate clustering. This requirement ensured that we could properly identify clusters that spanned multiple partitions. A comparison was made between the points identified as members of a cluster by each node to determine if any of the points between identified clusters overlapped. If an overlap existed between points in two or more clusters identified on different nodes, the points identified as members of those clusters are actually members of the same cluster and should be merged. Using this technique, greater than 50% of the input dataset in our test data needed to be aggregated. Since our dataset sizes reached into the multi-hundred gigabyte range, aggregation of half of the total dataset in the merging process would not be scalable. Resolving the issue of aggregation required that we determine a way to reduce the amount of data necessary to identify clusters spanning multiple partitions.

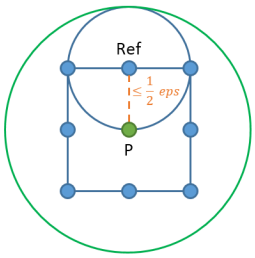

Figure 3: Any overlapping core point *P* must be within $\frac{1}{2} \times Eps$ of at least one corner or side of the grid cell. We label this point *Ref*. This means that the representative point for *Ref* must fall within a $\frac{1}{2} \times Eps$-neighborhood of *Ref*. Since this entire region (shown as the blue circle) is contained in the *Eps*-neighborhood of *P* this means that *P* is always within *Eps* of a representative point.

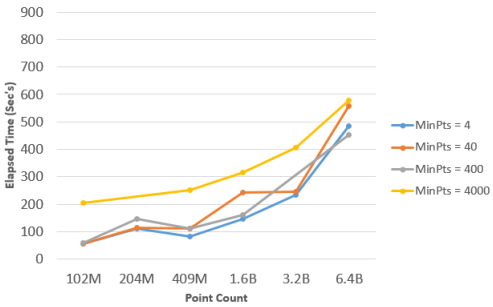

Figure 4: Elapsed time of Mr. Scan to cluster the Twitter dataset with *Eps*=0.1 and varied *MinPts*.

We developed three techniques to overcome the challenges in the clustering and merge phases: 1) Added *shadow regions* to our partitioning scheme to selectively overlap computation between nodes to reduce the amount of merging required, 2) created the *dense box algorithm* to reduce the complexity of performing DBSCAN on dense data regions, and 3) developed a merging algorithm that uses *representative points* of a cluster to reduce the number of points that need to be sent to merge clusters. We describe these techniques in Sections 4.1, 4.2, and 4.3 respectively.

4.1 Partitioning with Shadow Regions

In the partition phase, the input file is read by a partitioner that creates one partition per clustering process (one partition per leaf node). The input file can contain billions of points and can reach sizes up to 300 GB, so the partitioner is distributed using a hybrid MRNet/message passing model to parallelize this step. Each leaf node process starts by reading a unique section of the input file. Input points read from the file are then used to construct a $Eps \times Eps$ grid on each leaf node. The point counts of each grid cell are then sent by the leaf nodes up to the root of the tree. At the root, the grid cells are then used to generate the partitions for DBSCAN. Each partition contains a set of *Eps*-grid cells.

In addition to the basic goal of dividing an input dataset into n partitions given n leaf nodes, we have a secondary goal of creating partitions that reduce the amount of data required for merging the outputs from each leaf process to produce a correct result. To address this issue, we add a *shadow region* to each partition. The shadow region is the set of points not already included in the partition that lie *Eps* distance from the partition's boundary. A *shadow point* is a point that lies in a shadow region with respect to a partition, and a *partition point* is a point already included in the partition. When the shadow region is added to a partition, each partition point's *Eps*-neighborhood contains only partition points or shadow points, and thus is complete within the partition.

Each leaf node knows the composition of the clusters that it has discovered in terms of the number of shadow points they contain. A cluster that does not contain shadow points is defined as being complete since there is no possibility that the cluster can be merged. This guarantee can be given because clusters found in different partitions can only overlap at most *Eps* distance from one another. Since the shadow region is *Eps* distance in size, if a cluster contains no points in the shadow region it will not overlap with any cluster discovered on any other node. The data points contained in clusters with no shadow points do not need to be sent up the tree in the merge step.

4.2 Dense Box Algorithm

The dense box algorithm allows for points in dense regions of data to be marked as members of a cluster without incurring the cost of expanding each point individually. The dense box algorithm uses a modified form of the KD-tree [4] data structure already in use by CUDA versions of DBSCAN to eliminate the expansion phase of points in dense regions. Since the dense box algorithm reuses existing data structures, the complexity of DBSCAN in dense regions is reduced from $O(n^2)$ to roughly $O(n)$ with little added cost.

In its unmodified form, the KD-tree is used to identify points that may be within the *Eps-neighborhood* of a point p. The leaves of a standard KD-tree are composed of individual points while the upper levels partition the k-dimensional space. The major difference between the KD-Tree used by Mr. Scan and a standard KD-Tree is that leaf nodes represent *regions of data* instead of single points. Leaf nodes in the KD-Tree point to data within a (X,Y) value range while internal nodes point to leaf nodes containing data in distinct non-overlapping X value ranges. The nodes are stored in the tree in sorted order relative to their parent; the left-most child node contains the range with the lowest minimum value and the right-most child node contains the range with the largest maximum value. Each leaf node of the KD-Tree contains an offset that acts as a pointer into a array where the point data is stored. In addition, each leaf node of the KD-Tree also contains the number of points inside the leaf node's range.

After the construction of the modified KD-tree, the *dimension size* of all leaf nodes is examined. All points in a leaf node with *dimension size* less than or equal to $\frac{Eps}{2\sqrt{2}}$ *by* $\frac{Eps}{2\sqrt{2}}$ and $pointcount \geq MinPts$ will be marked as members of a cluster. We know that these points are members of the same cluster since they are all within *Eps* distance of one another. The points that are marked as being members of a cluster are not expanded when they are encountered by DBSCAN.

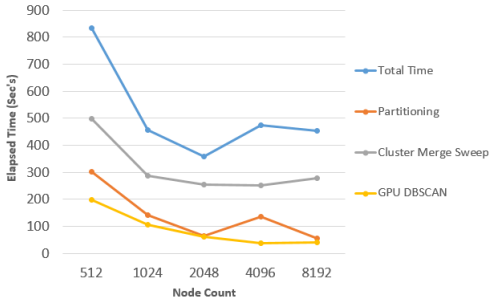

Figure 5: Strong scaling of 6.5 billion point twitter dataset

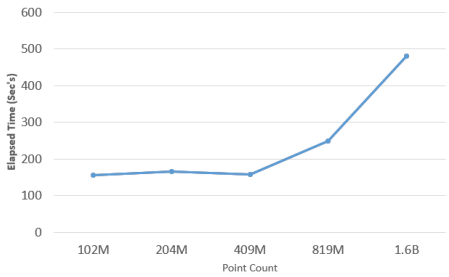

Figure 7: Elapsed time of Mr. Scan for the sky survey dataset with the parameters of 0.00015 *Eps* and 5 *MinPts*

4.3 Merging with Representative Points

Clusters identified in different partitions are merged if they have overlapping core points, points with greater than *MinPts* neighbors that overlap between partitions. A simple merge algorithm would compare the core points identified in each partition with one another to identify clusters that would need to be merged. The issue encountered with this approach is the number of points that needed to be aggregated to perform this check. When dealing with multi-billion point datasets, this simple approach is unusable. Adding to the challenge of merging, we do not want the reduction inside of the aggregation to impact the quality of the final clustering.

To address this issue, we select *representative points* that reduce the aggregation size while maintaining the quality of the final clustering. The set of representative points is the minimum set of core points from a single cluster that can correctly detect a merge inside a single grid cell. Clusters that have overlapping core points need to have at least one core point of overlap within the collective *Eps*-neighborhood that is formed by the set of representative points of the grid cell. We have determined that eight points can represent the core points of a grid cell of arbitrary density. The eight selected representative points are the points closest to the center of the sides of the grid cell and the corners of the grid cell. Figure 3 shows that when two clusters have an overlapping core point in a grid cell that at least one will be within the *Eps*-neighborhood of a representative point.

5 EVALUATION

We have two goals in evaluating Mr. Scan. The first goal is to test our ability to run DBSCAN on datasets that are several billion points in size in a reasonable amount of time. These datasets must represent real-world problems, and our experiments must use DBSCAN parameters that are useful to the problem. Datasets of this size have not been successfully clustered with any density-based clustering algorithm. The second goal is to evaluate whether Mr. Scan exhibits good scaling properties. This is a difficult proposition because memory limits make comparison to a single node implementation impossible.

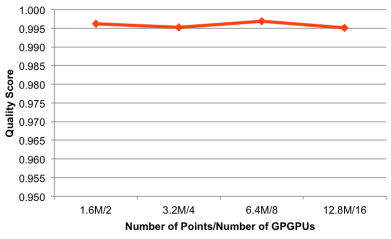

Figure 6: Quality measurement of Mr. Scan's output on the Twitter dataset using the DBDC [11] cluster quality metric (1.0 signifies a perfect clustering)

Mr. Scan was evaluated with datasets from both Twitter and the Sloan Digital Sky Survey. Mr. Scan is able to successfully cluster 6.5 billion points from the Twitter dataset with 8192 GPU equipped nodes. Clustering the 6.5 billion point twitter dataset can be accomplished between 453 and 576 seconds depending on the *MinPts* parameter used. Figure 4 shows the weak scaling results for the Twitter dataset. The strong scaling results with a breakdown of the total time for each phase of Mr. Scan are shown in Figure 5. Figure 6 shows the output quality of Mr. Scan in comparison to a single core reference DBSCAN implementation (ELKI 0.4.1 [2]).

The sky survey experiment consisted of a weak scaling experiment with a maximum point count of 1.6 billion points processed on 2048 nodes. This experiment was run with a fixed *Eps* value of 0.00015 and a fixed *MinPts* of 5. Figure 7 shows the weak scaling results for the sky survey experiment.

6 CONCLUSION

The successful scaling of Mr. Scan was only possible due to the spatial data reduction methods we developed. The computational power of the GPU allowed us to trade increased individual node workloads for a reduction in aggregation size. This trade begins in the partitioning phase where we use shadow regions to duplicate computation near partition boundaries. During the computation phase, the dense box algorithm is used to help the GPUs cope with the increased workload by reducing the computational complexity of regions with high point density. Finally, because of the extra computation that was performed, we could use a fixed number of representative points for aggregation without a loss in output quality. While the techniques we have presented are directly applied to the DBSCAN algorithm, the general idea of trading computation for a reduction in data size is not limited to clustering algorithms. Future expansions of our work would be to focus on studying the use of aggregation operations within the GPU itself and finding the increased compute for data reduction tradeoff in other algorithms.

REFERENCES

[1] 2013. Sloan Digital Sky Survey. (April 2013). www.sdss.org.

[2] Elke Achtert, Ahmed Hettab, Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. 2011. Spatial Outlier Detection: Data, Algorithms, Visualizations. In *Advances in Spatial and Temporal Databases*. Lecture Notes in Computer Science, Vol. 6849. Springer Berlin Heidelberg, 512–516. https://doi.org/10.1007/978-3-642-22922-0_41

[3] T. Ali, S. Asghar, and N.A. Sajid. 2010. Critical Analysis of DBSCAN Variations. In *International Conference on Information and Emerging Technologies 2010 (ICIET 2010)*. Karachi, Pakistan. https://doi.org/10.1109/ICIET.2010.5625720

[4] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975). https://doi.org/10.1145/361002.361007

[5] Bi-Ru Dai and I.-Chang Lin. 2012. Efficient Map/Reduce-Based DBSCAN Algorithm with Optimized Data Partition. In *IEEE 5th International Conference of Cloud Computing (IEEE CLOUD 2012)*. Honolulu, HI, USA.

[6] S. Davidoff and P. Wozniak. 2003. RAPTOR-scan: Identifying and Tracking Objects Through Thousands of Sky Images. In *Gamma-Ray Bursts: 30 Years of Discovery: Gamma-Ray Symposium*. Santa Fe, NM, USA.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *The Second International Conference on Knowledge Discovery and Data Mining (KDD '96)*. Portland, OR, USA.

[8] Todd Gamblin, Bronis R. de Supinski, Martin Schulz, Robert J. Fowler, and Daniel A. Reed. 2010. Clustering Performance Data Efficiently at Massive Scales. In *ACM/SIGARCH International Conference on Supercomputing (ICS 2010)*. Epochal Tsukuba, Tsukuba, Japan.

[9] Y. He. 2013. Personal communication. (19 March 2013).

[10] Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. 2011. MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce. In *The 17th IEEE International Conference on Parallel and Distributed Systems (ICPADS '11)*. Tainan, Taiwan.

[11] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. 2004. DBDC: Density Based Distributed Clustering. In *Int. Conf. on Extending Database Technology (EDBT '04)*. Heraklion, Crete, Greece, 88–105.

[12] Marzena Kryszkiewicz and Piotr Lasek. 2010. TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality. In *The Seventh International Conference of Rough Sets and Current Trends in Computing (RSCTC 2010)*. Warsaw, Poland.

[13] Marzena Kryszkiewicz and Lukasz Skonieczny. 2005. Faster Clustering with DBSCAN. In *International Conference on Intelligent Information Systems 2005: New Trends in Intelligent Information Processing and Web Mining (IIPWM 2005)*. Gdansk, Poland, 605–614.

[14] R Mills, Forrest M. Hoffman, Jitendra Kumar, and William W. Hargrove. 2011. Cluster Analysis-Based Approaches for Geospatiotemporal Data Mining of Massive Data Sets for Identification of Forest Threats. *Procedia CS* 4 (2011), 1612–1621.

[15] Md. Mostofa Ali Patwary, Nadathur Satish, Narayanan Sundaram, Fredrik Manne, Salman Habib, and Pradeep Dubey. 2014. Pardicle: Parallel Approximate Density-based Clustering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*. IEEE Press, Piscataway, NJ, USA, Article 2683655, 12 pages. https://doi.org/10.1109/SC.2014.51

[16] P. Roth, D. Arnold, and B. Miller. 2003. MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools. In *ACM/IEEE Supercomputing Conference 2003 (SC 2003)*. Phoenix, Arizona.

[17] Benjamin Welton and Barton P. Miller. 2014. The Anatomy of Mr. Scan: A Dissection of Performance of an Extreme Scale GPU-based Clustering Algorithm. In *Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA '14)*. IEEE Press, Piscataway, NJ, USA, Article 2691150, 7 pages. https://doi.org/10.1109/ScalA.2014.10

[18] Benjamin Welton, Evan Samanas, and Barton P. Miller. 2013. Mr. Scan: Extreme Scale Density-based Clustering Using a Tree-based Network of GPGPU Nodes. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*. ACM, New York, NY, USA, Article 84, 11 pages. https://doi.org/10.1145/2503210.2503262

[19] Xiaowei Xu, Jochen Jäger, and Hans-Peter Kriegel. 1999. A Fast Parallel Clustering Algorithm for Large Spatial Databases. *Data Mining and Knowledge Discovery* 3, 3 (1999), 263–290.