BD-CATS: Big Data Clustering at Trillion Particle Scale

Md. Mostofa Ali Patwary[1,†], Suren Byna[2], Nadathur Satish[1], Narayanan Sundaram[1], Zarija Lukić[2], Vadim Roytershteyn[3], Michael Anderson[1], Yushu Yao[2], Prabhat[2], Pradeep Dubey[1]

[1]Intel Corporation, [2]Lawrence Berkeley National Laboratory, [3]Space Science Institute
[†]Corresponding author: mostofa.ali.patwary@intel.com

ABSTRACT

Modern cosmology and plasma physics codes are now capable of simulating trillions of particles on petascale systems. Each timestep output from such simulations is on the order of 10s of TBs. Summarizing and analyzing raw particle data is challenging, and scientists often focus on density structures, whether in the real 3D space, or a high-dimensional phase space. In this work, we develop a highly scalable version of the clustering algorithm DBSCAN, and apply it to the largest datasets produced by state-of-the-art codes. Our system, called BD-CATS, is the first one capable of performing end-to-end analysis at trillion particle scale (including: loading the data, geometric partitioning, computing kd-trees, performing clustering analysis, and storing the results). We show analysis of 1.4 trillion particles from a plasma physics simulation, and a $10{,}240^3$ particle cosmological simulation, utilizing $\sim$100,000 cores in 30 minutes. BD-CATS is helping infer mechanisms behind particle acceleration in plasma physics and holds promise for qualitatively superior clustering in cosmology. Both of these results were previously intractable at the trillion particle scale.

Categories and Subject Descriptors

B.4.3 [**Input/Output and Data Communication**]: Interconnections—*Parallel I/O*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*; I.5.3 [**Pattern Recognition**]: Clustering—*Algorithms*

Keywords

Density-based clustering, DBSCAN, Parallel I/O, KDTree

1. INTRODUCTION

Clustering is an important data mining kernel used in many scientific applications including satellite image segmentation [36], finding halos in cosmology [30], noise filtering and outlier detection [4], prediction of stock prices [26], and bioinformatics [34]. Clustering classifies a given set of points into meaningful subclasses (called *clusters*) that minimizes intra-differences and maximizes inter-differences among them. Even though there exist several clustering algorithms such as K-means [33], K-medoids [37], STING [48] or WaveCluster [43], density based clustering (DBSCAN [16] and OPTICS [2, 31]) has received significant attention from the research community in recent years. This is mainly due to the unique nature of DBSCAN in identifying irregularly shaped clusters and filtering noise particles. The key idea of DBSCAN is that, for each data point in a cluster, the neighborhood in a given radius (called ϵ) has to contain at least a minimum number of points (*minpts*).

DBSCAN was known to be an inherently sequential algorithm for decades. Major progress has been made in parallelizing the algorithm, transitioning from master-slave based implementations [8, 17] to highly parallel techniques [39], GPU or map-reduce based implementations [50, 24], and even approximate algorithms [40]. Most of these efforts are centered around the core kernel and not much attention has been paid to the other steps (rather considered as either pre-processing or post-processing steps although they may take more time than DBSCAN itself). Some of these steps include reading the dataset, geometrically partitioning the dataset for better load balancing and reduced communication, and kd-tree construction for efficient neighborhood searches. Several of these steps (such as partitioning and kd-tree construction) have the same computational complexity as DBSCAN, $O(n \log n)$, where n is the number of particles. In most cases, these steps were either considered sequential or partially parallel [24, 40], hence not an end-to-end scalable solution for large scale scientific applications. To the best of our knowledge, only [50], a GPU based DBSCAN implementation, considered end-to-end analytics. However, due to memory requirements, their system could only handle a few tens of millions of particles per node, making trillion particle scale analysis infeasible.

In this paper, we develop the first end-to-end DBSCAN clustering solution at trillion particle scale on 10's of TB datasets to enable scientific analysis at previously unexplored scales. Our solution uses HDF5-based parallel I/O for reading the dataset and writing the clustering solution. We use a novel sampling based median computation technique (previously used in a different context, sample sort [18, 14]) to improve the load imbalance in geometric partitioning (from $2.7\times$ using previous implementations using geometric means down to $1.0\times$ using our technique). We also use the same technique in the kd-tree construction in contrast to the usual mean-based technique [40], leading to balanced binary trees and reducing the load imbalance in the nearest neighbor search in DBSCAN. Our hybrid implementation uses both MPI and OpenMP to take advantage of modern multithreaded architectures, and reduces overall communication time compared to previous implementations [39, 40, 50].

We apply our end-to-end clustering solution on two scientific

SC '15, November 15 - 20, 2015, Austin, TX, USA

ACM ISBN 978-1-4503-3723-6/15/11...$15.00

DOI: http://dx.doi.org/10.1145/2807591.2807616

applications: cosmology and plasma-physics. Their simulations are now reaching trillions of particles, producing 10's of TBs of data per single time output. Summarizing and visualizing these particle data is extremely challenging; using our scalable solution, we demonstrate end-to-end analysis on 1 and 1.4 trillion particles from cosmology and plasma physics simulation respectively utilizing $\sim 100,000$ cores.

Our specific contributions are the following:

- We have developed the first end-to-end clustering (BD-CATS) solution, which is capable of analyzing trillions of particles and 10's of TBs of data. This is more than two orders of magnitude larger than those ever attempted before. This has directly enabled scientific analysis of such large simulation results for the first time.
- Compared to previous work [40], BD-CATS improves load balancing and is more scalable. On 32 nodes, this leads to an end-to-end speedup of 7.7× over that work. This includes data reading, geometric partitioning, kd-tree construction, in-memory analytics, and storing clustering results.
- We demonstrate strong and weak scalability of BD-CATS up to 98,304 cores.
- BD-CATS has enabled new insights into trillion particle datasets for the first time. For cosmology datasets, we show that clustering solution using DBSCAN may qualitatively be superior — avoiding the cluster bridging problem — compared to a baseline method of friends-of-friends. For plasma physics applications, we are able to examine phase space composition of regions with large plasma densities, which helps infer mechanisms behind the acceleration of particles. Both of these results were previously intractable at the trillion particle scale.

The remainder of this paper is organized as follows. In Section 2, we discuss the motivation of this research in terms of scientific applications. We describe all the technical details and experimental setup in Section 3 and Section 4, respectively. We analyze the results in Section 5 with related works in Section 6. We conclude and propose future work in Section 7.

2. APPLICATIONS

Big Data Analytics is key for gaining scientific insights from large scale simulation output. In this paper, we consider two scientific domains: Cosmology and Plasma Physics, where large scale simulations are routinely used to test our theoretical models of the natural world. State-of-the-art simulations in both domains produce truly massive particle datasets, and any comprehensive analysis mandates scalable analytics capabilities. This paper makes a key contribution in developing a scalable, end-to-end clustering solution for the largest datasets in both domains. We now discuss the scientific motivation for this clustering analysis.

2.1 Cosmology

Over the last two decades, Cosmology has been one of the most rapidly advancing areas of physical sciences. That advance was enabled by increasingly precise sky surveys, covering a wide range of wavelengths, from X-ray to the radio part of the spectrum. Different sky surveys all arrive at the same picture of the Universe, where ∼25% of its content is in form of pressureless dark matter which interact only via gravity, and ∼70% is in spatially smooth component, consistent with cosmological constants (e.g. recent results by Planck collaboration [42]) What is actually dark matter, and what is the smooth component which drives the accelerated expansion of the universe remains a mystery.

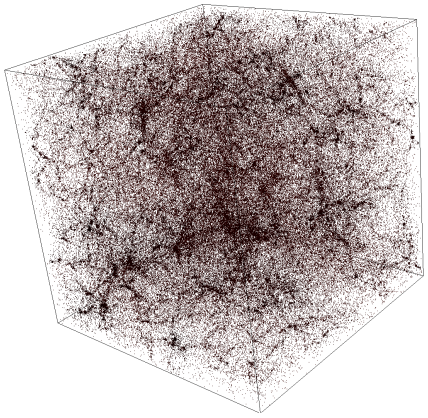


Figure 1: Visualization of one of cosmological simulations used in this paper. We scatter-plot particles, but for clarity we show only ∼250,000 particles out of 1024^3. In this image we see filaments — long and narrow features with density ∼ 10× the mean density, as well as halos — gravitationally bound regions of very high densities (in excess of 100× mean). In cosmological simulations, lot of focus is on characterizing halos, and the tool used to identify them in simulations is called "halo finder" [30].

To answer these important questions, sky surveys continue their rapid advance, but it is also imperative to have theoretical modeling at least as accurate as observational data (precision level of roughly 1%). By and large, future observations focus on the evolution of density fluctuations in the Universe, and the task for theory is to produce predictions for what those fluctuations are in different cosmological models. The picture is complicated by the fact that density fluctuations are difficult to probe directly, but we rather observe some biased tracer of the density field, like galaxies, clusters of galaxies, or Lyman-alpha flux decrement. The only way to reliably model the small scale non-linear fluctuations at $<\sim 100$ Mpc (megaparsec; 1 parsec = 3.26 light years, or 3.08×10^{16} meters) scales is via numerical simulations. Back of the envelope requirement for simulations of future sky surveys is a few Gpc box size, with resolution of ∼1 kpc. In order to resolve relevant structures by enough mass tracers, simulations will have trillion particles or more. Note that it is not a single "heroic" simulation that will suffice, but many such simulations for different cosmological models. The computational requirements are therefore tenacious, and cosmological simulations require both computational- and memory-efficient parallel codes.

Formation of structure in the universe is driven by gravitational instability, where small initial density fluctuations are rapidly enhanced in time [41]. The resulting density field has large void spaces, many filaments, and dense clumps of matter within filaments (Figure 1). The existence of those localized, highly overdense clumps of dark matter, termed halos, is an essential feature of non-linear gravitational collapse in current cosmological models. Dark matter halos occupy a central place in the paradigm of structure formation: gas condensation, resultant star formation, and eventual galaxy formation all occur within halos. A basic analysis task is to find these clusters of particles, and measure their properties like mass, velocity dispersion, density profile, and others. The most common method (although not the only one) for finding halos in cosmological simulations is friends-of-friends algorithm (FOF; first used in [15] and [13]), a percolation motivated

halo finder, which approximately tracks isodensity contours. FOF is an efficient method, but suffers from numerical insufficiencies, most notably the fact that total halo mass will change in a systematically biased way as we reduce number of sampling particles in a halo [49], and that different halos can be "bridged" in one by a tiny stream of particles [52, 32]. DBSCAN is similarly a percolation-based algorithm, and in fact, FOF is just a special case of a DBSCAN algorithm, where the minimum number of neighbors is set to one. Therefore DBSCAN can be used to give just the same results as FOF, but in significant addition to that, the extra parameter of DBSCAN can be utilized to reduce the number of bridged halos in the final halo catalog and improve overall accuracy.

2.2 Plasma Physics

Magnetic reconnection is a process of rapid topological rearrangement of magnetic fields embedded into plasma. Often accompanied by an explosive release of accumulated magnetic energy, this fundamental plasma physics process plays a significant role in the dynamics of many systems across a wide range of scales. Some important examples include laboratory fusion experiments, magnetospheres of the Earth and of other planets, the solar corona, and many astrophysical objects.

Magnetic reconnection arises as a nontrivial process because typical plasma viewed on large spatial and long temporal scales behaves as a superconducting fluid, such that the embedded magnetic field is constrained to move together with (or, as it is often said, be "frozen into") plasma. This often leads to accumulation of significant amount of magnetic energy in the system and the appearance of narrow spatial regions where magnetic field is highly stressed. In these narrow layers, termed current sheets, deviations from the idealized "frozen-in" approximation become essential and may lead to breakup of the current sheets and associated rapid relaxation in the system. Some of the most spectacular observable examples of events thought to be inherently associated with magnetic reconnection are solar flares and coronal mass ejections.

In high-temperature rarefied plasmas typical of many systems, the microscopic physical processes ultimately responsible for magnetic reconnection are of kinetic origin and can only be adequately described in a formalism that retains information about interaction of individual particles with electromagnetic field. Fully kinetic Particle-In-Cell (PIC) simulations have greatly advanced understanding of magnetic reconnection in recent decades. In fact, many of the specific goals of recently launched Magnetospheric Multiscale (MMS) NASA mission [1] dedicated to studies of magnetic reconnection are motivated by recent theoretical advances. At the same time, analysis of the simulation data becomes increasingly challenging as the size of the simulations grow. Fully kinetic PIC codes use Lagrangian markers to efficiently sample the six-dimensional coordinate-velocity phase space. With the advent of petascale computing, simulations with upwards of 10^{12} particles have become common. At the same time, the tools for analysis of such massive amount of data have lagged behind. Today, a typical analysis of the large-scale datasets produced by PIC simulations focuses on reduced information obtained for example by computing the moments of distribution function on a pre-defined spatial grid. At best, one might compute cumulative energy distributions for particles in a given region. Meanwhile, a more detailed view of the phase-space is of great interest. For example, magnetic reconnection sites are known to be efficient particles accelerators. Several competing microscopic acceleration mechanisms have been proposed in the literature and understanding of if and how they operate in a particular simulation requires clusters of high-energy particles to be identified and quantified. This is further complicated by the fact that magnetic reconnection in 3D current sheets is inherently turbulent [11], so that multiple reconnection and acceleration sites of complex geometry are present in the system.

In this work, we consider magnetic reconnection in collisionless pair plasmas comprised of electrons and positrons, relevant to astrophysical magnetically dominated systems such as Pulsar Wind Nebulae, relativistic jets in Active Galactic Nuclei, or Gamma Ray Bursts [27]. The simulation focuses on spontaneous development of magnetic reconnection in an isolated, initially stationary, 3D current sheet. The initial conditions and parameters of the simulation are detailed in Section 4. DBSCAN is used to identify high-density clusters, which are shown to be associated with high-energy particles and thus related to acceleration sites.

3. BD-CATS SYSTEM

BD-CATS is an end-to-end system for performing clustering analysis on large scale scientific data, consisting of a number of steps. BD-CATS internally uses DBSCAN, a density based clustering algorithm. Such clustering algorithms often require "pre-processing" steps (including geometrically partitioning the data and gathering ghost particles around partitions) as well as "post-processing" steps, such as computing cluster ids, file I/O etc that are not usually considered as part of the runtime of the clustering algorithm itself. However, such steps pose serious bottlenecks in the context of an end-to-end system that aims at analyzing large scientific data. Since our aim is to run BD-CATS to cluster a trillion particles on thousands of nodes, each step of the system has to be scalable for the overall system to scale. Moreover, at such large scales, it is also critical to make efficient use of memory, avoiding redundant copies of data wherever possible. In this section, we describe the algorithmic and implementation techniques with optimizations involved in making each step of our system scalable and efficient. The various steps involved in the end-to-end pipeline are shown in Algorithm 1. We describe these steps below:

Algorithm 1 Steps in the BD-CATS system. Input: A file I containing particles to cluster, a distance threshold ϵ, and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters C written to output file O.

1: **procedure** BD-CATS($I, \epsilon, minpts, O$)
2: $X \leftarrow$ READINPUTFILE(I)
3: $Y \leftarrow$ PARTITION(X)
4: $Y' \leftarrow$ GATHER(Y, ϵ)
5: $K \leftarrow$ CONSTRUCTKDTREE(Y')
6: $T \leftarrow$ DBSCAN($Y', K, \epsilon, minpts$) ▷ Includes local computation and merging, T is the union find tree
7: $C \leftarrow$ COMPUTECLUSTERID(T)
8: WRITEOUTPUTFILE(C, O)

3.1 Reading Data

The system begins by reading an input file containing the data to be clustered. We consider that the data files store properties of particles in arrays. For instance, the two trillion particle data files we analyzed in this study use HDF5 file format, where each property is stored as a HDF5 *Dataset*. Each property encodes particle attributes in one of the dimensions in which it resides. These properties are stored in a structure of arrays (SOA) format, where the first property for all particles are stored contiguously, followed by data for the second property and so on. This is in contrast to previous work [40], where data is read from an array of structure (AOS) format, and later converted to SOA format in memory for efficient

vectorization in distance computations. Here, we avoid this expensive in-memory conversion and directly store data in SOA format.

We use parallel I/O for reading the datasets into memory. Assuming that P MPI processes were used to read the data related to N particles, each HDF5 dataset is logically split into P equal partitions. Each process p reads a subset of particles starting from p * N/P to $(p+1)$ * N/P. The last MPI process in addition reads data related to the remainder of N/P.

To make use of parallelism at file system level, we have stored the data files on a Lustre parallel file system with striping across maximum available storage targets. We used HDF5 dataset read calls with MPI-IO in collective buffering mode to limit the number of MPI processes requesting the storage targets [46]. In this mode, MPI processes use two-phase I/O, where the first phase reads data from storage to aggregator nodes and the second phase distributes the data to MPI processes. The number of aggregators is equal to a multiple of the number of storage targets on Lustre file system. Based on our previous observations, using this mode of parallel I/O on peta-scale supercomputers achieves high I/O rates with large number of MPI processes [7]. Since the large systems are typically shared by hundreds of users, when running with large number of MPI processes, the I/O performance is better when using a significant fraction of the system.

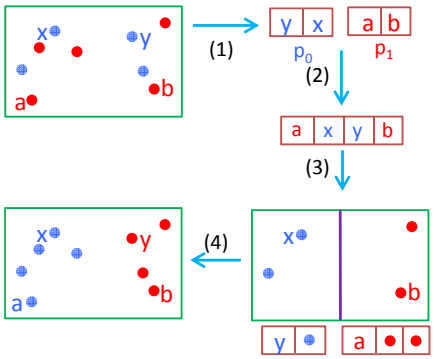


Figure 2: Partitioning using sampling based median computation. There are 2 nodes with the particles owned shown in different colors (blue and red). In Step (1), each node selects a random sample of its particles. These samples are merged and the median of the combined samples is found in Step (2). In Step (3), each node perform a local partitioning of the data according to the median. In Step 4, the two nodes exchange one of their partitions created above – node p_0 sends all particles greater than the median to node p_1; while node p_1 sends particles less than the median to node p_0.

3.2 Geometric Partitioning

After file reads, the nodes now exchange particles among them in such a way that each node owns a geometrically partitioned section of the data. The idea behind geometric partitioning is to improve locality of processing and avoid repeated inter-node communication in later steps. At the end of partitioning, each node owns a subset of data that could be completely different from what it read. Apart from locality, another important consideration during geometric partitioning is to ensure load balance by ensuring that each node receives roughly same amount of data after partitioning.

Geometric partitioning has been previously studied in the context of clustering algorithms [39, 40], with the main idea being to recursively divide particles among nodes based on (typically) axis-aligned splits of data. The choice of split dimension is fairly standard (the one with the largest spread of attributes), but the choice of split location along that dimension varies between implementations and has high impact on load balance. Ideally, one would compute an exact median which ensures perfect load balance; however computing this is expensive and is not commonly done in practice. Previous implementations of DBSCAN have used a simple average (mean) of attribute values along the split dimension as the split location [39, 40]. This is easy to compute with low overhead, but this results in severe load imbalance – in some real world datasets (*cosmo_small* in Table 1 on 32 nodes), one node gets 10 Million particles while another gets 91 Million particles, with an average of 33.5M particles per node (Section 5).

In contrast, we use a sampling based median computation technique, typically used in other application domains such as sorting [28], to compute the split location. The idea here is to compute a median based on a randomly selected subset of the data. As more such samples are selected, the median computed more precisely matches the real median. This scheme results in much better load balance – for the same example above, we see variations only between 33 M and 34 M particles per node.

Figure 2 shows the basic idea. We first randomly select a small set of data (samples) from each node. The location of each of these samples is a potential split location. Each node broadcasts its selection to all other nodes, and nodes sort this small set of collected samples to construct the median. Then each node partitions its data into two sets, one set each for particles on each side of the median. In the 2 node case shown in the figure, all particles on the left of the chosen partition (assuming partition dimension is along the x-axis) are communicated to node p_0 and all particles on the right to node p_1. This then completes the partitioning step for the 2-node case. In the more general case, where $P > 2$, this step is extended to have multiple pairs of nodes exchange particles. This works as follows: nodes p_0 and $p_{P/2}$ exchange particles as in the figure, as do nodes p_1 and $p_{P/2+1}$ and so on. These exchanges are independent. The results distribution of particles will be such that nodes between p_0 to $p_{\lfloor P/2 \rfloor}$ will get particles to the left of the median, and nodes $p_{\lceil P/2 \rceil}$ to p_{P-1} will get particles to the right of the median. Hence there is no overlap in the bounding boxes of these two sets of nodes. Each of these two sets of nodes then repeats ($O(\log(P))$ times) this process of computing a split and partitioning until each node gets one geometric partition of the data. At this point, the bounding boxes of the particles in different nodes will not overlap, and geometric partitioning is complete.

At large scales, the process of partitioning data can be computationally demanding and heavy in terms of memory usage. It is hence important to develop a hybrid implementation with multiple threads inside each node cooperatively working to create partitions. The main challenge for large scale data is to perform in-place exchange of data for efficient use of memory. Maintaining a contiguous set of data before and after data exchange can require compaction of "holes" that can be created when data is sent out and appending data received from other nodes. In order to do this efficiently, we divide the original data among threads, and have each thread first perform an in-place partition into left and right halves with the median as pivot (similar to quicksort). Then threads merely cooperatively concatenate the local partitions they want to send out into a contiguous buffer. The rest of the data, together with the received data are concatenated and form the partitioned data. The concatenation steps are done efficiently in large blocks using atomics. By using in-place per-thread partition and reusing the communication buffers across each of the $\log(P)$ partitioning steps, we achieve good scalability while ensuring we do not replicate the dataset by more than a factor of 2.

3.3 Gathering

Once the data is geometrically partitioned, the bounding boxes of the data stored at each node do not intersect. While this improves locality in nearest neighbor computations during clustering, it is however possible for neighbors to cross over the geometric partitions obtained. In order to avoid costly communication steps for nearest neighbor computations, we gather particles from a ghost region from neighboring nodes for use in the clustering step. DBSCAN specifies an ϵ radius for use in the neighbor computations, and it suffices to gather all particles that are in an extended bounding box with an extension of ϵ around the extent of the geometric partition. This step is fairly straightforward, with the idea being that all nodes exchange their extended bounding box coordinates with other nodes. Each node then iterates through the local dataset and checks it against each extended bounding box, which requires P point-box comparisons for each particle. To make this step efficient, we first check if there is any overlap of the current node's (say n) bounding box with the extended box of other nodes. If there is no overlap between node n and say node k, then we skip all intersection comparisons with points in n with the bounding box of k. This step can be multithreaded in a straightforward way.

For datasets in the cosmology domain, we have an additional step in that the boundaries are periodic and hence wrap around. For such cases, we have to perform additional intersection tests for each particle moved by +L and -L in each dimension, where L is the length of the global bounding box in that dimension. This can be computationally demanding for large datasets, and we see the effect of this in the cosmology trillion particle dataset in Section 5.

3.4 Clustering

Once particles in the ghost regions are gathered, the actual process of clustering is achieved by first running a clustering algorithm, here DBSCAN, on the local data gathered to each node, followed by a process of merging the local clusters so obtained into a set of global clusters.

A scalable version of DBSCAN has been described in previous work [40]. This work however has two assumptions: one, it uses Kdtrees for nearest neighbor computations but does not discuss how to construct these trees in a scalable manner that also achieves good load balance in neighbor computations. Second, the implementation in [40] uses a union-find algorithm to represent clusters; however, in most cases a cluster id has to be assigned to each particle of the union-find trees. [40] uses a sequential algorithm to do this last phase and in fact gathers all points to one node to perform this step. In this work, we develop scalable implementations of these steps – this is required for an end-to-end scalable solution.

3.4.1 Kd-tree construction

Kd-trees are constructed as an efficient data structure in order to effectively find nearest neighbors of points for local clustering within the node. Instead of computing distances from a source particle to all other particles in order to find the nearest neighbors, a kd-tree traversal allows for many regions of space to be pruned away and only a few distance computations to be performed.

Kd-tree construction follows a similar spatial partitioning of particles, and in fact can be viewed as the analog of the partitioning phase described previously, but done at a scale local to a node. Just like the partitioning phase, load balancing is again a concern – different threads will perform simultaneous nearest neighbor lookups that may go through different paths in the tree. Developing kd-trees that are relatively balanced in the number of particles at different leaf nodes will help reduce such issues. Previous state of the art [39, 40] uses the average (mean) of particle coordinates to find kd-tree splits, which can lead to high load imbalance (higher depth of the tree) in following steps. In this work, we use a sampling based median computation (as in the geometric partitioning step) to obtain good load balance. We noticed that in some real datasets ($cosmo\_small$ in Table 1), the maximum depth reduces by 20% using our technique.

Our second aim is to perform the kd-tree construction in a scalable fashion and we develop a multithreaded algorithm to achieve this. This is a two stage algorithm – (i) For creating the first few levels of the kd-tree, threads cooperate to perform median computation and swap particles - this is similar to the way the partitioning phase is done. This scheme works well as long as there are enough particles to process at each step. However, as we go down the tree, each kd-tree node has a small number of particles, and hence parallelization overheads can be high. Hence we stop cooperatively partitioning kd-tree nodes once we reach a fixed number of leaf nodes and switch parallelism modes. (ii) Once we have constructed at least as many leaf kd-tree nodes using step (i) as there are threads, we can switch to an independent scheme where each thread takes up responsibility for constructing the sub-tree (containing the remaining levels) that is rooted at one of the leaf nodes above. For load balancing reasons, we perform this step only when there are about 2-4 times as many sub-trees to construct as threads. Each thread stops partitioning when a subtree contains ≤ 32 particles.

In order to avoid high memory bandwidth overheads, we don't move the entire data when constructing each level of the kd-tree. Instead, we use an additional index array to keep track of the id's of the particles and we only reorder that array. In the end, we order the data according to the index array order. We use in-place memory operations wherever possible. Overall, our scheme scales well and generates balanced trees.

3.4.2 Local computation and merging

In this step, we call the DBSCAN algorithm (the core kernel in clustering) to compute clusters. DBSCAN has two user-defined parameters, ϵ and $minpts$. The key idea of DBSCAN is that, for each data point in a cluster, the neighborhood in a given radius, ϵ has to contain at least a minimum number of points (*minpts*).

Different algorithms for DBSCAN follow different parallelization schemes. The classical DBSCAN algorithm [16] adopts a breadth-first order of processing points but is inherently sequential and not suited to high performance implementations. The authors of [39] develop a massively parallel approach using a union-find data structure, which maintains a dynamic collection of clusters and supports fast UNION and FIND operations [38] to combine clusters or to identify the cluster to which a point belongs. This scheme was further modified in [40], where the authors combined the cache locality of the breadth-first algorithm while still allowing for parallel updates using union and find operations. We adopt this last approach in this work. The end result of the clustering phase is a set of union-find trees, where particles belonging to a single cluster are placed in the same tree. However, these trees can span across nodes, and even finding the root of the union-find trees can be time-consuming. The operations of local clustering, thread-level and node-level merging have been previously optimized and we follow the scalable implementation of [40] (exact version).

3.4.3 Computing cluster IDs

The result of clustering is a set of union-find trees where each tree contains all particles belonging to a single cluster. However, many science applications require per-particle cluster id to be computed and written to disk. Computing cluster ids from union-find trees is conceptually simple - one simply traverses the tree up-

wards starting from a particle until it reaches the root, at which point the cluster id of the root is assigned to that particle. However, the challenge comes from the fact that each tree (cluster) can span across multiple nodes, and hence traversing trees can involve several rounds of heavy communication. For this reason, previous implementations [39, 40] often gather points to a single node and perform this computation there. However, this is clearly not scalable and is not applicable to large datasets. In this work, we parallelize the process of computing cluster ids from the tree structures both across nodes and threads.

Only the points belonging to a cluster need the cluster ids. So, the key step is to traverse the union-find tree globally for all such points. The idea is that each point tries to find if the root is local to the node it belongs to. In that case, no communication is required and it gets the cluster id from the root immediately. Otherwise, a find root request is initiated. The request might traverse multiple nodes before reaching the root node. The root then sends the cluster id directly to the request initiator. The algorithm continues until there are no more requests to be forwarded.

The number of nodes that requests get forwarded to reflect the span of the final clusters. For some datasets that have very large clusters like the plasma physics dataset, we have found that some clusters can span as many as 68 nodes, which means requests can be forwarded 68 times. This makes this step highly communication limited. Moreover, in the last few rounds, only a few nodes may be active leading to reduced scalability.

3.5 Storing Results

Finally, the assigned cluster id at the end of the previous step needs to be written to disk. By the time the cluster ids are assigned, they are partitioned and geometrically distributed across nodes and are not in the original order of increasing global ids. One option could be to write the entire data with cluster IDs out of order, which leads to writing more data than the original file. Instead, we append the cluster IDs as a separate field to the original HDF5 file. However, this means that each particle will want to write to a different region of the file, leading to very inefficient I/O patterns. Hence we do the following.

3.5.1 Parallel sorting

In this step, we redistribute the particles (only cluster id and global particle id, not the data) among the nodes to match the initial order in which particles were read in. We use the global ids of the particles as a sorting key, and the cluster ids as the value, and perform a distributed in-memory (key,value) sort. This can be achieved in a simple manner since all keys are unique – we evenly assign particles to nodes according to their keys (global ids), and bucket particles into which node they should end up in. An all-to-all communication step then brings particles to the right destination nodes. Each node then performs a local sort of all particles they receive; this will then be in sorted order. We arrange particles such that the first few particles end up in node 0, the next few in node 1 and so on. This allows consecutive writes.

3.5.2 File writes

We have used H5Part [25], a veneer API on top of HDF5, to write the cluster IDs into a HDF5 file. H5Part files are valid HDF5 files and are compatible with other HDF5-based interfaces and tools. By constraining the usage scenario to particle-based data, H5Part is capable of encapsulating much of the complexity of implementing effective parallel I/O in HDF5. In writing the cluster IDs, we have striped the output files on Lustre file system to use all available storage targets. Identical to reading, we use MPI-IO in collective I/O mode to use a small number of aggregators to interact with the file system and hence achieve similar behavior.

3.6 Summary of Optimizations

We now summarize the various kinds of optimizations we performed. Overall, we parallelized all steps using hybrid MPI + OpenMP. We modified various data structures to enable vectorization and to improve the locality of cache accesses.

We also perform various optimizations to reduce memory overhead such as in-place local partitioning and avoid replicating data more than twice during this process. We use bit-vectors during the local computation phase to ensure that we do not insert the same particle more than once into the local processing queues. Finally, as a note, for large datasets, we frequently run into MPI limitations regarding the size of messages that can be sent in a single MPI call. Hence we need to block these to match MPI limitations. This happens in partitioning, gathering and computing cluster id steps.

4. EXPERIMENTAL CONFIGURATION

We now describe the experimental setup including the cluster configuration, datasets and software tools used for benchmarking the end-to-end BD-CATS pipeline.

4.1 Hardware Platform

We performed all the experiments on Edison, a Cray XC30 supercomputing system, at the National Energy Research Scientific Computing Center (NERSC). Edison's compute partition consists of 5576 compute nodes. Each compute node of Edison is configured with two 12-core Intel® Xeon® [1] E5-2695 v2 processors at 2.4 GHz and 64 GB of 1866-DDR3 memory. Compute nodes communicate using a Cray Aries interconnect that supports injection rates of 10 GB/s bi-directional bandwidth per node. Edison has a total of 7.4 PB of "scratch" storage provided by a Cray Sonexion 1600 Lustre appliance. This aggregate storage is divided into three partitions. We have used the partition with 3.2 PB capacity that is configured with 36 Lustre Object Storage Servers (OSSs) and 144 Object Storage Targets (OSTs). This partition has a peak bandwidth of 72 GB/s. Depending on the size of the datasets, we varied the striping of files used in this study on the Lustre file system.

Table 1: Structural properties of the testbed (cosmology, $cosmo*$ and plasma physics, $plasma*$ datasets) with the input parameters and the taken time in seconds using p cores. K, B, and T stands for thousand, billion and trillion, respectively.

Name	Particles	Bounding Box	$(\epsilon, minpts)$	Time(s)	Cores(p)
$cosmo\_small$	1.07 B	$500 \times 500 \times 500$	0.09766, 1	227	96
$cosmo\_medium$	8.60 B	$1K \times 1K \times 1K$	0.09766, 1	249	768
$cosmo\_large$	68.72 B	$2K \times 2K \times 2K$	0.09766, 1	364	6144
$cosmo\_xlarge$	1.07 T	$8K \times 8K \times 8K$	0.15625, 1	1146	98304
$plasma\_large$	188.84 B	$330 \times 330 \times 132$	0.1, 240	689	24576
$plasma\_xlarge$	1.40 T	$330 \times 330 \times 132$	0.1, 600	1869	98304

4.2 Datasets

A summary of all the datasets used is presented in Table 1. We give more details about the datasets below.

4.2.1 Cosmology Data

To test the accuracy and efficiency of BD-CATS, we have produced a set of 3 cosmological N-body simulations using Gadget code [45]. We have run flat ΛCDM cosmological model with parameters: Hubble constant $h = 0.72$, total matter content $\Omega_m =$

[1] Intel and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

0.25, baryonic matter content $\Omega_b = 0.0432$, normalization of linear power spectrum of density fluctuations $\sigma_8 = 0.8$, and primordial spectral index $n_s = 0.97$. Simulations are started at redshift $z = 200$, and evolved till today when $z = 0$, which is the output we analyze in this paper. Above parameters are cited purely for completeness; in no way our choice of cosmological parameters affect any of the conclusions we present here, nor the scaling properties of BD-CATS. This simulation set is excellent for exploring weak-scaling properties of BD-CATS – each simulation has the same gravitational softening parameter $\epsilon = 10h^{-1}$kpc, and the same particle mass, $m_p = 1.12 \times 10^{10} M_\odot$ (solar mass; 1 $M_\odot \approx 2\times10^{30}$kg). The volume of simulations and the number of particles are increasing synchronously, the 3 runs have 1024^3, 2048^3, and 4096^3 particles in a cubic box with $500h^{-1}$Mpc, $1000h^{-1}$Mpc, and $2000h^{-1}$Mpc side, respectively. Note that we do not completely maintain the same amount of work per MPI rank as we scale the problem up. The initial conditions in cosmological simulations are Gaussian random field of density fluctuations, therefore in larger volume we better sample the tail of distribution, meaning we initially produce very rare density peaks, which will result in very massive halos at $z = 0$. This is nevertheless as close as we can get to ideal weak-scaling requirements, while still analyzing realistic cosmological simulations.

In addition to these 3 runs, we also use the biggest cosmological N-body simulation run to date, with $10{,}240^3$ particles [44]. This simulation was produced with a different code, using a set of different cosmological and numerical parameters, thus it is not just the "bigger" version of other runs in this paper. We analyze this simulation to demonstrate that BD-CATS can successfully be applied to one of the largest datasets in cosmology. Cosmological simulations use periodic boundaries in all 3 dimensions. Particle properties in all the cosmology datasets include spatial location (x,y,and z), and particle velocities (vx, vy, and vz). The biggest dataset is stored in a single HDF5 file with a size of 24 TB, where each particle property organized into a HDF5 dataset. The smaller datasets, i.e., 1024^3, 2048^3, 4096^3 data, also follow the same file organization and are of size ∼24 GB, ∼192 GB, and ∼1.6 TB, respectively.

4.2.2 Plasma Physics Data

The 3D simulation of magnetic reconnection in electron-positron plasma was performed using high-performance fully relativistic PIC code VPIC [5]. The initial conditions correspond to Harris current sheet equilibrium [22] $B_x = B_0 \tanh(z/\delta)$ with a guide magnetic field of equal strength to the reconnecting component $B_y = B_0$, such that the rotation angle of the magnetic field through the layer is 90°. The initial current sheet thickness is $\delta = 1c/\omega_{pe}$, where c is the speed of light and ω_{pe} is the plasma frequency. The background plasma density $n_b = 0.3n_0$, where n_0 is the peak density of the Harris current sheet. The ratio of electron to positron temperature is $T_e/T_p = 1$, and the ratio of electron plasma frequency to the electron cylotron frequency is $\omega_{pe}/\omega_{ce} = 3$. The simulation is performed in a 3D domain with open boundary conditions [12] of size $(330 \times 330 \times 132)c/\omega_{pe}$ with $2000 \times 2000 \times 800$ cells. The average initial particle density is 320 particles per species per cell, so that the simulation started tracking roughly two trillion particles (one trillion electrons and one trillion ions) and as it progressed more particles were added due to the open boundaries.

Particle properties of interest in this simulation include spatial location (x,y,z), kinetic energy $E = m_e c^2(\gamma - 1)$, and individual components of particle velocity U_x, U_y, and U_z. The electron data we used was written in the ∼23,000 time step that contained the properties of ∼1.4 trillion particles. The particle data was stored in a single HDF5 file of ∼36 TB, with each property of the particles was organized as a HDF5 dataset. For a smaller scale run, we have used a subset of the large file when we extracted all the data related to particles with $E > 1.1m_e c^2$. The resulting sizes of the subset file is ∼5 TB, where the number of particles is ∼188.8 billion.

4.3 Design of Scaling Experiments

All code was implemented in C/C++ and was compiled using Intel® C++ compiler v.15.0.1[2]. The code was parallelized using OpenMP and MPI. We used Intel® MPI library v.5.0.2.

We demonstrate strong scaling on the large cosmology and plasma physics datasets (see Table 1) in Section 5.3.2. We cluster using the 3 spatial coordinates with Euclidean distance metric computation (with periodic boundaries for cosmology). We show detailed performance and runtime breakdown for the xlarge (trillion+ point) datasets in Section 5.4. We use the small, medium and large cosmology datasets to demonstrate weak scaling using 96 to 6144 cores. For these datasets, the corresponding DBSCAN input parameters (ϵ, $minpts$) are shown in Table 1.

For cosmology datasets, a very common value for FOF search radius is 1/5th of the mean interparticle spacing, which roughly corresponds to the iso-density surface of ∼82× the mean matter density in a simulation (see, e.g. [35]). We use this as ϵ; in addition, we focus here on $minpts = 1$, as this choice of parameters eases validation against known FOF results. We show results for $minpts = 3$ in Section 5.5.1. For plasma physics data, the structures arising in reconnection layers are known to have a certain characteristic scale length. Therefore, the value of ϵ was chosen to be reasonably close to that scale. For a given ϵ, the value of $minpts$ was estimated from local values of plasma density.

5. RESULTS

5.1 Improvement in end-to-end performance

We compare our end-to-end clustering time to previous work, Pardicle [40] (exact version) on $cosmo\_small$ dataset using 32 nodes. The results of the comparison are tabulated in Table 2. Although Pardicle is not an end-to-end optimized system (focused only on improvements to the DBSCAN kernel), it was recently shown to be the best performing DBSCAN implementation to date [40]. We present this comparison to show how the other steps impact the performance of DBSCAN kernel. Due to a combination of better load balancing and other code optimizations, the performance of the DBSCAN kernel itself improves by a factor of 5.1×. Considering all steps, BD-CATS shows an end-to-end speedup of 7.7× over an end-to-end Pardicle run.

Table 2: End-to-end performance comparison of BD-CATS with the exact version of Pardicle [40]

	Reading	Partitioning	Gathering	Kd-tree	DBSCAN	Cluster_ids	Storing	Total
Pardicle	18.4	33.4	14.4	78.5	91.1	8.6	27.4	271.7
BD-CATS	6.7	3.1	0.2	2.5	17.7	1.2	4.1	35.5
Speedup	2.7	10.8	62.1	31.6	5.1	7.3	6.7	7.7

We explain the component speedups below. Reading and writing

[2]Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel micro-architecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

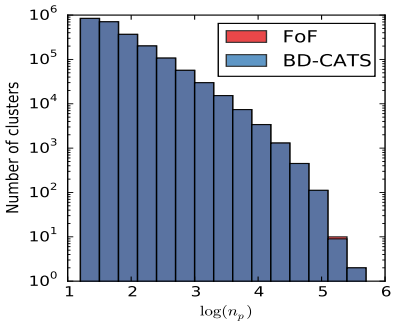


Figure 3: Histogram of cluster masses in 1024^3 cosmological simulation using FOF and DBSCAN halo finder.

times improved due to the use of HDF5 formats and better use of parallel I/O through striping as explained in Section 3. Partitioning and gathering time improved because of multithreading. We used a sampling-based median computation for partitioning (as discussed in Section 3.2), which improves load balancing for downstream tasks. Most of the overall performance gain was achieved due to much better load balancing in the most compute-expensive portions of the code (Kd-tree construction and DBSCAN). The range of particles per node was [10M, 91M] for Pardicle, whereas BD-CATS produces partitions in the range of [33M, 34M]. This improves the skew (max/mean) from 2.7 to 1.0. We also found that within a node Kd-tree construction is more balanced due to use of better median in our case. While Pardicle's Kd-trees have 21-25 levels, we found that BD-CATS consistently produces Kd-trees with 20 levels ($cosmo\_small$ dataset). These improvements in load balancing improved the runtime of DBSCAN kernel even when no code optimizations were done here. We observed a speedup of 2.6x over Pardicle due to this factor alone. In addition, code optimizations such as improved parallelizations and cache utilization gave an additional 1.9x speedup. Kd-tree construction itself was also optimized as explained in Section 3.4.1. Final clustering stage was serialized in Pardicle i.e. one node gathered all points to compute the cluster ids. This step is fully distributed in BD-CATS (Section 3.4.3). We saw similar speedups for other datasets as well.

5.2 Accuracy

To validate BD-CATS, we have compared results of DBSCAN with $minpts$ parameter set to 1 with a parallel FOF halo (cluster) finder originally developed in Los Alamos [21], and tested against many other halo finder codes in use within the cosmology community [29] (see "LANL finder"). While in principle the two codes could produce identical results, in practice that will not happen as both algorithms rely on a floating-point comparison, i.e. they test if a neighboring particle is inside a search radius or not. The most straightforward comparison is to go on a halo by halo basis and check the level of agreement; in practice that is unnecessarily cumbersome, as cosmological simulations produce very large number of small halos (Figure 3). We have conducted one-to-one mapping for a random subset of 20 halos with 100,000 or more particles. We observed a 0.001% difference in the number of points between the two algorithms. Information about the three largest clusters is presented in Table 3. The level of agreement between DBSCAN and FOF is great as expected, generally within few particles. Comparing BD-CATS with Pardicle, both systems use the same DBSCAN clustering algorithm, and hence produce results of identical quality.

5.3 Scalability

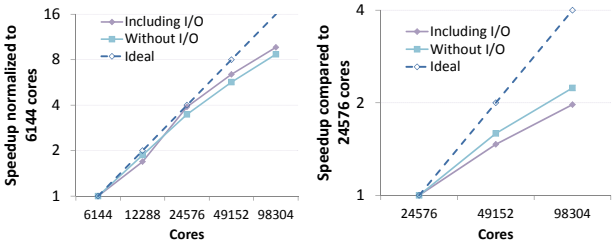
We showcase strong and weak scaling of BD-CATS on cosmology and plasma physics datasets below.

Table 3: Comparison of number of particles in top 3 biggest clusters with FOF[21]

BD-CATS	FOF	Difference
362,885	362,889	4
288,333	288,336	3
243,590	243,589	1

5.3.1 *Strong scaling*

In order to demonstrate strong scaling, we ran BD-CATS on varying number of cores using two large datasets, $cosmo\_large$ and $plasma\_large$ with 69B and 189B particles, respectively. $cosmo\_large$ dataset was run on five scales with core counts ranging from 6,144 to 98,304, whereas the $plasma\_large$ dataset was run on three scales from 24,576 to 98,304 cores. Since the plasma physics dataset is much larger than the cosmology dataset, hence it requires more memory and could not be run on fewer than 24,576 cores. Figure 4 shows the results of our strong scaling experiments.



(a) Cosmology (b) Plasma physics

Figure 4: Strong scaling on $cosmo\_large$ (69B particles) and $plasma\_large$ (189B particles) normalized to the time taken on 6,144 and 24,576 cores respectively.

We observe a speedup of $9.7\times$ excluding I/O and $8.7\times$ including I/O while running the cosmology dataset on 98,304 cores compared to running on 6,144 cores ($16\times$ cores). For the plasma physics dataset, increasing the core count by $4\times$ improves runtime by $2.0\times$ and $2.2\times$ when excluding and including I/O runtimes, respectively. For the plasma physics dataset, we achieved relatively low speedups due to high communication overheads for large scale runs in the find cluster ids step. Specifically, clusters in these datasets are large and span several nodes, and the span increases with increasing node count. Hence the number of communication rounds required to reach the root of the union-find tree increases with node count. Most of the nodes remain idle during the later rounds, resulting in low scalability. In contrast, the clusters in cosmology datasets are not as large and hence performance on these datasets scales well.

5.3.2 *Weak scaling*

We fix the number of particles per node to be $\sim$250M and run weak scaling experiments on the cosmology datasets. Figure 5 shows the result of running the small, medium and large cosmology datasets on 96, 768 and 6,144 cores, respectively. We see good scalability in end-to-end runtime ($1.6\times$ runtime increase when scaled to $64\times$ more cores and data). Weak scaling was not possible on plasma physics datasets as the dataset characteristics change when using different energy filters, so work per node does not remain constant while scaling down.

5.4 Trillion particle clustering

We demonstrate BD-CATS, the first end-to-end trillion particle clustering system on both cosmology and plasma physics xlarge datasets. These datasets are 24TB (1T particles) and 36TB (1.4T particles) in size, respectively. We used 98,304 cores to perform the experiment. The complete end-to-end run took 20 and 30 minutes,

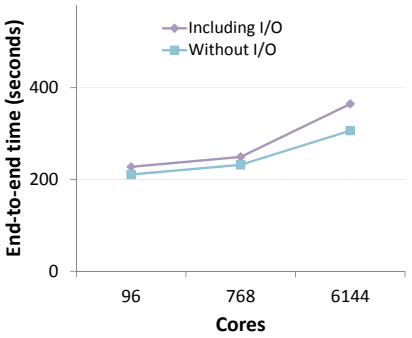
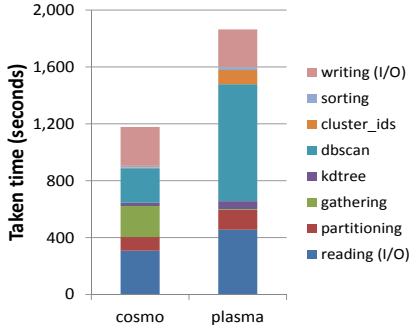
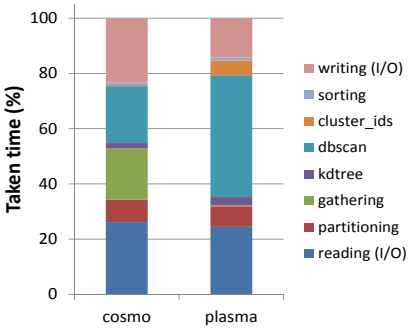


Figure 5: Weak scaling on small, medium and large cosmology datasets using 96, 768, and 6,144 cores, respectively.



(a) Actual time (seconds)



(b) Relative time (%)

Figure 6: Taken time by different steps of BD-CATS on the largest cosmology ($cosmo\_xlarge$, 1 trillion particles) and plasma physics ($plasma\_xlarge$, 1.4 trillion particles) datasets using 98,304 cores.

respectively. The detailed breakdown is shown in Figure 6.

For the trillion particle clustering, I/O is the most dominant component of runtime, taking about 50% and 40% of the total runtime for the cosmology and plasma physics datasets respectively. DBSCAN is the next largest component taking around 20% and 44% respectively. For the plasma physics dataset, the number of particles per node is 40% higher than cosmology. The datasets also have difference characteristics - each particle in plasma physics dataset has on average ~ 410 neighbors, whereas cosmology dataset has only $\sim$ 25 neighbors. These factors show up as increased DBSCAN runtime for the plasma physics dataset. For the cosmology dataset, gathering takes significant time (19%). This is due to the periodic boundary condition, leading to more intersection tests (see Section 3.3) and an increased number of points being gathered. We continue to see that the finding cluster ids step takes more time on the plasma physics dataset (5% of overall time) as compared to the cosmology dataset (0.4% of overall time).

5.5 Science interpretations

5.5.1 *Cosmology*

As discussed previously in Section 2.1, DBSCAN can be thought of a superset of FOF algorithm. Although prolifically used, FOF has some known drawbacks, most notably the easy linking of two different clusters into one via a narrow stream of particles connecting them (see e.g. left panel in Figure 7). This situation is very common in cosmological applications, as smaller halos are driven by gravity into merging with bigger halos, and tidal stripping produces particle "bridges" much before centers of two halos will collide. In Figure 7 we present an example of how the $minpts$ parameter in DBSCAN can be used to improve application accuracy via removing such "bridges". In this example, changing the $minpts$ parameter from 1 (equivalent to FOF) to 3 clearly improves the cluster quality. As middle panel shows, $minpts = 3$ didn't affect the "main" part of a halo in a significant way, but has removed links to most of smaller, in-falling halos. Thus DBSCAN can not only reproduce FOF results, but also offers promise of improving them via optimally chosen $minpts$ parameter. Finding the optimal choice is beyond the scope of this paper, and is something we will pursue in the future.

5.5.2 *Plasma physics*

Theoretical analysis and experimental observations suggest that magnetic reconnection can efficiently accelerate charged particles to very high energies. While a variety of competing mechanisms have been proposed in the literature, a great deal of outstanding questions remain. Quantifying distribution of accelerated particles in space and energy is important since those distributions reflect basic properties of the acceleration mechanism operating in a given scenario. As an example, here we discuss analysis of the $E > 1.5 m_e c^2$ dataset that contains a high-energy subset of all the particles in the simulation. To the best of our knowledge, this is the first application of clustering analysis to the problem of magnetic reconnection.

Figure 8 shows the spatial distribution of clusters identified by DBSCAN. The high-density clusters are predominantly localized within the current sheet and appear as narrow structures elongated along the direction of local magnetic field. The availability of detailed information on the particles comprising the clusters allowed us to perform analysis not possible previously. For example, the left panel in Fig. 9 shows distribution of average particle energy inside the clusters and compares it with the distribution of all the particles in the data set. It is apparent that the clusters are associated with a particular, narrow energy range. Moreover, the particles within each cluster are characterized by a very narrow distribution of energies. This is illustrated in the right panel of Fig. 9, which shows normalized variance of particle energy inside the clusters. A possible interpretation of these results is that the particles comprising the clusters have been accelerated in a process where they gain a fixed amount of energy in a relatively narrow region of space.

6. RELATED WORK

Halo finding is an essential analysis steps in N-body cosmological simulations, and there are many different approaches to it with tens of independently written codes. We refer the reader to the recent review on halo finding [30]. The FOF halo finder we use as benchmark in this paper (listed as "LANL finder" in the mentioned review) is presently used as *in situ* halo finder in the state-of-the-art HACC code [19, 20]. In contrast, clustering analysis of particle space in plasma physics applications is used rarely, if at all. Exist-

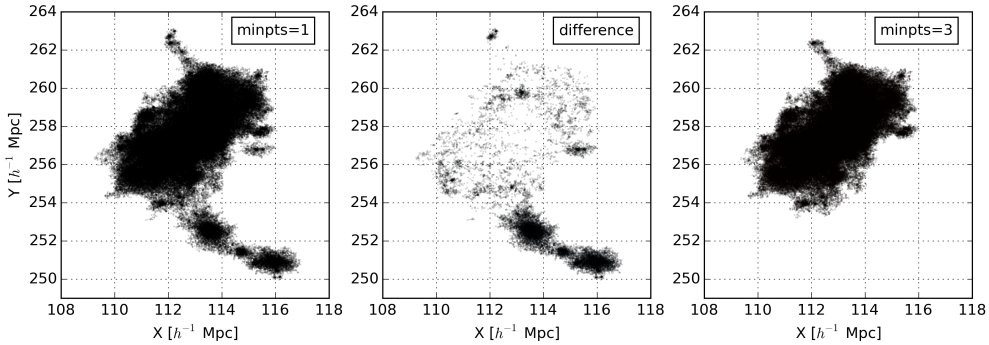


Figure 7: Example of improvement DBSCAN can offer in cosmological halo finding. In the left panel, we show the most massive halo in our 1024^3 simulation using FOF halo finder or equivalently DBSCAN with $minpts = 1$. In the right panel we show particles which DBSCAN defines as a part of halo when we set $minpts = 3$. Particles which appear in $minpts = 1$ halo, but not in $minpts = 3$ halo are scatter plotted in the middle panel, and we can clearly see that most significant bridges are removed, while the "main" part of the object is almost unchanged.

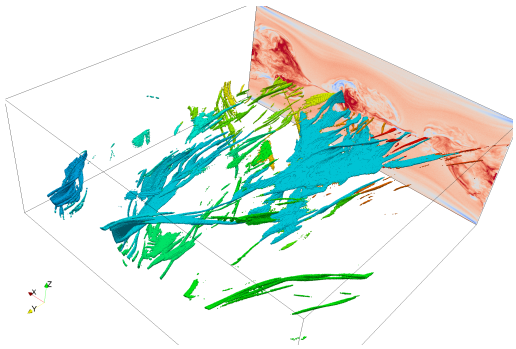


Figure 8: Clusters identified in the plasma physics $E > 1.5m_ec^2$ dataset with $\epsilon = 0.3$ and $minpts = 300$. The back panel shows plasma density in the simulations. Clusters are colored by the ID.

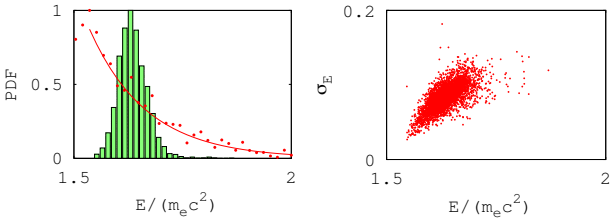


Figure 9: Analysis of energy distribution of clusters identified in the plasma physics $E > 1.5m_ec^2$ dataset with $\epsilon = 0.3$ and $minpts = 300$. Left panel shows probability distribution function (PDF) of mean particle energy associated with clusters (bars), $\langle E_k$. For reference, we also show PDF of all particles in the datasets (dots) and its power law approximation (solid line). Right panel shows normalized variance σ_E of particle energy inside each cluster.

ing methods typically focus on clustering of grid data, for example identifying current sheets in fluid turbulence simulations [47].

As mentioned earlier, many existing parallelizations of DBSCAN adopt the master-slave model [6, 3, 8, 9, 53, 54]. The idea is that the data is equally partitioned and distributed among the slaves. Each slave then computes the clusters locally and sends back the results to the master, which then merges the clusters sequentially to obtain the final clusters. This strategy incurs high communication overhead between the master and slaves, and a low parallel efficiency during the merging process. Since the master has to hold the entire clustering solution, this does not scale to today's massive datasets. Several map-reduce or hadoop-based approach were presented in [17, 23, 10]. None of these are end-to-end solutions and assumes several pre-processing or post-processing steps.

Recently, DBSCAN has been re-designed to break the inherent sequential nature using the concept of using union-find trees. The authors demonstrated high parallelism (6k speedup using 8k cores) on datasets of size 100 million particles[39]. This work has later been extended to an approximate version with high quality [40]. Although these works progressed DBSCAN significantly, they mainly improved parallelism of the core algorithm, not paying attention to assumed pre-processing or post-processing steps such as partitioning, kd-tree-construction or finding cluster ids. These steps were partially parallel or even sequential on a single node, hence not suitable for scientific applications with trillion range datasets.

To the best of our knowledge, the only available end-to-end system is a GPU based DBSCAN implementation [51, 50]. They showed that 6.5 billion 2D particles can be processed using 8,192 GPU nodes in 7.5 minutes. Even assuming perfect scalability, their approach would requires $\sim$ a million nodes to process a trillion particles. In contrast, we process 1.4 Trillion 3D particles using 4,096 nodes in 30 minutes.

7. CONCLUSION AND FUTURE WORK

This paper has presented a BD-CATS, a highly scalable, end-to-end framework for performing clustering on massive simulation output. We have demonstrated weak and strong scaling for all stages in the BD-CATS pipeline on up to $\sim$ 100,000 cores of a Cray XC30 system. In the first exercise of its kind, we have successfully demonstrated application of BD-CATS to a trillion particle cosmology simulation, and a 1.4 trillion particle plasma physics simulation dataset. Both of these applications have facilitated first-time scientific insights into important scientific questions; an en-

deavor that has been previously intractable due to the sheer size of datasets. We believe that such highly scalable clustering tools will be critical for analysis of both simulation and observational datasets. As future work, we intend to use BD-CATS to gain insights into large bioimaging datasets.

8. ACKNOWLEDGMENT

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract number DE-AC02-05CH11231. ZL acknowledges support by the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy Office of Advanced Scientific Computing Research and the Office of High Energy Physics. This research used resources of the National Energy Research Scientific Computing Center. The authors would like to acknowledge the excellent support extended by NERSC staff (Tina Declerck, Tine Butler, Zhenji Zhao and Jay Srinivasan) in facilitating these runs. This work made use of the NASA Astrophysics Data System and of the astro-ph preprint archive at arXiv.org.

9. REFERENCES

[1] Magnetospheric multiscale (mms) mission. http://mms.gsfc.nasa.gov/.

[2] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD*, pages 49–60, New York, NY, USA, 1999. ACM.

[3] D. Arlia and M. Coppola. Experiments in parallel clustering with DBSCAN. In *Euro-Par 2001 Parallel Processing*, pages 326–331. Springer, LNCS, 2001.

[4] D. Birant and A. Kut. ST-DBSCAN: An algorithm for clustering spatial-temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[5] K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulationa). *Physics of Plasmas (1994-present)*, 15(5):–, 2008.

[6] S. Brecheisen, H. Kriegel, and M. Pfeifle. Parallel density-based clustering of complex objects. *Adv. in Know. Discovery and Data Mining*, pages 179–188, 2006.

[7] S. Byna, A. Uselton, D. K. Prabhat, and Y. He. Trillion particles, 120,000 cores, and 350 tbs: Lessons learned from a hero i/o run on hopper. In *Cray User Group meeting*, 2013.

[8] M. Chen, X. Gao, and H. Li. Parallel DBSCAN with priority r-tree. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*, pages 508–511. IEEE, 2010.

[9] M. Coppola and M. Vanneschi. High-performance data mining with skeleton-based structured parallel programming. *Parallel Computing*, 28(5):793–813, 2002.

[10] B.-R. Dai and I. Lin. Efficient map/reduce-based dbscan algorithm with optimized data partition.

[11] W. Daughton, V. Roytershteyn, H. Karimabadi, L. Yin, B. J. Albright, B. Bergen, and K. J. Bowers. Role of electron physics in the development of turbulent magnetic reconnection in collisionless plasmas. *Nat Phys*, 7(7):539–542, July 2011.

[12] W. Daughton, J. Scudder, and H. Karimabadi. Fully kinetic simulations of undriven magnetic reconnection with open boundary conditions. *Physics of Plasmas (1994-present)*, 13(7):072101, 2006.

[13] M. Davis, G. Efstathiou, C. S. Frenk, and S. D. M. White. The evolution of large-scale structure in a universe dominated by cold dark matter. *Astrophysical Journal*, 292:371–394, May 1985.

[14] D. J. DeWitt, J. F. Naughton, and D. A. Schneider. Parallel sorting on a shared-nothing architecture using probabilistic splitting. In *Parallel and Distributed Information Systems, 1991., Proceedings of the First International Conference on*, pages 280–291. IEEE, 1991.

[15] J. Einasto, A. A. Klypin, E. Saar, and S. F. Shandarin. Structure of superclusters and supercluster formation. III Quantitative study of the local supercluster. *Monthly Notices of the Royal Astronomical Society*, 206:529–558, Feb. 1984.

[16] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, volume 1996, pages 226–231. AAAI Press, 1996.

[17] Y. Fu, W. Zhao, and H. Ma. Research on parallel DBSCAN algorithm design based on mapreduce. *Advanced Materials Research*, 301:1133–1138, 2011.

[18] M. T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

[19] S. Habib, V. Morozov, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, T. Peterka, J. Insley, D. Daniel, P. Fasel, N. Frontiere, and Z. Lukić. The Universe at Extreme Scale: Multi-Petaflop Sky Simulation on the BG/Q. *SC '12*, Nov. 2012.

[20] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukić, S. Sehrish, and W.-k. Liao. HACC: Simulating Sky Surveys on State-of-the-Art Supercomputing Architectures. *ArXiv e-prints*, Oct. 2014.

[21] S. Habib, A. Pope, Z. Lukić, D. Daniel, P. Fasel, N. Desai, K. Heitmann, C.-H. Hsu, L. Ankeny, G. Mark, S. Bhattacharya, and J. Ahrens. Hybrid petacomputing meets cosmology: The Roadrunner Universe project. *Journal of Physics Conference Series*, 180(1):012019, July 2009.

[22] E. Harris. On a plasma sheath separating regions of oppositely directed magnetic field. *Il Nuovo Cimento Series 10*, 23(1):115–121, 1962.

[23] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan. Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data. *Frontiers of Computer Science*, 8(1):83–99, 2014.

[24] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan. Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 473–480. IEEE, 2011.

[25] M. Howison, A. Adelmann, E. W. Bethel, A. Gsell, B. Oswald, and Prabhat. H5hut: A High-Performance I/O Library for Particle-Based Simulations. In *Proceedings of 2010 Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10)*, Heraklion, Crete, Greece, Sept. 2010. LBNL-4021E.

[26] S. Huo. *Detecting Self-Correlation of Nonlinear, Lognormal, Time-Series Data via DBSCAN Clustering Method, Using Stock Price Data as Example*. PhD thesis, The Ohio State University, 2011.

[27] D. Kagan, L. Sironi, B. Cerutti, and D. Giannios. Relativistic magnetic reconnection in pair plasmas and its astrophysical

applications. *Space Science Reviews*, pages 1–29, 2015.

[28] C. Kim, J. Park, N. Satish, H. Lee, P. Dubey, and J. Chhugani. Cloudramsort: Fast and efficient large-scale distributed ram sort on shared-nothing cluster. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 841–850, New York, NY, USA, 2012. ACM.

[29] A. Knebe, S. R. Knollmann, S. I. Muldrew, F. R. Pearce, M. A. Aragon-Calvo, Y. Ascasibar, P. S. Behroozi, D. Ceverino, S. Colombi, J. Diemand, K. Dolag, B. L. Falck, P. Fasel, J. Gardner, S. Gottlöber, C.-H. Hsu, F. Iannuzzi, A. Klypin, Z. Lukić, M. Maciejewski, C. McBride, M. C. Neyrinck, S. Planelles, D. Potter, V. Quilis, Y. Rasera, J. I. Read, P. M. Ricker, F. Roy, V. Springel, J. Stadel, G. Stinson, P. M. Sutter, V. Turchaninov, D. Tweed, G. Yepes, and M. Zemp. Haloes gone MAD: The Halo-Finder Comparison Project. *Monthly Notices of the Royal Astronomical Society*, 415:2293–2318, Aug. 2011.

[30] A. Knebe, F. R. Pearce, H. Lux, Y. Ascasibar, P. Behroozi, J. Casado, C. C. Moran, J. Diemand, K. Dolag, R. Dominguez-Tenreiro, P. Elahi, B. Falck, S. Gottlöber, J. Han, A. Klypin, Z. Lukić, M. Maciejewski, C. K. McBride, M. E. Merchán, S. I. Muldrew, M. Neyrinck, J. Onions, S. Planelles, D. Potter, V. Quilis, Y. Rasera, P. M. Ricker, F. Roy, A. N. Ruiz, M. A. Sgró, V. Springel, J. Stadel, P. M. Sutter, D. Tweed, and M. Zemp. Structure finding in cosmological simulations: the state of affairs. *Monthly Notices of the Royal Astronomical Society*, 435:1618–1658, Oct. 2013.

[31] H.-P. Kriegel and M. Pfeifle. Hierarchical density-based clustering of uncertain data. In *Data Mining, Fifth IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[32] Z. Lukić, D. Reed, S. Habib, and K. Heitmann. The structure of halos: Implications for group and cluster cosmology. *The Astrophysical Journal*, 692(1):217, 2009.

[33] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. USA, 1967.

[34] S. Madeira and A. Oliveira. Biclustering algorithms for biological data analysis: a survey. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 1(1):24–45, 2004.

[35] S. More, A. V. Kravtsov, N. Dalal, and S. Gottlöber. The Overdensity and Masses of the Friends-of-friends Halos and Universality of Halo Mass Function. *Astrophysical Journal Supplement*, 195:4, July 2011.

[36] A. Mukhopadhyay and U. Maulik. Unsupervised satellite image segmentation by combining SA based fuzzy clustering with support vector machine. In *Proceedings of 7th ICAPR'09*, pages 381–384. IEEE, 2009.

[37] H. Park and C. Jun. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36(2):3336–3341, 2009.

[38] M. Patwary, J. Blair, and F. Manne. Experiments on union-find algorithms for the disjoint-set data structure. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA 2010)*, pages 411–423. Springer, LNCS 6049, 2010.

[39] M. A. Patwary, D. Palsetia, A. Agrawal, W.-k. Liao, F. Manne, and A. Choudhary. A new scalable parallel dbscan algorithm using the disjoint-set data structure. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 62:1–62:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[40] M. M. A. Patwary, N. Satish, N. Sundaram, F. Manne, S. Habib, and P. Dubey. Pardicle: Parallel approximate density-based clustering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 560–571, Piscataway, NJ, USA, 2014. IEEE Press.

[41] P. J. E. Peebles. *The large-scale structure of the universe*. 1980.

[42] Planck Collaboration, P. A. R. Ade, N. Aghanim, M. Arnaud, M. Ashdown, J. Aumont, C. Baccigalupi, A. J. Banday, R. B. Barreiro, J. G. Bartlett, and et al. Planck 2015 results. XIII. Cosmological parameters. *ArXiv e-prints*, Feb. 2015.

[43] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: a wavelet-based clustering approach for spatial data in very large databases. *The VLDB Journal*, 8(3):289–304, 2000.

[44] S. W. Skillman, M. S. Warren, M. J. Turk, R. H. Wechsler, D. E. Holz, and P. M. Sutter. Dark Sky Simulations: Early Data Release. *ArXiv e-prints*, July 2014.

[45] V. Springel. The cosmological simulation code GADGET-2. *Monthly Notices of the Royal Astronomical Society*, 364:1105–1134, Dec. 2005.

[46] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO. In *Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation*, FRONTIERS '99, pages 182–, Washington, DC, USA, 1999. IEEE Computer Society.

[47] V. M. Uritsky, A. Pouquet, D. Rosenberg, P. D. Mininni, and E. F. Donovan. Structures in magnetohydrodynamic turbulence: Detection and scaling. *Phys. Rev. E*, 82:056326, Nov 2010.

[48] W. Wang, J. Yang, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the International Conference on Very Large Data Bases*, pages 186–195. IEEE, 1997.

[49] M. S. Warren, K. Abazajian, D. E. Holz, and L. Teodoro. Precision Determination of the Mass Function of Dark Matter Halos. *Astrophysical Journal*, 646:881–885, Aug. 2006.

[50] B. Welton and B. P. Miller. 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*, pages 54–60. IEEE Press, 2014.

[51] B. Welton and B. P. Miller. Mr. scan: A hybrid/hybrid extreme scale density based clustering algorithm. 2014.

[52] M. White. The mass of a halo. *Astronomy and Astrophysics*, 367:27–32, Feb. 2001.

[53] X. Xu, J. Jäger, and H. Kriegel. A fast parallel clustering algorithm for large spatial databases. *High Performance Data Mining*, pages 263–290, 2002.

[54] A. Zhou, S. Zhou, J. Cao, Y. Fan, and Y. Hu. Approaches for scaling DBSCAN algorithm to large spatial databases. *Computer science and technology*, 15(6):509–526, 2000.