# Streaming Algorithms for Halo Finders

Zaoxing Liu* [1], Nikita Ivkin* [1], Lin F. Yang† [2], Mark Neyrinck† [3], Gerard Lemson† [3],
Alexander S. Szalay† [3], Vladimir Braverman* [4], Tamas Budavari†, Randal Burns*, Xin Wang† [3]
*Department of Computer Science
†Department of Physics & Astronomy
Johns Hopkins University
Baltimore, MD 21218, USA

***Abstract*—Cosmological $N$-body simulations are essential for studies of the large-scale distribution of matter and galaxies in the Universe. This analysis often involves finding clusters of particles and retrieving their properties. Detecting such "halos" among a very large set of particles is a computationally intensive problem, usually executed on the same super-computers that produced the simulations, requiring huge amounts of memory.**

**Recently, a new area of computer science emerged. This area, called streaming algorithms, provides new theoretical methods to compute data analytics in a scalable way using only a single pass over a data sets and logarithmic memory.**

**The main contribution of this paper is a novel connection between the $N$-body simulations and the streaming algorithms. In particular, we investigate a link between halo finders and the problem of finding frequent items (heavy hitters) in a data stream, that should greatly reduce the computational resource requirements, especially the memory needs. Based on this connection, we can build a new halo finder by running efficient heavy hitter algorithms as a black-box. We implement two representatives of the family of heavy hitter algorithms, the Count-Sketch algorithm (CS) and the Pick-and-Drop sampling (PD), and evaluate their accuracy and memory usage. Comparison with other halo-finding algorithms from [1] shows that our halo finder can locate the largest haloes using significantly smaller memory space and with comparable running time. This streaming approach makes it possible to run and analyze extremely large data sets from $N$-body simulations on a smaller machine, rather than on supercomputers. Our findings demonstrate the connection between the halo search problem and streaming algorithms as a promising initial direction of further research.**

[1]This material is based upon work supported in part by the NSF under Grant No. 1447639 and by DARPA grant N660001-1-2-4014. Its contents are solely the responsibility of the authors and do not represent the official view of DARPA or the Department of Defense.

[2]This material is based upon work supported in part by the National Science Foundation under Grant No. 1447639

[3]This material is based upon work supported in part by a grant in Data-Intensive Science from the Gordon and Betty Moore and Alfred P. Sloan Foundations.

[4]This material is based upon work supported in part by the NSF under Grant No. 1447639, by the Google Faculty Award and by DARPA grant N660001-1-2-4014. Its contents are solely the responsibility of the authors and do not represent the official view of DARPA or the Department of Defense.

## I. Introduction

The goal of astrophysics is to explain the observed properties of the universe we live in. In cosmology in particular, one tries to understand how matter is distributed on the largest scales we can observe. In this effort, advanced computer simulations play an ever more important role. Simulations are currently the only way to accurately understand the nonlinear processes that produce cosmic structures such as galaxies and patterns of galaxies. Hence a large amount of effort is spent on running simulations modelling representative parts of the universe in ever greater detail. A necessary step in the analysis of such simulations involves locating mass concentrations, called "haloes", where galaxies would be expected to form. This step is crucial to connect theory to observations – galaxies are the most observable objects that trace the large-scale structure, but their precise spatial distribution is only established through these simulations.

Many algorithms have been developed to find these haloes in simulations. The algorithms vary widely, even conceptually. There is no absolutely agreed-upon physical definition of a halo, although all algorithms give density peaks, i.e. clusters of particles. Galaxies are thought to form at these concentrations of matter. Some codes find regions inside an effective high-density contour, such as Friends-of-Friends (FoF) [2]. In FoF, particles closer to each other than a specified linking length are gathered together into haloes. Other algorithms directly incorporate velocity information as well. Another approach finds particles that have crossed each other as compared to the initial conditions, which also ends up giving density peaks [3]. FoF is often considered to be a standard approach, if only because it was among the first used, and is simple conceptually. The drawbacks of FoF include that the simple density estimate can artificially link physically separate haloes together, and the arbitrariness of the linking length. A halo-finding comparison project [4] evaluated the results of 17 different halo-finding algorithms; further analysis appeared in [1]. We take the FoF algorithm as a fiducial result for comparison, but compare to results from some other finders, as well.

Halo-finding algorithms are generally computationally intensive, often requiring all particle positions and velocities to be loaded in memory simultaneously. In fact most are executed during the execution of the simulation itself, requiring comparable computational resources. However, in order to understand the systematic errors in such algorithms, it is often necessary to run multiple halo-finders, often well after the original simulation has been run. Also, many of the newest simulations have several hundred billion to a trillion particles, with a very large memory footprint, making such posterior computations quite difficult. Here, we investigate a way to apply streaming algorithms as halo finders, and compare the results to those of other algorithms participating in the Halo-Finding Comparison Project.

Recently, streaming algorithms [5] have become a popular way to process massive data sets. In the streaming model, the input is given as a sequence of items and the algorithm is allowed to make a single or constant number of passes over the input data while using sub-linear, usually poly-logarithmic space compared to the storage of the data. Streaming algorithms have found many applications in networking ([6], [7], [8]), machine learning ([9], [10]), financial analytics ([11], [12], [13]) and databases ([14], [15]).

In this paper, we apply streaming algorithms to the area of cosmological simulations and provide space and time efficient solutions to the halo finding problem. In particular, we show a relation between the problem of finding haloes in the simulation data and the well-known problem of finding "heavy hitters" in the streaming data. This connection allows us to employ efficient heavy hitter algorithms, such as Count-Sketch [16] and Pick-and-Drop Sampling [17]. By equating heavy hitters to haloes, we are implicitly defining haloes as positions exceeding some high density threshold. In our case, these usually turn out to be density peaks, but only because of the very spiky nature of the particle distributions in cosmology. Conceptually, FoF haloes are also regions enclosed by high density contours, but in practice, the FoF implementation is very different from ours.

## II. Streaming Algorithm

In this section, we investigate the application of streaming algorithms to find haloes using a strong relation between the halo-finding problem and the heavy hitter problem, which we discuss in section II-A4. Heavy hitter algorithms find the $k$ densest regions, that may physically correspond to haloes. In our implementation, we carefully choose $k$ to get the desired outcome. This parameter $k$ is as also discussed in section II-A4. We first present in the next sub-section the formal definition of streaming algorithms and the connection between heavy hitter problem and halo-finding problem. After that, we presents the basic procedures of the two heavy hitter algorithms: Count-Sketch and Pick-and-drop Sampling.

### *A. Streaming Data Model*

*1) Definitions:* A data stream $D = D(n, m)$ is an ordered sequence of objects $p_1, p_2, \dots, p_n$, where $p_j = 1 \dots m$. The elements of the stream can represent any digital object: integers, real numbers of fixed precisions, edges of a large graphs, messages, images, web pages, etc. In the practical applications both $n$ and $m$ may be very large, and we are interested in the algorithms with $o(n + m)$ space. A streaming algorithm is an algorithm that can make a single pass over the input stream. The above constraints imply that a streaming algorithm is often a randomized algorithm that provides approximate answers with high probability. In practice, these approximate answers often suffice.

We investigate the results of cosmological simulations where the number of particles will soon reach $10^{12}$. Compared to offline algorithms that require the input to be entirely in memory, streaming algorithms provide a way to process the data using only megabytes memory instead of gigabytes or terabytes in practice.

*2) Heavy Hitter:* For each element $i$, its frequency $f_i$ is the number of its occurrences in $D$. The $k^{th}$ frequency moment of a data stream $D$ is defined as $F_k(D) = \sum_{i=1}^{m} f_i^k$. We say that an element is "heavy" if it appears more times than a constant fraction of some $L_p$ norm of the stream, where $L_p = (\sum_i f_i^p)^{1/p}$ for $p > 1$. In this paper, we consider the following heavy hitter problem.

**Problem 1.** *Given a stream $D$ of $n$ elements, the $\epsilon$-approximate $(\phi, L_p)$-heavy hitter problem is to find a set of elements $T$:*

- *$\forall i \in [m]$, $f_i > \phi L_p \implies i \in T$.*
- *$\forall i \in [m]$, $f_i < (\phi - \epsilon) L_p \implies i \notin T$.*

We allow the heavy hitter algorithms to use randomness; the requirement is that the correct answer should be returned with high probability. The heavy hitter problem is equivalent to the problem of approximately finding the $k$ most frequent elements. Indeed, the top $k$ most frequent elements are in the set of $(\phi, L_1)$-heavy hitters in the stream, where $\phi = \Theta(1/k)$. There is a $\Omega(1/\epsilon^2)$ trade-off between the approximation error $\epsilon$ and the memory usage. Heavy hitter algorithms are building blocks of many data stream algorithms ([18], [19]).

We treat the cosmological simulation data from [4] as a data stream. To do so, we apply an online transformation that we describe in the next section.

*3) Data Transformation:* In a cosmological simulation, dark matter particles form structures through gravitational clustering in a large box with periodic boundary conditions representing a patch of the simulated universe. The box we use [4] is of size 500 Mpc$/h$, or about 2 billion light-years. The simulation data consists of positions and velocities of $256^3$, $512^3$ or $1024^3$ particles, each representing a huge number of physical dark-matter particles. They are distributed rather uniformly on large scales ($\gtrsim 50$ Mpc$/h$) in the simulation box, clumping together on smaller scales. A halo is a clump of particles that are gravitationally bound.

To apply the streaming algorithms, we transform the data. We discretize the spatial coordinates so that we will have a finite number of types in our transformed data stream. We partition the simulation box into a grid of cubic cells, and bin the particles into them. The cell size is chosen to be 1 Mpc$/h$ as to match a typical size of a large halo; there are thus $500^3$ cells. This parameter can be modified in practical applications, but it relates to the space and time efficiency of the algorithm. We summarize the data transformation steps as follows.

- Partition the simulation box into grids of cubic cells. Assign each cell a unique integer ID.
- After reading a particle, determine its cell. Insert that cell ID into the data stream.

Using the above transformation, streaming algorithms can process the particles in the same way as an integer data stream.

*4) Heavy Hitter and Dense Cells:* For a heavy-hitter algorithm to save memory and time, the distribution of cell counts must be very non-uniform. The simulations begin with an almost uniform lattice of particles, but after gravity clusters them together, the density distribution in cells can be modeled by a lognormal PDF ([20], [21]):

$$P_{LN}^{(1)}(\delta) = \frac{1}{(2\pi\sigma_1^2)^{1/2}} \exp\left\{-\frac{[\ln(1+\delta)+\sigma_1^2/2]^2}{2\sigma_1^2}\right\} \frac{1}{1+\delta}, \tag{1}$$

where $\delta = \rho/\bar{\rho} - 1$ is the overdensity, $\sigma_1^2(R) = \ln[1 + \sigma_{\rm nl}^2(R)]$, and $\sigma_{\rm nl}^2(R)$ is the variance of the nonlinear density field in spheres of radius $R$. Our cells are cubic, not spherical; for theoretical estimates, we use a spherical top-hat of the same volume as a cell.

Let $N$ be the number of cells, and $P_c$ be the distribution of the number of particles per cell. The $L_p$ heaviness $\phi_p$ can be estimated as

$$\phi_p \approx \frac{P_{200}}{(N\langle {P_c}^p\rangle)^{1/p}}, \tag{2}$$

where $P_{200}$ is the number of particles in a cell with density exactly $200\bar{\rho}$. This density threshold is a typical minimum density of a halo, coming from the spherical-collapse model. We theoretically estimated $\sigma_{\rm nl}$ for the cells in our density field by integrating the nonlinear power spectrum (using the fit of [22], and the cosmological parameters of the simulation) with a spherical tophat window. The grid size in our algorithm is roughly 1.0 Mpc ($500^3$ cells in total), giving $\sigma_{\rm nl}(\text{Cell}) \approx 10.75$. We estimated $\phi_1 \approx 10^{-6}$ and $\phi_2 \approx 10^{-3}$, matching order-of-magnitude with the measurement of the actual density variance from the simulation cells. These heaviness values are low enough to presume that a heavy-hitter algorithm will efficiently find cells corresponding to haloes.

### *B. Streaming Algorithms for Heavy Hitter Problem*

The above relation between the halo-finding problem and the heavy hitter problem encourages us to apply efficient streaming algorithms to build a new halo finder. Our halo finder takes a stream of particles, performs the data transformation described in section II-A3 and then applies a heavy hitter algorithm to output the approximate top $k$ heavy hitters in the transformed stream. These heavy hitters correspond to the densest cells in the simulation data as described in section II-A4. In our first version of the halo finder, we use Count-Sketch algorithm [16] and Pick-and-Drop Sampling [17].

*1) The Count-Sketch Algorithm:* For a more generalized description of the algorithm, please refer to [16]. For completeness, we summarize the algorithm as follows. The Count-Sketch algorithm uses a compact data structure to maintain the approximate counts of the top $k$ most frequent elements in a stream. This data structure is an $r \times t$ matrix $M$ representing estimated counts for all elements. These counts are calculated by two sets of hash functions: let $h_1, h_2, \ldots, h_r$ be $r$ hash functions, mapping the input items to $\{1, \ldots, t\}$, where each $h_i$ is sampled uniformly from the hash function set $H$. Let $s_1, s_2, \ldots, s_r$ be hash functions, mapping the input items to $\{+1, -1\}$, uniformly sampled from another hash function set $S$. We can interpret this matrix as an array of $r$ hash tables, each containing $t$ buckets.

There are two operations on the Count-Sketch data structure. Denote $M_{i,j}$ as the $j^{th}$ bucket in the $i^{th}$ hash table:

- $Add(M, p)$: For $i \in [1, r]$, $M_{i,h_i[p]} += s_i[p]$.
- $Estimate(M, p)$, return $median_i\{h_i[p] \cdot s_i[p]\}$

The $Add$ operation updates the approximate frequency for each incoming element and the $Estimate$ operation outputs the current approximate frequency. To maintain and store the estimated $k$ most frequent elements, CountSketch also needs a priority queue data structure. The pseudocode of Count-Sketch algorithm is presented in Figure 1. More details and theoretical guarantees are presented in [16].

*2) The Pick-and-Drop Sampling Algorithm:* Pick-and-Drop Sampling is a sampling-based streaming algorithm to approximate the heavy hitters. To describe the idea of Pick-and-Drop sampling, we view the data stream as a sequence of $r$ blocks of size $t$. Define $d_{i,j}$ as the $j^{th}$ element in the $i^{th}$ block and $d_{i,j} = p_{k(i-1)+j}$ in stream $D$. In each block of the stream, Pick-and-Drop sampling will pick one random sample and record its remaining frequency in the block. The algorithm maintains a sample with the largest current counter

```
procedure CountSketch($r, t, k, D$)  ▷ $D$ is a stream
    Initialize an empty $r \times t$ matrix $M$.
    Initialize an min-priority queue $Q$ of size $k$
    (particle with smallest count is on the top).
    for $i = 1, \dots, n$ and $p_i \in D$ do
        $Add(M, p_i)$;
        if $p_i \in Q$ then
            $P_i.count$++;
        else if $Estimate(M, p_i) > Q.top().count$ then
            $Q.pop()$;
            $Q.push(p_i)$;
        end if
    end for
    return $Q$
end procedure
```

Figure 1: Count-Sketch Algorithm

and drops previous samples. The pseudocode of Pick-and-Drop sampling [17] is given in Figure 2 and we need the following definitions in Figure 2. For $i \in [r]$, $j, s \in [t]$, $q \in [m]$ define:

$$f_{i,q} = |\{j \in [t] : d_{i,j} = q\}|, \tag{3}$$

$$a_{i,s} = |\{j^* : s \leq j^* \leq t, d_{i,j^*} = d_{i,s}\}|. \tag{4}$$

```
procedure PickDrop($r, t, \lambda, D$)
    Sample $S_1$ uniformly at random on $[t]$.
    $L_1 \leftarrow d_{1,S_1}$,
    $C_1 \leftarrow a_{1,S_1}$,
    $u_1 \leftarrow 1$.
    for $i = 2, \dots, r$ do
        Sample $S_i$ uniformly at random on $[t]$.
        $l_i \leftarrow d_{i,S_i}$, $c_i \leftarrow a_{i,S_i}$
        if $C_{i-1} < \max(c_i, \lambda u_{i-1})$ then
            $L_i \leftarrow l_i$,
            $C_i \leftarrow c_i$,
            $u_i \leftarrow 1$
        else
            $L_i \leftarrow L_{i-1}$,
            $C_i \leftarrow C_{i-1} + f_{i,L_{i-1}}$,
            $u_i \leftarrow q_{i-1} + 1$
        end if
    end for
    return $\{L_r, C_r\}$
end procedure
```

Figure 2: Pick-and-Drop Algorithm

The detail implementation is in Section III-B.

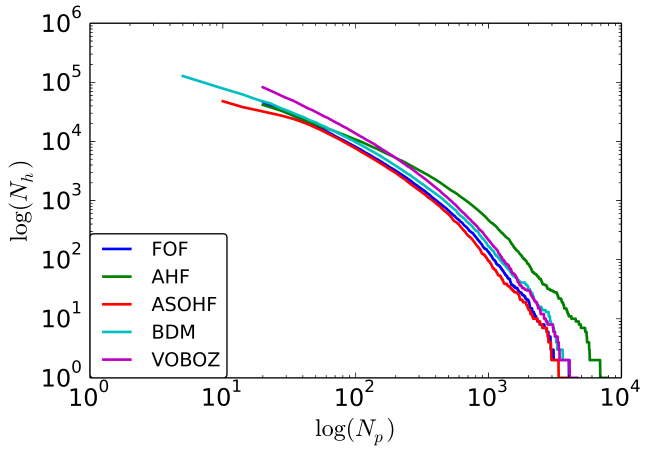


Figure 3: Halo mass distribution of various halo finders.

## III. Implementation

### A. Simulation Data

The $N$-body simulation data we use as the input to our halo finder was used in the halo-finding comparison project [4] and consists of various resolutions (numbers of particles) of the MareNostrum Universe cosmological simulation [23]. These simulations ran in a 500 $h^{-1}$Mpc box, assuming a standard $\Lambda$CDM (cold dark matter and cosmological constant) cosmological model.

In the first implementation of our halo finder, we consider two halo properties: center position and mass (the number of particles in it). We compare to the the fiducial offline algorithm FoF. The distributions of halo sizes from different halo finders are presented in Fig. 3.

Since our halo finder builds on the streaming algorithms of finding frequent items, the algorithms need to transform the data as described in section II-A3 — dividing all the particles into different small cells and label each particle with its associated cell ID. For example, if an input dataset contains three particles $p_1, p_2, p_3$ and they are all included in a cell of ID $= 1$, then the transformed data stream becomes $1, 1, 1$. The most frequent element in the stream is obviously $1$ and thus the cell $1$ is the heaviest cell overall.

### B. Implementation Details

Our halo finder implementation is written using C++ with GNU GCC compiler 4.9.2. We implemented Count-Sketch and Pick-and-Drop sampling as two algorithms to find heavy hitters.

*1) Count-Sketch-based Halo Finder:* There are three basic steps in the Count-Sketch algorithm, which returns the heavy cells and the number of particles associated with them.

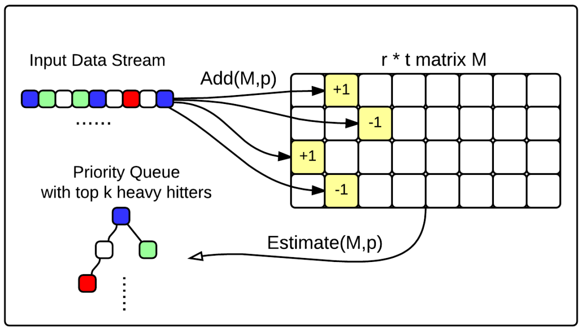


Figure 4: Count-Sketch Algorithm

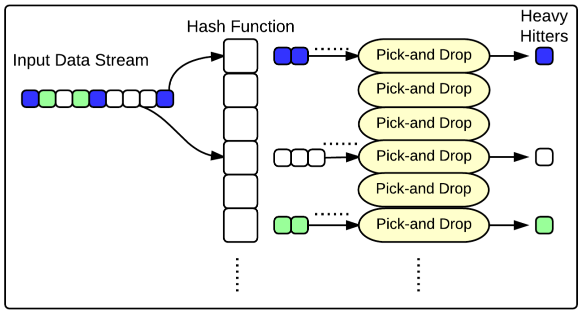


Figure 5: Pick-and-Drop Sampling

(1) Allocate memory for the CountSketch data structure to hold current estimates of cell frequencies; (2) use a priority queue to record the $k$ most frequent elements; (3) return the positions of the top $k$ heavy cells. Figure 4 presents the process of the Count-Sketch.

The Count-Sketch data structure is an $r \times t$ matrix. Following [24], we set $r = \log(\frac{n}{\epsilon})$ and $t$ to be sufficiently large (>1 million) to achieve an expected approximation error $\epsilon = 0.05$. We build the matrix as a 2D array with $r \times t$ 0's. For each incoming element in the stream, an $Add$ operation has to be executed and an $estimate$ operation needs to be executed only when this element is not in the queue.

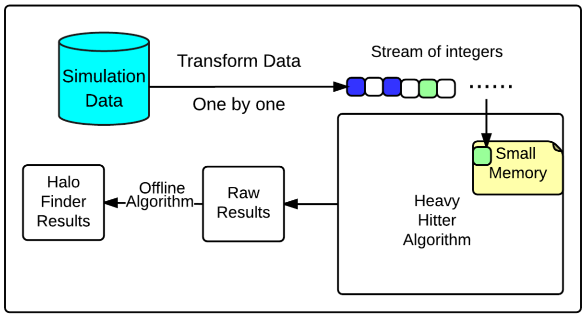


Figure 6: Halo Finder Procedure

*2) Pick-and-Drop-based Halo Finder:* In the Pick-and-Drop sampling based halo finder, we implement a general hash function $H$: $\mathbb{N}^+ \rightarrow \{1, 2, \ldots, ck\}$, where $c \geq 1$, to gain the probability of success to approximate the $k$ heaviest cells. We apply the hash function $H$ on every incoming element and put the elements with the same hash value together such that the original stream is divided into $ck$ smaller sub-streams. Meanwhile, we initialize $ck$ instances of Pick-and-Drop sampling so that each PD instance will process one sub-stream. The whole process of approximating the heavy hitters is presented in Figure 5. In this way, the repeated items in the whole stream will be distributed into the same sub-stream and they are much heavier in this sub-stream. With high probability, each instance of Pick-and-Drop sampling will output the heaviest one in each of the sub-streams, and in total we will have $ck$ output items. Because of the randomness in the sampling method, we will expect some of inaccurate heavy hitters among the total $ck$ outputs. By setting a large $c$, most of the actual top $k$ most frequent elements should be inside the $ck$ outputs (raw result).

To get precise properties of haloes, such as the center, and mass, an offline algorithm such as FoF [2] can be applied to the particles inside the returned heavy cells and their neighbor cells. This needs an additional pass over the data but we only need to store a small amount of particles to run those offline in-memory algorithms. The whole process of the halo finder is represented in Figure 6, where heavy hitter algorithms can be regarded as a black box. That is, any theoretically efficient heavy hitter algorithms could be applied to further improve the memory usage and practical performance.

## C. Shifting Method

In the first pass of our halo finder, we only use the position of a heavy cell as the position of a halo. However, each heavy cell may contain several haloes and some of the haloes located on the edges between two cells cannot be recognized because the cell size in the data transformation step is fixed. To recover those missing haloes, we utilize a simple shifting method:

- Initialize $2^d$ instances of Count-Sketch or Pick-and-Drop in parallel, where $d$ is the dimension. Our simulation data reside in three dimensions, so $d = 3$.
- Move all the particles to one of the $2^d$ directions with a distance of 0.5 Mpc$/h$ (half of the cell size). In each of the $2^d$ shifting processes, assign a Count-

Sketch/Pick-and-Drop instance to run. By combining the results from $2^d$ shifting processes, we expect that the majority of $k$ largest haloes are discovered. All the parallel instances of the CountSketch/Pick-and-Drop are enabled by OpenMP 4.0 in C++.

## IV. Evaluation

To evaluate how well streaming based halo finders work, we mainly focus on testing it in the following three aspects:

- **Correctness:** Evaluate how close are the positions of $k$ largest haloes found by the streaming-based algorithms to the top $k$ large haloes returned by some widely used in-memory algorithms. Evaluate the trade-off between the selection $k$ and the quality of result.
- **Stability:** Since streaming algorithms always require some randomness and may produce some incorrect results, we want to see how stable are streaming based heavy hitter algorithms are.
- **Memory Usage:** Linear memory space requirement is a "bottle neck" for all offline algorithms, and it is the central problem that we are trying to overcome by applying streaming approach. Thus it is significantly important to theoretically or experimentally estimate the memory usage of Pick-and-Drop and Cound-sketch algorithms.

In the evaluation, all the in-memory algorithms we choose to compare were proposed in the Halo-Finding Comparison Project [4]. We test against the fiducial FOF method, as well as four others that find density peak:

1) **FOF** by Davis et al.[2]
"Plain-vanilla" Friends-of-Friends.
2) **AHF** by Knollmann & Knebe [25]
Density peaks search with recursively refined grid
3) **ASOHF** by Planelles & Quilis. [26]
Finds spherical-overdensity peaks using adaptive density refinement.
4) **BDM** [27], run by Klypin & Ceverino "Bound Density Maxima" – finds gravitationally-bound spherical-overdensity peaks.
5) **VOBOZ** by Neyrinck et al [28]
"Voronoi BOund Zones" – finds gravitationally bound peaks using a Voronoi tessellation.

### A. Correctness

As there is no agreed upon rule how to define the center and the boundary of a halo, it is impossible to theoretically define and deterministically verify the correctness of any halo finder. Therefore a comparison to the results of previous widely accepted halo finders seems to be the best practical verification of a new halo finder. To compare the outputs of two different halo finders we need to introduce some formal measure of similarity. The most straight forward way to compare them is to consider one of them $H$ as a ground truth, and another one $E$ as an estimator. Among this the FOF algorithm is considered to be the oldest and the most widely used, thus in our initial evaluation we decided to concentrate on the comparison with FOF. Then the most natural measure of similarity is number of elements in $H$ that match to elements in output of $E$. More formally we will define "matches" as: for a given $\theta$ we will say that center $e_i \in E$ matches the element $h_i \in E$ if $dist(e_i, h_i) \leq \theta$, where $dist(\cdot,\cdot)$ is Euclidean distance. Then our measure of similarity is:

$$Q(\theta) = Q(E_k, H_k, \theta) = |\{h_i \in H_k : \min_{e_j \in E_k} dist(h_i, e_j) < \theta\}|,$$

where $k$ represents $k$ heaviest halos.

We compare the output of both streaming-based halo finders to the output of in-memory halo finders. We made comparisons for the $256^3, 512^3$ and $1024^3$-particle simulations, finding the top 1000 and top 10000 heavy hitters. Since the comparison results in all cases were similar, the figures presented below are for the $256^3$ dataset, and $k = 1000$.

On the Figure 7a we show for each in-memory algorithm the percentage of centers that were not found by streaming-based halo finder. We can see that both the Count-Sketch and Pick-and-Drop algorithms missed not much more than 10 percent of the haloes in any of the results from the in-memory algorithms.

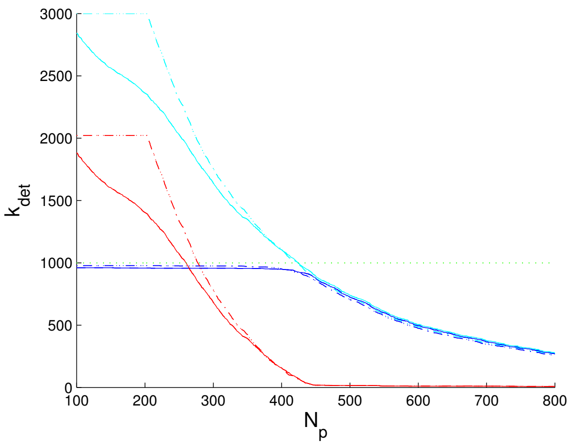
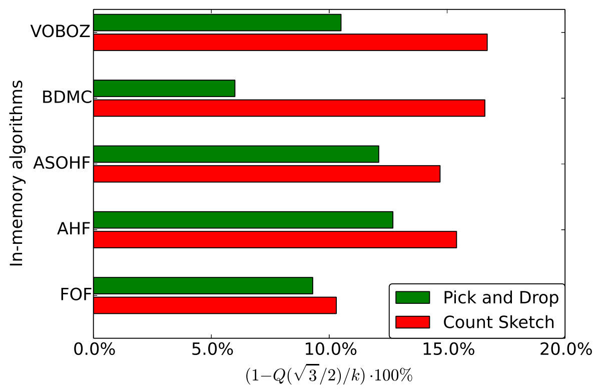
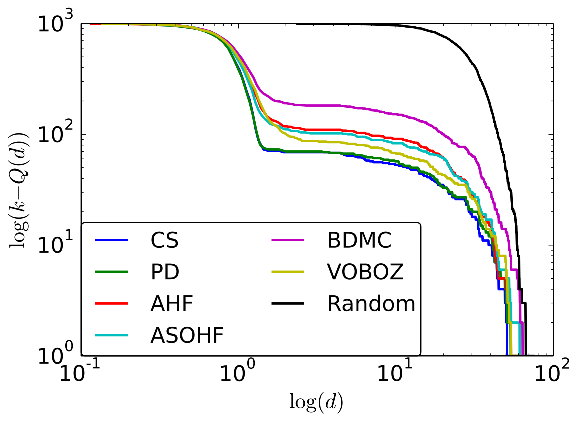


Figure 8: Number of detected halos by our two algorithms. The solid lines correspond to (CS) and the dashed lines to (PD). The dotted line at $k = 1000$ shows our selection criteria. The $x$ axis is the threshold in the number of particles allocated to the heavy hitter. The cyan color denotes the total number of detections, the blue curves are the true positives (TP), and the red curves are ethe false positives (FP).

To understand whether the 10 percent means two halo catalogs are close to each other or not, we will choose one of the in-memory algorithms as a ground truth and compare how close the other in-memory algorithms are. Again, we



(a)



(b)

Figure 7: (a) Measures of the disagreement between PD and CS, and various in-memory algorithms. The percentage shown is the fraction of haloes farther than a half-cell diagonal ($0.5\sqrt{3}$ Mpc/$h$) from PD or CS halo positions. (b) The number of top-1000 FoF haloes farther than a distance $d$ away from any top-1000 halo from the algorithm of each curve.

choose FOF algorithm as a ground truth. The comparison is depicted in Fig. 7b. From this graph you can see that the outputs of Count-Sketch and Pick-and-Drop based halo finders are closer to the FOF haloes, than other in-memory algorithms. It can be easily explained, as after finding heavy cells we apply the same FOF to these heavy cells and their neighborhoods, the output should always have similar structure to the output of in-memory FOF on the full dataset. Also from this graph you will see that each line can be represented as a mix of two components, one of which is the component of random distribution. It means that after a distance of $\sqrt{3}/2$ all matches are the same if we just put bunch of points at random.

The classifier is using a top-$k$ to select the halo candidates. Figure 8 shows how sensitive the results are to the selection threshold of $k = 1000$. It shows several curves, including the total number of heavy hitters, the ones close to an FoF group – we can call these true positive (TP) – and the ones detected, but not near an FoF object (false positives FP). From the figure, it is clear that the threshold of 1000 is close to the optimal detection threshold, preserving TP and minimizing FP. This corresponds to a true positive detection rate (TPR) of 96% and a false positive detection rate of 3.6%. If we lowered our threshold to $k = 900$, our TPR drops to 91% but the FPR becomes even lower, 0.88%.

These tradeoffs can be shown on a so-called ROC-curve (receiver operating characteristic), where the TPR is plotted against the FPR. This shows how lowering the detection threshold increases the true detections, but the false detection rate increases much faster. Using the ROC curve, shown below we can see the position of the $k = 1000$ threshold as a circle and the $k = 900$ as a square.

Finally, we should also ask, besides the set comparison, how do the individual particle cardinalities counted around

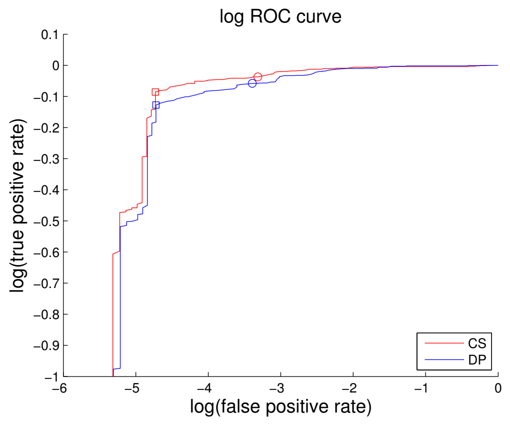


Figure 9: This ROC curve shows the tradeoff between true and false detections as a function of threshold. The figure plots TPR vs FPR on a log-log scale. The two thresholds are shown with symbols, the circle denotes 1000, and the square is 900.

the heavy hitters correspond to the FoF ones. Our particle counting is restricted to neighboring cells, while the FoF is not, so we will always be undercounting. To be less sensitive to such biases, we compare the rank ordering of the two particle counts in the two samples in Fig. 10. The rank 1 is assigned to the most massive objects in each set.

## B. Stability

As most of the streaming algorithms utilize randomness, we estimate how stable our results are compared to the results from a deterministic search. In the deterministic search algorithm, we find the actual heavy cells by counting the number of particles inside them; we perform the comparison

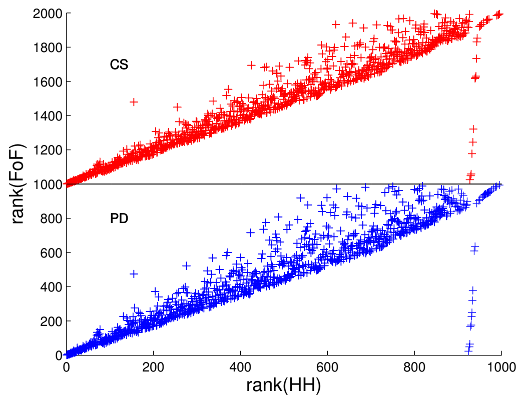


Figure 10: The top 1000 heavy hitters are rank-ordered by the number of their particles. We also computed a rank of the corresponding FoF halo. The linked pairs of ranks are plotted. One can see that if we adopted a cut at $k = 900$, it would eliminate a lot of the false positives.

for the dataset containing $256^3$ particles. To perform this evaluation we run 50 instances of each algorithm (denoting the outputs as $\{C^i_{cs}\}^{50}_{i=1}$ and $\{C^i_{pd}\}^{50}_{i=1}$). We also count the number of cells of each result that match the densest cells returned by the deterministic search algorithm $C_{ds}$. The normalized number of matches will be $\rho^i_{pd} = \frac{|C^i_{pd} \cap C_{ds}|}{|C_{ds}|}$ and $\rho^i_{cs} = \frac{|C^i_{cs} \cap C_{ds}|}{|C_{ds}|}$ correspondingly. Our experiment showed:

$$\mu(\rho^i_{cs}) = 0.946, \ \sigma(\rho^i_{cs}) = 2.7 \cdot 10^{-7}$$

$$\mu(\rho^i_{pd}) = 0.995, \ \sigma(\rho^i_{pd}) = 6 \cdot 10^{-7}$$

This means that the approximation error caused by randomness is very small compared with the error caused by transition from overdense cells to halo centers. This fact can also be caught from the Fig. 11. On that figure you can see that shaded area below and above the red line and green line, which represents the range of outputs among 50 instances, is very thin. Thus the output is very stable.

### C. Memory Usage

Comparing with current halo finding solutions, streaming approachs' low memory usage is one of the most significant advantages. To the best of our knowledge even for the problem of locating 1000 largest haloes in the simulation data with $1024^3$ particles, there is no way to run other halo finding algorithms on a regular PC since $1024^3$ particles already need $\approx$ 12GB memory to only store all the particle coordinates; a computing cluster or even supercomputer is necessary. Therefore, the application of streaming techniques introduces a new direction on the development of halo-finding algorithms.

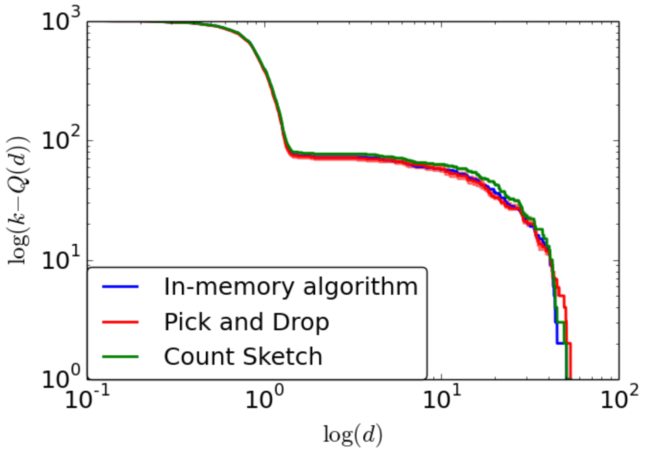


Figure 11: Each line on the graph represents the top 1000 halo centers found with Pick-and-Drop sampling, Count-Sketch, and in-memory algorithms, as described in section III-B. The comparison with FOF is shown in Fig.7b. The shaded area (too small to be visible) shows the variation due to randomness.

To find top $k$ heavy cells, Count-Sketch theoretically requires following amount of space:

$$O(k \log \frac{n}{\delta} + \frac{\sum_{q'=k+1}^{m} f^2_{q'}}{(\epsilon f_k)^2} \log \frac{n}{\delta}),$$

where $1 - \delta$ is probability of success, $\epsilon$ is an $Q_k$ estimation error, and $Q_k$ is the frequency of $k$-th heaviest cell. It is worth mentioning that in application to the heavy cell searching problem the second term is the dominating one. The first factor in the second term represents the linear dependency of memory usage on the heaviness of top $k$ cells. Thus we can expect linear memory usage for small dataset. But as dataset grows the dependency becomes logarithmic if we assume the same level of heaviness. Experiments verify this observation, as for small dataset with $256^3$ particles Count-Sketch algorithm used around 900 megabytes memory, while for the large $1024^3$-dataset, the memory usage was increased to nearly 1000 megabytes. Thus the memory grows logarithmically with the size of dataset if we assume almost constant heaviness of the top $k$ cells; that is why such approach is scalable for even larger datasets.

In the experiments using this particular simulation data, Pick-and-Drop sampling shows much better performance in terms of memory usage than Count-Sketch. The actual usage of memory was around 20 megabytes for the dataset with $256^3$ particles and around 30 megabytes for the dataset with $1024^3$ particles.

## V. Conclusion

In this paper we find a novel connection between the problem of finding the most massive halos in cosmological N-Body simulations and the problem of finding heavy hitters in data streams. According to this link, we have built a halo finder based on the implementation of Count-Sketch algorithm and Pick-and-Drop sampling. The halo finder successfully locates most ($> 90\%$) of the $k$ largest haloes using sub-linear memory. Most halo-finders require the entire simulation to be loaded into memory. But our halo finder does not and could be run on the massive $N$-body simulations that are anticipated to arrive in the near future with relatively modest computing resources. We will continue to improve the performance of our halo finder, something we have as yet not paid much attention to. In the very first implementation we evaluated here, we mainly focus on the verification of precision instead of performance. But both Count-Sketch and Pick-and-Drop sampling can be easily parallelized further to achieve significantly better performance. The majority of the computation on Count-Sketch is spent on the calculations of $r \times t$ hash functions. A straight forward way to improve the performance is taking advantage of the highly parallel GPU streaming processors to improve the performance of calculating a large number of hash functions. Similarly, Pick-and-Drop sampling is also a good candidate for more parallelism since the Pick-and-Drop instances are running independently.

We also note that this halo finder finds only the $k$ most massive haloes. These are features of interest in the simulation, but some further work is required for our methods to return a complete set of haloes as an in-memory algorithm.

Future work:

1) Optimize the current methods using Count-Sketch and Pick-and-Drop sampling. Our goal is to provide a halo finder tool that can be running on personal PCs or even laptops, and provide comparatively accurate results in a reasonable running time.
2) An application of interest to cosmologists would be to run a streaming algorithm similar to this that includes velocity information; this is important in distinguishing small "subhaloes" from FoF-type haloes. Including additional attributes/dimensions in our algorithms clustering is quite easy, and will be investigated in the near future.

## References

[1] A. Knebe, F. R. Pearce, H. Lux, Y. Ascasibar, P. Behroozi, J. Casado, C. C. Moran, J. Diemand, and K. Dolag, "Structure finding in cosmological simulations: the state of affairs," *MNRAS*, vol. 435, pp. 1618–1658, Oct. 2013.

[2] 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," *ApJ*, vol. 292, pp. 371–394, May 1985.

[3] B. L. Falck, M. C. Neyrinck, and A. S. Szalay, "ORIGAMI: Delineating Halos Using Phase-space Folds," *ApJ*, vol. 754, p. 126, Aug. 2012.

[4] 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, and K. Dolag, "Haloes gone MAD: The Halo-Finder Comparison Project," *MNRAS*, vol. 415, pp. 2293–2318, Aug. 2011.

[5] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 20–29. [Online]. Available: http://doi.acm.org/10.1145/237814.237823

[6] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '04. New York, NY, USA: ACM, 2004, pp. 101–114. [Online]. Available: http://doi.acm.org/10.1145/1028788.1028802

[7] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '06/Performance '06. New York, NY, USA: ACM, 2006, pp. 145–156. [Online]. Available: http://doi.acm.org/10.1145/1140277.1140295

[8] H. C. Zhao, A. Lall, M. Ogihara, O. Spatscheck, J. Wang, and J. Xu, "A data streaming algorithm for estimating entropies of od flows," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 279–290. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298345

[9] J. Beringer and E. Hüllermeier, "Efficient instance-based learning on data streams," *Intell. Data Anal.*, vol. 11, no. 6, pp. 627–650, Dec. 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1368018.1368022

[10] E. Liberty, "Simple and deterministic matrix sketching," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 581–588.

[11] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous trend-based clustering in data streams," in *Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery*, ser. DaWaK '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 251–262.

[12] L. Serir, E. Ramasso, and N. Zerhouni, "Evidential evolving gustafson-kessel algorithm for online data streams partitioning using belief function theory." *Int. J. Approx. Reasoning*, vol. 53, no. 5, pp. 747–768, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijar/ijar53.html

[13] B. Ball, M. Flood, H. Jagadish, J. Langsam, L. Raschid, and P. Wiriyathammabhum, "A flexible and extensible contract aggregation framework (caf) for financial data stream analytics," in *Proceedings of the International Workshop on Data Science for Macro-Modeling*. ACM, 2014, pp. 1–6.

[14] F. Rusu and A. Dobra, "Statistical analysis of sketch estimators," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07. New York, NY, USA: ACM, 2007, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/1247480.1247503

[15] J. Spiegel and N. Polyzotis, "Graph-based synopses for relational selectivity estimation," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 205–216. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142497

[16] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ser. ICALP '02. London, UK, UK: Springer-Verlag, 2002, pp. 693–703. [Online]. Available: http://dl.acm.org/citation.cfm?id=646255.684566

[17] V. Braverman and R. Ostrovsky, "Approximating large frequency moments with pick-and-drop sampling," *CoRR*, vol. abs/1212.0202, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1212.html

[18] V. Braverman, J. Katzman, C. Seidell, and G. Vorsanger, "An Optimal Algorithm for Large Frequency Moments Using $O(n^{1-2/k})$ Bits," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), K. Jansen, J. D. P. Rolim, N. R. Devanur, and C. Moore, Eds., vol. 28. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 531–544. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2014/4721

[19] P. Indyk and D. Woodruff, "Optimal approximations of the frequency moments of data streams," in *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05. New York, NY, USA: ACM, 2005, pp. 202–208. [Online]. Available: http://doi.acm.org/10.1145/1060590.1060621

[20] P. Coles and B. Jones, "A lognormal model for the cosmological mass distribution," *MNRAS*, vol. 248, pp. 1–13, Jan. 1991.

[21] I. Kayo, A. Taruya, and Y. Suto, "Probability distribution function of cosmological density fluctuations from a gaussian initial condition: Comparison of one-point and two-point lognormal model predictions with n-body simulations," *The Astrophysical Journal*, vol. 561, no. 1, p. 22, 2001. [Online]. Available: http://stacks.iop.org/0004-637X/561/i=1/a=22

[22] R. E. Smith, J. A. Peacock, A. Jenkins, S. D. M. White, C. S. Frenk, F. R. Pearce, P. A. Thomas, G. Efstathiou, and H. M. P. Couchman, "Stable clustering, the halo model and non-linear cosmological power spectra," *MNRAS*, vol. 341, pp. 1311–1332, Jun. 2003.

[23] S. Gottlöber and G. Yepes, "Shape, Spin, and Baryon Fraction of Clusters in the MareNostrum Universe," *ApJ*, vol. 664, pp. 117–122, Jul. 2007.

[24] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1530–1541, Aug. 2008. [Online]. Available: http://dx.doi.org/10.14778/1454159.1454225

[25] S. R. Knollmann and A. Knebe, "Ahf: Amiga's halo finder," *The Astrophysical Journal Supplement Series*, vol. 182, no. 2, p. 608, 2009.

[26] S. Planelles and V. Quilis, "Asohf: a new adaptive spherical overdensity halo finder," *Astronomy & Astrophysics*, vol. 519, p. A94, 2010.

[27] A. Klypin and J. Holtzman, "Particle-Mesh code for cosmological simulations," *ArXiv Astrophysics e-prints*, Dec. 1997.

[28] M. C. Neyrinck, N. Y. Gnedin, and A. J. S. Hamilton, "VOBOZ: an almost-parameter-free halo-finding algorithm," *MNRAS*, vol. 356, pp. 1222–1232, Feb. 2005.