A Parallel GPU-Based Approach to Clustering Very Fast Data Streams

Pengtao Huang, Xiu Li and Bo Yuan[*]
Intelligent Computing Lab, Division of Informatics
Graduate School at Shenzhen, Tsinghua University
Shenzhen 518055, P.R. China
hpt13@mails.tsinghua.edu.cn, li.xiu@sz.tsinghua.edu.cn, yuanb@sz.tsinghua.edu.cn

ABSTRACT

Clustering data streams has become a hot topic in the era of big data. Driven by the ever increasing volume, velocity and variety of data, more efficient algorithms for clustering large-scale complex data streams are needed. In this paper, we present a parallel algorithm called PaStream, which is based on advanced Graphics Processing Unit (GPU) and follows the online-offline framework of CluStream. Our approach can achieve hundreds of times speedup on high-speed and high-dimensional data streams compared with CluStream. It can also discover clusters with arbitrary shapes and handle outliers properly. The efficiency and scalability of PaStream are demonstrated through comprehensive experiments on synthetic and standard benchmark datasets with various problem factors.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering; I.5.3 [**Clustering**]: Algorithms

General Terms

Algorithms, Experimentation

Keywords

PaStream, data stream, clustering, GPU

1. INTRODUCTION

Data stream is a special form of data,which is very common in the real world. Unlike traditional static data stored in databases or data warehouses, data stream is a dynamic, continuous, massive, unbounded and rapid sequence of data. Examples of data stream include multimedia data, trajectory data, web click data and financial data[4, 23].

[*]Corresponding author.

CIKM'15, October 19–23, 2015, Melbourne, VIC, Australia
© 2015 ACM. ISBN 978-1-4503-3794-6/15/10 ...$15.00
DOI: http://dx.doi.org/10.1145/2806416.2806545.

Due to the characteristics of data stream, it is infeasible to control the order in which elements arrive, nor is it possible to store all elements of a data stream. Consequently, each new element in the data stream should be processed instantly when it arrives and there is no enough time to examine it more than once. Furthermore, the memory requirement should be confined finitely, although new data elements arrive continuously [16, 17].

Clustering is a popular technique of data mining, which aims at discovering groups of similar data elements measured by a given set of criteria [21]. Most traditional clustering algorithms do not apply to data streams because of the one pass constraint and storage limitation. Barbará [6] summarized three basic requirements for clustering data streams: compactness of representation, fast and incremental processing of new data points, clear and fast identification of outliers. Aggarwal et al. [2] further claimed that the quality of clusters should be assured when the data evolves considerably over time and the capability of discovering clusters over different portions of stream should also be maintained. A number of stream clustering algorithms have been proposed, which more or less meet the above requirements. However, in recent years, people are faced with some new challenges in data stream mining.

With the rise of big data, the volume and speed of data streams are far beyond the capacity of most algorithms running on traditional computing architecture and infrastructure. For example, the vast amount of security monitoring data in a city needs to be analyzed in real time so that we can timely handle emergencies. So it is increasingly urgent to explore new efficient data stream mining algorithms based on more powerful hardware. The emerging heterogeneous platforms such as GPU, Hadoop, FPGA and DSP bring us powerful tools to cope with massive and complex big data. Among them, GPU is a lightweight, efficient and promising platform for both personal supercomputing and large scale supercomputing.

In this paper, we present PaStream, a parallel algorithm framework for clustering very high speed and high dimensional data streams with arbitrary cluster shapes based on NVIDIA GPUs. PaStream follows the basic online-offline framework of the well-known CluStream [2]. For the online part, we propose a parallel mechanism to fully exploit the computing power of GPUs, making PaStream hundreds of times faster than CluStream on maintaining micro-clusters. For the offline part, we extend one of the latest clustering algorithms [25] so that it can be used to cluster micro-clusters and find macro-clusters of arbitrary shapes and handle outliers properly. Furthermore, it is accelerated using GPUs and can achieve dozens of times speedup. In summary, the key contributions in PaStream include a new parallel approach to online micro-cluster maintenance and a more powerful method for offline macro-cluster creation.

The remainder of this paper is organized as follows. In Section 2, we review existing work on data stream clustering, and analyse their strengths and weaknesses. We also give an introduction of the CUDA programming model based on NVIDIA GPUs. An overall description of PaStream is given in Section 3. We present the details of the GPU-based algorithm for online micro-clustering in Section 4 and a powerful new approach to offline macro-clustering in Section 5. Section 6 reports the performance of our algorithm and compares it with CluStream. The conclusion and future work of this paper are presented in Section 7.

2. RELATED WORK

2.1 Data Stream Clustering

Zhang et al. [30, 31] proposed BIRCH to efficiently cluster data in very large databases. The algorithm builds a CF Tree to incrementally and dynamically cluster incoming multi-dimensional metric data points. It can typically find a good clustering pattern with a single scan of the data and refine the quality further with a few additional scans.

Guha et al. [19, 18] investigated data stream clustering using K-median and proposed constant-factor approximation algorithms for the K-median problem in the data stream model with a single pass. Charikar et al. [11] gave an improved streaming algorithm for the K-median problem with an arbitrary distance function.

O'Callaghan et al. [24] proposed LOCALSEARCH and STREAM algorithms for high-quality clustering. They are also single-pass algorithms, but they evaluate the performance by a combination of SSQ (Sum of Squared Distance) and the number of centers used. In many applications, they exhibit superior performance compared with algorithms such as BIRCH.

Barbara and Chen [7] presented the Fractal Clustering (FC) algorithm. The algorithm clusters points incrementally and places them in certain clusters. After adding the point, the change in the fractal dimension of the cluster should be the least. It can effectively handle large data sets with high-dimensionality and noise and is capable of recognizing clusters of arbitrary shapes.

Babcock et al. [5] extended the STREAM algorithm with sliding window and solved two important and related problems in the sliding window model: maintaining variance and maintaining the K-median clustering.

Most algorithms above are one-pass algorithms and can address the scalability issue of data stream clustering, but they can hardly cope with fast evolving data streams. Aggarwal et al. [2] proposed the CluStream algorithm, a novel framework for clustering fast evolving data streams. It divides the clustering process into an online phase, which periodically stores detailed summary statistics, and an offline phase, which only relies on these summary statistics. CluStream provides desired functionality in discovering and exploring clusters over different portions of the stream, but it can not discover clusters with arbitrary shapes. Afterwards they proposed HPStream [3], which is a framework for projected clustering of high dimensional data streams.

In order to find clusters of arbitrary shapes in data streams, a number of new algorithms were proposed, such as CDS-Tree [27] based on an improved space partition, DenStream [9] and D-Stream [12] based on density, ACluStream [34] and MGDDS [33] based on grid and FCluStream [20] based on fractal. Most of them followed the online-offline framework of CluStream and incorporated new methods to find clusters of arbitrary shapes.

Dedicated software tools and libraries have also been developed for data stream analysis. For example, Bifet et al. [8] developed a software environment called MOA (Massive Online Analysis)[22] for online learning from evolving data streams. Zhang and Mueller [32] provided a C++ template library for parallel data streaming applications based on the streaming abstraction and GPUs.

2.2 GPU High Performance Computing

In recent years, GPUs have evolved into highly parallel, multi-threaded, many-core processors and are widely used for general purpose computing [14]. Compared with CPU based distributed systems such as Hadoop, GPU based parallel computing systems are more lightweight, portable and energy-efficient.

CUDA (Compute Unified Device Architecture) is a general purpose parallel computing platform and programming model, which was first introduced by NVIDIA in November 2006. CUDA makes it convenient to exploit the parallel capabilities of NVIDIA GPUs to solve computationally intensive problems much more efficiently. CUDA is designed to support various programming languages and interfaces such as CUDA C, OpneCL, Python [1] and FORTRAN.

Threads and kernels are the most important concepts in CUDA. Threads are lightweight processes executed on independent processors in GPU, and they are easy to be created and synchronized. Kernels are functions that are executed on the GPU in parallel by massive threads organized into blocks and grids [13].

There are different types of memory in GPUs, which can significantly affect the performance of programs. Each thread has its private local memory called register, which is the fastest type of memory. Each thread block features shared memory accessible by all threads within a block, which can be as fast as registers if accessed properly. All threads have access to the same global memory, which is the largest and slowest storage and the only memory visible to the CPU. Constant memory and texture memory are two read-only memory spaces accessible by all threads [29]. The global, constant, and texture memory spaces are persistent across kernel launches by the same application.

For devices of compute capability 3.5 or higher, a new functionality of CUDA called dynamic parallelism is available, which enables a CUDA kernel to create new work directly from the GPU without the intervention of CPU, reducing the need to transfer instructions and data between host and device. This feature is very helpful for the parallelization of algorithms and programming patterns that contain recursion, irregular loop structure, or other constructs that do not fit a flat, single-level of parallelism [14].

Cao and Zhou [10] made some attempts to accelerate data stream clustering algorithms using GPUs. However, they only focused on calculating and comparing distances in parallel without making necessary changes to the algorithm flow. Due to the limitations of the algorithm architecture and early GPU architectures, the GPU-based algorithms were only about 7 times faster than the CPU-based algorithms. Meanwhile, Roger et al. [26] presented an efficient algorithm to remove unwanted elements from a stream of outputs on the GPU and Verner et al. [28] proposed an efficient and practical algorithm for processing streams with hard real-time constraints on heterogeneous systems.

3. PRELIMINARIES

Similar to CluStream [2], PaStream consists of an online phase and an offline phase. The online phase groups data points into micro-clusters and the offline phase aggregates micro-clusters into macro-clusters. The schematic diagram of our algorithm is illustrated in Fig. 1.

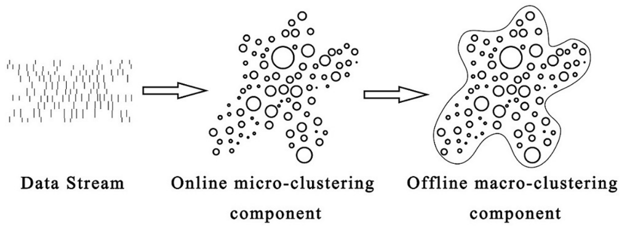
Data Stream
Online micro-clustering component
Offline macro-clustering component

Figure 1: Schematic diagram of PaStream

In our model, each micro-cluster can be regarded as a hyper-sphere containing a set of d-dimensional data points. Using the concept of *clustering feature* (CF) [30], we can represent the i-th micro-cluster containing data points $\overline{X_{i_1}}$, $\overline{X_{i_2}}, \cdots, \overline{X_{i_{n_i}}}$ with time stamps $T_{i_1}, T_{i_2}, \cdots, T_{i_{n_i}}$ by $(\overline{CF2^x_i}, \overline{CF1^x_i}, CF2^t_i, CF1^t_i, n_i)$, where $\overline{CF2^x_i}$ and $\overline{CF1^x_i}$ are d-dimensional vectors and the other three are scalar values.

$\overline{CF2^x_i}$ maintains the sum of squares of the data points, whose j-th entry can be expressed as:

$$CF2^x_{i_j} = \sum_{k=1}^{n_i} (X_{i_{k_j}})^2 \tag{1}$$

$\overline{CF1^x_i}$ maintains the sum of the data points, whose j-th entry can be expressed as:

$$CF1^x_{i_j} = \sum_{k=1}^{n_i} X_{i_{k_j}} \tag{2}$$

$CF2t_i$ maintains the sum of squares of the time stamps, which can be expressed as:

$$CF2^t_i = \sum_{k=1}^{n_i} T^2_{i_k} \tag{3}$$

$CF1t_i$ maintains the sum of the time stamps, which can be expressed as:

$$CF1^t_i = \sum_{k=1}^{n_i} T_{i_k} \tag{4}$$

n_i is the number of data points contained in the i-th micro-cluster. In this way, a micro-cluster is represented by a $(2 * d + 3)$ tuple instead of a $(n_i * d)$ tuple, reducing the memory requirements significantly for a typical $n_i \gg 2$.

From the five attributes above, we can get the *centroid*, *radius* and *relevance stamp* of a micro-cluster. The j-th entry of the centroid of the i-th micro-cluster is:

$$Centroid_{i_j} = \frac{CF1^x_{i_j}}{n_i} \tag{5}$$

Radius reflects the dispersion of the data points in a micro-cluster, which can be measured by the RMS (Root Mean Square) deviation of the data points from the centroid:

$$Radius_i = \frac{1}{d} \sum_{j=1}^{d} \sqrt{\frac{CF2^x_{i_j}}{n_i} - \left(\frac{CF1^x_{i_j}}{n_i}\right)^2} \tag{6}$$

In the micro-clustering phase, whether a data point can be absorbed by a micro-cluster is determined by its *maximal boundary*. For a fast evolving data stream, it is difficult to judge whether to absorb a data point since the radii of micro-clusters may be unstable. If the criterion of absorbing a data point relies on the radius of the nearest micro-cluster, there may be many new micro-clusters created and many existing micro-clusters merged, which is much slower than the absorbing process. To improve efficiency, we introduce an expanding factor so that micro-clusters can expand gradually in each time.

Relevance stamp reflects the recency of the last m data points of a micro-cluster, which is defined as the time of arrival of the $\frac{m}{2 \cdot n_i}$-th percentile of the points. The time-stamps of the data points are assumed to obey normal distribution, the j-th entry of whose mean and standard deviation are:

$$\mu_{i_j} = \frac{CF1^t_{i_j}}{n_i} \tag{7}$$

$$\sigma_{i_j} = \sqrt{\frac{CF2^t_{i_j}}{n_i} - \left(\frac{CF1^t_{i_j}}{n_i}\right)^2} \tag{8}$$

Actually, each micro-cluster can be regarded as a hyper-sphere containing a set of d-dimensional data points. At the very beginning of the algorithm, we need to establish q micro-clusters by buffering *InitNum* data points on disk and clustering them using a standard k-means algorithm. Then, whenever a new data point arrives, the online phase determines whether it is absorbed by a micro-cluster, or a new micro-cluster is created after an old micro-cluster being deleted or two closest micro-clusters being merged. We define the proportion of absorbed points as *absorption rate.* At the same time, the micro-clusters are stored as snapshots following a pyramidal time pattern [2], so that for a given time horizon, the offline phase can determine the micro-clusters by using two related snapshots.

The efficiency of the online phase determines the speed of processing and should be as simple as possible and easy to be parallelized. It is obvious that online micro-clustering methods based on spherical clusters are much simpler than those based on density grids [9, 12] or space partition [27, 34]. So, there is a need to investigate whether spherical micro-clusters can meet the requirements of discovering macro-clusters of arbitrary shapes in the offline phase. To show its feasibility, we consider the following proposition.

Proposition 1. *A confined space of any shape can be approximated infinitely well by an arrangement of non-overlapping spheres of the same dimension.*

A typical circle packing problem[1] is to find an arrangement in which the circles can fill as large a proportion of the space as possible. Note that Proposition 1 describes a similar problem where both the number of circles and the radii of circles are flexible. So, it is feasible to approximate macro-clusters of any shapes by many micro-clusters.

4. ONLINE MICRO-CLUSTERING

In this section, we focus on the key challenges in designing the parallel framework for online micro-clustering in order to handle fast and complex data streams efficiently.

Challenge 1. Transferring data points from host (CPU) to device (GPU) on a one by one basis can be very time consuming.

In a data stream, only a single data point arrives at any given moment. Consequently, data points should be processed one after another according to the order of arrival. However, in order to conduct computing on the GPU, there is a need to transfer data from CPU to GPU, which is a time consuming operation compared with the computing power of the GPU. If each data point is transferred to the GPU upon its arrival, significant amount of time will be wasted on the transferring operation. To reduce the average communication cost of each data point, we create a buffer where *BufferSize* data points can be stored in the CPU memory for a short time and then transferred to device together. Furthermore, as shown in Fig. 2, if the CPU is not able to finish processing a data point before another point arrives, the time lag will get larger over time. By contrast, all the points in the buffer may be timely processed on the GPU. The practicability and accuracy of our method will be verified by experiments in Section 6.

Challenge 2. The shapes of macro-clusters may be irregular or complex, and the number of them may vary tremendously.

For macro-clusters of regular shapes, a small number of micro-clusters can approximate them well enough and excessive number of micro-clusters may waste memory and time. However, for macro-clusters of complex shapes, many more micro-clusters will be needed to approximate them. So, the number of micro-clusters should be self-adaptive within the limits of memory. We propose a scheme according to which adjacent micro-clusters are merged based on their merging factor instead of the number of unabsorbed points. The merging factor is defined as Definition 1, which means any micro-clusters closer than their merging factor will merge.

Definition 1. (Merging Factor) The merging factor Mg_{ij} of the i-th and the j-th micro-clusters is defined as a factor of the sum of the radii of the two adjacent micro-clusters, which can be expressed as:

$$MG_{ij} = t(Radius_i + Radius_j) \qquad (9)$$

where t is a relaxation factor, typically between 0.5 and 2.

[1]In three dimensions, it is called sphere packing problem. In higher dimensions, it is called hyper-sphere packing problem.

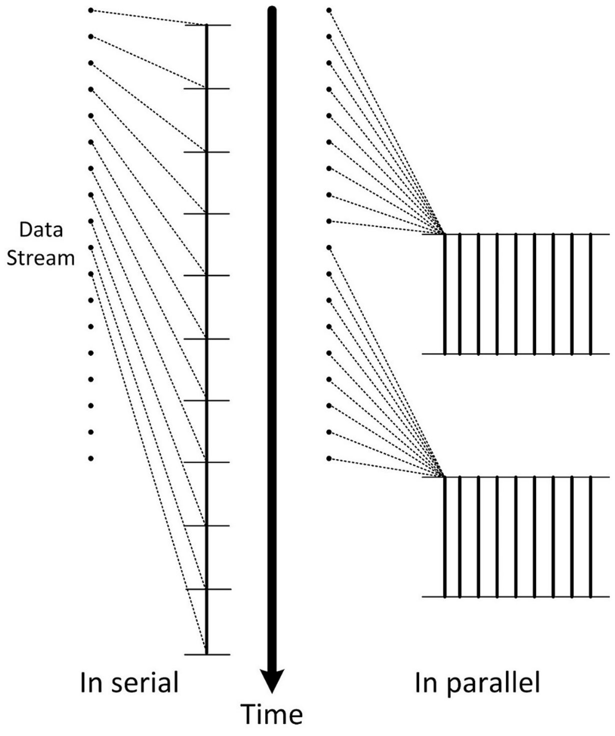

Figure 2: Micro-clustering in serial and in parallel

Based on the discussion above, the parallel micro-clustering framework based on GPU is shown as Algorithm 1 using pseudo-code.

There are several computation-intensive parts in the algorithm, which are very suitable for running on the GPU:

- ☐ Obtaining the cluster label of each point in the initialization part. For each iteration in the loop, there are *PointNum*ClusterNum* Euclidean distances to compute.
- ☐ Finding the closest micro-clusters. For each pass, there are *BufferSize*q* Euclidean distances to compute. The minimum distance and its index are also needed.
- ☐ Finding very old micro-clusters. For each pass, there are q relevance stamps to compute. The minimum distance and its index are also needed.
- ☐ Finding very adjacent micro-clusters. For each pass, there are $\frac{q(q-1)}{2}$ Euclidean distances to compute.

When finding the closest micro-clusters, each point can be processed in a single *block*, and a specific number of Euclidean distances are computed by a single *thread*. For very high dimensionality, we can take advantage of *dynamic parallelism* to launch some child kernels to deal with each dimension within the limits of hardware resource. Furthermore, the data needed by all the threads in the same block can be placed in *shared memory* for faster access, such as the centers of micro-clusters.

According to Amdahl's Law, the maximum speedup that can be achieved depends on the proportion of a program

Algorithm 1 GPU Based Online Micro-clustering

```
Input: Micro-clusters, imic; Buffered data points, data; Time stamps, t
Output: New micro-clusters, omic
 1: function MICROCLUSTERING(imic, data, t)
 2:     Get centers, $CF2t$, $CF1t$ and point numbers of imic
 3:     for $i = 0 \to BufferSize - 1$ in parallel do
 4:         for $j = 0 \to MicroClusterNum - 1$ do
 5:             Calculate $DistMat[i, j]$
 6:         end for
 7:     end for
 8:     for $i = 0 \to BufferSize - 1$ do
 9:         $MinDist \leftarrow min(DistMat(i, :))$
10:         $MinIdx \leftarrow argmin(DistMat(i, :))$
11:         Calculate $MaxBoundary$ of the $MinIdx$-th micro-cluster
12:         if $MinDist <= MaxBoundary$ then AbsorbPoint(data(i), t(i))
13:         end if
14:     end for
15:     for $i = 0 \to BufferSize - 1$ in parallel do
16:         Calculate $RelevStmpVec$
17:         if $RelevStmpVec[i] < threshold$ then DeleteCluster(i)
18:         end if
19:     end for
20:     for $i = 0 \to BufferSize - 2$ in parallel do
21:         for $j = i \to BufferSize - 1$ do
22:             Calculate $ClusterDistMat[i, j]$
23:             if $ClusterDistMat[i, j] <= MG_{ij}$ then MergeClusters(i,j)
24:             end if
25:         end for
26:     end for
27:     for each remain point do
28:         CreateNewClusters()
29:     end for
30:     return omic
31: end function
```

that can be made parallel and the number of processors [15]. Let us denote the number of processors using n and the proportion using P, the theoretical speedup of the parallel code over the serial code can be expressed as:

$$S(n) = \frac{1}{(1-P) + P/n} \tag{10}$$

With the increase of the number of micro-clusters and dimensionality, P gets larger, and more processors can be used. As a result, (1-P) gets smaller while P/n almost remains the same, resulting in higher $S(n)$ value. In this way, the algorithm is expected to feature good scalability with regard to the number of micro-clusters and dimensionality. However, the number of processors in a GPU may be up to thousands but still limited, and the memory space in a GPU is also limited. If the number of micro-clusters or dimensionality is out of range, processors or memory space will be short supply and the maximum speedup will increase at a much lower speed and can not go beyond n. In this case, more powerful computing devices are needed to break through the limitation.

In addition to the above factors, there are also some parts that consume significant time but are very difficult to run in parallel, such as the deletion of old clusters and the creation of new clusters. This is because these parts involve object creation and destruction, resulting in memory allocation and release, a slow operation on the GPU. Consequently, for fast evolving data streams, the speedup rate will drop with the decrease of *absorption rate*. In this case, we propose a *Distance Matrix*, which maintains the Euclidean distance between each pair of micro-clusters. Each time a micro-cluster is modified, created or deleted, the *Distance Matrix* updates the corresponding value in it, making it easier to find adjacent objects frequently.

5. OFFLINE MACRO-CLUSTERING

In this section, we describe how to cluster micro-clusters into macro-clusters of arbitrary shapes and detect outliers and present GPU accelerated version of the algorithm.

Rodriguez and Laio [25] proposed a powerful and concise clustering algorithm in 2014, which is able to recognize a large number of clusters with random noise regardless of their shapes and dimensionality. The core idea is that clusters are characterized by a higher density than their neighbours and by a relatively large distance from points with high densities. However, for the offline macro-clustering phase in PaStream, the objects to be clustered are not common data points, but micro-clusters containing different numbers of data points, or weighted points. So, we need to make some improvement to meet the new requirements.

There are two quantities to evaluate the i-th micro-cluster, which are defined as follows:

Definition 2. (Local Density) The local density ρ_i with a Gaussian kernel is defined as the total Gaussian weight of all the data points, which can be expressed as:

$$\rho_i = \sum_{j=1}^{n} \frac{d_{ij}}{d_c} \exp(-\frac{d_{ij}}{d_c}) \tag{11}$$

where n is the number of data points, and d_{ij} is the Euclidean distance between the i-th and the j-th micro-cluster, and d_c is a cutoff distance.

Definition 3. (Local Distance) The local distance ψ_i is defined as the minimum distance between the i-th micro-cluster and any other micro-cluster with higher local density, which can be expressed as:

$$\psi_i = min_{j,\rho_j>\rho_i}(d_{ij}) \tag{12}$$

First of all, for all the cluster centers, we need to calculate the Euclidean distances between each other, which is stored in *CenterDistMat*. Afterwards, for each micro-cluster, we calculate its *local density* and *local distance*, and then we can plot the *decision graph*. The *spiral* dataset is plotted in Fig. 3, and its *decision graph* is shown in Fig. 4.

Typically, the *local distance* of the local density maxima is much larger than others, which makes it very easy to distinguish the number and centers of the clusters.

The parallel macro-clustering framework based on GPU is shown as Algorithm 2 using pseudo-code.

The tasks listed below are computation-intensive but very convenient to parallelize:

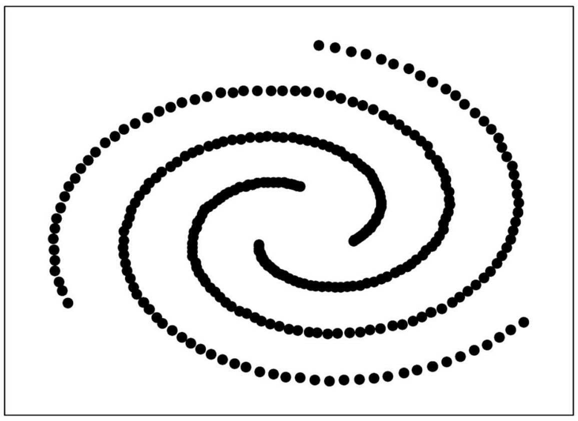

Figure 3: Spiral

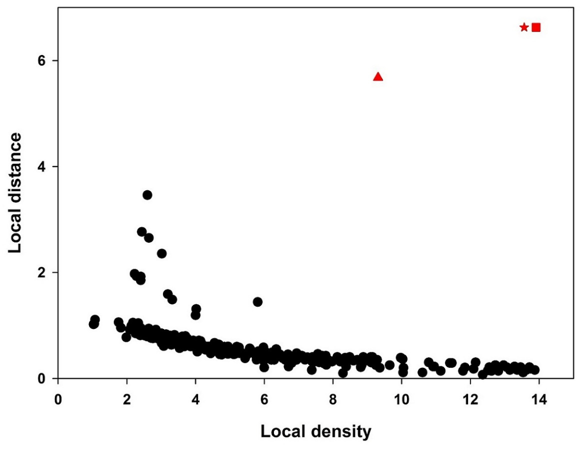

Figure 4: Decision graph of spiral

Algorithm 2 GPU Based Offline Macro-clustering

```
Input: Micro-clusters, $mic$
Output: Macro-clusters, $mac$:
 1: function MACROCLUSTERING($mic$)
 2:     for $i = 0 \to MicNum - 2$ in parallel do
 3:         for $j = i \to MicNum - 1$ do
 4:             Calculate $CenterDistMat[i, j]$
 5:         end for
 6:     end for
 7:     Get $d_c$ by sorting the $CenterDistMat$
 8:     for $i = 0 \to MicNum - 1$ in parallel do
 9:         Calculate $\rho[i]$
10:     end for
11:     for $i = 0 \to MicNum - 1$ in parallel do
12:         Calculate $\psi[i]$
13:     end for
14:     Plot Decision Graph and choose macro-cluster centers
15:     for $i = 0 \to MicNum - 1$ in parallel do
16:         Assign cluster labels
17:     end for
18:     return $mac$
19: end function
```

- ☐ Calculating *CenterDistMat*. There are $\frac{n(n-1)}{2}$ Euclidean distances to compute.

- ☐ Calculating local densities. There are $\frac{n(n-1)}{2}$ Gaussian wights to compute.

- ☐ Calculating local distances. There are $\frac{n(n-1)}{2}$ comparisons needed.

- ☐ Assigning cluster labels. There are n comparisons needed to assign cluster labels for all the normal points. Besides, in order to distinguish outliers, $\frac{n(n-1)}{2}$ comparisons are needed.

Similarly, each task can be divided into many independent parts to be executed by different blocks and threads. Within the limits of processors and memories, we should try to use more threads to simplify the task of a single thread. For sorting operations, *Thrust* offers several functions such as *thrust::sort* and *thrust::stable_sort* to sort data or rearrange data according to a given criterion very efficiently.

6. EXPERIMENTAL RESULTS

We conducted the experiments on a PC with Intel-i7 CPU (3.40GHz, 4 cores / 8 threads) and NVIDIA GeForce GTX 750 GPU. The programming environment was Visual Studio 2012 with CUDA 6.5 running on Windows 8.1 (64bit).

Both synthetic datasets and real datasets were used in our experiments. The synthetic datasets have a varying number of clusters ranging from 1 to 1000, and dimensionality ranging from 2 to 100. The widely used real dataset in data stream clustering KDD-CUP'99, which contains a set of Network Intrusion Detection stream data, was also included in experiments. This dataset contains 4898431 records, 5 clusters and 42 attributes of which 34 are continuous and the other 8 are discrete. In our experiments, all the 34 continuous attributes were used and normalized to [0, 1].

6.1 Micro-Clustering Performance

We evaluated the online micro-clustering phase of PaStream by its efficiency and quality compared with CluStream [2], including the (single-threaded) CPU version and the OpenMP version.

Fig. 5 shows the running time of PaStream compared with the CPU version and OpenMP version CluStream with varying numbers of micro-clusters, as well as the speedup rates relative to the CPU version CluStream. We can see that the speedup rate of the OpenMP version CluStream varied between 2 and 3.5 while the speedup rate of PaStream kept growing with the increase of the number of micro-clusters,which shows excellent scalability with regard to the micro-cluster number. However, too many micro-clusters will increase the running time significantly, so the number of micro-clusters should be controlled within a certain scope.

Fig. 6 shows the running time of PaStream compared with the CPU version and OpenMP version CluStream with varying dimensionality, as well as their speedup rates relative to the CPU version CluStream. We can see that the speedup rate of PaStream kept growing when the dimensionality increased from 2 to 70 and the running time remained below 10 seconds, which shows excellent scalability with regard

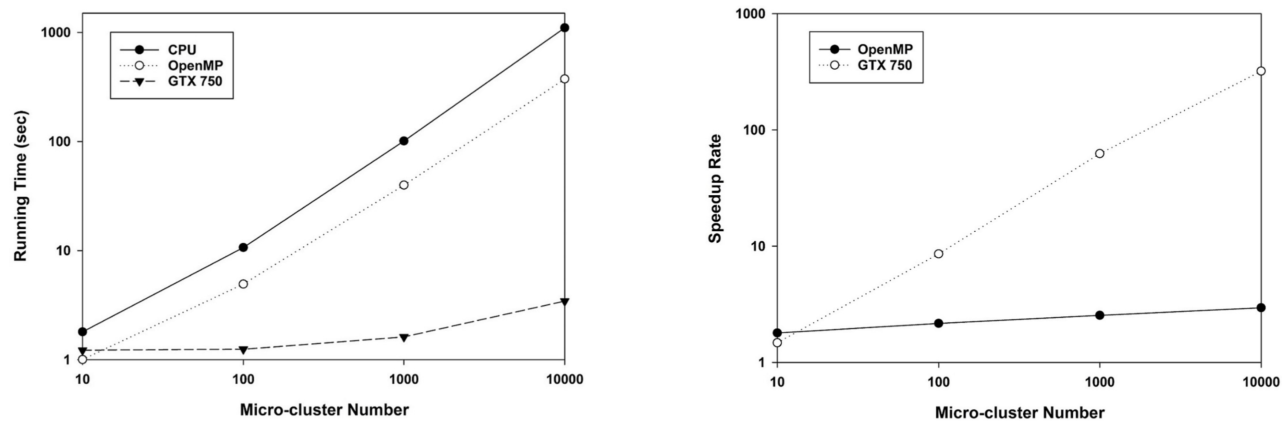

Figure 5: Running time and speedup rate comparison with varying numbers of micro-clusters (Dimensionality = 2, Buffer Size = 1000, Absorption Rate = 95%)

to dimensionality. However, when the dimensionality exceeded 70, the speedup dropped a bit and stopped increasing, which is caused by the limits of processor number and memory space in the GPU. If the dimensionality is too high, too much data will remain in the global memory of GPU, reducing the computing speed.

Fig. 7 shows the running time of PaStream with varying buffer sizes, as well as their speedup rates relative to the CPU version CluStream. We can see that the running time of PaStream dropped significantly when the buffer size increased from 1 to 10 and the speedup rate kept growing with the increase of buffer size, which is consistent with the analysis above. Obviously, the buffer size should be selected rationally to keep the balance between speedup rate and clustering quality. If the buffer size is too large (relative to data stream speed), speedup rate will increase much slower while clustering quality will drop significantly.

Fig. 8 shows the speedup rate of PaStream with varying absorption rates. We can see that the speedup rate decreased for fast evolving data streams because of the frequent deletion of old clusters and creation of new clusters, but the algorithm still shows very high efficiency.

6.2 Macro-Clustering Performance

We evaluated the offline phase of PaStream by its efficiency and quality compared with its CPU version.

Fig. 9 shows the 7 macro-clusters by clustering 1000 micro-clusters. We can see that the macro-clusters were of high quality and most outliers (marked as black dots) were correctly detected.

Fig. 10 shows the SSQ and clustering quality of macro-clustering algorithm with varying buffer sizes based on the KDD-CUP'99 dataset. We can see that the SSQ increased very slowly with the increase of buffer size and the clustering quality was only affected slightly. In practice, the buffer size can be chosen properly according to the speed and complexity of data stream.

Fig. 11 shows the running time and speedup of the GPU version macro-clustering algorithm compared with its CPU version with varying numbers of micro-clusters. We can see that the GPU version can reach up to more than one hundred times faster for complex data streams.

Fig. 12 shows the running time and speedup of the GPU version macro-clustering algorithm compared with its CPU version with varying dimensionality. We can see that the running time of the GPU version algorithm showed excellent scalability with regard to dimensionality and the speedup kept increasing within the permitted scope of hardware resources.

7. CONCLUSION AND FUTURE WORK

In this paper, we present PaStream, a highly efficient parallel algorithm for clustering very fast and complex data streams based on NVIDIA GPUs. With the help of a new parallel framework and a series of innovative techniques, the online micro-clustering phase of PaStream can achieve hundreds of times speedup using inexpensive commodity GPUs and features excellent scalability with regard to cluster number and dimensionality. By incorporating a local density based algorithm into the macro-clustering phase, PaStream can discover clusters with arbitrary shapes and handle outliers effectively. Meanwhile, the macro-clustering algorithm can be accelerated by dozens of times using GPUs, without compromising the clustering quality.

As to future work, it will be a tremendous advantage if we can reduce the I/O overhead significantly and run the algorithm totally on the GPU. Other advanced high-performance computing platforms such as Mars, Spark etc. can also be exploited to handle very fast and complex data streams. It is equally important to consider some special data types (e.g., spatio-temporal data) and more realistic scenarios (e.g., uncertainties and spatial constraints).

8. ACKNOWLEDGMENTS

This work was supported by the NVIDIA CUDA Teaching Center awarded to Tsinghua University.

9. REFERENCES

[1] Accelerate Continuum Documentation. http://docs.continuum.io/accelerate.

[2] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 81–92, 2003.

[3] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for projected clustering of high dimensional

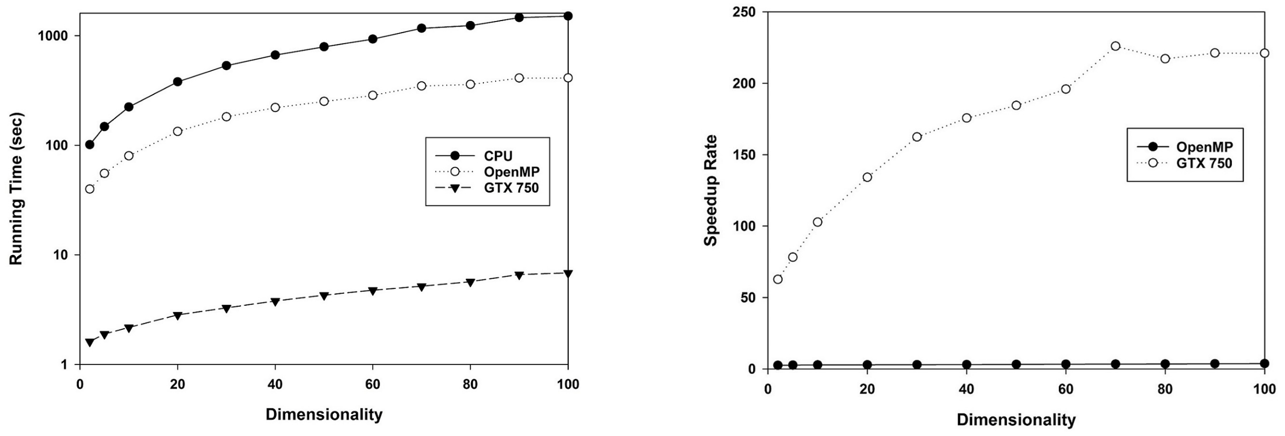

Figure 6: Running time and speedup rate comparison with varying dimensionality (Micro-cluster Number = 1000, Buffer Size = 1000, Absorption Rate = 95%)

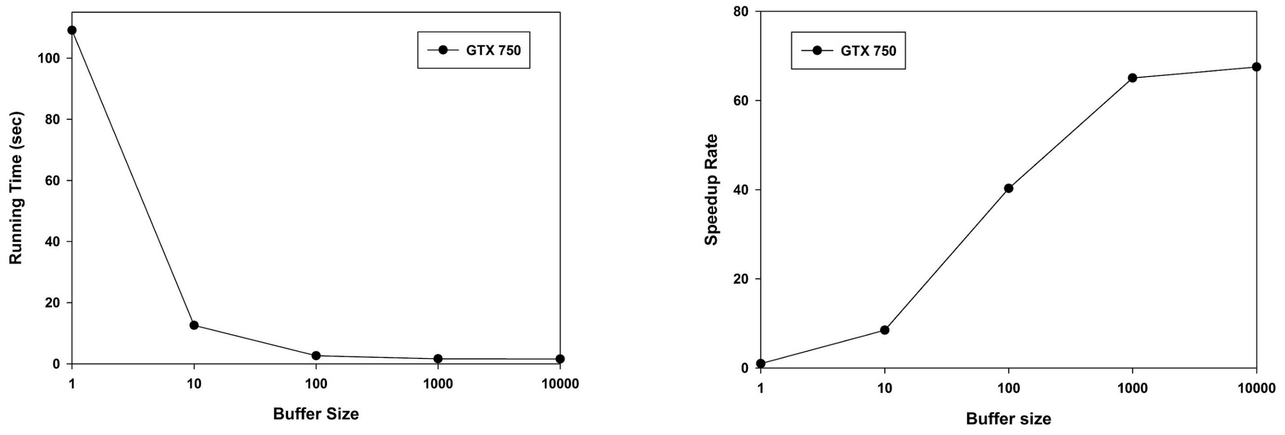

Figure 7: Running time and speedup rate with varying buffer sizes (Micro-cluster Number = 1000, Dimensionality = 2, Absorption Rate = 95%)

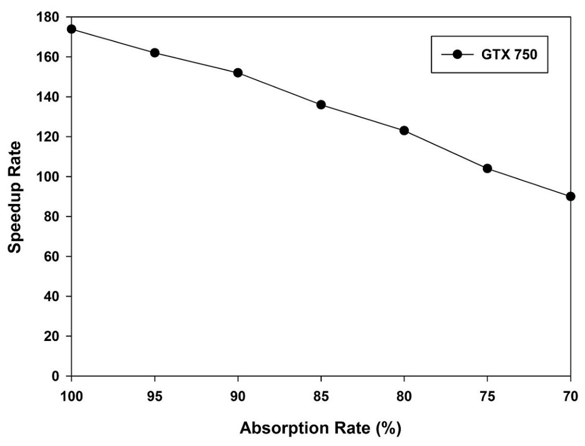

Figure 8: Speedup comparison with varying absorption rates (Micro-cluster Number = 1000, Dimensionality = 30, Buffer Size = 1000)

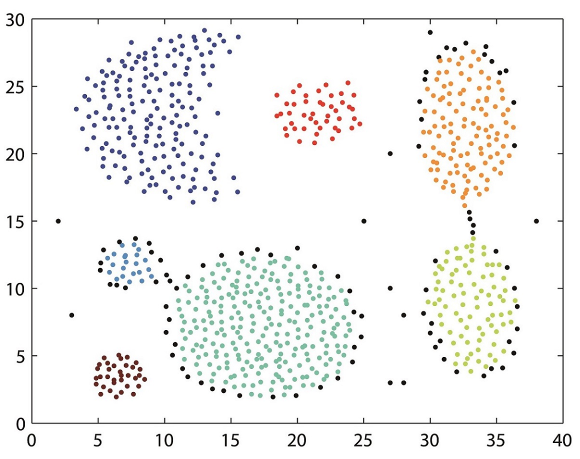

Figure 9: Macro-clusters with outliers

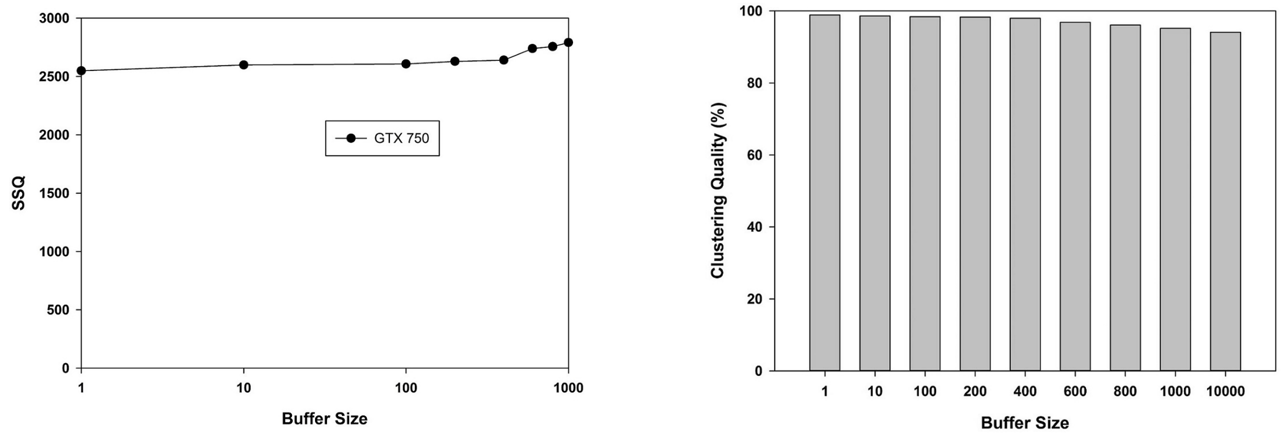

Figure 10: SSQ and clustering quality with varying buffer sizes (Micro-cluster Number = 1000, Dimensionality = 2, Absorption Rate = 95%)

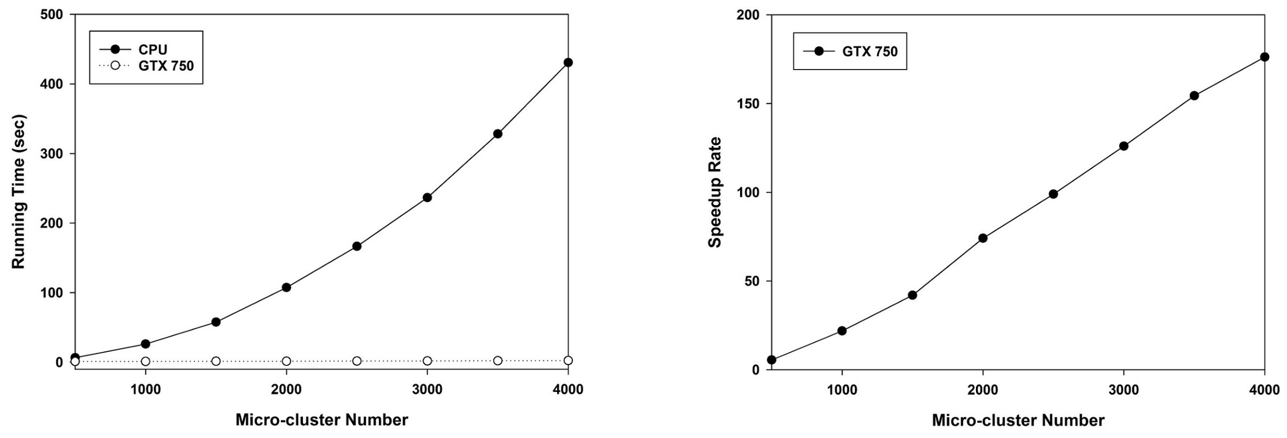

Figure 11: Running time and speedup with varying numbers of micro-clusters (Dimensionality = 60)

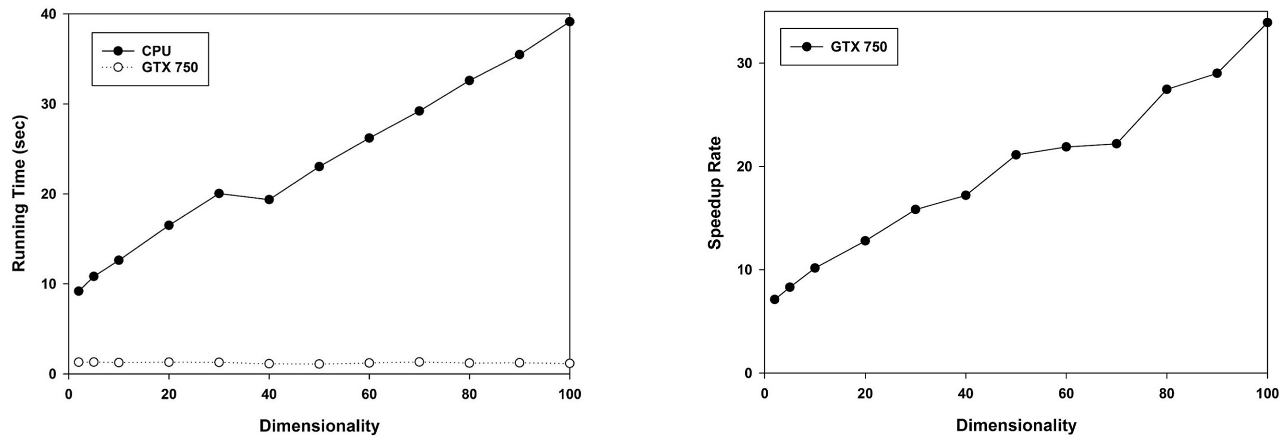

Figure 12: Running time and speedup with varying dimensionality (Micro-cluster Number = 1000)

data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 852–863, 2004.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16, New York, USA, 2002. ACM Press.

[5] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '03*, pages 234–243, New York, USA, June 2003. ACM Press.

[6] D. Barbará. Requirements for clustering data streams. *ACM SIGKDD Explorations Newsletter*, 3(2):23–27, 2002.

[7] D. Barbara and P. Chen. Using the Fractal Dimension to Cluster Datasets. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 260–264, New York, USA, 2000.

[8] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA : Massive Online Analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[9] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-Based Clustering over an Evolving Data Stream with Noise. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 328–339, 2006.

[10] F. Cao and A. Zhou. Fast Clustering of Data Streams Using Graphics Processors. *Journal of Software*, 18(2):291–302, 2007.

[11] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the Thirty-fifth ACM Symposium on Theory of Computing - STOC '03*, pages 30–39, New York, USA, 2003. ACM Press.

[12] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, New York, USA, 2007. ACM Press.

[13] S. Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Newnes, 2012.

[14] CUDA Toolkit Documentation. http://docs.nvidia.com/cuda/index.html.

[15] R. Farber. *CUDA Application Design and Development*. Elsevier, 2011.

[16] M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26, 2005.

[17] L. Golab and M. T. Özsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, June 2003.

[18] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, May 2003.

[19] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 359–366. IEEE Comput. Soc, 2000.

[20] Y. Luo, Z. Ni, and Z. Yangge. New fractal clustering algorithm on data stream. *Computer Engineering and Applications*, 46(6):136–138, 2010.

[21] A. Mahdiraji. Clustering data stream: A survey of algorithms. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 13:39–44, 2009.

[22] MOA. http://moa.cms.waikato.ac.nz/.

[23] H. Nasereddin. Stream Data Mining. *International Journal of Web Applications (IJWA)*, 1(4):183–190, 2009.

[24] L. O'Callaghan, N. Mishra, and A. Meyerson. Streaming-data algorithms for high-quality clustering. In *Proceedings of the 18th International Conference on Data Engineering*, pages 685–694, Washington, DC, 2002.

[25] A. Rodriguez and A. Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, June 2014.

[26] D. Roger, U. Assarsson, and N. Holzschuch. Efficient stream reduction on the GPU. In *Workshop on General Purpose Processing on Graphics Processing Units*, Boston, United States, 2007.

[27] H. Sun, G. Yu, Y. Bao, F. Zhao, and D. Wang. CDS-Tree : An Effective Index for Clustering Arbitrary Shapes in Data Streams. In *Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)*, pages 81–88. IEEE, 2005.

[28] U. Verner, A. Schuster, and M. Silberstein. Processing data streams with hard real-time constraints on heterogeneous systems. In *Proceedings of the 25th International Conference on Supercomputing*, Tucson, Arizona, 2011.

[29] N. Wilt. *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley, 2013.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, volume 1, pages 103–114, 1996.

[31] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.

[32] Y. Zhang and F. Mueller. GStream: A general-purpose data streaming framework on GPU clusters. In *International Conference on Parallel Processing (ICPP)*, number 1, pages 245 – 254, Taipei City, 2011. IEEE.

[33] Y. Zheng, Z. Ni, S. Wu, and L. Wang. Data stream cluster algorithm based on mobile grid and density. *Computer Engineering and Applications*, 45(8):129–131, 2009.

[34] W. Zhu, J. Yin, and Y. Xie. Arbitrary Shape Cluster Algorithm for Clustering Data Stream. *Journal of Software*, 17(3):379–387, 2006.