Research Issues of Outlier Detection in Trajectory Streams Using GPUs

Eleazar Leal
Department of Computer Science
University of Minnesota Duluth
Duluth, MN, USA
eleal@d.umn.edu

Le Gruenwald
School of Computer Science
University of Oklahoma
Norman, OK, USA
ggruenwald@ou.edu

ABSTRACT

The widespread availability of sensors like GPS and traffic cameras has made it possible to collect large amounts of spatio-temporal data. One such type of data are trajectories, each of which consists of a time-ordered sequence of positions that a moving object occupies in space as time goes by. Trajectories can be streamed in real time from sensors, and because of this, they capture the current state of moving objects. For this reason, trajectories can be used in applications such as the real-time detection of senior citizens who have just fallen or who have just gotten lost outdoors, the real-time detection of drunk drivers, and the real-time detection of enemy forces in the battlefield. These applications involve the identification of trajectories with anomalous behaviors, and require fast processing in order to take immediate preventive action. However, outlier detection poses challenges stemming from both the complexity of the data and of the task. One way to address this is through parallel architectures like GPUs. In this paper, we present the problem of outlier detection in trajectory streams, and discuss the research issues that should be addressed by new outlier detection techniques for trajectory streams on GPUs.

Keywords

Trajectory data; data streams; trajectory streams; outlier detection; GPUs

1. INTRODUCTION

Outlier detection is the data mining task that consists in finding those elements, called outliers, that are substantially different from other elements [1] in a dataset so as to arouse the suspicion that they may have been generated by a different process [2]. Outliers are inevitable because of human and instrument errors, catastrophes, malicious behaviors, etc. [1]. It is desirable to find outliers because they can reveal information that is not present in the non-anomalous elements of the dataset.

One characteristic that the outlier detection problem shares with other data mining tasks is that the type of data [3] has a profound influence on the complexity of the problem; in other words, outlier detection algorithms are very data type specific. For example, an outlier detection technique for time series data needs to address the issue of temporal continuity because temporally adjacent data points in a time series exhibit a strong correlation. This is usually not the case in multi-dimensional record data, where different data points are independent, so they have no temporal continuity.

Among the different types of data in which outliers can be searched for are trajectories. A trajectory is a polygonal line consisting of the points that a moving object occupies in space as time goes by. One way of constructing these polygonal lines is by periodically sampling the positions of the objects being tracked, a sampling that can be done through the use of location sensors like GPS. We see then that since trajectories consist of a time-ordered sequence of points, there is a temporal dependency between them. For this reason, trajectories are a special case of multivariate time series.

Just like in the case of time series, the problem of trajectory outlier detection can be off-line or online. In the off-line problem, the trajectories are fixed and known, while in the online one, the trajectories keep growing as more and more points arrive from the location sensors. Since an online trajectory is essentially the same as a data stream [4] [5] whose constituent elements are the positions of the trajectory, we call the problem of online trajectory outlier detection *trajectory stream outlier detection*.

The definition of trajectory outlier depends heavily on the nature of the application. An outlier element in one application might not be such in another application. For this reason, the semantics of trajectory stream outliers can be defined in several different ways [6]. For example, a trajectory stream is considered an outlier with respect to a set of trajectory streams if it has fewer neighbor trajectories within a given temporal window than most streams. In this definition, the notion of neighbor trajectories takes into consideration both the spatial proximity of the trajectories and the time duration of that spatial proximity. Another example is to define outlier trajectory streams to be those that have significantly large distances to their K-nearest neighbor trajectories in a temporal window.

There are many applications for outlier detection in trajectory streams. For example, some senior citizens may require constant monitoring, especially when they are outside of their homes; the same is the case for children. It is desirable to know in real time, if they just took the wrong bus, if they just had a fall, or if they just got lost [7]. One way of addressing this problem is by having a record of the trajectories that each senior citizen has traversed in the past, and by also keeping track of their current location when they are outside. With this information, it is possible to determine if the trajectory being currently traversed by the senior citizen is an anomalous one, and then take preventive and immediate action.

A second application of outlier detection in trajectory streams consists in the instantaneous detection of dangerous driving behavior [8]. By using traffic cameras, streams of trajectory points [9] can be collected from the vehicles on the road. Then, by finding outliers among the trajectory streams, it is possible to find those vehicles whose trajectories significantly deviate from the normal driving behavior, and which could correspond to drunk drivers, speeding drivers, or drivers with malfunctioning cars. In all these cases, the immediate recognition of these anomalous behaviors is essential to avoid potentially dangerous situations. In this application, since in practice there are many vehicles, each providing the data for a different trajectory stream, there is a need for scalable outlier detection in trajectory streams.

A third example application of outlier detection in trajectory streams is in security surveillance, where the idea is to identify in real time those individuals with suspicious behaviors that could be a safety threat for others. For example, visitors in a military base that do not stay with their designated group can be identified as those with anomalous trajectories [10]. Also related to this application is the real-time detection of approaching enemies in the battlefield. In this application, military forces deploy in an area a network of sensors to protect troops and towns, by monitoring the movements of potential intruders, which would correspond to those with anomalous trajectories [11].

The problem of outlier detection in trajectory streams is one of Big Data. The reason is that it involves a high volume of uncertain data that are also arriving at a high velocity. For the example applications outlined before, there is a need to keep track of multiple trajectory streams at the same time because a system may be capable of tracking the movements of several senior citizens. Each stream generates new position updates at every time instant, each new position arriving with high velocity. In addition to the high volume and high velocity, trajectory data are characterized by an inherent uncertainty, stemming from the noise of location-sensing devices like GPS. Finally, on top of all these challenges, there is the issue of trajectory outlier detection, which in general has high computational complexity.

One way of tackling these Big Data issues is through the use of parallel computer architectures like Graphics Processing Units (GPUs). GPUs are the co-processors installed on graphics cards, and are an example of highly parallel SIMD architecture, which is capable of achieving up to an order of magnitude of higher floating point instruction throughput than comparable multicore CPUs [12]. This throughput advantage combined with the facts that GPUs are available in many types of computers, from cellphones to supercomputers, and that they are highly energy efficient makes GPUs an ideal architecture to leverage against the Big Data challenges of trajectory stream outlier detection.

Despite all the advantages of GPUs, developing scalable algorithms for this type of parallel architectures can be challenging. Among the reasons for this is that GPUs have a relatively small memory space, and that they are connected to the host computer through relatively low throughput interfaces that can adversely affect the performance of the algorithms. However, there exists no work discussing the research issues that need to be addressed when developing trajectory stream outlier detection algorithms for GPUs. This paper aims to fill this gap.

The remainder of this paper is organized as follows. Section 2 introduces the concepts of trajectory, trajectory stream, outlier detection in trajectory streams, and GPU computing. Section 3 discusses the research issues that an outlier detection technique for trajectory streams on GPUs needs to address. Finally, Section 4 provides conclusions and future research directions.

2. PRELIMINARIES

In this section, we discuss the fundamental concepts of trajectories, trajectory streams, outlier detection in trajectory streams, and GPU computing.

2.1 Trajectories

Informally, a *trajectory* is a polygonal line consisting of the points that a moving object occupies in space as time goes by. One way of constructing these polygonal lines is by periodically sampling the positions of the objects being tracked over time through the use of location sensors like GPS. More formally, given a set $S = \{(x_i, y_i, t_i)$, with $t_i \leq t_{i+1}$, $1 \leq i < n\}$ of points in R^3 sampled from the movement of an object with a location sensor, a trajectory over S is a continuous function τ where $\tau(i) = (x_i, y_i, t_i)$ for all integers i in $[1, \ldots n]$ and such that $\tau(x)$, with x in $[t_i, t_i + 1)$, is the interpolated value between $\tau(i)$ and $\tau(i + 1)$ [13].

In the above definition, the tuple (x_i, y_i, t_i) means that at time t_i the object traversing this trajectory was at the position (x_i, y_i). Figure 1 shows an example of a moving object's trajectory with five points $p[1]$ to $p[5]$, where the start point $p[1]$ occurs at the location that has the latitude 2, longitude 10, at the timestamp 9:01:56 am, and the end point $p[5]$ occurs at the location with the latitude 5, longitude 17, and at the timestamp 9:02:17 am.

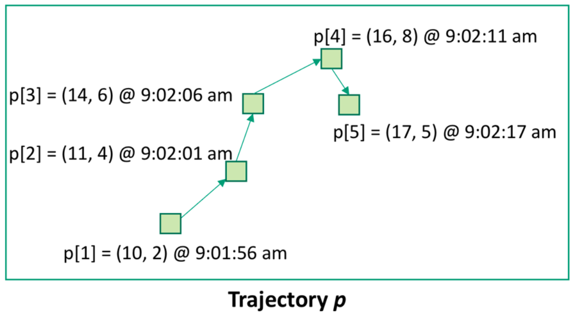
p[4] = (16, 8) @ 9:02:11 am
p[3] = (14, 6) @ 9:02:06 am
p[5] = (17, 5) @ 9:02:17 am
p[2] = (11, 4) @ 9:02:01 am
p[1] = (10, 2) @ 9:01:56 am
Trajectory *p*

Figure 1. Example of a Trajectory

2.2 Trajectory Streams

A *trajectory stream* S is a data stream such that its constituent elements are points from the trajectory of a moving object. One way of obtaining a trajectory stream is by periodically sampling the movements of an object with the help of a location sensor, like a GPS.

2.3 Outlier Detection in Trajectory Streams

Informally, the problem of outlier detection in trajectory streams can be stated as follows. Given a set of n trajectory streams $\{S_i : 1 \leq i \leq n\}$, find those trajectory streams S_k that exhibit an unusual or anomalous behavior when compared to either their respective previous behaviors, or other nearby trajectories streams in the dataset. By unusual behavior of S_k we refer to the kinematic characteristics of the trajectory: for example, S_k has a very different shape (position), velocity, or acceleration, when compared to all other trajectory streams.

As is usually the case in outlier detection, outliers in trajectory streams are no exception in that the concrete definition of outlier is very application dependent [6]. In the few works that exist [3], there are at least two different definitions of trajectory outliers, which can be generalized to trajectory streams. These definitions are now discussed.

2.3.1 Distance Outliers

Given a set of n trajectory streams $\{S_i : 1 \leq i \leq n\}$, find those trajectory streams S_k such that their corresponding moving objects have locations which are very different from those of their neighboring objects' [14].

2.3.2 Velocity Outliers

Given a set of n trajectory streams $\{S_i : 1 \leq i \leq n\}$, find those trajectory streams S_k such that their corresponding moving objects move in directions (i.e., following velocities) which are very different from those of their neighboring objects' [9]. A velocity outlier, unlike a distance outlier, may be located very closely to its neighbors; however, its singular characteristic is that it moves in directions that are very different from those of its neighbors.

As an example, one way of solving the problem of drunk driver detection is by finding velocity outliers. This is because drunk drivers tend to drive at irregular speeds and following irregular directions, as illustrated in Figure 2.

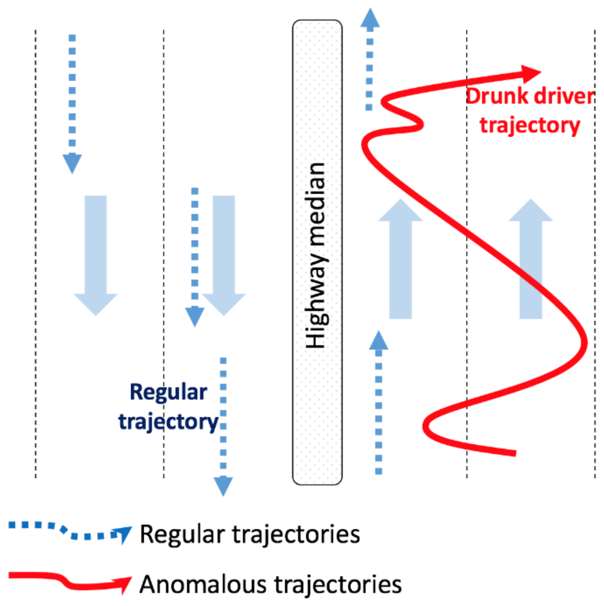

Figure 2. Drunk Driver Detection can be solved by finding velocity outliers

2.4 GPU Computing

In this section, we introduce Graphics Processing Units (GPUs) and describe those characteristics of GPUs that can impact the design of outlier detection algorithms for trajectory streams.

GPUs are co-processors installed on graphics cards in order to perform the necessary calculations to render graphical models. For this reason, GPUs were designed as a parallel architecture capable of simultaneously performing many floating point operations. However, GPUs are designed not only for rendering graphics, but also for general purpose parallel programming.

Among the many advantages of GPUs are that they are present in many kinds of computers, from mobile devices to supercomputers; on certain algorithms that exhibit lots of parallelism, they can achieve up to an order of magnitude of higher floating point instruction throughput than multicore CPUs [12]; and they are very energy efficient [15]. Another advantage of GPUs is that there are works [16] that allow GPU processing from within the popular Spark parallel computing framework [17], so that the high instruction throughput of GPUs can be combined with the scalability, ease of use and fault-tolerance of the Spark framework. All these advantages of GPUs make them excellent tools for tackling the computational challenges associated with outlier detection in trajectory streams.

We now discuss the programming model of GPUs [18] using the vocabulary of CUDA [19], which is one of the popular GPU programming models. GPUs follow a parallelism model that is very similar to SIMD (Single Instruction Multiple Data), where different threads perform the same instruction in parallel over different data. To accomplish this, the programmer must specify the total number of threads that will run in the GPU. Once this is done, during runtime, the system will assign a unique identification number to every thread; it is in this manner that different threads can perform the same instruction and work on different data. GPUs are designed to run portions of code called kernels, which look like regular C-language functions and are called from within the CPU execution flow. However, there is one inconvenience with GPUs, which is that these cards have a separate memory address space from the host computer's main memory. So, before kernels are launched, the CPU must call a special function to transfer the data from the host computer's main memory to the device's memory space. In a similar fashion, once the kernel finishes its execution, the CPU must call another special function to transfer the results from the device's memory space back to the host's main memory.

GPUs can be thought of as a highly parallel architecture where execution threads form the most essential part of the execution hierarchy. At the top of this hierarchy is the grid, which is composed of all threads launched with the kernel. All the threads in a grid can access the GPU's global memory, which is a memory space that is big (in the order of gigabytes) and has high latency. All the threads in a grid are grouped at the time that the kernel is launched into thread blocks, each of which is a collection of threads that can communicate through shared memory. This is illustrated in Figure 3 which shows three thread blocks with three threads each (each GPU thread block has a number of threads which is a multiple of 32), and also shows the shared memory corresponding to every thread block. Shared memory is a memory space private to each thread block that is both smaller (in the order of tens of kilobytes) and faster (around 10 times) than global memory [20]. The threads within a thread block are grouped into sets of 32 threads called warps, each of which is a collection of threads that execute the same instruction (maybe with different operands) in lockstep.

This hierarchy determines not only which threads can communicate, but also how threads can synchronize. Only the threads within a block can use barrier synchronization, and the only way to run barrier synchronization among threads belonging to different blocks is to exit the current kernel and launch a new one. The reason for this is that not all thread blocks run simultaneously.

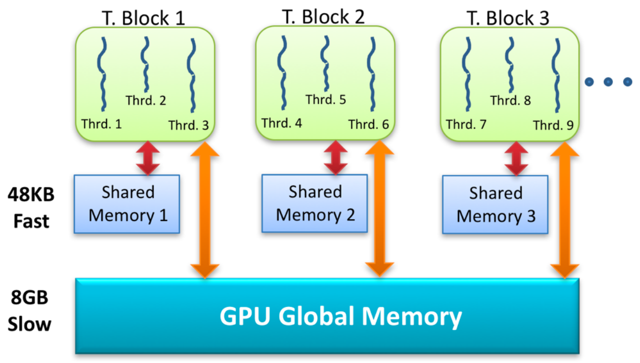

Figure 3. GPU Memory Diagram

3. RESEARCH ISSUES

This section first discusses, in Section 3.1, the general issues, i.e., the challenges, that arise when detecting outliers in trajectory streams. Then, in Section 3.2, we study the specific challenges associated with using GPUs for outlier detection in trajectory streams.

3.1 Challenges of Outlier Detection in Trajectory Streams

In this section, we discuss the general challenges concerning outlier detection in trajectory streams. These challenges are presented in no particular order.

3.1.1 Infinite Size

The sizes of trajectory streams, just like those of general data streams, are infinite. This is because it is assumed that after an object stops moving, it will resume its movement again in the future. Taxis are an example of this because after they finish a trip, they will start another one within a certain period of time [21]. This impacts the outlier detection procedure because the whole stream cannot be stored in memory. Rather, it is only possible to store and make decisions about the outlier-ness of a stream element (stream data point) based on the information provided by a small portion of the stream.

Even if the trajectory streams were not infinite, it would be very challenging to elaborate a model that captures the complete behavior of the trajectories in a dataset across all possible timestamps (past, present and future). This is because of the streaming nature of the data being dealt with, where the data arrives one point at a time, which makes it difficult to anticipate the future behavior of the streams. Even if such a comprehensive model could be built, then there would be the difficulty of efficiently updating it as more data points arrive [8].

Challenge: A technique for outlier detection in trajectory streams should be able to ascertain the outlier-ness of an incoming stream element based on the knowledge that it has gathered after observing a finite number of stream elements.

3.1.2 High Sampling Rates

The sampling rate of a stream is the frequency at which the stream emits new points. A trajectory outlier detection technique for data streams should take the sampling rate into consideration because if the sampling rate is much faster than the processing time, then, because of the infinite size issue, several of the incoming data points might have to be discarded. The issue with this is that these discarded points could be crucial in the decision of whether the trajectory stream in question is an outlier or not.

Challenge: To avoid discarding points from a trajectory stream, a technique for outlier detection in trajectory streams should be able to quickly ascertain the outlier-ness of an incoming data point before the next data point arrives.

3.1.3 Asynchronous Sampling Rates

Asynchrony refers to the fact that moving objects need not emit trajectory updates at the same time instants, i.e., they may emit trajectory updates out of synchrony. For example, in the case of drivers' trajectories, the drivers may turn on their location sensors at different times, and this could lead to having position updates that are not synchronized across drivers. This makes outlier detection more challenging because at the moment when a decision needs to be made regarding whether a trajectory is an outlier, the technique may not have an up-to-date picture of the positions of all objects, which may affect the classification of the outlier trajectory streams.

Challenge: A technique for outlier detection in trajectory streams should take into account the fact that different trajectory streams can have different sampling rates, when determining if a trajectory stream is an outlier.

3.1.4 Concept Drift

The probability distribution from which data originate is not stationary, i.e., the probability distribution of the data changes in time. The impact of this is that trajectory outlier detection techniques cannot assume a fixed probability distribution, and need to adapt to changing distributions. However, this is a difficult problem because that distribution needs to be estimated from the data itself. Nonetheless, due to the infinite size of a data stream, it is not possible to use the complete data stream to estimate this changing distribution, which makes the problem even more difficult.

This is also an issue in the case of trajectory streams because different moving objects can follow different dynamics. For example, if the moving objects of a data stream are car drivers, then each driver may have a different driving style. Additionally, the driving style is dependent upon many other environmental factors like traffic, time of the day, geographic location, weather conditions, etc., and these environmental factors change constantly in time.

Challenge: A technique for outlier detection in data streams should be able to adapt to non-stationary, i.e., time changing, spatio-temporal distributions of trajectory data.

3.1.5 Transiency

The points of a trajectory stream are *transient*, meaning that these points are important for only a brief period of time after they have been generated [22], and become progressively less important over time. The reason for this is that the points keep coming at a high speed, and the probability distribution of trajectory streams is non-stationary. Therefore, since this distribution is evolving over time, then the most recent points reflect the current distribution more accurately than the older ones. Moreover, even if the probability distribution were stationary, the most recent points are often more important in many stream applications. For example, if a senior citizen is wandering around an area, only the most recent points capture the current position of this person.

For this reason, outlier detection algorithms must take the ages, i.e., the time elapsed since each point was generated, of trajectory elements into consideration, and quickly determine if the incoming element is an outlier, so as not to delay the processing of the next element in the stream.

Challenge: A technique for outlier detection in trajectory streams should take into consideration the fact that the most recent points in a trajectory stream are the most important because they convey the most current information about the physical state of the moving object that is traversing that trajectory. This technique should also take into consideration the fact that the importance of the points decreases over time.

3.1.6 Measurement Uncertainty

Location sensor devices have an inherent measurement error. This can negatively impact the classification of a trajectory as an outlier because such classification depends on the moving object's estimated positions. An example of this situation is the following. Suppose that an elder citizen has gotten lost in the city. If this citizen's trajectory is fairly close to, but different from, another of his or her usual trajectories, then, because of the measurement uncertainty, it may not be possible to determine that the citizen is indeed lost.

One of the noise sources related to the measurement process is the noise inherent to GPS device measurements [23] [24]. This noise arises because no measurement is perfectly accurate, but also arises from the environmental conditions surrounding the sensor at the moment when the measurement is made. For example, when collecting the trajectory data from flying mallards [25], the GPS measurement errors can be greater if there are overcast skies in the place where the animals are, or if the animals have tampered with their GPS collars, etc.

Challenge: A technique for outlier detection in trajectory streams should take the uncertainty of the trajectory data into consideration when detecting outlier trajectories.

3.1.7 Model Uncertainty

The commonly used trajectory model [26] assumes that each object moves in a straight line between two consecutive points of its trajectory. Because moving objects may not necessarily move in a straight line between any two consecutive sampled points, it follows that the linear interpolation model for trajectories cannot approximate the true paths of the moving objects with perfect accuracy. The problems originating from this model uncertainty are not as evident when sampling rates are relatively high with respect to the speed of the moving object. This is because the higher the sampling rates are, the smaller the approximation errors are between the trajectory and the object's true path. Model uncertainty is an important issue in applications like taxi monitoring, where the sampling rates are intentionally low in order to reduce the energy consumption of the location sensors.

This model uncertainty issue can adversely impact the detection of trajectory outliers because such detection depends on the model used to predict the true paths of the objects describing the trajectories. If, for example, the sampling rate of an object is very low when compared to its speed, then there is an uncertainty concerning the actual location of the object in between two consecutive sampled data points, making it significantly more challenging to determine if the trajectory in question is an outlier.

Challenge: A technique for outlier detection in trajectory streams should take into consideration the possibility that, if the sampling rate is too low with respect to the speed of the object, then the true path of the moving object might deviate from the straight line between two consecutive trajectory points.

3.1.8 Continuity

Moving object trajectories, unlike arbitrary data streams, have very smooth dynamics, meaning that as time goes by, the position of a moving object cannot exhibit abrupt changes [6]. The challenge then lies in how to adequately exploit this smoothness in order to detect outlier trajectory streams efficiently.

Challenge: A technique for outlier detection in trajectory streams should exploit their continuity property.

3.1.9 Collective Behavior

When searching for outlier trajectory streams, the goal lies not only in finding a single outlier point within a trajectory stream, but also in finding a sub-sequence of consecutive trajectory points such that they all collectively exhibit an anomalous behavior. This makes the problem of outlier detection in trajectory streams more challenging than that of outlier detection in ordinary point streams because the objects that are being classified as outliers are significantly more complex.

Challenge: A technique for outlier detection in trajectory streams should consider not only the behavior of each individual data point in the streams, but also the behavior of the sub-trajectories in the streams when making outlier-ness decisions.

3.1.10 Contextual Behavior

When searching for outliers in a dataset, not just of trajectory data streams, there usually are two sets of attributes: behavioral attributes, and contextual attributes. The set of behavioral attributes contains all those attributes that are of interest to an application. For example, when searching for outliers in a data stream of stock prices, the stock price is the behavioral attribute. The set of contextual attributes contains all those attributes that determine the "proximity" between data points. It is expected that points that are nearby according to the values of their contextual attributes will exhibit similar behavioral attribute values. In the case of a stock price data stream, time is the contextual attribute because it is expected that there is some correlation between the prices of a stock at nearby time instants.

According to the informal definition of trajectory outlier in Section 2.3, a trajectory stream is an outlier if it exhibits an unusual behavior when compared to its previous behavior, or to the behavior of other nearby trajectories in the dataset. Therefore, from this definition, we see that time is a contextual attribute of the trajectory stream outlier detection problem because the points that make up a trajectory are smooth functions of the time parameter. Similarly, the spatial attributes, which are behavioral attributes, can also be part of the set of contextual attributes because the trajectories of nearby objects tend to exhibit similar behaviors.

To accurately classify a trajectory stream T as an outlier, it is important to take into account the *context of* T, which includes both the time and space attribute sets. This is because T might or might not be an outlier, depending on the current behavior of other nearby trajectories (spatial context). For example, with drivers on the road, a vehicle that takes a detour might be an outlier in ordinary circumstances. However, if there is an accident on the road, then that trajectory with a detour is not an outlier because many other trajectories from other drivers are likely taking that same path. Similarly, a trajectory T might or might not be an outlier, depending on the time context. For example, a person might walk to school during the Spring, but might drive to school in the Winter. In other words, to classify a trajectory as an outlier, there is a need for computing the conditional probability of a trajectory being an outlier given the context.

This challenge is significant because of at least two reasons. The first is that it may be difficult to determine the extent of the temporal and spatial contexts of T. In other words, it may be difficult to decide which points of T are significant to determine its outlier-ness, and it may be difficult to determine which trajectory streams are considered as being close to T. The second is that, even if the temporal and spatial contexts can be successfully identified, then there is a scalability issue during processing because computing the conditional probability of a trajectory stream being an outlier given the context could potentially involve examining a large amount of data.

Challenge: A technique for outlier detection in trajectory streams should take the temporal and spatial contexts of the streams into consideration.

3.2 Challenges of Outlier Detection in Trajectory Streams using GPUs

In addition to the issues discussed in Section 3.1, a GPU technique for outlier detection in trajectory streams needs to address another set of GPU-specific issues, which are now discussed.

3.2.1 *Low Throughput of the Host-GPU Bus*

GPUs are connected to their host machines through the PCI-express (PCIe) bus, and this bus has a remarkably lower throughput than the GPU's global memory [27], and the host computer's RAM memory [28]. This low PCIe throughput, combined with the high sampling rates of trajectory streams, represents a challenge when processing outlier trajectories because this may hinder the transfer of the data points that make up the trajectories to and from the GPU. The reason for this is that since the PCIe bus has a low throughput, relative to the high instruction throughput of GPUs, it is not cost efficient to send the incoming trajectory points to the GPU as soon as they arrive. Rather, these points may need to be buffered in the host's RAM until the number of points collected there reaches a size that makes it cost effective to send these buffered points to the GPU [29]. This may, however, force a trajectory stream outlier detection algorithm to make a decision based on outdated data, negatively impacting the accuracy of the technique.

The challenge therefore lies in determining how long to wait before sending and retrieving the data to and from the GPU in order to optimize the transfer times, while at the same time guaranteeing the following two things. First, the data with which the GPU algorithm is working is not too obsolete; and second, the anomaly detection results are sent back to the host in a timely manner that allows taking any immediate action in response to those anomalies found.

This challenge can also produce a synergistic adverse impact when it interacts with the concept drift challenge. The reason is that it is conceivable that in scenarios where the spatio-temporal distribution of the trajectories changes at a high rate, then, because of the low PCIe bus throughput, the trajectory stream outlier detection algorithm for GPUs might not be able to quickly adapt to this non-stationary behavior. One example application that illustrates this scenario is that of drunk driver detection in a highway. In this case, the spatial distribution of the trajectories can change very quickly as a consequence of weather, big events, other accidents, etc.

Challenge: A technique for outlier detection in trajectory streams using GPUs should work in a way that minimizes the cost of the PCIe communication, but that also guarantees two things. First, the decision about the outlier-ness of an incoming trajectory is made based on an up-to-date probabilistic model for the data; and second, the anomaly detection results are sent back to the host on time to take any actions in response to the moving object's behavior.

3.2.2 *Small Memory Space of GPUs*

Compared to the sizes of the host computer's RAM, the memory space of GPUs is very small: in the order of 10s of GBs in the high-end GPU models. In the era of Big Data, this is incredibly small [30]. This challenge, when combined with the infinite size of trajectory streams and the low throughput of the PCIe bus, means that a trajectory stream outlier detection technique needs to detect the trajectory outlier streams using only a small portion of the overall data at a time. This could in turn entail having to build a concise summary of both the trajectory dataset and of the trajectory stream that is under scrutiny.

Challenge: To deal with the small memory space of GPUs, a technique for outlier detection in trajectory streams should work in an incremental fashion by keeping a data structure that summarizes the behavior of the trajectories in the dataset, and that avoids storing all the trajectories in the GPU's memory space.

3.2.3 *Load Balancing*

Load balancing refers to striving to ensure that all computational units in a GPU (Streaming Multiprocessors, SMXs, GPU threads, etc.) perform a similar amount of work in order to avoid that the execution time of a single unit ends up dominating the execution time of the whole algorithm. In the case of trajectory stream outlier detection on GPUs, this problem is of special significance. The reason is that a GPU may be working on more than one trajectory stream at the same time. However, because of the challenges of different sampling rates, some trajectories may have more constituent points than others, and this can make the problem of load balancing more complex because some trajectories will be bigger than others. Moreover, because of the concept drift challenge, the distributions of the different trajectories change in time, which means that the relative sizes of the trajectories will also vary in time, making the problem of load balancing of trajectories more complicated.

Challenge: A technique for outlier detection in trajectory streams using GPUs should distribute the work among the threads in a block, and among different blocks such that no single unit ends up determining the overall execution time of the algorithm.

3.2.4 *Low Global Memory Bandwidth Relative to the Number of Threads*

In GPUs, there are often thousands of threads contending for access to the GPU's slow global memory. This implies that every time there is a global memory read instruction, thousands of memory transactions need to be performed (one per thread). To deal with this problem, GPUs (just like regular CPUs) are equipped with caches that can exploit the spatial locality of global memory accesses in order to reduce the traffic through the memory controller. However, in order to take advantage of such caches, threads in a GPU need to access the global memory following patterns that respect the spatial locality. When threads access the global memory respecting this spatial locality, the cache can reduce the contention for memory bandwidth, in which case it is said that the GPU has *coalesced those global memory accesses*.

The conditions that are required for coalescing global memory accesses vary according to the type of the GPU. In general, a coalesced memory access occurs when threads in a warp, the smallest unit of parallelism in a GPU, simultaneously access a sequence of contiguous locations in the GPU's global memory. For example, if *a* is an integer array, memory accesses to *a* can be coalescing when the threads $t_{32}, t_{33}, \ldots, t_{63}$ access the array elements *a*[*0*], *a*[*1*], ..., *a*[*31*]. The advantage of memory coalescing is that only a single global memory transaction, instead of multiple separate transactions, is performed to access all those locations, and this reduces the demand for memory bandwidth, which, in the case of GPUs, constitutes one of the dominating factors for performance [28].

An example of a place where global memory coalescing could be potentially challenging is when appending the most recent data points into the trajectory data structures in the GPU. This is because the set of the most recent data points, which contains the latest points of all trajectories, is likely going to be arranged in an array. However, consecutive elements in this array of points may correspond to different trajectories, so that when merging this new set of points with the preexisting trajectory data structures on the GPU, there could be non-coalesced global memory accesses.

Challenge: A technique for outlier detection in trajectory streams using GPUs should arrange data in the GPU's global memory in a way that minimizes the number of un-coalesced global memory accesses.

4. CONCLUSIONS

There exist many applications of outlier detection in trajectory streams, like senior citizen monitoring, drunk driver detection, alerting the military about the presence of nearby enemies, etc. This problem is one of Big Data because it can involve potentially large amounts of uncertain data coming at high speeds from location sensors. These applications are critical in the sense that they require a timely identification of outliers. One way to address this need for real-time outlier detection is through the use of GPUs. This paper discusses the research issues that an outlier detection technique should address when leveraging GPUs to detect outliers in trajectory streams.

REFERENCES

[1] V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR),* vol. 41, no. 3, 2009.

[2] D. Hawkins, Identification of Outliers, Chapman and Hall, 1980.

[3] M. Gupta, J. Gao, C. Aggarwal and J. Han, Outlier Detection for Temporal Data, Morgan Claypool Synthesis Lectures, 2014.

[4] M. Stonebraker, U. Çetintemel and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record,* vol. 34, no. 4, pp. 42-47, 2005.

[5] Z. Galić, Spatio-temporal Data Streams, Springer, 2016.

[6] C. C. Aggarwal, Outlier Analysis, 2nd ed., Springer, 2017.

[7] Y. Bu, L. Chen, A. W.-C. Fu and D. Liu, "Efficient anomaly monitoring over moving object trajectory streams," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[8] Y. Yu, L. Cao, E. A. Rundensteiner and Q. Wang, "Outlier Detection over Massive-Scale Trajectory Streams," *ACM Transactions on Database Systems (TODS),* vol. 42, no. 2, June 2017.

[9] Y. Ge, H. Xiong, Z.-h. Zhou, H. Ozdemir, J. Yu and K. C. Lee, "Top-Eye: top-k evolving trajectory outlier detection," in *CIKM '10 Proceedings of the 19th ACM international conference on Information and knowledge management*, 2010.

[10] Y. Yu, L. Cao, E. A. Rundensteiner and Q. Wang, "Detecting Moving Object Outliers in Massive-Scale Trajectory Streams," in *Knowledge Discovery and Data Mining (KDD)*, 2014.

[11] L.-a. Tang, X. Yu, S. Kim, J. Han, C.-C. Hung and W.-C. Peng, "Tru-Alarm: Trustworthiness Analysis of Sensor Networks in Cyber-Physical Systems," in *IEEE International Conference on Data Mining*, 2010.

[12] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal and P. Dubey, "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in *37th annual international symposium on Computer architecture*, 2010.

[13] H. Cao, O. Wolfson and G. Trajcevski, "Spatio-temporal data reduction with deterministic error bounds," vol. 15, no. 3, pp. 211-228, 2006.

[14] J.-G. Lee, J. Han and X. Li, "Trajectory Outlier Detection: A Partition-and-Detect Framework," in *Proceedings of the IEEE 24th International Conference on Data Engineering*, 2008.

[15] D. Lustig and M. Martonosi, "Reducing GPU offload latency via fine-grained CPU-GPU synchronization," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[16] P. Li, Y. Luo, N. Zhang and Y. Cao, "HeteroSpark: A heterogeneous CPU/GPU Spark platform for machine learning algorithms," in *IEEE Conference on Networking, Architecture and Storage*, 2015.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

[18] M. Garland and D. Kirk, "Understanding Throughput-oriented Architectures," *Communications of the ACM,* vol. 53, no. 11, p. 58–66, 2010.

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

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

[21] Y. Ge, H. Xiong, C. Liu and Z.-H. Zhou, "A Taxi Driving Fraud Detection System," in *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, 2011.

[22] M. Stonebraker, U. Çetintemel and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record,* vol. 34, no. 4, pp. 42-47, December 2005.

[23] C. Ma, H. Lu, L. Shou and G. Chen, "KSQ: Top-k Similarity Query on Uncertain Trajectories," *IEEE Transactions on Knowledge and Data Engineering,* vol. 25, no. 9, 2013.

[24] F. Cagnacci, L. Boitani, R. A. Powell and M. S. Boyce, "Animal ecology meets GPS-based radiotelemetry: a perfect storm of opportunities and challenges," *Philosophical Transactions of the Royal Society of Biological Sciences,* vol. 365, no. 1550, 2010.

[25] N. J. Hill, I. T. M. Hussein, K. R. Davis, E. J. Ma, T. J. Spivey, A. M. Ramey, W. B. Puryear, S. R. Das, R. A. Halpin, X. Lin, N. B. Fedorova, D. L. Suares, W. M. Boyce and J. A. Runstadler, "Reassortment of Influenza A Viruses

in Wild Birds in Alaska before H5 Clade 2.3.4.4 Outbreaks," *Emerging and Infectious Diseases,* vol. 23, no. 4, 2017.

[26] H. Cao, O. Wolfson and G. Trajcevski, "Spatio-temporal data reduction with deterministic error bounds," *The VLDB Journal,* vol. 15, no. 3, 2006.

[27] Nvidia, "Cuda C Best Practices Guide v.9.1," March 2018. [Online].

[28] D. Kirk and W.-m. Hwu, Programming Massively Parallel Processors: A Hands on Approach, San Francisco: Morgan Kauffman, 2013.

[29] Nvidia, "CUDA Programming Guide v.9.1.85," 2018 March. [Online].

[30] X. Wu, X. Zhu, G.-Q. Wu and W. Ding, "Data Mining with Big Data," *IEEE Transactions on Knowledge and Data Engineering,* vol. 26, no. 1, 2014.