# Towards a GPU Accelerated Spatial Computing Framework

Harshada Chavan Rami Alghamdi Mohamed F. Mokbel

Department of Computer Science and Engineering
University of Minnesota, Minneapolis, MN 55455
{chava057, alghamdi, mokbel}@cs.umn.edu

***Abstract*—Ease of availability of spatial data has increased the interest in the domain of spatial computing. Various services such as Uber, Google maps, and Blue Brain Project have been developed that consume and process such spatial data. Spatial data processing is not only data intensive but also compute intensive. A lot of efforts have been made by the spatial computing community to tackle the problems due to huge volumes of data. However, unfortunately, not enough attention has been given to address the compute intensive nature of the problem. In parallel to the advancements in spatial domain, Graphics Processing Units (GPUs) have emerged as compelling computing units. A lot of work has been done in spatial domain to leverage the computing power of GPUs. However, to the best of our knowledge, none of the work present a holistic system. In this paper, we propose a vision for a GPU accelerated end-to-end system for performing spatial computations. Our envisioned system supports a plethora of spatial operations ranging from basic operations, computational geometry operations to Open Geospatial Consortium (OGC) compliant operations. Our system exploits the power of CPU-GPU co-processing by scheduling the execution of spatial operators either on CPU or GPU based on a cost model. Within the framework of our system we discuss the challenges and open research problems in building such a system. We also provide some preliminary results to show the computational gain achieved by performing spatial operations on GPUs.**

## I. Introduction

Recently, the ease of accessibility of spatial data has increased interest in the field of spatial computing. Examples of freely available spatial data include the Landsat satellite images made available by United States Geological Survey (USGS) [1], geo-tagged Twitter data [2], and the world map available at OpenStreetMap [3]. Numerous services such as Uber [4], Google Maps [5], and Blue Brain Project [45], have been developed taking advantage of such data. Though such available spatial data is very rich and can enable a myriad of important spatial applications and findings, processing and mining useful information out of such data is hindered by two main challenges. First, such datasets come in huge amounts, making spatial data processing data intensive. For example, NASA archive of satellite earth images has more than 1 PB and increases daily by 25 GB [6], medical devices produce spatial images (Xrays) at a rate of 50 PB per year [7]. Second, operations on such spatial data are computationally intensive. For example, it takes roughly 20 hours on a single machine to compute the spatial self-join of a polyline table with 73 million records representing road networks in USA [53].

The data intensive challenge is largely addressed by the recently developed Big Spatial Data systems, e.g., SpatialHadoop [26], Hadoop-GIS [15], MD-HBase [51], and SciDB [59]. Yet, unfortunately, the computationally intensive challenge has not received enough attention. For example, spatial joins are typically performed in two steps, namely *filtering* and *refinement*. The *filtering* step selects the candidate objects using their minimum bounding rectangles (MBRs). In the *refinement* step, the final result is computed by comparing the actual geometries of candidate objects with the query geometry. Total cost of the refinement step consists of two factors: the I/O cost and computational cost. In case of complex geometries i.e., polygons with large number of vertices, the computational cost largely dominates the total cost of the refinement step [60]. For example, the *refinement* step in SpatialHadoop is the bottleneck in further improvement of the performance of spatial join [26].

Meanwhile, advancements in Graphics Processing Unit (GPU) chips in terms of more functionality and programmability have established them as a compelling computing platform for compute intensive tasks. GPUs were transformed from being co-processors beside the CPUs for very narrow tasks, to being used side by side with the CPUs [46]. Taking advantage of the CPU-GPU co-processing seems rather practical toward speeding up computations. During the last decade, GPUs have been used for accelerating computations in a wide range of domains that include physics [49], medical image processing [56], and biology [21]. Availability of thousands of cores make GPUs a promising hardware for performing compute intensive tasks.

So far, few attempts have been made to leverage the enormous power of GPUs to execute the computationally intensive spatial operations (e.g., see [40], [60], and [64]). However, to the best of our knowledge, all existing approaches represent stand alone solutions as opposed to an end-to-end system. In other words, these approaches are designed as algorithmic designs for specific spatial operations, e.g., processing streams of spatial k-NN queries [40], expediting the refinement phase in spatial join [60], and spatial cross-comparison [64]. Such stand alone approaches would have limited functionality if they are isolated from a full-fledged usable system.

In this paper, we present our vision for the first holistic GPU-aware system for spatial computing. The system would

This work is partially supported by the National Science Foundation, USA, under Grants IIS-1525953, CNS-1512877, IIS-0952977 and IIS-1218168.

exploit the computing power of GPUs for indexing, querying, visualizing, and analyzing big spatial data. We believe that having an end-to-end system is strongly preferred over having several scattered solutions. An end-to-end system provides a complete execution package for all spatial operations. Thus, avoiding the need to switch between programs for different spatial operations. Our envisioned system takes advantage of the CPU-GPU co-processing. It wisely decides where to execute a query operation for optimal execution- on CPU or GPU. A query optimizer equipped with a cost model helps in taking this decision. For executing spatial operations on GPUs, the execution engine consists of GPU kernels written for each of the spatial operators. The job of the query processor is to invoke the execution of the operator either on CPU or on GPU, as per the optimized query plan.

There are several open challenges in building such a GPU-aware system for spatial computing. The challenges can be classified into three main categories. (1) **System implementation**. System implementation poses challenges related to the approach of building the system, portability of the system and formulating the cost-model. The system could be built from scratch or by extending an existing big spatial data system. It is interesting to evaluate the pros and cons of both the approaches. Secondly, tt is important to decide the programming language used for implementing the system. This decision will determine the GPU hardware supported by the system i.e. the *portability* of the system. The programming toolkits available at present bind the software with a specific GPU vendor. For example, a program implemented in CUDA [8] can only be executed on NVIDIA GPUs. There are some generic toolkits which are supported by all hardware vendors, for example OpenGL [9]. Yet, programs written in such languages are not as optimized because they do not take advantage of the underlying hardware architecture. Therefore, it is important to make a decision between portability and efficiency. Additionally, we need to build a *cost model* that will help the query optimizer to decide the target hardware for executing a query operator. This decision combines data and hardware parameters, e.g., metadata, input size, operation type, and available resources on each hardware. (2) **Algorithmic design**. Implementing GPU aware spatial operations is much more than merely creating multi-threaded versions of single threaded algorithms. Algorithms should be designed in a way that keeps the workload balanced across all cores. Multiple cores must work together and synchronize their CPU memory accesses for effective query processing and to ensure high memory bandwidth [40]. Since GPUs have limited device memory, the algorithm must make minimum data transfers between CPU and GPU to avoid delays due to low bandwidth bus. (3) **Data representation**. GPUs are good at processing array-based data storage [34]. On the other hand, CPUs are capable of handling more complex data structures which are more efficient than arrays. Deciding on data representation format that is suitable and efficient on both CPU and GPU is a challenge.

The rest of the paper is organized as follows. Section II highlights related work. Section III gives a background knowledge about GPUs. Architecture of our envisioned system is given in Section IV. The challenges in building our system are discussed in Section V. Finally, Section VI concludes the paper.

## II. Related Work

This section highlights related work to this paper in terms of supporting spatial operations in big data systems, building systems for big spatial data, and spatial operations on the GPU.

**Spatial operations in Big Data systems.** Due to the need to process big spatial data, several attempts were made to support various spatial operations on top of existing big data systems. This includes: (1) constructing *R-trees* through data partitioning on a Hadoop cluster [20], (2) supporting *range queries* by scanning partitions in Hadoop Distributed File System (HDFS) [44], [73], (3) supporting various kinds of $k$*-nearest-neighbor* ($k$NN) queries on Hadoop using either a brute force approach for traditional $k$NN queries and *all-nearest-neighbor* queries [73] or a Voronoi diagram for traditional $k$NN queries [17] and *reverse-nearest-neighbor* queries [17], (4) supporting *spatial join* on either Hadoop map-reduce framework [73] or Spark system [70], and (5) supporting *kNN join* on map-reduce-based frameworks [43], [71].

**Big spatial data systems.** Following the need for more efficient processing of big spatial data, it becomes of essence to go beyond supporting specific spatial operations to building full-fledged systems for big spatial data. To this end, various systems were introduced, which include: (1) SpatialHadoop [25], [26] that injects spatial-awareness inside every layer of the core Hadoop engine including indexing [22], query processing [24], and visualization [27], [28]. (2) HadoopGIS [15], [16] which extends Hive [63] to support spatial indexing and query processing. (3) MD-HBase [50], [51], which extends HBase [10] for spatial operations. (4) GeoSpark [69], which extends the core of Spark engine [70] to support in-memory spatial operations. (5) Sphinx [23], which injects spatial indexing and query operations inside the engine of Cloudera Impala [38]. (6) Parallel SECONDO [32], [42] which supports trajectory analysis using SECONDO instances [30], [31] on a hadoop cluster. (7) various industrial systems that include ESRI GIS tools for Hadoop [66], GeoMesa [11], and Geotrellis [12].

**Spatial operations on the GPU.** Existing work that leverages the power of the GPU to support spatial operations is only limited to specific operations that include: (1) supporting *range queries* [18], [60] through introducing an intermediate GPU-based filtering step to accelerate spatial selections. (2) Supporting *spatial intersection* queries [64] through a GPU-based algorithm for finding the overlap between two polygons. (3) Supporting *kNN* queries through GPU-based implementations for Voronoi diagrams on road networks [29], parallel locality-sensitive hashing [52], truncated bitonic sort [57], a tree search method on $kd$-tree [48], multi-staged framework for continuous queries [65], or a framework over moving objects [40]. (4) extending Impala to support *range queries*, *aggregate queries*, and *spatial joins* on the GPU [68]. Here, the role of Impala is to manage spatial data and control spatial operations in the cluster. Each node offloads spatial indexing, and spatial filtering and refinement operations to the local host GPU. Data retrieved on each node are locally indexed on the fly on the GPU using *R-trees* index [67] and queries are locally executed on the GPU [72]. The system proposed here lacks a lot of features that other big spatial data systems support such as global spatial indexes on on Hadoop storage layer. Moreover, it does not support other important spatial

operations (e.g., $k$NN queries and computational geometry operations), and visualization. It is far from being an end-to-end system that fully exploit the GPU for spatial operations.

**Our proposed work.** As it is clear from the above related work, processing big spatial data had started by supporting only specific operations. Then, full-fledged systems were developed, which proved to be much more valuable. However, when leveraging the power of the GPU, existing work is limited to only specific operations. So, it becomes natural to exploit the idea of building a GPU-aware full-fledged system for spatial operations, which is what we are proposing in this paper. Similar to the success of full-fledged systems developed for big spatial data over supporting specific spatial operations, we envision that building a GPU-aware full-fledged system will be way superior and usable over supporting specific spatial operations on the GPU.

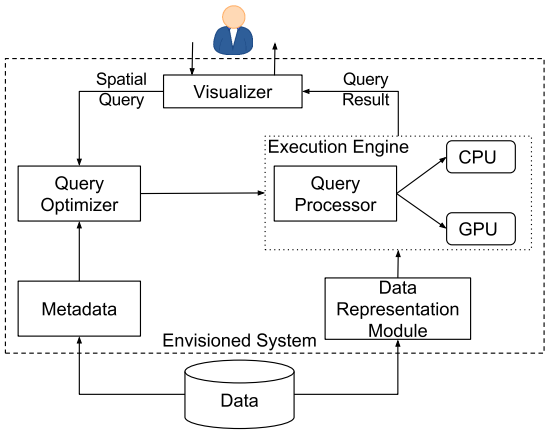



Fig. 2: System Architecture

## III. Background

This section briefly gives a background on GPUs starting from its early adaptation as a new hardware (Section III-A) and going through various architectures for integrating with CPU (Section III-B).

### A. The GPU for Display Systems

The very first invention for GPUs came from the need to accelerate display routines in display systems. The first system allowing graphical interaction with users, Sketchpad [61], along with other systems at that time [33], [37] needed a separate computer, named *display system*, for processing display tasks for three reasons: (1) to free the main computer from the repetitive tasks of display routines, (2) to overcome the limits of the communication lines at the time that hampered the transfer rate of display data, in a time-shared central computer environment, and (3) to have a flicker-free display, in 2D and 3D applications, the display have to be updated 30 times per second which led to occupying the main processor most of the time. It was not clear whether to have a full-capable processor or a customized one with less functionality for display routines or not [47]. Then, in 1968, the clipping divider [58] came out as the first specialized hardware unit designed for the purpose of redrawing lines efficiently in 2D and 3D environments. The clipping divider is considered to be the first fixed hardware pipeline that accelerates graphics tasks beside the processor. It was used in a head-mounted three dimensional display [62] and in a commercial system called Line Drawing System Model 1, which is considered the first GPU in the market. As graphics applications kept emerging, the GPU kept improving for accelerating graphics tasks in applications such as real-time video systems, pilot simulators, and 3D modeling.

### B. The CPU-GPU Architecture

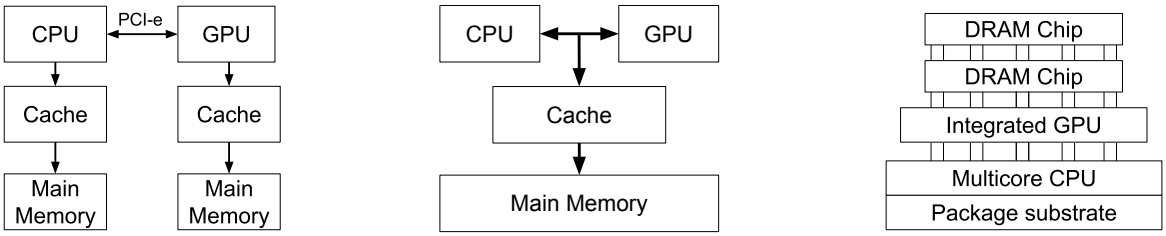
There are three CPU-GPU architectures (Figure 1): (1) The discrete (dedicated) GPU (Figure 1a), where the GPU card is installed on the computer and communicates with the CPU through the PCI-e bus. The PCI-e has a low bandwidth and can become a bottleneck in the GPU performance where transfers of large data are involved [35]. Meanwhile, the discrete GPU is used for high performance computing tasks due to having high number of cores, large memory size and bandwidth and it is costly compared to other GPU architecture. For example, the NVIDIA GPU card GTX TITAN X [13] has 3072 CUDA cores with 1000 MHz base clock speed and memory bandwidth of 336.5 GB/s. (2) The coupled (integrated) architecture (Figure 1b), where the CPU and the GPU share the cache and the physical memory. This is more affordable and more common on personal computers for routine tasks. The HD 4000 GPU in Intel's new Ivy Bridge CPU is an example of a coupled GPU architecture. Integrated architecture eliminates the need to transfer data over PCI-e bus, but provides less performance since both the CPU and the GPU has to compete over memory. (3) The 3D die-stacking (Figure 1c) is a new emerging technology where the silicon dies are stacked on top of one another. In this architecture, the GPU accelerator can be tightly-integrated with the CPU.

## IV. System Architecture

The ultimate goal of our envisioned system is to provide an end-to-end support for the spatial queries using GPUs. The system accepts SQL-like spatial queries from users, optimizes them using a full-fledged query optimizer, executes the query plans using CPU-GPU co-processing, and provides a module for visualizing the results. The architecture of the system is shown in Figure 2. The components of our system share similarities with the components of traditional data management systems. However, there are functional differences between them. The presence of a different type of hardware (i.e., GPU) adds responsibility to some of the existing components. In our system, we envision to extend three main components, namely, data representation (Section IV-A), query processor (Section IV-B), and query optimizer (Section IV-C).

### A. Data Representation Module

Architectural and functional differences between CPU and GPU lead to different requirements in terms of data storage format. GPUs prefer data to be stored in contiguous memory locations such as arrays. Array format provides two advantages: (1) It hides the inefficiency of GPUs in pointer chasing [54]. In arrays, locations can be computed using index arithmetic, instead of following non-contiguous memory pointers. (2) It helps in making coalesced read requests from GPU to CPU memory (we discuss more about this in Section V-B). On the




(a) Discrete. (b) Integrated. (c) 3D die-stacking.

Fig. 1: CPU-GPU architectures

other hand, CPUs can efficiently process data stored in more complex but efficient data structures. In order to cater to the needs of both CPU and GPU, our envisioned system provides a data representation module that takes care of representing data in a suitable format for both, CPU and GPU. For example, if B+-Tree index search operation is to be performed on GPUs, the tree structure needs to be represented in an array format. The array-based version of a B+-Tree, CSS-Tree [54], which stores the keys into an array can be used for representing data. This adds a new responsibility of representing data in a GPU friendly format, to the data-representation module. The data representation module takes care of adding array based B-Tree, R-Tree, and spatial hash indexes for GPUs.

### B. Query Processor

In addition to the CPU-based implementations of spatial operations, query processor contains GPU specific kernel implementations of these operators. Our envisioned system supports various spatial operators that can be classified into the following three categories: (1) *Basic spatial operators*. These operators are necessary for any spatial data processing engine, and include operators like spatial joins, spatial range, and k-NN. (2) *Computational geometry operators*. Computational geometry operations are important for visualization. They are also used for supporting basic spatial operations. For example, Voronoi diagrams can be used for supporting spatial k-NN operations. In addition, computational geometry operations are compute intensive. For example, to calculate the farthest pair for a set of given points, we first need to compute the convex hull of all points, then calculate the distance between every pair of points on the convex hull to find the farthest pair. These operations can be easily parallelized and therefore can be accelerated using GPUs. Other examples of computational geometry operations include polygon union, skyline, and closest pair. (3) *Open Geospatial Consortium (OGC [14]) compliant operations*. To follow the industry standards and to increase the adaptability of the system, our envisioned system supports OGC-compliant operations that include *Overlaps* to test if the given two geometries overlap, *Within* to test if one geometry is completely inside the other geometry. Some steps in many of these operations are compute intensive. For example, the *refinement* step in spatial join and range queries for complex geometries. Such steps should be executed on GPU to expedite the computation. The responsibility of the query processor is to execute the operator either on CPU or on GPU as per the optimized query plan.

### C. Query Optimizer

Traditionally, the query optimizer takes into account the metadata information and hardware specifications to generate an optimal query plan. In our envisioned system, the query optimizer has an additional responsibility of determining the optimal hardware for executing an operator. Our query optimizer is equipped with a cost model that takes this decision. The cost model considers the following four parameters while making its decision [19]: (1) The operation type, (2) the features of the underlying data, i.e., data size, data type, and operation selectivity, (3) hardware specifications, i.e., number of cores and memory bandwidth, and (4) present state of the hardware, e.g., even if an operation runs faster on GPU, it should be executed on CPU if GPU is overloaded.

Among the above four parameters, we have already investigated the first parameter, i.e., *the operation type*. We performed a preliminary experiment to evaluate the performance of a multi-key index search operation on CPU and GPU. Multi-key index search is an important step in various spatial operations such as spatial hash joins [41], k-NN queries [40], and range queries [18], [60]. We implemented an array-based B+-Tree index, namely CSS-Tree [54] on GPU using CUDA. CSS-Tree can be used to index partition ordering generated by space filling curves [55] such as Z-curve or Hilbert curve. Space filling curves are heavily used in spatial domain for multi-dimensional indexing [39], [40]. For this experiment, we used an Intel(R) Xeon(R) CPU E5405 with 2 GHz clock speed, 8GB memory and 8 cores CPU and an NVIDIA GeForce GTX 470 with 625 MHz clock speed, 1.25 GB memory and 448 cores GPU. We performed the experiment for varying values of *tree order* and for searching 500 key values. Here, *tree order* acts as an indication of the index size. The results of the experiment are shown in Figure 3. It can be seen from the results that the execution time on GPU is significantly smaller than that of CPU. The multi-key index search operation performs 6 to 13 times faster than that on CPU.

The difference in execution time increases exponentially with the increase in the index size. Such behavior can be explained by looking into the architectures of CPU and GPU. The architecture and therefore, the program execution model of GPUs is significantly different than that of CPUs. GPUs are well suited for easily parallelizable operations [53] like multi-key index search. Multiple cores of GPUs can be exploited to perform parallel index searches. However, the multi-key search on CPU is executed sequentially.

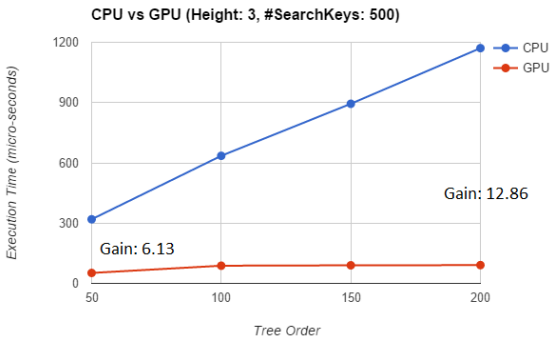



Fig. 3: Effect of index size on execution time

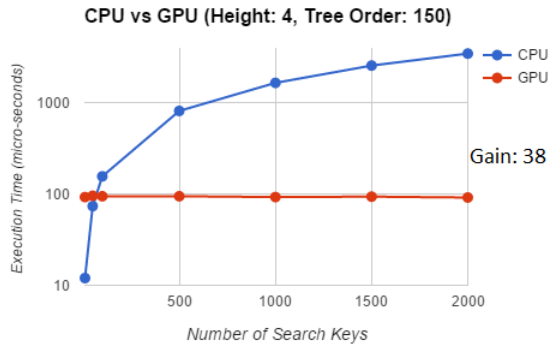



Fig. 4: Effect of degree of parallelism on execution times

We performed another experiment to evaluate the effect of increasing parallelism on the execution time of multi-key index search operation on CPU and GPU. For this experiment, the CSS-Tree index under consideration was of height 4 and the order of the tree was 150. We performed multi-key search operation for varying degree of parallelism indicated by the number of search keys. The results of our experiment are shown in Figure 4. It can be seen from the results that increase in number of search keys does not affect the execution time on GPUs but it increases the execution time on CPUs exponentially. Since the parallelism on GPUs is hardware supported, it does not incur much operational overheads due to increase in degree parallel execution. An interesting behavior can be observed for the lower values of number of keys. For number of keys less than 100, the execution time of CPU is lower than that of GPU. This is due to the facts that a single core of CPU is faster than a single core of GPU. It is the presence of a large number of cores that makes GPUs faster than CPUs in the end. With the help of multi-threading, CPUs can also be used for performing parallel index search. However, this parallelism is not hardware supported. Thus, it incurs overheads as the degree of parallelism increases. In both the experiments the index was fully loaded in the GPU and CPU memory.

We performed a third experiment to evaluate the performance of the *filtering* step of spatial join on GPU compared to CPU. Two datasets with 2.5K and 300K records were used as input. We performed a nested loop filtering step where the number of comparisons were equal to the cross product of the sizes of two datasets. For this experiment, we used an Intel i7 with 2.5GHz of clock speed as CPU and an NVIDIA GeForce GT 750M with 0.9GHz clock speed, 2GB memory and 384 cores as GPU. We implemented the *filtering* step using CUDA. The results of our experiment are shown in Figure 5. The filtering step in spatial join is easily parallelizable operation. Multiple cores of GPU perform multiple comparisons in parallel to speed up the computation, whereas CPU performs all comparisons sequentially. This leads to a eight times performance gain on GPU.

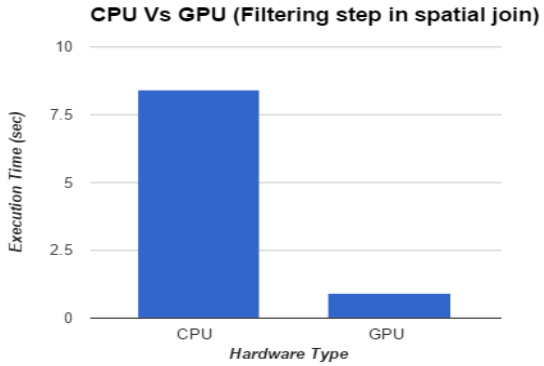



Fig. 5: MBR filtering performance

These experiments prove that executing easily parallelizable operations on GPU yields better performance in terms of execution time than on CPUs.

On the other hand CPUs are far better at executing programs with complex control flows [53]. GPUs are known to be inefficient at computing complex operations such as spatial overlay and join. These operations involve building and accessing irregular data structures such as graphs, priority queues. The accesses to the underlying data structures are irregular and data-dependent [53]. Therefore, operation type is an important parameter in deciding the execution placement.

## V. Challenges

This section outlines the main challenges and research directions towards building a GPU accelerated framework for spatial computing. These challenges can be divided into three categories: (1) Challenges related to system implementation (Section V-A), (2) challenges related to algorithmic changes (Section V-B), and (3) challenges related to data representation (Section V-C).

### A. System Implementation

System implementation poses several challenges related to the approach of building the system, making the system portable, and formulating a cost model to decide the execution placements of operators.

**Building the system.** There are two primary approaches of building such a system. First approach is to build all the components of our envisioned system from scratch. This approach involves a lot of redundant work and is very time consuming. Second approach is to extend an existing system. Existing Big Spatial Data systems such as SpatialHadoop [26] and HadoopGIS [15] can be extended to be GPU aware. This

approach is more effective and saves redoing the redundant work. It is still a challenge to select the most suitable system out of all the existing ones. However, with this approach, the system inherits the limitations of the underlying system. It is an interesting problem to evaluate the pros and cons of each of the approaches.

**Portability.** There are multiple software development toolkits available for programming GPUs. Moreover, certain toolkits bind the software program to a specific vendor. For example, programs written in CUDA [8] can only be executed on NVIDIA GPUs. The simplest solution to this problem could be implementing different versions of the same algorithm for each hardware type. However, this approach incurs a lot of development and maintenance cost. There are some general toolkits that are hardware agnostic such as OpenGL [9], but programs written in OpenGL are less efficient as they do not take advantage of the underlying optimizations within the hardware. Therefore, there is a trade-off between efficiency and portability. One solution to this problem is to use kernel-adapter based design incorporated in OmniDB [74]. OmniDB implements a portable query processing kernel (qKernel) that contains parameters and configurations that are architecture specific. These parameters are provided by the adapters to adapt kernel as per the target architecture. However, OmniDB does not support spatial data types.

**Formulating cost-model.** Formulating a cost model that considers data and hardware information to decide the execution placement of a spatial operator is not trivial. Extensive experiments need to be done to study the effect of each of the parameters on the execution time. Additionally, the cost model must be able to adapt to the system load.

### B. Algorithmic Design

Implementing a spatial operation for GPUs is much more than simply transforming the single threaded version of the program into a multi-threaded version. The algorithm should utilize the limited memory on GPU effectively, reduces data transfers between CPU and GPU, and keep load balanced across all the cores of GPU.

**Limited GPU memory.** GPUs have limited device and on-chip memory. The device memory is usually insufficient for data intensive spatial applications that incur a lot of data transfer between the CPU memory and the device memory. These transfers take place over the low bandwidth (4–8 GB/sec [35]) PCI-e bus, leading to delays in computations. Therefore, to achieve performance gains on the GPUs, data transfers should be minimized. There are a couple of ways for doing this. First, is to use coupled CPU-GPU architecture [35] instead of discrete GPUs. However, the GPU in the coupled architecture is much less powerful than the GPUs in the discrete architecture [36]. The second way of achieving less data transfers is to coalesce the memory accesses from GPU to CPU memory, which is achieved by making algorithmic changes.

**Algorithmic change.** In order to exploit the full potential of GPUs, new GPU specific algorithms are needed. Single threaded algorithms for problems which are embarrassingly parallel (e.g., matrix multiplication and array-based operations) can be easily converted to a parallel multi-threaded algorithms. However, more complex algorithms for problems with irregular communication patterns (e.g., spatial joins and spatial overlays) are difficult to convert from a single threaded version to multi-threaded algorithms [53]. Also, algorithmic efforts are required to reduce data transfers over PCI-e bus. One way of reducing data transfers is by coalescing accesses to CPU memory. If multiple threads are running in parallel and accessing contiguous memory, read/write request from these threads can be clubbed together to avoid multiple data transfers over PCI-e bus. In order to take advantage of memory coalescing, multiple cores must co-ordinate their activities.

**Load balancing.** Uniform load balancing across all cores of GPU enhances the performance of computations on GPUs by ensuring full utilization of all the processing units. Load balancing is especially critical for spatial data computations due to potential large data skew. This leads to the need for careful partitioning of data and operations. Load imbalance can cause inactivity bubbles and severely degrades the performance [40]. This is especially more challenging with spatial data since partitioning should preserve spatial locality for better performance. Techniques such as space-filling curves [55] (e.g., Z curve or Hilbert curve) can be realized on the GPU.

### C. Data Representation

Our envisioned system consists of a data representation module to make the data available as per the needs of the hardware. The naïve way of fulfilling this requirement is to store data sets in two formats, one that is more suitable to CPU and one that is more suitable to GPU. However, the complexity of the problem increases if a number of different indexes are to be created on the same data. This approach will end up creating twice the number of indexes. Clearly, we need to invent a smarter technique to solve the problem of data representation.

## VI. Conclusion

The volume and complexity of processing spatial data is increasing rapidly. Existing approaches for processing spatial data are falling short of providing support for the high computational complexity. On the other side, GPUs have emerged as a competitive general computing platform in the last decade. Recent advancement in the GPUs hardware and their development tools have enabled general processing on GPUs. GPUs have been used in a wide range domain to solve compute intensive problems. They have also been used to solve spatial operations. However, none of the existing approaches present a GPU accelerated end-to-end system to support spatial computations. In this paper, we presented a detailed background on GPUs and related work. We envisioned a holistic system to support spatial operations using GPUs. We presented the system architecture and also discussed the open challenges and research directions in building our system. From our experiments, we presented the computational benefits of performing easily parallelizable spatial operations using GPUs. We believe that GPUs have the potential to be a successful spatial computing framework.

## References


[1] http://landsat.usgs.gov/Landsat_Search_and_Download.php.

[2] https://dev.twitter.com.

[3] http://planet.openstreetmap.org/.

[4] https://www.uber.com.

[5] https://www.google.com/maps.

[6] https://lpdaac.usgs.gov/about.

[7] http://www.eiroforum.org/activities/scientific_highlights/2012/201209_XFEL/index.html.

[8] http://developer.nvidia.com/object/cuda.html.

[9] https://www.opengl.org/.

[10] https://hbase.apache.org/.

[11] http://www.geomesa.org/.

[12] http://geotrellis.io/.

[13] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/.

[14] http://www.opengeospatial.org.

[15] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel H. Saltz. Hadoop-gis: A high performance spatial data warehousing system over mapreduce. *PVLDB*, 6(11), 2013.

[16] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel H. Saltz. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *PVLDB*, 6(11), 2013.

[17] Afsin Akdogan, Ugur Demiryurek, Farmoush Banaei-Kashani, and Cyrus Shahabi. Voronoi-based Geospatial Query Processing with MapReduce. In *International Conference on Cloud Computing Technology and Science*, 2010.

[18] Nagender Bandi, Chengyu Sun, Amr El Abbadi, and Divyakant Agrawal. Hardware Acceleration in Commercial Databases: A Case Study of Spatial Operations. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2004.

[19] Sebastian Breí, Felix Beier, Hannes Rauhe, Kai-Uwe Sattler, Eike Schallehn, and Gunter Saake. Efficient Co-processor Utilization in Database Query Processing. *Information Systems*, 38(8), 2013.

[20] Ariel Cary, Zhengguo Sun, Vagelis Hristidis, and Naphtali Rishe. Experiences on processing spatial data with mapreduce. In *Scientific and Statistical Database Management, 21st International Conference, SSDBM 009, New Orleans, LA, USA, June 2-4, 2009, Proceedings*, 2009.

[21] Lorenzo Dematté and Davide Prandi. GPU Computing for Systems Biology. *Briefings in Bioinformatics*, 2(3), 2010.

[22] Ahmed Eldawy, Louai Alarabi, and Mohamed F. Mokbel. Spatial Partitioning Techniques in Spatial Hadoop. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 8(12), 2015.

[23] Ahmed Eldawy, Mostafa Elganainy, Ammar Bakeer, Ahmed Abdelmotaleb, and Mohamed Mokbel. Sphinx: Distributed Execution of Interactive SQL Queries on Big Spatial Data. November 2015.

[24] Ahmed Eldawy, Yuan Li, Mohamed F. Mokbel, and Ravi Janardan. CG_Hadoop: Computational Geometry in MapReduce. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM GIS*, November 2013.

[25] Ahmed Eldawy and Mohamed F Mokbel. A Demonstration of Spatialhadoop: An Efficient Mapreduce Framework for Spatial Data. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 6(12), 2013.

[26] Ahmed Eldawy and Mohamed F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *Proceedings of the IEEE International Conference on Data Engineering, ICDE*, 2015.

[27] Ahmed Eldawy, Mohamed F. Mokbel, and Christopher Jonathan. A Demonstration of HadoopViz: An Extensible MapReduce System for Visualizing Big Spatial Data. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 8(12), 2015.

[28] Ahmed Eldawy, Mohamed F. Mokbel, and Christopher Jonathan. HadoopViz: A MapReduce Framework for Extensible Visualization of Big Spatial Data. In *Proceedings of the IEEE International Conference on Data Engineering, ICDE*, May 2016.

[29] Marta Fort and Joan Antoni Sellarès. GPU-based Computation of Distance Functions on Road Networks with Applications. In *Proceedings of the ACM Symposium on Applied Computing, SAC*, 2009.

[30] Ralf Hartmut Güting, Thomas Behr, and Christian Düntgen. SECONDO: A platform for moving objects database research and for publishing and integrating research implementations. *IEEE Data Engineering Bulletin*, 33(2), 2010.

[31] Ralf Hartmut Güting, Victor Teixeira de Almeida, and Zhiming Ding. Modeling and querying moving objects in networks. *The International Journal on Very Large Data Bases, VLDB Journal*, 15(2), 2006.

[32] Ralf Hartmut Güting and Jiamin Lu. Parallel SECONDO: scalable query processing in the cloud for non-standard applications.

[33] Barrett Hargreaves, John D. Joyce, George L. Cole, Ernest D. Foss, Richard G. Gray, Elmer M. Sharp, Robert J. Sippel, Thomas M. Spellman, and Robert A. Thorpe. Image processing hardware for a man-machine graphical communication system. In *Proceedings of the October 27-29, 1964, Fall Joint Computer Conference, Part I*, 1964.

[34] Bingsheng He, Ke Yang, Rui Fang, Mian Lu, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational joins on graphics processors. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2008.

[35] Jiong He, Mian Lu, and Bingsheng He. Revisiting Co-processing for Hash Joins on the Coupled CPU-GPU Architecture. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 6(10), 2013.

[36] Jiong He, Shuhao Zhang, and Bingsheng He. In-cache Query Co-processing on Coupled CPU-GPU Architectures. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 8(4), 2014.

[37] Timothy E. Johnson. Sketchpad iii: A computer program for drawing in three dimensions. In *Proceedings of Spring Joint Computer Conference*, 1963.

[38] Marcel Kornacker, Alexander Behm, Victor Bittorf, Taras Bobrovytsky, Casey Ching, Alan Choi, Justin Erickson, Martin Grund, Daniel Hecht, Matthew Jacobs, Ishaan Joshi, Lenni Kuff, Dileep Kumar, Alex Leblang, Nong Li, Ippokratis Pandis, Henry Robinson, David Rorke, Silvius Rus, John Russell, Dimitris Tsirogiannis, Skye Wanderman-Milne, and Michael Yoder. Impala: A Modern, Open-Source SQL Engine for Hadoop. In *Proceedings of the International Conference on Innovative Data Systems Research, CIDR*, January 2015.

[39] Jonathan K. Lawder and Peter J. H. King. Using Space-Filling Curves for Multi-dimensional Indexing. In Brian Lings and Keith Jeffery, editors, *Advances in Databases*, volume 1832 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2000.

[40] Francesco Lettich, Salvatore Orlando, and Claudio Silvestri. Processing streams of spatial k-nn queries and position updates on manycore gpus. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM GIS*, 2015.

[41] Ming-Ling Lo and Chinya V. Ravishankar. Spatial Hash-joins. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, June 1996.

[42] Jiamin Lu and Ralf Hartmut Güting. Parallel SECONDO: A Practical System for Large-scale Processing of Moving Objects. In *Proceedings of the IEEE International Conference on Data Engineering, ICDE*, April 2014.

[43] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient Processing of k Nearest Neighbor Joins using MapReduce. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2012.

[44] Qiang Ma, Bin Yang, Weining Qian, and Aoying Zhou. Query processing of massive trajectory data based on mapreduce. In *Proceedings of the First International CIKM Workshop on Cloud Data Management, CloudDb 2009, Hong Kong, China, November 2, 2009*, 2009.

[45] Henry Markram. The Blue Brain Project. *Nature Reviews Neuroscience*, 7(2), 2006.

[46] Sparsh Mittal and Jeffrey S. Vetter. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Computing Surveys*, 47(4), 2015.

[47] T. H. Myer and I. E. Sutherland. On the design of display processors. *Communications of the ACM*, 11(6), 1968.

[48] Naohito Nakasato. Implementation of a parallel tree method on a GPU. *Journal of Computer Science*, 3(3), 2012.

[49] Cristóbal A. Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A Survey on Parallel Computing and its Applications in Data-Parallel Problems using GPU Architectures. *Communications in Computational Physics*, 15(2), 2014.

[50] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Lo-

cation Aware Services. In *Proceedings of the International Conference on Mobile Data Management, MDM*, June 2011.

[51] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Hbase: design and implementation of an elastic data infrastructure for cloud-scale location services. *Distributed and Parallel Databases*, 31(2), 2013.

[52] Jia Pan and Dinesh Manocha. Fast gpu-based locality sensitive hashing for k-nearest neighbor computation. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM GIS*, 2011.

[53] Sushil K. Prasad, Michael McDermott, Satish Puri, Dhara Shah, Danial Aghajarian, Shashi Shekhar, and Xun Zhou. A Vision for GPU-accelerated Parallel Computation on Geo-Spatial Datasets. *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM GIS*, 6(3), 2014.

[54] Jun Rao and Kenneth A. Ross. Cache Conscious Indexing for Decision-Support in Main Memory. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, September 1999.

[55] Hans Sagan. Space-Filling Curves. Springer New York, 1994.

[56] Lin Shi, Wen Liu, Heye Zhang, Yongming Xie, and Defeng Wang. A survey of GPU-based Medical Image Computing Techniques. *Quantitative Imaging in Medicine and Surgery*, 2(3), 2012.

[57] Nikos Sismanis, Nikos Pitsianis, and Xiaobai Sun. Parallel search of k-nearest neighbors with synchronous operations. In *IEEE Conference on High Performance Extreme Computing, HPEC*, 2012.

[58] Robert F. Sproull and Ivan E. Sutherland. A clipping divider. In *Proceedings of Fall Joint Computer Conference*, 1968.

[59] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. The Architecture of SciDB. In *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*, July 2011.

[60] Chengyu Sun, Divyakant Agrawal, and Amr El Abbadi. Hardware Acceleration for Spatial Selections and Joins. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2003.

[61] Ivan E. Sutherland. Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE Design Automation Workshop*, 1964.

[62] Ivan E. Sutherland. A Head-mounted Three Dimensional Display. In *Proceedings of Fall Joint Computer Conference*, December 1968.

[63] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - A Warehousing Solution Over a Map-Reduce Framework. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2(2), 2009.

[64] Kaibo Wang, Yin Huai, Rubao Lee, Fusheng Wang, Xiaodong Zhang, and Joel H. Saltz. Accelerating Pathology Image Data Cross-Comparison on CPU-GPU Hybrid Systems. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 5(11), 2012.

[65] Liao Wei, Zhang Zhiming, Yuan Zhimin, Fu Wei, and Wu Xiaoping. Parallel continuous k-nearest neighbor computing in location based spatial networks on gpus. In *Computational and Information Sciences, ICCIS*, 2013.

[66] Randall T. Whitman, Michael B. Park, Sarah M. Ambrose, and Erik G. Hoel. Spatial indexing and analytics on hadoop. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM GIS*, November 2014.

[67] Simin You, Jianting Zhang, and Le Gruenwald. GPU-based Spatial Indexing and Query Processing using R-Trees. In *ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, BigSpatial*, November 2013.

[68] Simin You, Jianting Zhang, and Le Gruenwald. Scalable and efficient spatial data management on multi-core CPU and GPU clusters: A preliminary implementation based on impala. In *International Workshop on Big Data Management on Emerging Hardware, HardBD*, April 2015.

[69] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM GIS*, November 2015.

[70] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, June 2010.

[71] Chi Zhang, Feifi Li, and Jeffrey Jestes. Efficient Parallel kNN Joins for Large Data in MapReduce. In *Proceedings of the International Conference on Extending Database Technology, EDBT*, 2012.

[72] Jianting Zhang, Simin You, and Le Gruenwald. Parallel Online Spatial and Temporal Aggregations on Multi-core CPUs and Many-core GPUs. *Information Systems*, 44, 2014.

[73] Shubin Zhang, Jizhong Han, Zhiyong Liu, Kai Wang, and Zhiyong Xu. SJMR: Parallelizing spatial join with MapReduce on clusters. In *CLUSTER*, 2009.

[74] Shuhao Zhang, Jiong He, Bingsheng He, and Mian Lu. OmniDB: Towards Portable and Efficient Query Processing on Parallel CPU/GPU Architectures. *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 6(12), 2013.