# Towards GPU-Accelerated Web-GIS for Query-Driven Visual Exploration

Jianting Zhang[1,2], Simin You[2], and Le Gruenwald[3]

[1] Department of Computer Science
The City College of the City University of New York
138th Convent Avenue,New York, NY 10031
jzhang@cs.ccny.cuny.edu
[2] Department of Computer Science
The Graduate Center of the City University of New York
365 Fifth Avenue,New York, NY 1006
syou@gradcenter.cuny.edu
[3] Department of Computer Science
The University of Oklahoma
110 W. Boyd St. Norman, OK, USA, 73019
ggruenwald@ou.edu

**Abstract.** Web-GIS has played an important role in supporting accesses, visualization and analysis of geospatial data over the Web for the past two decades. However, most of existing WebGIS software stacks are not able to exploit increasingly available parallel computing power and provide the desired high performance to support more complex applications on large-scale geospatial data. Built on top our past works on developing high-performance spatial query processing techniques on Graphics Processing Units (GPUs), we propose a novel yet practical framework on developing a GPU-accelerated Web-GIS environment to support Query-Driven Visual Explorations (QDVE) on Big Spatial Data. An application case on visually exploring global biodiversity data is presented to demonstrate the feasibility and the efficiency of the proposed framework and related techniques on both the frontend and backend of the prototype system.

**Keywords:** Web-GIS, GPU, QDVE, Spatial Join, Biodiversity

## 1 Introduction

Since the inception of Web-GIS more than two decades ago, Web-GIS systems have played important roles in extending desktop GIS functionality to Web environments that allow users to visualize and query geographical information anywhere and anytime. In the coming Big Data era, the volume, variety and velocity (3Vs) of geo-referenced data increase proportionally, if not more rapidly, when compared with the mainstream relational data. However, although considerable new techniques have been developed for relational Big Data by exploiting newly emerging hardware, platforms and algorithms for higher performance, it seems that Web-GIS systems are still largely relying on the frameworks and techniques matured decades ago. As such, Web-GIS users are often forced to limit their applications to pre-aggregated data generated by third-party systems before feeding them to existing Web-GIS systems. This not only significantly increases end-to-end system complexity but also makes the overall performance very poor, as data exchanges among multiple systems through disks and network interfaces are known to be expensive. The low performance likely makes more performance demanding applications in Web-GIS environments impractical, such as query-driven visual exploration on large-scale geo-referenced data.

Although HTML5 has been quickly adopted and new Web techniques are increasingly feature-rich, the development of server side techniques for large-scale geospatial data is more complicated. Existing leading Big Data techniques, such as Hadoop and Spark, are mostly designed for offline batch processing on cluster computers. Although those techniques have been extended to process large-scale geospatial data (e.g. [1][2][3][4]), the technical complexity, hardware cost and high-latency make them generally unattractive for Web-GIS applications. Parallel techniques on multiple processors [5], such as multi-core CPUs and many-core Graphics Processing Units (GPUs), offer alternative solutions to cluster-based distributed computing techniques, although those two categories of techniques can be combined in principle. While efforts on parallelizing geospatial operations on multi-core CPUs for higher performance can be dated back to decades ago, using GPUs for general computing did not come into being until 2007 when Compute Unified Device Architecture (CUDA) was announced by Nvidia[6]. There have been growing interests in using the massively data parallel processing power provided by GPUs for geospatial processing (see Section 2 for more details), however, to the best of our knowledge, there have been little systematic efforts on using GPUs to speed up Web-GIS applications on large-scale geo-referenced data.

In this study, we propose to exploit parallel processing power on modern processors, adopt new parallel designs and implementations of spatial join query processing and refactor existing Web-GIS processing pipelines to support visual query exploration on large-scale geo-referenced data. Our specific contributions in this paper are the following. First, we present a novel yet practical framework on integrating GPU techniques, geospatial techniques and Web-GIS pipelines to enable query driven visual explorations on large-scale geo-reference data to achieve the desired high performance. Second, we design and implement the relevant components and develop a prototype system for feasibility and performance studies. Third, we perform experiments on a real world large-scale geospatial dataset at global scale to demonstrate its feasibility and realizable performance. The rest of the paper is arranged as follows. Section 2 introduces background, motivation and related work. Section 3 presents system arhitecture and the design and implementation details. Section 4 reports experiments and results. Finally, Section 5 is the conclusion and future work directions.

## 2 Background and Related Work

### 2.1 Web-GIS and Parallel and Distributed Processing

A Web-GIS can be considered as a distributed computing system that allows visualizing and querying geo-referenced data in Web browsers. Standards such as Web Map Services (WMS) and Web Feature Services (WFS) are developed to standardize data communication protocols between server side and client side in a Web-GIS application. While it is desirable to expose as many desktop GIS functionalities to Web browsers as possible, functionalities that are supported by WebGIS are typically limited when compared with desktop GIS, due to security restrictions and limited accesses to native or raw computing power through Web browsers in the browser-server computing model whose strength and weakness are well understood. Query-Driven Visual Explorations (QDVE) integrates dynamic queries with traditional visualization techniques to support users explore large-scale datasets interactively (e.g. [7]). There have been growing interests in enabling QDVE in a Web environment. We consider Web-based QDVE on large-scale geospatial data a special application of Web-GIS and is a natural extension to traditional Web-GIS applications that primarily focus on mapping and simple queries. It is easy to see that QDVE imposes significant performance challenges on both client and server sides of Web-GIS systems in this setting.

While client machines (including both PCs and mobile devices) hosting Web browsers have significantly improved their processing power due to hardware progresses on single processors, the hardware performance improvements on server side are mostly due to the increased parallelisms, including multi-core CPUs and many-core GPUs. Indeed, while parallelizing a sophisticated general-purpose browser program is very difficult, it is relatively easier to parallelize individual server programs which are very often domain-specific and with abundant parallelisms to exploit. Unfortunately, most of existing Web-GIS applications are built on top of legacy commercial or open-source Web-GIS systems that are aware of neither application parallelisms nor the underlying parallel hardware. It is thus both conceptually interesting and practically useful to match the intrinsic parallelisms in processing large-scale geospatial data with the increasingly available parallel hardware for desired high performance in Web-GIS applications, especially for QDVE applications that demand interactive responses in real time.

Parallel processing of geospatial data is not a new idea and can be traced back to at least two decades ago (see the dedicated book by Healey[8] for details). However, as discussed in [9], research on parallel and distributed processing of geospatial data prior to 2003 has very little impact on mainstream geospatial data processing applications, possibly due to the accessibility of hardware and infrastructures in the past. The past ten years have seen two major technical trends in scaling up large-scale data processing, one is MapReduce/Hadoop based techniques for distributed computing and the other is GPU related techniques for parallel computing on a single computing node. MapReduce/Hadoop based techniques provide a distributed execution engine and a runtime system to automatically distribute Map and Reduce tasks to distributed computing nodes and assemble results. So long as users can decompose their problems into independent Map and Reduce tasks, they do not need to write sophisticated and error-prone distributed programs while are able to achieve significant speedups. More efficient successors, such as Spark-based techniques that utilize in-memory processing, are likely to play a leading role in Big Data market in the near future. Although there were attempts to integrate MapReduce/Hadoop based techniques for interactive visualization (e.g. [2]), their inherent high latency (due to start-up delay and batch-oriented processing schema) has made the attempts largely unsuccessful. Although Spark-based techniques are generally more efficient, there are still significant gaps between hardware potential and realizable capabilities. Our focus in this work is to maximize the utilization of increasing parallel computing power on a single computing node and understand its realizable performance on commodity hardware in the context of Web-GIS applications.

A typical modern computing node is equipped with 2-4 CPUs each with 4-16 processing cores (totaling 8-64 CPU cores) and multiple hardware accelerators, such as Nvidia GPUs [6]. While multi-core CPUs feature large-memory capacity and multi-level large caches to accommodate tasks with complex logics and irregular data accesses, GPUs typically have much larger number of cores ($10^2$-$10^4$), higher memory bandwidth ($10^2$-$10^3$ GB/s) and significantly higher floating point computing power (10+ TFLOPs) than multi-core CPUs [5]. Although a single CPU core may only have a few to a few tens of GFLOPs computing power, a modern computing node can easily achieve several tens of TFLOPs when SIMD computing power of both GPUs and multi-core CPUs are fully utilized. When properly used, the three orders higher computing power may produce orders of magnitude of higher performance.

Towards this end, we have been working on data parallel designs of major operations that support spatial query processing and we refer to an invited ACM SIGSPATIAL Special contribution [11] for a summary. Preliminary results have demonstrated thousands of times of speedups over traditional techniques that reflect the combined improvements of data structures, algorithms, in-memory process and GPU accelerations on a single computing node. This effectively reduces processing time from tens of hours to tens of seconds on several real world datasets on GPUs (e.g. [12][14]). However, as runtimes are not crucially important to offline processing, we believe that interactive applications, especially for QDVE on large-scale geospatial data in a Web-GIS environment where real time response is essential, are better suited for the newly developed high-performance geospatial techniques. While most of the data parallel designs and implementations utilized in this study were initially developed for offline applications as improvements to traditional single-node and cluster computing environments, we believe adopting and adapting these techniques for Web-GIS applications to support interactive QDVE is reasonably novel and could be interesting to the Web-GIS community, especially at the tipping point that hardware progresses demand software renovations due to the changed cost models, to better support real world applications.

## 2.2 Spatial Joins on GPUs

Spatial joins techniques are fundamental to spatial query processing in spatial databases [17][18], which are typically the core of a Web-GIS application backend, especially when working with vector geo-referenced data. For example, both MapServer and ESRI ArcGIS Server can delegate spatial operations written in SQL to major database systems that support spatial data, such as PostgreSQL (with PostGIS), Oracle and Microsoft SQLServer. Compared with point/location query and window/range query that involve querying a point/box against a single dataset which typically only incurs linear complexity at most, spatial joins typically involve multiple datasets and their tuples need to be related based on certain spatial operations. The indexing and query processing capabilities provided by such database systems are fundamental to the performance of Web-GIS applications when datasets are getting large. The efficiencies of indexing schemes and query execution plans, which may vary significantly among aforementioned spatially-enhanced database systems, have been one of the primary driving forces for spatial databases research. While hundreds of spatial indexing techniques and numerous query processing techniques have been proposed [19][20], most of them are designed for serial executions on uniprocessors.

Our efforts on developing spatial indexing and query processing techniques on GPU-accelerated machines are largely based data parallel deigns and implementations[11]. By carefully identifying parallelisms on spatial indexing (including grid-file based, quadtree based and R-Tree based) and spatial joins (including point-to-polyline/polygon distance based, point-in-polygon test based and polyline-intersection test based), we are able to chain parallel primitives [37], such as map/transform scan, reduce and sort, to partition spatial data into blocks (partitioning), index the Minimum Bounding Boxes (MBBs) of these blocks (indexing) and join them based on spatial intersections (filtering) before developing fine-grained parallel designs for geometric computation on GPUs (refinement). The behavior of the underlying parallel primitives is well understood and their efficient implementations on multiple parallel hardware (including both multi-core CPUs and GPUs) are available either by hardware vendors or parallel computing research community. Since spatial partitioning and indexing are typically one-time cost and spatial filtering are typically cheaper than refinement, the design choice represents a reasonable tradeoff between complexity and efficiency. On the other hand, spatial refinement generally involves floating-point intensive geometric computation and very often dominates the overall cost in spatial joins. As such, it is crucial to maximize its performance by exploiting GPU specific hardware features and we refer to our individual reports on computing point-to-polyline distance (NN)[13], point-to-polygon distance (KNN) [12], point-in-polygon test [10][14] and polyline intersection [16] based relationships. As a general strategy, in spatial refinement, we allocate a joined pair (P, Q) to a GPU thread block and let all the basic elements in P (e.g., points in a quadrant or a grid cell and vertices of a polyline or a polygon) loop through all the basic elements in Q to achieve coalesced memory accesses and reduce control divergence, both are important in maximizing GPU performance. We have also developed efficient lightweight data structures (e.g., array-based queues) on block-specific shared memory (fast but with very limited size [6]) and use atomic operations whereas appropriate to further maximize GPU hardware utilization.

While our data parallel spatial join techniques were motivated by GPU hardware, we have found that our in-memory columnar data layouts using flat arrays [12] have contributed significantly to the orders of magnitude of performance improvement when compared with traditional spatial databases that are typically row/tuple based and disk resident. By storing columns or column groups as arrays and only load columns that are relevant into CPU memory (and subsequently transferred to GPU memory), both disk I/Os and memory footprints can be significantly reduced. Furthermore, since array offsets can be used in lieu of pointers and data accesses in data parallel deigns are well behaved, the columnar layouts are cache friendly on CPUs and memory accesses are largely coalesced on GPUs, both are highly desirable with respect to memory system performance, which are becoming increasingly important on modern hardware.

## 2.3 Previous Efforts on WebGIS for QDVE on Large-Scale Geospatial Data

Our previous efforts on supporting QDVE in a WebGIS framework mostly focus on improving spatial indexing and query processing in traditional disk-resident systems (PostgreSQL in particular) and main-memory systems. In [21], we decompose millions of polygons from thousands of bird species range maps into linear quadtree nodes and represent them as strings. Subsequently, we use the LTREE module available in PostgreSQL[1] to index the strings and support window/range queries based on string matching. By using a same quadtree based spatial tessellation for both polygons and query windows, the technique essentially transforms a spatial query into a string query. While it does not seem to be elegant from a spatial databases research perspective, experiments have demonstrated the desired high performance than querying PostgreSQL/PostGIS directly with 6-9.5X speedups.

To further improve performance, we have developed a memory-resident Multi-Attributed Quadtree (MAQ-Tree) structure by re-using the linear quadtree nodes derived previously [22]. Basically, individual linear quadtree nodes that represent range maps of thousands of bird species were spatially aggregated based on binary linear codes. As such, polygon identifies are now associated with both intermediate and leaf quadtree nodes. A window/range query can be efficiently processed by traversing the multi-attributed quadtree while evaluating the query in memory at each quadtree node. The new technique, which was integrated with a OpenLayers based WebGIS, has reduced the end-to-end response time from high tens of seconds to below a second in a Web environment and is suitable for interactive location/window query processing.

We have also experimented a simple brute-force based technique to support simple QDVE in a WebGIS environment [15]. By adopting the columnar based design for point and polygon data discussed previously and utilizing multi-core CPU parallel processing power, we have successfully demonstrated that interactively querying hundreds of millions of taxi trip records that fall within a few Regions of Interests (ROIs) that are defined by users interactively in a Web environment (using Google Map client APIs) while achieving sub-second end-to-end performance without sophisticated indexing is possible. It is conceivable that a WebGIS backend that can leverage both spatial indexing and parallel processing is essential to provide desired performance when handling more complex QDVE tasks on more complex spatial data types such as polygons and polylines.

Similar to indexing and querying vector data on GPUs, we have also developed a server side indexing technique called Binned Min-Max Quadtree (BMMQ-Tree) to support efficient query processing on large-scale raster data [23]. The technique was integrated with an ArcGIS Server for QDVE on monthly climate data using the 1-km global WorldClim datasets [24]. The BMMQ-Tree construction algorithm was later parallelized on GPUs with significant performance improvements [25][26]. We further have developed a technique that converts query results on a BMMQ-Tree to tiled images to be overlaid with base map data at the client side for highlighting purposes [27]. As the on-the-fly derived query results are provided as dynamic WMS services, the technique imposes virtually no data management overhead to the server side (the CPU overhead of generating binary image tiles along with query processing is negligible) and is easy to use on client side (by using conventional XYZ tile request).

### 2.4 Other Related Work

It is beyond the scope of this paper to provide a comprehensive review of large bodies of related works on spatial databases, parallel processing of geospatial data, WebGIS and QDVE applications in various application contexts. Besides what have been discussed in the previous section and subsections, we refer to the SP-GIST project [28] for developing a general index framework for space partitioning trees and its implementation in PostgreSQL. Several research groups have developed GPU-based techniques for large scale geospatial data for different purposes [29][30]. However, to the best of our knowledge, these techniques have not been applied to Web-GIS environment. There are several works that integrate high-end computing facilities at supercomputer centers and visualization software (e.g., ParaView[2]) to facilitate identifying and understanding patterns from large-scale geospatial data, such as weather and climate data[7]. A hybrid architecture that integrates the data storage and processing power from remote Cloud resources (Microsoft Azure in particular) and visual analytics functionality of local workstations is demonstrated to be beneficial for visual explorations of large-scale ocean data[2]. However, these systems are typically developed for domain experts that have accesses to supercomputer resources and are not available to the general public. Furthermore, the techniques behind are significantly different from those that Web-GIS applications are built upon.

## 3 System Design and Prototyping

The prototype system being presented in this work integrates several techniques we have developed previously, including spatial indexing and spatial joins on GPUs, dynamic tiled map services and client side geometry APIs for query optimization. In this section, we will first introduce the overall system design and then present details of the relevant modules. Due to time limit, not all modules discussed have been tested and they are left for future work. The overall system design follows a typical Web-GIS framework: a server side backend for data processing, a client side frontend runs in Web browsers and the backend and the frontend communicate using WMS, WFS and other relevant Web services. Different from classic WebGIS backends that delegates spatial query processing to spatial databases, our prototype integrates spatial query processing with the backend to conveniently exploit parallel processing power on modern hardware. While the frontend can be implemented on top of various Web Map Javascript libraries, we have chosen Google Map API for popularity, ease-of-use (API) and performance (fast base map loading). The overall system architecture is shown in Fig. 1.

The left side of Fig. 1 illustrates the major modules of the proposed Web-GIS backend: data storage, spatial indexing, spatial filtering and spatial refinement. The right side of the figure also shows the major modules of our WebGIS frontend for QDVE applications. While the frontend and the backend communicate through convential Web standards, including WMS, WFS and JSON, both the backend and the frontend are enhanced with the targeted application: QDVE on large-scale geospatial data. As discussed previously, the backend is accelerated by GPU hardware which requires a complete new set of techniques for data storage, indexing, filtering and refinement. While the backend is designed to support major geospatial data types (point, polyline and polygons) and popular spatial operations (e.g., point-to-polyline distance computation, point-to-polyline/polygon distance computation, polyline-to-polyline distance computation and point-in-polygon topological test), we will focus on point-in-polygon test based spatial operation in this study. The frontend is enhanced with customized Graphics User Interface (GUI) to facilitate realizing typical QDVE schemes (see below for details) and a simple yet effective Javascript geometry for performance optimization. We next present more design and implementation details on the relevant modules.

## 3.1 On-Demand Data Parallel Spatial Joins on GPUs

As discussed in Section 2.2, spatial join processing is indispensable in supporting query-driven visual explorations on large-scale data. Our previous works have developed data parallel spatial indexing, filtering and refinement techniques on GPUs, including utilizing quadtree indexing for point and rasters and grid-file and R-Tree indexing for MBRs of polylines and polygons, which have demonstrated excellent performance. However, these techniques were developed for static data and offline processing. In this study, we adapt these techniques for online interactive QDVE with a set of strategies to effectively support QDVE for the following considerations.

First, queries in QDVE are typically ad-hoc and it is impractical to pre-build indices for all possible queries for both raw and intermediate data. On the other hand, as long as the response time is in the order of sub-second to a few seconds, users on QDVE tasks would hardly perceive major differences. As such, for QDVE tasks, it is possible to more aggressively exploit the concept of ”disposable indexing” for on-demand spatial joins. The idea is that, instead of loading pre-built index from disks and use it for the subsequent spatial filtering (as in traditional databases), we build index on the data partitions of interests in real time and use it to relate relevant data partitions on both sides of a spatial join for the final spatial refinement based on the desired spatial operation (e.g., point-in-polygon test). While the end-to-end runtime of a spatial join significantly depends on the spatial distributions of the underlying spatial datasets to be joined, the average performance of each of the three steps in a spatial join (indexing, filtering and refinement) with respect to the number of data items can be roughly estimated based on historical experiments. This is because our implementations are based on data parallel designs which mostly involve element-wise operations that include transform (map), scan, reduce and binary search [37]. Furthermore, the most popular sorting algorithm on GPUs is radix sort, which also incurs linear complexity[38]. Subsequently, it is possible to roughly estimate whether disposable indexing is feasible and the level of granularity of indexing and to decide the tradeoffs between filtering and refinement that are needed to achieve response time requirement.

Second, the four popular QVDE schemes [35][36] listed in the top-right part of Fig. 1 exhibit certain patterns in processing a QVDE task and mapping them to sequences of spatial databases operations, including spatial joins, location/window queries and relational queries. For example, *Overview* typically happens at the beginning of a QVDE task while *Details on Demand* happens after the other three types of schemes. *Context+Focus*, typically works as an intermediate step between *Overview* and *Details on Demand* and mostly involves the neighborhood of the region that a user is currently exploring. The *Filter and Zoom* scheme may involve both spatial and non-spatial attributes

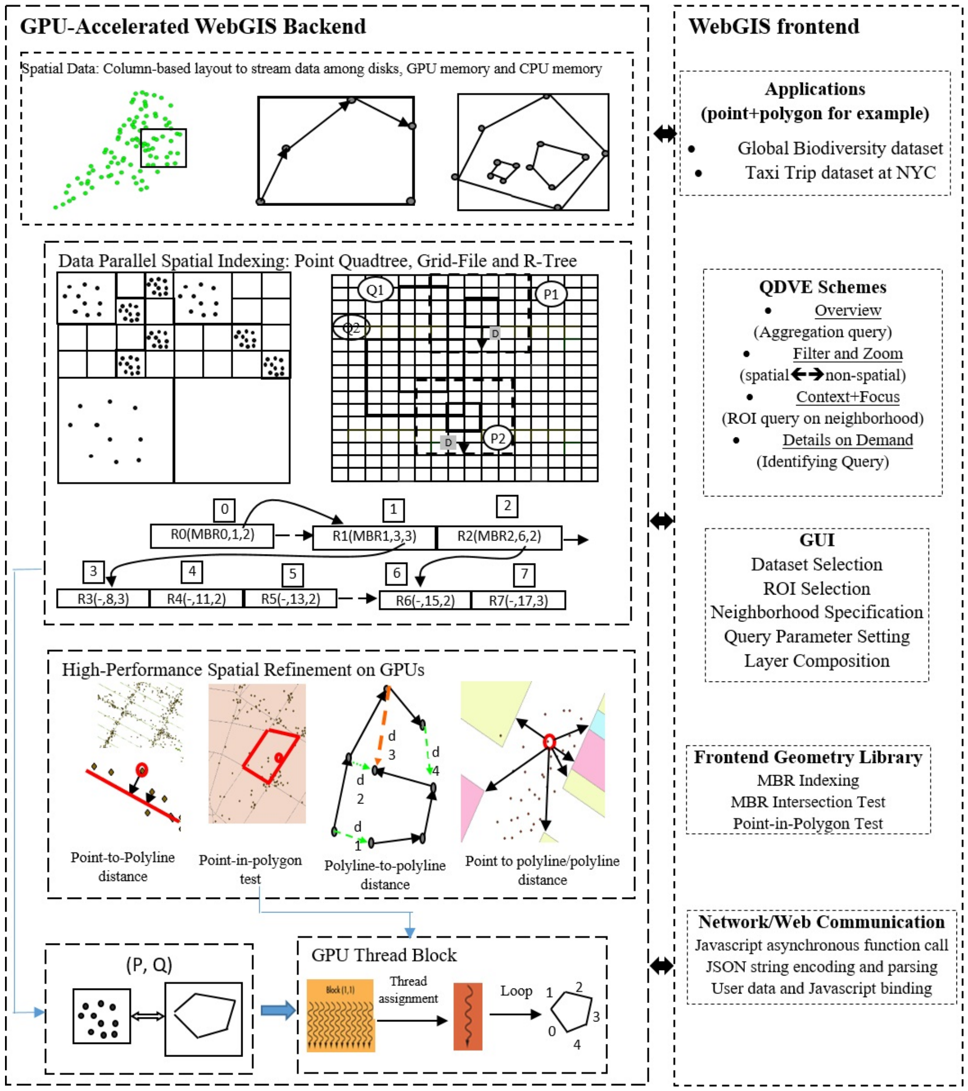
GPU-Accelerated WebGIS Backend
Spatial Data: Column-based layout to stream data among disks, GPU memory and CPU memory
Data Parallel Spatial Indexing: Point Quadtree, Grid-File and R-Tree
Q1
P1
Q2
P2
D
0
1
2
R0(MBR0,1,2)
R1(MBR1,3,3)
R2(MBR2,6,2)
3
4
5
6
7
R3(-,8,3)
R4(-,11,2)
R5(-,13,2)
R6(-,15,2)
R7(-,17,3)
High-Performance Spatial Refinement on GPUs
Point-to-Polyline distance
Point-in-polygon test
Polyline-to-polyline distance
Point to polyline/polyline distance
(P, Q)
GPU Thread Block
Block (1,1)
Thread assignment
Loop
WebGIS frontend
Applications (point+polygon for example)
Global Biodiversity dataset
Taxi Trip dataset at NYC
QDVE Schemes
Overview
(Aggregation query)
Filter and Zoom
(spatial↔non-spatial)
Context+Focus
(ROI query on neighborhood)
Details on Demand
(Identifying Query)
GUI
Dataset Selection
ROI Selection
Neighborhood Specification
Query Parameter Setting
Layer Composition
Frontend Geometry Library
MBR Indexing
MBR Intersection Test
Point-in-Polygon Test
Network/Web Communication
Javascript asynchronous function call
JSON string encoding and parsing
User data and Javascript binding

**Fig. 1.** Prototype System Architecture and Modules.

and queries may require two-way correspondence between spatial and non-spatial data. As such, we can precompute and cache queries related to *Overview* to reduce startup delay and improve response time. In a similar way to the tiling mechanism exploited by WebGIS for simple visualization, we actively buffer query results based on the neighborhood of the current region being explored to serve *Context+Focus* better. Location-dependent queries are reevaluated when their results are not in the cache or cannot fully cover the neighborhood anymore. For filtering based on relational attributes, the results may potentially cover the whole study area and require highlighting and coloring to help identify ROIs for further explorations. We reuse the dynamic tiling technique to generate image tiles that represent the relational filtering results to help users spatially zooming to multiple ROIs in a convenient manner. Subsequently *Context+Focus* and *Details on Demand* schemes can be applied. We note that *Details on Demand* typically incurs very light overhead as retrieving a limited number of records from memory is generally very fast and the response time is well beyond the sub-second reponse time requirement, even through it takes extra time to assemble column values from different arrays to form records (tuples) due to our column-based data layout desgin.

Third, collaborations between CPUs and GPUs are crucial for the end-to-end performance. Despite that GPU memory capacity is increasingly fast (up to 24GB), it is still 10X-50X smaller than CPU memory capacity on a typical machine. The data transfer speed between CPU memory and GPU memory is limited to PCI-E bandwidth (16/32 GB/s). Fortunately, all our spatial indexing and spatial filtering techniques designed for GPUs are implemented based on data parallel primitives that are also supported on multi-core CPUs. Even though our spatial refinement implementations are GPU specific, we have developed multi-core CPU implementations on top of Intel TBB parallel library [37] with the same interfaces. As such, it is both desirable and feasible to execute certain queries on multi-core CPUs to avoid the data transfer overhead between CPUs and GPUs, which naturally improves the overall performance. The large CPU memory capacity also makes it suitable to cache GPU results as discussed above.

## 3.2 WebGIS Frontend Optimized for QDVE

Different from our previous works on Web-GIS applications using OpenLayers [22] and ESRI ArcGIS API for Flex [23], we have chosen to use Google Map Javascript API for our WebGIS frontend in this study. Several modules listed in the lower-right part of Fig. 1 are inherited from our previous work on visual analytics of taxi trip Origin-Destination (OD) data in a Web environment [15] with enhancements which will be described shortly.

The GUI module is the interfaces to interact with users which is largely application specific. Nevertheless, there are several interfaces that are common to many applications, including selecting dataset(s) to start a QVDE task, choosing/switching base map layers and turning on/off auxiliary and cosmetic layers, tools to interactively specifying ROIs and neighborhood, and dynamic interfaces to set both mandatory and optional parameters for spatial queries during a QVDE task. Those interfaces can be relatively easily implemented using Javascript and HTML and we skip the details due to space limit. The network/Web Communication modules listed in the bottom-right part of Fig. 1, while important, are quite standard in implementation and whose details are also skipped for the same reason. We note that the preference of JSON over GML or KML as a data format to communicate between the WebGIS backend and the frontend is mostly due to efficiency concerns, although other data formats including GML and KML can also be used with an additional conversion step before the parsed data can be used by Javascript APIs. We next focus on the frontend geometry API module which is unique to the targeted QDVE applications.

The optional *GetFeatureInfo* operation defined in the WMS specification allows to retrieve the underlying data including geometry and attribute values for a pixel location on a map. Since the operation can also return unique FeatureID, it is possible to combine it with the *GetFeature* operation defined in the WFS specification to retrieve specific attribute values to support visualization and information display at the Web frontend. However, while those techniques work reasonably well for a small number of features in traditional WebGIS applications, working at the individual feature level through WMS/WFS is not likely to be efficient for QEDV tasks where a large number of features can be involved, such as results of queries for *Overview*, *Filter and Zoom* and *Context+Focus* schemes. While dynamically generating WMS/WFS layers at the backend side and make them ready to be pulled by the frontend is a practically useful solution, efficiently filtering features at the frontend (to reduce the backend overhead of generating and maintaining a large number of dynamic WMS/WFS layers) is complementarily beneficial. Towards this end, we reuse the MBR intersection test and point-in-polygon test Javascript code initially implemented in [15] to spatially filter out FeatureIDs at the Web frontend that cannot be part of queries, such as those involve ROIs that are interactively drawn by users and those involve neighborhood dynamically set by users, before the queries are sent for server side processing. We note that the server side processing of such filtering can be several orders of magnitude faster than the performance of the Javascript code performance at the frontend. However, the benefit here is mostly on reducing data transmission overhead over the Web and reducing the associated encoding/decoding overhead, which could be much more costly than spatial filtering on the frontend using Javascript code.

We are aware of the techniques and applications of integrating Google Map and Google BigQuery[3] APIs to spatially visualize ”small” query results from ”big” relational tables in Cloud where the number of features (after combining geometry and relational attribute values) is limited. However, we beleive these techniques are generally incapable of handling QEDV tasks on large-scale geospatial datasets where the features can be large in quantity and complex in topology and thus a pure Web frontend solution is inapplicable. Our strategy is to hybridize the traditional WMS/WFS based techniques that mostly rely on backend processing and the Google Map/BigQuery based techniques that mostly rely on frontend processing. The idea is to use the size of the set of unique FeatureIDs to determine whether a backend or a frontend processing is more beneficial. When size is large, we generate dynamic WMS/WFS layers so that query results in different visual representations can be pulled by the frontend. On the other hand, when the size is small, we choose to request feature data (including both geometry and attribute values) and create cosmetic layer(s). Either the dynamic layers, the cosmetic layers or their combinations can be composited with existing layers for better visualization and visual explorations.

## 4 Application Case, Experiments and Results

Quantifying species-environment relationships, i.e., analyzing how species are distributed on the Earth has been one of the fundamental questions studied by biogeographers and ecologists for a long time [33]. Several enabling technologies have made biodiversity data available at much finer scales in the past decade [32]. The complex relationships among environmental variables and species distribution patterns make query-driven visual explorations desirable. The uninterrupted exploration processes are likely to facilitate novel scientific discoveries effectively. As discussed in [31][22], the relevant data for large-scale biodiversity exploration can be categorized into three types: taxonomic (T), geographical (G) and environmental (E). Taxonomic data are the classifications of organisms (e.g., Family, Genus and Species) and environmental data are the measurements of environmental variables (e.g., precipitation and temperature) on the Earth. The geographical data defines the spatial tessellation of how the taxonomic data and the environmental data are observed/measured, which can be based on either the vector polygonal or the raster grid tessellation[31]. The potential research in exploring species distributions and their relationships with the environment is virtually countless given the possible combinations of geographic/ecological regions, species groups and environmental variables [31]. Previously, we have developed a desktop application called Linked Environment for Exploratory Analysis of Large-Scale Species Distribution Data (LEEASP, [31]) to visually explore 4000+ birds species range maps in the West hemisphere and 19 WorldClim environmental variables[24]. The taxonomic hierarchy is represented as a tree and visualized using Prefuse[4] with rich user interaction functionality. The software has several sophisticated mechanisms to coordinate the multiple views representing the three types of biodiversity data. A subset of the functionality of the desktop application has been provided by a WebGIS application supported by a main-memory database engine using MAQ-Tree data structure to represent rasterized polygons [22], as discussed in Section 2.2.

In this study, as an application case of the proposed framework and techniques for QDVE on large-scale geospatial data in a WebGIS environment discussed in the previous three sections, we use a different set of species distribution data and geographical data to explore global biodiversity patterns (linking with environmental data will be added in future work). For the species distribution data, we use the Global Biodiversity Information Facility (GBIF[5]) global species occurrence dataset which has 375+ million species occurrences records (as of 2012). Our preprocessing results have shown that the dataset contains 1,487,496 species, 168,280 genus, 1,142 families in 262 classes, 109 phyla and 9 kingdoms. Different from LEEASP that uses a raster grid tessellation which exploits polygon rasterization in a preprocessing step to improve response time during a QDVE task with reduced accuracy (as limited by raster grid cell size), in this study, we use vector tessellation to achieve a maximum possible spatial accuracy. Towards this end, although virtually any global administrative data (e.g., country boundary) or ecological zone data can be used for tessellation, we have chosen the World Wild Fund (WWF) ecoregion dataset[6] for such a purpose. The WWF ecoregion data has 14,458 polygons, 16,838 rings and 4,028,622 points. Linking the point species distribution data and the polygonal geographical data largely depends on the functionality and efficiency of spatial joins based on the point-in-polygon test spatial operation (known as

Zonal Summation). Our previous work on the spatial join using both the full species occurrence and the WWF ecoregion datasets indicate that it takes less than 100 seconds on a Nvidia Quadro 6000 GPU (released in 2010), which represents a 4-5 orders of magnitude of higher performance than a standalone program using libspatialindex [7]for spatial indexing and GDAL[8] for point-in-polygon test on a single CPU core [14]. The offline GPU processing performance also suggests that it may be possible to reduce the processing time to sub-second level when processing a subset of species distribution data. This is often the case for the majority of queries in QDVE tasks and the conjecture largely motivated this work.

Our experiments in this application case has several goals. First, as a WebGIS application, to examine that each component is implemented correctly and data flows among different components as expected. Second, to check that GUI interfaces work properly and ensure that users are able to use the interfaces to complete their desired QDVE tasks. Third, to demonstrate the desired efficiency and high performance by using GPU acceleration. Among the three goals, the first two goals are realized by using traditional Web techniques, such as Javascript/PHP/Python programming and HTML/XML/JSON for coding/encoding and we leave their verifications for interactive tests. We next focus on the experiments for the third goal on real time spatial join performance, which focus on backend side GPU design and implementations and are unique to this study.

Towards this end, we have randomly picked five groups of species in two selected categories, although users are allowed to pick up any combinations of species by clicking on the taxonomic tree, to evaluate the performance of spatial joins on these five species groups. The spatial join result, which tells the number of occurrences of the species in the group in each ecoregion, will be used to generate dynamic image tiles for the WebGIS frontend, as intrduced in Section 3.2. Conceptually, this is an example of applying the *Filter and Zoom* scheme in QDVE where species identifications are used for filtering the full species occurrence dataset. On the other hand, this example can also be considered as an application of the *Overview* scheme to help users understand the distributions of the species occurrence records of the subset of the species occurrence dataset with respect to the species in the chosen group.

For notation convenience, the two categories are named as C34, where the number of occurrence records of a species is between 1000 and 10,000, and C45, where the number of occurrence records of a species is between 10,000 and 100,000, respectively. There are 22,421 species in C34 and 3223 in C45, out of the 1,487,496 total species in the full dataset. Species in the two categories represent the most widely distributed ones. In our experiments, three groups of species are randomly chosen in C34 with 100, 50 and 20 species, and, two groups of species are randomly chosen in C45 with 25 and 10 species, respectively. We have chosen to use a 8192*8192 grid for grid-file based spatial indexing which seems to be appropriate. Note that using grid-file for spatial indexing is different from rasterization as used in our previous works [31][21][22] and does not sacrifice spatial accuracy. The experiment results of the four groups are shown in Table 1 using a 2013 Nvidia GTX Titan GPU with 2688 cores and 6GB memory[9]. All experiments are repeated several times to ensure that the results are consistent. The runtimes reported in the table are end-to-end times that include the runtimes of all the three modules in a spatial join (indexing for both points and polygons, filtering and refinement) but excluding disk I/O times as we assume all data are memory-resident for QDVE. Note that the polygon indexing times are the same as the WWF ecoregion data is used for all the five groups.

**Table 1.** Runtimes of Experimenting Five Selected Species Groups on GPUs (in milliseconds)

| Group | $1(C34)$ | $2(C34)$ | $3(C34)$ | $4(C45)$ | $5(C45)$ |
|---|---|---|---|---|---|
| #of species | 100 | 50 | 20 | 25 | 10 |
| # of records | 264,917 | 114,883 | 58,332 | 746,302 | 279,808 |
| Point Indexing Time (ms) | 3.0 | 2.4 | 2.0 | 4.5 | 3.1 |
| Polygon MBR Indexing Time (ms) | 29 | 29 | 29 | 29 | 29 |
| Filtering Time (ms) | 595 | 285 | 259 | 759 | 399 |
| Refinement Time (ms) | 785 | 329 | 163 | 860 | 590 |
| Total Time (ms) | 1412 | 645 | 453 | 1653 | 1021 |

From Table 1 we can see that there are two groups that have a total runtime significantly below one second and there are two groups that have a total runtime around 1.5 seconds. The average of the runtimes is around 1.0 second, which is very close to the desired sub-second level. Note that the GPU we use in the experiment was released in 2013. As more powerful GPUs are available in the past three years and the near future, we expect the average runtime can be further reduced to below 0.5 second on these GPUs. The results suggest that QDVE on global biodiversity data with support from a GPU-accelerated WebGIS backend is quite possible on a commodity personal workstation. While we have not had a chance to compare with the performance of a conventional spatial database directly for the subsets of the species occurrence data we have chosen, based on the performance on the full species occurrence dataset [14], we believe that the performance can be orders of magnitude higher and we leave a full comparison for future work.

## 5 Conclusions and Ongoing Work

In this study, motived by the increasingly available parallel hardware and the low performance of the traditional WebGIS software stacks, we have proposed a new WebGIS framework and the techniques to leverage GPU-accelerated spatial join techniques for query driven visual explorations on large-scale geospatial data. The design and implementation considerations are discussed in details and an application case on exploring global biodiversity data is presented. The experiment results have demonstrated the feasibility of the proposed framework and the desired high performance on exploring large-scale geospatial data through extensive queries.

The presented work is preliminary in nature from a system development perspective. While the major components, including query processing on the backend and QDVE related GUIs on the frontend have been designed and implemented, our prototype currently is largely geared towards the specific application case. Evolving the prototype towards a more mature system that can serve as a generic platform to accommodate more application cases certainly requires significant amount of efforts in the future. Furthermore, as CPUs are also getting more powerful with significantly larger number of cores and memory capacities, integrating both CPUs and GPUs to further enhance backend performance is not only interesting but practically useful. Finally, compared with traditional WebGIS applications, QDVE workflows are significantly more complex and require novel ideas on designing effective GUI interfaces and the direction is also left for our future work.

## References

1. Cary, A., Sun, Z., Hristidis, V., Rishe, N.: Experiences on processing spatial data with mapreduce. Proc. SSDBM'09, 302-319, 2009.
2. Grochow, K., Howe, B., et al.: Client + Cloud: Evaluating Seamless Architectures for Visual Data Analytics in the Ocean Sciences Proc. SSDBM'10, 114-131, 2010.
3. Aji, A. , Wang,F., et al.: HadoopGIS: A High Performance Spatial Data Warehousing System over Mapreduce Proc. VLDB Endow., vol. 6, no. 11, pp. 1009-1020, 2013
4. Eldawy,A., Mokbel, M F., Jonathan, C.: HadoopViz: A MapReduce framework for extensible visualization of big spatial data Proc. ICDE'16, 601-612, 2016
5. Hennessy,J. L., Patterson,D. A. Computer Architecture: A Quantitative Approach (5th Ed.) Morgan Kaufmann, 2011
6. Kirk,D. B., Hwu,W.-m. W.: Programming Massively Parallel Processors: A Hands-on Approach, 2nd ed., Morgan Kaufmann, 2012.
7. Kendall, W., Glatter, M., et al.: Terascale data organization for discovering multivariate climatic trends. In: SuperComputing'09. (2009) 1–12
8. Healey,R., Dowers,S. , et al: Parallel Processing Algorithms for GIS. CRC, 1997
9. Clematis,A. , Mineter,M.,Marciano,R.: High performance computing with geographical data. Parallel Computing, 29(10):1275-1279, 2003.
10. You,S., Zhang,J.,Gruenwald,L. : Parallel spatial query processing on GPUs using R-trees Proc. BigSpatial@SIGSPATIAL 23-31, 2013
11. Zhang,J., You,S., Gruenwald,L. : Large-Scale Spatial Data Processing on GPUs and GPU-Accelerated Clusters ACM SIGSPATIAL Special, vol. 6, no. 3, pp. 27-34, 2014.
12. Zhang,J., You,S., Gruenwald,L. : High-Performance Spatial Query Processing on Big Taxi Trip Data Using GPGPUs Proc. IEEE International Congress on Big Data, 72-79, 2014.
13. Zhang,J., You,S., Gruenwald,L. : Parallel online spatial and temporal aggregations on multi-core CPUs and many-core GPUs Inf. Syst. 44: 134-154, 2014
14. Zhang,J., You,S., Gruenwald,L. : Efficient Parallel Zonal Statistics on Large-Scale Global Biodiversity Data on GPUs. Proc. BigSpatial@SIGSPATIAL 35-44, 2015
15. Zhang,J.,You,S., Xia,Y. : Prototyping A Web-based High-Performance Visual Analytics Platform for Origin-Destination Data: A Case study of NYC Taxi Trip Records Proc. UrbanGIS@SIGSPATIAL 16-23, 2015
16. You,S.,Zhang,J., Gruenwald,L. : High-performance polyline intersection based spatial join on GPU-accelerated clusters. Proc. BigSpatial@SIGSPATIAL 42-49, 2016
17. Shekhar S., Chawla,S.: Spatial Databases: A Tour. Pearson(2003)
18. newblock Jacox, E. H., Samet, H. Spatial join techniques ACM TODS, 32(1), No. 7., 2007
19. Gaede, V., Gunther, O.: Multidimensional access methods. Computing Surveys, 30(2), 170-231,1998
20. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc. (2005)
21. Zhang,J., Gertz,M., Gruenwald, L.: Efficiently managing large-scale raster species distribution data in PostgreSQL. Proc. ACM-GIS'09,316-325, 2009

22. Zhang J.: A high-performance web-based information system for publishing large-scale species range maps in support of biodiversity studies Ecological Informatics 8: 68-77, 2012
23. Zhang J., You S., Supporting Web-based Visual Exploration of Large-Scale Raster Geospatial Data Using Binned Min-Max Quadtree Proc. SSDBM'10, 379-396, 2010.
24. Hijmans, R.J., Cameron, S.E., et al: Very high resolution interpolated climate surfaces for global land areas. Int.J. of Climatology 25(15), 1965-1978, 2005 `http://www.worldclim.org/current`
25. Zhang,J., You, S., Gruenwald, L.: Parallel quadtree coding of large-scale raster geospatial data on GPGPUs Proc. ACM-GIS'11, 457-460, 2011.
26. Zhang,J., You, S., Gruenwald, L.: High-performance quadtree constructions on large-scale geospatial rasters using GPGPU parallel primitives IJGIS, 27(11): 2207-2226, 2013
27. Zhang,J., You, S.: Dynamic tiled map services: supporting query-based visualization of large-scale raster geospatial data Proc. COM.GEO'10, 2010.
28. Aref,G. A., Ilyas,I.F.: SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees Journal of Intelligent Information Systems (JIIS), 17(2/3), 215 -240, 2001
29. Aji, A., Teodoro, G., Wang, F.: Haggis: turbocharge a MapReduce based spatial data warehousing system with GPU engine. Proc. ACM BigSpatial@SIGSPATIAL 15-20,2014
30. Chavan,H., Alghamdi,R.,Mokbel, M.F.: Towards a GPU accelerated spatial computing framework. Proc. ICDE Workshops, 135-142,2016
31. Zhang,J.,Gruenwald, L.: Embedding and extending GIS for exploratory analysis of large-scale species distribution data. Proc. ACM-GIS'08, No. 28, 2008
32. Bisby, F. A.: The quiet revolution: Biodiversity informatics and the Internet Science, 289 (5488), pp. 2309-2312, 2000
33. Cox, C., Moore, P.: Biogeography: An Ecological and Evolutionary Approach (7th Ed.) Wiley, 2005
34. Shneiderman, B.: The eyes have it: A task by data type taxonomy for information visualizations. Proc. IEEE Symposium on Visual Languages, 336-343, 1996
35. Shneiderman, B.: The eyes have it: A task by data type taxonomy for information visualizations. Proc. IEEE Symposium on Visual Languages, 336-343, 1996
36. Keim, D.A., Mansmann, F.,et al.: Challenges in visual data analysis Proc. IEEE Conference on Information Visualization, 9-16, 2006
37. McCool, M., Robison, A.D. , Reinders, J.: Structured Parallel Programming: Patterns for Efficient Computation, Morgan Kaufmann, 2012.
38. Merrill, D., Grimshaw, A. S., : High performance and scalable radix sorting: a case study of implementing dynamic parallelism for GPU computing, Parallel Processing Letters, 21 (2), pp. 245-272, 2011.

## Notes

[1]`http://www.sai.msu.su/~megera/postgres/gist/ltree/`
[2]`http://www.paraview.org/`
[3]`https://cloud.google.com/bigquery/`
[4]`http://www.prefuse.org/`
[5]`http://www.gbif.org/`
[6]`http://www.worldwildlife.org/biomes`
[7]`https://libspatialindex.github.io/`
[8]`http://www.gdal.org/`
[9]`http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan/specifications`