# A Vision for GPU-accelerated Parallel Computation on Geo-Spatial Datasets

Prof. Sushil K. Prasad, PhD Students: Michael McDermott, Satish Puri, Dhara Shah and Danial Aghajarian
Department of Computer Science, Georgia State University, USA
Prof. Shashi Shekhar, Department of Computer Science, University of Minnesota, USA
Dr. Xun Zhou, Department of Management Sciences, University of Iowa, USA

**Abstract**

*We summarize the need and present our vision for accelerating geo-spatial computations and analytics using a combination of shared and distributed memory parallel platforms, with general-purpose Graphics Processing Units (GPUs) with 100s to 1000s of processing cores in a single chip forming a key architecture to parallelize over. A GPU can yield one-to-two orders of magnitude speedups and will become increasingly more affordable and energy efficient due to mass marketing for gaming. We also survey the current landscape of representative geo-spatial problems and their parallel, GPU-based solutions.*[1]

## 1 Introduction

Geo-spatial datasets are large and the related computations and analytics are computationally intensive. For example, it takes about 13 minutes for polygonal overlay of two GIS shapefiles *USA Detailed Water Bodies* and *USA Block Group Boundaries* containing about 600K polygons using ArcGiS 10.1 on Intel Core i5 processor, and it takes roughly 20 hours to compute the spatial join of a polyline table with 73M records representing the contiguous USA with itself on an Amazon EC2 instance [31]. Therefore, harnessing parallel processing capabilities of modern hardware platforms is essential. Parallel processing technology is not just for real-time monitoring and steering, making scientific and policy decisions within reasonable time, visualizing huge datasets, and for enabling memory-constrained computations which cannot otherwise be solved on sequential computers. This technology's vast speed gains will also result in new capabilities for the masses that are barely imagined today, analogous to what airplanes have meant over ships and trains.

The modern computing platforms are parallel, distributed and heterogeneous, ranging from mobile devices to desktops to high performance computing (HPC) clusters and cloud data-centers. Mobile devices already harness the backend cloud servers over the Internet for daily chores such as mapping and navigation. These are expected to launch high-end computations and data analytics for future applications. For GIS and other scientific applications such as multi-agent epidemic simulations, hot spot detections, and overlay computations, scientists will increasingly employ the powerful *shared-memory parallel processing* capabilities of their laptops and desktops. The individual compute nodes of these devices (as of clusters) consist of multi-core CPUs containing tens of processing cores and many-core GPUs containing hundreds to thousands of cores, both with

[1] For readers interested in shared-memory parallel programming for geo-spatial problems, we are setting up a work-in-progress webpage at `http://www.cs.gsu.edu/~dimos/?q=GIS-Resouces`, which currently contains source code and input and output data for (i) 1-D interesting region discovery using C/CUDA and Hadoop+CUDA (ii) bilateral filtering using C/CUDA, and (iii) Polygonal overlay system using Java/Hadoop and C# over Azure Cloud.

shared-memory architectures. For larger scale computing over huge data sets, possibly from multiple online sources, these devices will work in tandem with dedicated clusters and will increasingly employ on-demand HPC cloud clusters (*distributed-memory parallel processing*). In fact, for small-to-medium scale computations, laptops/desktops would be cost-effective alternatives to the clusters as these are not readily accessible to most scientists not involved with HPC.

Some scientists have begun employing distributed processing over clusters using map-reduce programming such as Hadoop (e.g., Spatial-Hadoop [9]) and message-passing programming such as MPI (Message-Passing Interface - e.g. PySAL [32], hydrology [35]). However, these projects currently do not employ the shared memory capabilities of the compute nodes. A few have employed multi-core capabilities of CPUs using OpenMP/Pthread (e.g., for plane-sweep [23, 19]), but not GPUs. Not harnessing the many-core GPUs typically results in a loss of at least 200%-300% speedup (2-3 fold) compared to a multi-core CPU, which can be obtained with reasonable porting effort. More often, the loss is one to two orders of magnitude for many computations, but that does require medium-to-expert level parallelization effort. The latter class of computations are typically either *embarrassingly parallel*, such as independent bags of uniform tasks requiring little communication among the processing cores, or are *regular computations*, those which have regular communication patterns among neighboring tasks. Some are *irregular computations* with no well-defined communication pattern due to irregular spatial and/or temporal task or data distributions, but can yield an order of magnitude speedup with sophisticated data structures and algorithms.

Our vision is that the bulk of geo-spatial software packages will employ both distributed and shared memory parallel processing including the GPUs toward scalable solutions. How will this happen? Clearly, this will require the collective effort of the GIS community and collaboration with computer scientists. Computer scientists and their graduate students will be interested in such interdisciplinary collaborations because of an excellent potential for computer science research in algorithms, data structures, database, data mining and systems over interesting datasets and domain applications. Such an investment is even more sensible given that the GPU accelerators are here to stay, will continually improve both in performance and energy foot-print, and will become more affordable due to mass usage by the gaming industry. The promising news is that a few groups are employing both distributed and shared memory parallel computing including GPUs (e.g., polygonal overlay over medical images using MPI and CUDA [37]).

In this article, we explain how you can recognize which of the three categories your favorite problems fall into by understanding the nature of similar problems and get a sense of the potential speedup and how much parallelization effort may be involved. You will also obtain pointers to some representative state-of-art projects, specially for the non-embarrassignly parallel problems. For embarrassing parallel problems, Section 2 illustrates several GIS and other problems, including hotspot detection problem for a fixed radius neighborhood. Section 3 describes regular computations, some related projects, and gives detailed description of interesting path/region discovery problem and our CUDA-based analytics solutions. For irregular computations, Section 4 reviews the literature with recent GPU parallelization efforts and provides details on our polygonal overlay MPI-GIS system.

## 2 Embarrassingly Parallel Problems

An embarrassingly parallel problem, is the easiest kind of parallelism as these problems exhibit little to no task dependency [11]. Most of these problems can be solved with little communication between parallel tasks and this makes them distinguishable from other problems which require exchanging intermediate results among several tasks. In other words, an embarrassingly parallel problem imposes the least amount of overhead to be parallelized and, therefore, usually does not suffer from parallel slowdown.

A celebrated example of an embarrassingly parallel problem is shared-memory matrix multiplication in which for two $n \times n$ matrices, the multiplication process is split into $n^2$ tasks, each responsible for calculating one element of the result matrix [5]. In this case, each task needs one row of the first input matrix and one

column of second input matrix to compute its result. However, it does not need to communicate with any other task. All $n^2$ tasks can function independently. [20] reports 776 to 11183-fold speedups using CUDA on a GPU for 4096X4096 matrices in comparison with a traditional sequential code on a single CPU core, using a range of improvements and parameter tunings including tile/thread-block dimensions, thread occupancy on cores, computational load per thread, and hiding memory latency between device memory shared by all streaming multipocessors and smaller memory shared by 8 cores within a multiprocessor considering number/volume of memory transctions, coalesced access, bandwidth utilization, caching, bank conflicts, register usage, etc. These GPU implementations correspond to 27 to 407-fold speedups relative to a 4-thread OpenMP cache-optimized code on the multicore CPU. These experiments are reported for a 2.8GHz Xeon quad-core CPU and 240-core NVIDIA Tesla C1060 GPU with 8 streaming multiprocessors each with 30 cores. Another example is searching big files to find specific phrases. As the searching process over different part of one file or over several files is independent, the search operation can be done in parallel; concurrent tasks do not require communication with each other at all. Sifting through large volume shape files to find specific objects has been extensively used in GIS applications [14].

Some fundamental building blocks of data mining algorithms can be formulated as embarrassingly parallel problem. Some examples in this category are finding area in a region, generating minimal bounding rectangles of individual polygons, points in polygon test, update phases of k means clustering, etc. In points in polygon test [15] for a set of given points and a polygon, the membership of each point in the given polygon is to be determined. Membership status of each point is independent of the status of other points, making it embarrassingly parallel. Similarly, the problem of forming k clusters in a given GIS grid is embarrassingly parallel. Initially $m$ random points are selected as cluster centers and each center is moved to mean value of distances of $k$ nearest neighbors from each center. These $m$ centers are updated iteratively. New mean value of each cluster depends only on old mean value of that cluster; hence the problem is task and data independent ([10] reports 13X speedup on NVIDIA GeForce 8800 GTX Ultra with 128 cores). Most pixel-based or fixed neighborhood based computations over raster data such as intersection over multiple data layers and simple stencil-based image processing tasks fall into this category ([4] reports 169 fold speedup for bilateral filtering on Nvidias GTX 280 GPU with 240 cores). It is also easy to implement these embarrassingly parallel problems using map-reduce paradigm, wherein the intermediate results calculated by each slave nodes are conveyed to a master node.([40] reports linear speedup in a distributed cluster for k means clustering.)

**Hotspot detection over a fixed neighborhood:** Hotspots are regions exhibiting unusual aggregation or outcome of a phenomenon. Some examples of hotspot detection are detecting aggregation of some epidemic, snowfall prediction, detecting most profitable regions to open a store for a business, regions of high level of radiations on a planet, etc. In simple formulations, hotspot detection depends on the value of the point in relation to its neighbors. In mathematical terms, on a given domain of data set, each point of the domain holds a value of discrete/continuous random variable. The value of the random variable is compared to the extant of a fixed radius using some predefined algorithm, and based on these comparisons, a spatial pattern of its distribution is determined. The hotspots are the regions exhibiting the desired values/shapes of this pattern [27]. This problem is embarrassingly parallel because usually each unit of the domain is processed with the same algorithm and is independent of the rest of the data set outside its radius of comparison (although, not all formulations are embarrassingly parallel). [21] reports up to 400X speedup on NVIDIA GTX280 GPUs.

# 3 Regular Computations

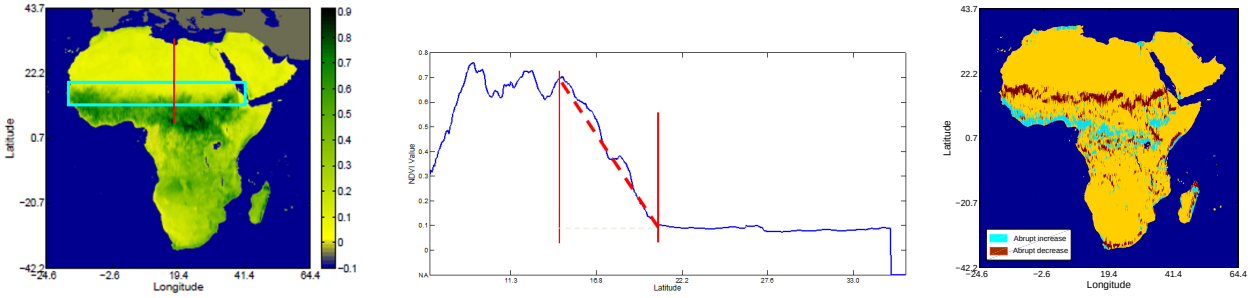
We now introduce the regularly structured problems and data which are the first tier of what is considered non-trivial parallelization. Some raster data computation, complex matrix manipulation problems, problems with many data independent loops, and certain types of graph problems typically fall within this category. Paral-

lelization is achieved by exploiting the independent nature of the instructions and/or the data in order to achieve good speedup. Matrix manipulation and raster data in particular lend themselves very well to GPU computation due to their reliance on basic data types and array-based structures. Graph algorithms can be performed on GPUs typically using their adjacency matrix representations which can then be stored conveniently on the GPU [18]. All-to-all Floyd-Warshall shortest path algorithm or Gaussian elimination for a system of linear equations, for example, manifest regular pattern of data access when parallelized [13].

The data collected in Earth Science are usually stored as raster data, where space and time are partitioned into regular grids (e.g., latitude and longitude, weeks/months/years) with attributes imposed on the grids (e.g., rainfall, vegetation cover index). Examples of such data include remotely sensed images of Earth surface at various resolutions [8, 24], ground observations of temperature, precipitation, etc. [16]. Pattern discovery on such datasets are potentially suitable for parallel computing due to the regular structures (grid) in the data. We introduce one example from our recent work, namely, the interesting sub-path/sub-region discovery problem [41].

## Interesting Sub-path/Region Discovery Problem

Given a spatiotemporal (ST) dataset and a path embedded in its spatiotemporal framework, the goal of the interesting sub-path discovery problem is to identify all qualifying contiguous subsets of the given path according to the given interest measure and interestingness test. The ability to discover interesting sub-paths is important for many application domains. For example, vegetation cover is often used to study the response of ecological zones to climate change, which may vary across different ecological zones in the world. Given a path (e.g., along a longitude in Africa) and an interest measure of abrupt change, one can find sub-paths (e.g., the paths across the Sahel) with sharp increases (decreases) of vegetation cover. Such sub-paths may outline the spatial footprint of the transitional areas (known as ecotones [25]) between ecological zones. Due to their vulnerability to climate changes, finding and tracking ecotones gives us important information about how the ecosystem responds to climate changes. Figure 1(a) shows the Sahel region (highlighted area in the box), a well-known ecotone in Africa. The color represents the amount of vegetation measured in the normalized difference of vegetation index (NDVI). The vegetation cover along a spatial path (in red) is plotted in Figure 1(b) as a sequence. As can be observed, the sub-path between 15N and 20N latitude exhibit an abruptly decreasing trend. This reflects the transition from tropical savannah to desert.




(a) Vegetation cover in Africa (from GIMMS Dataset [8, 36])

(b) Vegetation cover along highlighted longitudinal path (South to North)

(c) The footprint of Sahel discovered

Figure 1: An example of interesting sub-path in vegetation cover data (best viewed in color).

In the specific application scenario described above, the goal is to discover all the sub-paths with abrupt increase/decrease along each longitude of the input dataset/image. Particularly, we require that an abrupt change sub-path should not be a subset of others so that redundant results could be removed. By performing this analysis on each longitudinal column of the input data image, we are able to outline the entire Sahel region by combining the footprint of Sahel along each longitudinal band. Figure 1(c) shows the output of the problem

with $min\_change$ = the 90th percentile of all the longitudinal unit changes, where maroon and aqua represent decreases and increases of vegetation cover from south to north, respectively.

## Sequential and GPU Solutions

The sequential solution to the above problem is non-trivial to design in that (1) the patterns lack of monotonicity (decreasing segments in a increasing sub-path), (2) Unknown maximum length of change ranging from 0 to the entire path, and (3) large volume of input data. A naive solution to this problem has two steps. Step 1: enumerate all the sub-paths in each longitudinal path in the dataset and test the interest measure function to find a list of candidate patterns. Step 2: enumerate every pair of sub-paths in the candidate list to eliminate dominated (i.e., subset) sub-paths. This solution has high time cost. For a single input longitudinal path with $n$ locations, the time complexity reaches $O(n^4)$ in the worst case, bringing the total time complexity on a data image with $m$ such longitudinal columns to $O(m \cdot n^4)$ in the worst case.

In our previous work, a sub-path enumeration and pruning (SEP) approach was proposed to improve the performance. First, a linear scan of all the units in the input data path is performed to precompute the basic function needed in the interest measure and store them in a set of lookup tables. This step allows a constant-time evaluation of each sub-path. Second, a row-wise traversal order is used to enumerate all the sub-paths from longer ones to short ones. Once a sub-path is identified as an interesting sub-path, all its subsets will be pruned from the enumeration to avoid further cost. Finally, the algorithm filters dominant sub-paths in the candidate list. The SEP row-wise algorithm takes $O(n)$ (to build the lookup table) in best scenario and $O(n^2)$ in the worse case. For an entire input image ($m \times n$), the time complexity is reduced to $O(m \cdot n^2)$.

**GPU Implementation:** The GPU implementation of the interesting subpath problem is quite straightforward and leverages the fact that each worst-case possible path is data independent from every other query and each thread is largely performing the exact same set of computations. We can perform the naive algorithm in parallel by launching a thread for every possible interesting interval, in effect eliminating a theoretical order of magnitude from the total complexity. In many respects this gives a very nice speedup, however the SEP algorithm was found to be roughly 50 times faster [28].

Implementing the SEP algorithm on the GPU required some re-engineering in order to accommodate concepts that do not translate nicely to the GPU architecture such as dynamic data structures [26] and code branching. However, as the overall behavior of the algorithm is quite regular it produced very nice results that, with some constraints relaxed, allowed for near real-time computation and rendering of results to be achieved [28]. Instead of naively launching a thread for every conceivable outcome only half as many threads were launched. These threads were launched more intelligently with kernel dimensions that limited the potential effects of branching while also coalescing global memory read and write access. This also allowed leveraging shared memory on the GPU which drastically lowered compute time by reducing the time necessary for data access. Overall this sped up computation from 836ms for sequential Row-wise SEP to 35.2ms on the GPU for a single raster image (finding all intervals in the image), and from 576s to 20.1s for the entire raster image dataset.

This particular problem lent itself very well to implementation on the GPU as the dataset was raster data [8] and the individual pieces of the total dataset were rather small (2 MB). This allowed further parallelization to be undertaken. Each image being independent allowed us to further distribute this problem using the Hadoop framework across multiple GPU device nodes and achieve a near linear speedup for each additional node that was added. It also allowed us to run multiple instances of the program on a single GPU, however this is much harder as the intermediary data gets quite expensive to hold in memory and the GPU (at the time of experiment, NVidia GTX480) is resource limited to roughly 1 GB of memory on a standard consumer grade GPU.

# 4 Irregular Computations

GPUs are very effective at exploiting parallelism in regular, data-parallel computations, and we have seen some examples of efficient GPU parallelizations. However, many algorithms have to build, traverse, and update irregular data structures such as trees, graphs, and priority queues to solve a given problem. Examples of irregular computations includes breadth-first search, single-source shortest paths, n-body simulation, etc. In the context of spatial algorithms, spatial overlay and join are examples of irregular computation. In these examples, accesses to tree/graph based data structure is unpredictable and data-dependent. Moreover, parallelization of these data structure is hard because of the underlying tree topology and the fine-grained computation.

*Spatial overlay* is the process of interrelating several spatial features (points, lines, or polygons) from multiple datasets or layers of data, which creates a new output layer, visually similar to stacking several maps of the same region together. Spatial overlay has been accelerated by NVIDIA Tesla GPU in [7, 22]. McKenney et al. [22] developed GPU implementation of line segment intersection and the arrangement problem for overlay computation. For some datasets, authors show that their implementation of geospatial overlay on GPU runs faster than CPU-based plane-sweep implementation. Audet et al. [7] developed CUDA implementation of Uniform Grid based overlay algorithm originally designed by Franklin et al. [12].

*Spatial Join* is a type of table join operation in which fields from one layer's attribute table are appended to another layer's attribute table based on the relative locations of the features in the two layers. A typical example of a spatial join is "Find all pair of rivers and cities that intersect." Recent work on GPU based parallel spatial join are discussed in [39, 33, 38, 6, 34]. Authors in [38] describe how a spatial join operation with R-Tree can be implemented on a GPU. Hadoop map-reduce based spatial join is accelerated with GPU in [6] where GPU threads perform spatial join computation in parallel by first converting spatial data to pixel format.

### Polygonal overlay

Arguably, polygonal overlay computation is one of the difficult computations due to irregular distribution of arbitrary shaped polygons. Our Azure cloud based parallel system for polygon overlay known as *Crayons* is described in [1, 3]. It achieves 90x speedup for its task processing phase on Azure cloud - an embarrassingly parallel phase once pairs of potentially intersecting polygons have been identified constituting independent tasks. However, the end-to-end speed up is only 3 fold. An MPI version of *Crayons* is described in [2]. Polygon overlay processing algorithms using Hadoop map-reduce framework are described in [30], with three versions depending on the nature of the datasets, yielding up to 10x speedup. GPU-based parallelization of key tree-based data structures, namely R-tree [29], with 200 fold speedup for construction, and heap [17], with 40X speedup for concurrent inserts and min-deletes, have been explored recently. The parallel R-tree has been employed in *MPI-GIS*, our polygon overlay system, which yields 40x to 70x speedup in comparison to ArcGIS [28]. The architecture of *MPI-GIS* is shown in Figure 2a. Figure 2b shows the execution time for overlaying two real-world GIS datasets.

# 5 Conclusions

We presented three classes of computations, namely embarrassignly parallel, regular and irregular compytations, and illustrated each with representative parallelization of GIS computations and analytics over GPUs.

# References

[1] D. Agarwal and S. K. Prasad. Lessons learnt from the development of GIS application on azure cloud platform. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 352–359. IEEE, 2012.

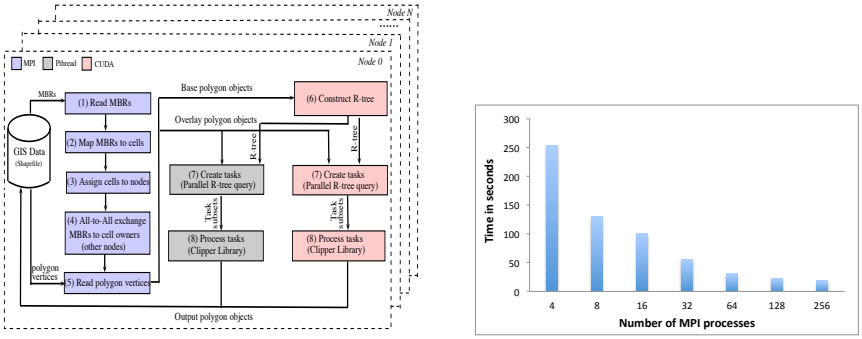



Figure 2: (a) Architecture of *MPI-GIS* for Polygon Overlay using MPI + Pthread + GPU. (b) Execution time of *MPI-GIS* with varying number of MPI processes for overlaying *USA Detailed Water Bodies* and *USA Block Group Boundaries* on a cluster with 32 compute nodes having 8 cores/node


[2] D. Agarwal, S. Puri, X. He, and S. K. Prasad. A system for GIS polygonal overlay computation on linux cluster-an experience and performance report. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1433–1439. IEEE, 2012.

[3] D. Agarwal, S. Puri, X. He, and S. K. Prasad. Cloud computing for fundamental spatial operations on polygonal GIS data. *Cloud Futures*, 2012.

[4] D. Agarwal, S. Wilf, A. Dhungel, and S. K. Prasad. Acceleration of bilateral filtering algorithm for manycore and multicore architectures. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 78–87. IEEE, 2012.

[5] R. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39:39–5, 1995.

[6] A. Aji, G. Teodoro, and F. Wang. Haggis: Turbocharge a MapReduce based spatial data warehousing system with GPU engine. In *Proceedings of the ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. ACM, 2014.

[7] S. Audet, C. Albertsson, M. Murase, and A. Asahara. Robust and efficient polygon overlay on parallel stream processors. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 294–303. ACM, 2013.

[8] C.J.Tucker, J.E. Pinzon, M.E. Brown. Global inventory modeling and mapping studies. Global Land Cover Facility, University of Maryland, College Park, Maryland, 1981-2006, 2006.

[9] A. Eldawy, Y. Li, M. F. Mokbel, and R. Janardan. CG_Hadoop: Computational geometry in MapReduce. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 284–293. ACM, 2013.

[10] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell. A parallel implementation of k-means clustering on gpus. In *PDPTA*, pages 340–345, 2008.

[11] I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[12] W. R. Franklin, C. Narayanaswami, M. Kankanhalli, D. Sun, M.-C. Zhou, and P. Y. Wu. Uniform grids: A technique for intersection detection on serial and parallel machines. In *Proceedings of Auto-Carto*, volume 9, pages 100–109, 1989.

[13] A. Grama, G. Karypis, V. Kumar, , and A. Gupta. *Introduction to Parallel Computing*. 2004.

[14] E. group. ESRI Shapefile Technical Description. 1998.

[15] E. Haines. Graphics gems iv. chapter Point in Polygon Strategies, pages 24–46. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[16] I. Harris, P. Jones, T. Osborn, and D. Lister. Updated high-resolution grids of monthly climatic observations–the cru ts3. 10 dataset. *International Journal of Climatology*, 34(3):623–642, 2014.

[17] X. He, D. Agarwal, and S. K. Prasad. Design and implementation of a parallel priority queue on many-core architectures. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10. IEEE, 2012.

[18] S. Hong, S. K. Kim, T. Oguntebi, and K. Olukotun. Accelerating cuda graph algorithms at maximum warp. *SIGPLAN Not.*, 46(8):267–276, Feb. 2011.

[19] A. B. Khlopotine, V. Jandhyala, and D. Kirkpatrick. A variant of parallel plane sweep algorithm for multicore systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(6):966–970, 2013.

[20] S. R. Li, Junjie and S. Sahni. Gpu matrix multiplication. *Multicore Computing: Algorithms, Architectures, and Applications*, 2013.

[21] Y. Li, K. Zhao, X. Chu, and J. Liu. Speeding up k-means algorithm by gpus. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 115–122. IEEE, 2010.

[22] M. McKenney, G. De Luna, S. Hill, and L. Lowell. Geospatial overlay computation on the GPU. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 473–476. ACM, 2011.

[23] M. McKenney and T. McGuire. A parallel plane sweep algorithm for multi-core systems. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 392–395. ACM, 2009.

[24] NASA Land Processes Distributed Active Archive Center (LP DAAC). The Moderate Resolution Imaging Spectroradiometer (MODIS) data collections, USGS/Earth Resources Observation and Science (EROS) Center, Sioux Falls, South Dakota, 2001.

[25] I. Noble. A model of the responses of ecotones to climate change. *Ecological Applications*, 3(3):396–403, 1993.

[26] I. Nvidia. Cuda c best practices guide. http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/, August 2014.

[27] J. K. Ord and A. Getis. Local spatial autocorrelation statistics: distribution issues and an application. *Geographical Analysis*, 27:286–306, 1995.

[28] S. K. Prasad, S. Shekhar, M. McDermott, X. Zhou, M. Evans, and S. Puri. Gpgpu-accelerated interesting interval discovery and other computations on geospatial datasets: A summary of results. In *Proceedings of the 2Nd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, BigSpatial '13, pages 65–72, New York, NY, USA, 2013. ACM.

[29] S. K. Prasad, S. Shekhar, M. McDermott, X. Zhou, M. Evans, and S. Puri. Gpgpu-accelerated interesting interval discovery and other computations on geospatial datasets: a summary of results. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, pages 65–72. ACM, 2013.

[30] S. Puri, D. Agarwal, X. He, and S. K. Prasad. MapReduce algorithms for GIS polygonal overlay processing. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1009–1016. IEEE, 2013.

[31] S. Ray, B. Simion, A. D. Brown, and R. Johnson. A parallel spatial data analysis infrastructure for the cloud. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 274–283. ACM, 2013.

[32] S. J. Rey and L. Anselin. Pysal: A python library of spatial analytical methods. In *Handbook of applied spatial analysis*, pages 175–193. Springer, 2010.

[33] B. Simion, S. Ray, and A. D. Brown. Speeding up spatial database query execution using gpus. *Procedia Computer Science*, 9:1870–1879, 2012.

[34] C. Sun, D. Agrawal, and A. El Abbadi. Hardware acceleration for spatial selections and joins. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 455–466. ACM, 2003.

[35] D. Tarboton. High Performance Hydrologic Terrain Analysis and CyberGIS. *CyberGIS*, 2014 (http://cybergis.illinois.edu/events/cybergis14/program.html).

[36] C. Tucker, J. Pinzón, M. Brown, D. Slayback, E. Pak, R. Mahoney, E. Vermote, and N. El Saleous. An extended AVHRR 8-km NDVI dataset compatible with MODIS and SPOT vegetation NDVI data. *International Journal of Remote Sensing*, 26(20):4485–4498, 2005.

[37] K. Wang, Y. Huai, R. Lee, F. Wang, X. Zhang, and J. H. Saltz. Accelerating pathology image data cross-comparison on cpu-gpu hybrid systems. *Proceedings of the VLDB Endowment*, 5(11):1543–1554, 2012.

[38] T. Yampaka and P. Chongstitvatana. Spatial join with R-tree on graphics processing units, IC2IT, 2012.

[39] J. Zhang and S. You. Speeding up large-scale point-in-polygon test based spatial join on GPUs. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, pages 23–32. ACM, 2012.

[40] W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on mapreduce. In *Cloud Computing*, pages 674–679. Springer, 2009.

[41] X. Zhou, S. Shekhar, P. Mohan, S. Liess, and P. K. Snyder. Discovering interesting sub-paths in spatiotemporal datasets: A summary of results. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 44–53. ACM, 2011.