Haggis: Turbocharge A MapReduce based Spatial Data Warehousing System with GPU Engine

Ablimit Aji[*]
HP Labs
ablimit@hp.com

George Teodoro
University of Brasília
teodoro@cic.unb.br

Fusheng Wang
Emory University
fusheng.wang@emory.edu

ABSTRACT

Spatial query processing involves complex multidimensional objects and compute intensive spatial operations, and therefore requires a high performance approach to meet the rapid data analytics requirements of modern spatial applications. Recently, MapReduce based spatial query systems have become a viable solution for many *data intensive* query tasks, and gained widespread adoption in both academia and industry. At the same time, GPUs have been successfully utilized in many applications that require high performance computation. Both approaches, GPU and MapReduce, have their own limitations and advantages, and have been separately utilized in spatial query processing tasks to boost application performance. However, it is unclear that how MapReduce and GPU, two vastly different parallelization techniques, can be fused together to effectively deal with the spatial big data challenges. In this paper, we explore such synergy of parallelization techniques for large scale spatial query processing. We extend Hadoop-GIS, a MapReduce based spatial query system, and provide GPU accelerated spatial query processing capability at the engine level. We evaluate the system on a real world dataset, and demonstrate that GPU accelerated system can gain considerable performance improvements. We also show how other factors such as partition granularity, task scheduling between CPU and GPU can impact the query performance.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial Database and GIS, Systems*

General Terms

Experimentation, Performance

[*]paper is based on work performed at Emory University as a Ph.D. student.

BigSpatial'14, November 04-07 2014, Dallas/Fort Worth, TX, USA
ACM 978-1-4503-3132-6/14/11 ... $15.00
http://dx.doi.org/10.1145/2676536.2676539 .

Keywords

Spatial Query Processing, Spatial Data Partition, MapReduce, Load Balancing, GPU

1. INTRODUCTION

In a wide range of application domains, data is being collected at an unprecedented scale, and used to drive innovation and business success. Scientific research and business decisions, that used to be based on limited amount of information or complete guesswork, now can be performed more effectively in a data-driven manner by utilizing massive amounts of data.

Data analytics on spatial and spatio-temporal data is one of the fast growing areas in the Big Data landscape that is increasingly suffering from inability to deal with massive amounts of data using traditional analytics systems. Recently, MapReduce [12] based data processing systems have emerged as a viable solution to the big data challenges of our time. By partitioning the data and running queries in parallel on large number of commodity hardware, MapReduce achieves high scalability and fault tolerance at low cost. Spatial data analytics [7, 13, 20, 23] also benefits from the rising tide of MapReduce, and conventional systems start to embrace the approach [1].

MapReduce is a simple and effective parallelization approach for many large scale query processing tasks. Previous research on spatial query workload characterization shows that spatial queries are a lot more compute-intensive than conventional non-spatial query workloads [25]. While MapReduce can effectively address data-intensive aspects of large scale spatial queries, it is not well suited to handle compute-intensive aspect of spatial queries. The multidimensional nature of spatial data analytics and the complexity of spatial queries requires a high performance approach that can leverage parallelism at multiple levels for query processing. In recent years, GPGPU has become mainstream as many-core computer architecture and programming techniques become mature. Now, a single machine may be equipped with multiple processors, e.g., multi-core CPUs and GPUs, and such hardware configurations is readily available on major cloud computing platforms. In the coming years, such heterogeneous parallel architecture will become dominant, and software systems must fully exploit this heterogeneity to deliver performance growth [10].

In this paper, we study opportunities for such synergy between two different parallelization techniques, MapReduce and GPU, for large scale spatial query processing. Build upon a major MapReduce based spatial data warehousing system Hadoop-GIS [7, 6, 5, 4], we develop ***Haggis*** – **Ha**doop-**GIS** accelerated with a **G**PU engine. Due to the difference in the parallelization model, there are several problems that need to be addressed to leverage the available computing power of hybrid systems. First, integration of two programing models is not trivial. Previous research efforts [16] propose to modify the MapReduce programming model to bring those two parallelization frameworks together in a unified framework. In Haggis, we take a decoupled approach in which the GPU is used as an engine accelerator without modifying the underlying MapReduce programming model. Second, as GPU computation requires the data to be moved to the device memory which incurs off-chip memory movement cost. Such data movement needs to be well orchestrated to reduce the I/O cost and fully exploit the computation power of GPU. Third, to achieve best system performance, the task scheduler needs to judiciously place tasks on CPU or GPU, so that the system resource is fully utilized. There are several design decisions towards building an integrated system, and we dicuss those issues in detail in this paper.

The rest of the paper is organized as follows. We describe the characteristics of spatial query processing and motivations of our work in Section 2. The system design is presented in Section 3. We empirically evaluate the proposed solution in Section 4. The related work is discussed in Section 5, and we conclude the paper in Section 6.

2. BACKGROUND

2.1 Characteristics of Spatial Queries

Most spatial queries are compute-intensive [25] as they involve geometric computations on complex multi-dimensional spatial objects. Spatial objects have complex extent that generally described with multi-dimensional data points. For example, typical spatial objects such as lines and polygons are represented with a set of points in a two dimensional space (in a 2D Euclidean coordinate system), and those data points are stored and processed together. Therefore, even simple operations on those objects incur considerable computation cost and I/O cost.

To avoid the high cost of the geometry computation and reduce unnecessary disk I/O, spatial queries employ a *filter-and-refine* strategy in which queries are processed in two phases [21]. During the *filter* phase, spatial objects are approximated with minimum bounding rectangles (MBRs), and objects that do not satisfy the query predicates even on the MBRs are eliminated. During the refinement step, candidate objects from the filter step are processed with real geometric operations, and the objects that satisfy the query predicates are reported as final query results. While spatial filtering through MBRs can be accelerated through spatial access methods, spatial refinements such as polygon intersection verification are highly expensive operations. For example, spatial join queries, such as spatial cross-matching or overlaying of multiple sets of spatial objects on an image/map, can be very expensive to process.

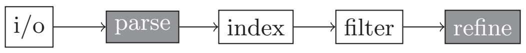

Figure 1: Spatial join query processing pipeline

To illustrate the high computational cost of spatial queries, we run a single threaded spatial join query on a small dataset. We process the join query in five steps, and figure 1 illustrates the query processing pipeline. During the *i/o* step, we read in two sets of polygons from a file that resides on the hard disk (10838 polygon objects); during the *parse* step, we parse polygon objects and transform them from text representation (WKT) into an in-memory data structure; during the *index* step, we construct two Hilbert R-Tree indexes, one index for each set of polygons, using bulk loading methods [18]; during the *filter* step we perform an indexed spatial join operation on the polygon MBRs using the indexes constructed in the previous step; finally during the *refine* step we calculate the Jaccard Similarity [29]. Here, the refinement operator includes both geometry based refinement operation for checking polygon intersection and the measurement of intersecting area.

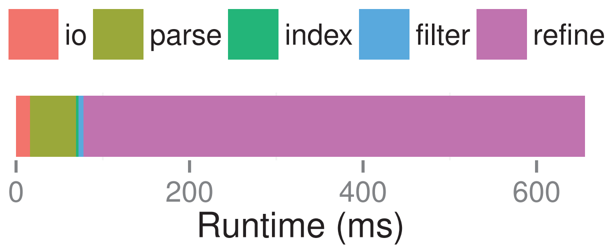

Figure 2: Spatial join query cost breakdown on CPU

Figure 2 shows the cost of each query component. Clearly, the cost of computation component (refine) is much higher than other query components, and becomes the main performance bottleneck.

2.2 Spatial Queries on GPUs

Graphics processing units (GPUs) have been successfully utilized in many applications that require high performance computation. Mainstream GPUs come with hundreds of cores and can execute thousands of threads in parallel. Compared to the multi-core computer systems (dozens of cores), GPUs can scale to a large number of threads in a cost effective manner. Therefore, GPUs have the great potential to improve the performance of spatial queries by eliminating the computation bottleneck.

Most spatial algorithms are designed for executing on CPU, and the branch intensive nature of CPU based algorithms require them to be redesigned for running efficiently on GPUs. PixelBox [29] is an algorithm that specifically designed for accelerating cross-matching queries on the GPU. It first transforms the vector based geometry representation into raster representation using a pixelization method. The pixelization method reduces the geometry calculation problem into simple pixel position checking problem, which is well suited for executing on GPUs. Since testing the position of one pixel is independent of another, the computation can be parallelized by having multiple threads processing the pixels in parallel. Since the positions of different pixels are computed against the same pair of polygons, the operations performed by different threads follow the SIMD fashion. The detailed description of the algorithm and related discussions can be found on the original paper.

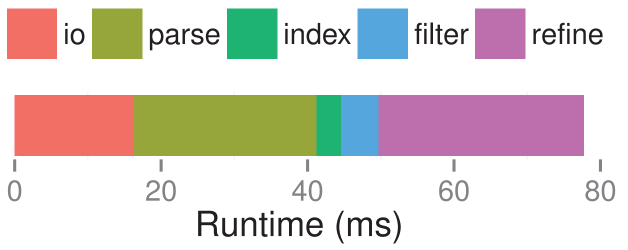

Figure 3: Spatial join query cost breakdown on GPU

Figure 3 shows the query performance breakdown of PixleBox on the same data we used for Figure 2. As the figure shows, PixelBox achieves almost an order of magnitude speedup compared to the CPU implementation, and significantly reduces the cost of computation. Note that here we perform *parse* and *refinement* steps on GPU, and the query cost includes the time to ship the intermediate data back and forth between host memory and device memory.

3. ARCHITECTURE AND IMPLEMENTATION OF HAGGIS

Haggis is an extension of Hadoop-GIS that supports the use of GPUs. Haggis utilizes MapReduce for multi-node query parallelization, and accelerates queries using GPU within a single node.

3.1 Architectural Details

Recently, several MapReduce based spatial query systems [7, 13] have emerged to support scalable spatial query processing on large datasets. While these systems vary in implementation details and at the query language layer, they are conceptually similar. Figure 4 shows the architecture of Haggis which uses Hadoop-GIS [7] under the hood. As the figure shows, data is spatially partitioned and staged to HDFS; spatial queries are expressed as a set of operators that can be translated to MapReduce tasks during the runtime. Tasks run on the partitioned input for parallel query processing. Queries are implicitly parallelized through MapReduce, and a *tile* (aka spatial partition) is the parallelization unit that a Mapper/Reducer can process independently. Within a Mapper/Reducer task, spatial operations such as index building and processing are performed with a high performance spatial query engine – RESQUE. Hadoop-GIS is designed to be extensible. Within a Mapper/Reducer task, Hadoop-GIS relies on RESQUE for spatial processing. Therefore, the MapReduce parallelization is decoupled from the actual spatial query operations, and we can easily extend the query engine to add new features.

In Haggis, the RESQUE engine is extended with GPU based spatial query operators. During the query processing phase, the RESQUE engine can arbitrarily choose CPU or GPU for processing the given partition, and the process is transparent to the user. Query optimizer is responsible for selecting the device(CPU/GPU) to run the query, and such decision depends on a number of factors such as potential speedup gain, and data movement cost. One major advantage of Haggis is flexibility. In a computer system equipped with GPUs, Haggis can use the extra computation power to

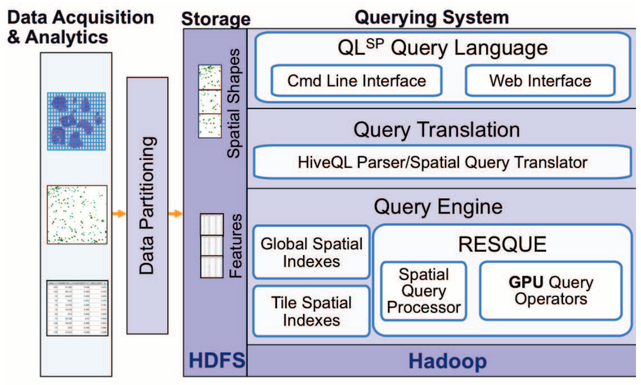

Figure 4: System Architecture

increase the query performance. If such resource is not available, Haggis still functions by relying on the CPU engine for query processing.

3.2 Task Assignment

One critical issue for GPU based parallelization is task assignment – which device to choose processing a given input. As Haggis uses MapReduce based parallelization at the higher level, tasks arrive in the form of data partitions along with spatial query operation to be performed on the data. For each partition, the query optimizer has to choose a device to process this partition, and assign a tag. Most often, the potential speedup gain is used to make such decision, and tasks with larger speedup are scheduled to run on the GPU. If tasks with small speedup are scheduled for running on the GPU, we may achieve little performance benefit and loose the opportunity for using the device for more appropriate tasks.

In Haggis, we use a predictive modeling approach for making such decision. Similar to the speculative execution model in Hadoop, we sample certain amounts of data (10% in our experiment) for performance profiling. In this phase, we process the sampled data on both GPU and CPU, and collect performance statistics and dataset characteristics. We use a polynomial line fitting algorithm to derive the performance model on the sampled dataset using the collected statistics as the input to the model. Then for upcoming tasks, we use this model to estimate the potential speedup factor. If the speedup factor is higher than a threshold, we assign the task to the GPU engine, otherwise to the CPU engine. Haggis uses following features as predictors: number of objects contained in a partition, number of total polygon vertexes. The outcome is the task run-time.

3.3 Effects of Task Granularity

To process a task on the GPU device, data needs to be shipped to the device memory. Such data movement incurs certain I/O cost. Although the memory bandwidth between GPU and CPU is much higher compared to the bandwidth between main memory and the disk, such data movement should be minimized to achieve optimal system performance. Therefore, Haggis needs to adjust the partition granularity to fully utilize system resources. While larger partitioning is ideal for achieving higher speedup on GPU, it may cause data skew. At the same time, a very fine granular partitioning generates large number small tasks that increase the data movement overhead between the CPU and GPU.

4. EXPERIMENTS

4.1 Experimental Setup

Hadoop-GIS: We use a cluster with 7 nodes and 112 cores. Each of these 7 nodes comes with 16 cores (AMD 6172 at 2.1GHz), 2.7TB hard drive at 7200rpm, 128GB memory, and OS is CentOS 5.6 (64 bit). Nodes are connected with a 1Gb network. Our Hadoop installtion is Cloudera Hadoop 2.0.0-cdh4.0.0, and most of the configuration parameters are set to their default values. The system is configured to run a maximum of 16 map or reduce tasks on each node. Datasets are uploaded to the HDFS and the replication factor chosen is 3 on each data node. Each node is equipped with a NVIDIA Tesla M2070 [2].

We use a pathology imaging dataset coming from an image analysis study. In this dataset, spatial objects are derived algorithmically by segmenting boundaries of micro-anatomic objects such as nuclei and tumor regions. Spatial boundaries are validated, normalized, and represented in WKT format. As a result the dataset contains roughly 16.5 million objects. For benchmarking, we use the spatial cross matching query described in [29, 7].

4.2 Effects of CPU for co-processing

In the first set of experiments we study the impact of the number of CPUs on the query performance. Specifically, rather than using all the available CPU cores, the Haggis scheduler only utilizes a given number of CPUs for co-query processing. Ideally, if there are extra computation resources, the system should utilize such resources and try to improve the query performance. However, due to scheduling issues, such objective is hard to achieve.

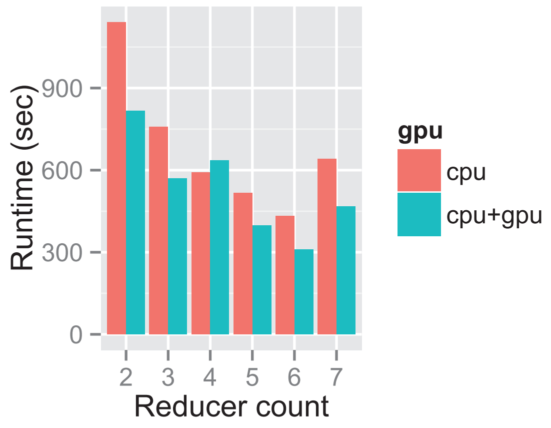

Figure 5: Single CPU for co-processing

Figures 5 and 6 show the performance of spatial join query. The horizontal axis represents MapReduce level parallelization and consequently the number of reducers instantiated for query processing job. The vertical axis represents the query runtime. In Figure 5 only single CPU core is used for query co-processing, whereas in Figure 6, 8 CPU cores are used for such purpose. We can learn three information from the figures. First, as Figure 5 shows, GPU can be helpful for improving query performance, and we can see that the GPU accelerated system outperforms the CPU only system. However, the speedup gain in not very high. We were expecting to see same speedup factor as we show earlier in Section 2. While this can be attributed in part to the data skew, data transfer overhead, and the disk based persistence in MapReduce, the result seems to be less than satisfactory.

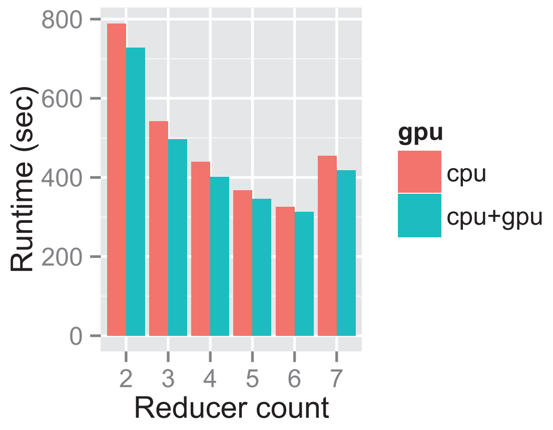

Figure 6: Eight CPUs for co-processing

Second, as Figure 6 shows, with sufficient number of CPU cores for query processing, the performance gain achieved by adding GPU seems to be trivial. We are planning to investigate why this is the case.

4.3 Effects of MR Parallelization

Next, we study how the number of available reducer nodes effects query performance. Specifically, for a fixed cluster capacity, we change the number of CPU cores available for co-processing and measure the query performance. Figures 7 and 8 show the performance of spatial join query on different cluster settings. The horizontal axis represents the number of available cores available for query co-processing, and the vertical axis represents the query runtime.

As the figures show, the query performance can be improved significantly by using a larger number of parallel MapReduce nodes. For example, with a single CPU core for co-processing, the query time is reduced from 1211 to 424 seconds. This also illustrates the advantage of such hybrid system which can benefit from the scalability of MapReduce. With the help of GPUs, this number is further reduced to 309 seconds. However, for a fixed cluster setting, as the number of CPU cores increases, the advantage of GPU start to fade away.

5. RELATED WORK

In [11], an approach is proposed on bulk-construction of R-Trees through MapReduce. In [32], a spatial join method on MapReduce is proposed for skewed spatial data, using an in-memory based strip plane sweeping algorithm. It uses a duplication avoidance technique which could be difficult to generalize for different query types. Hadoop-GIS takes a hybrid approach on combining partitioning with indexes and

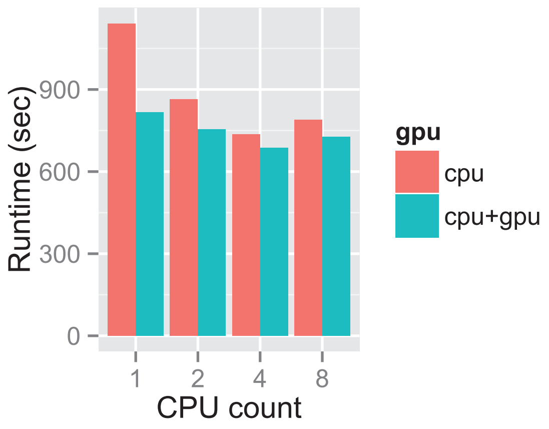

Figure 7: Two node cluster

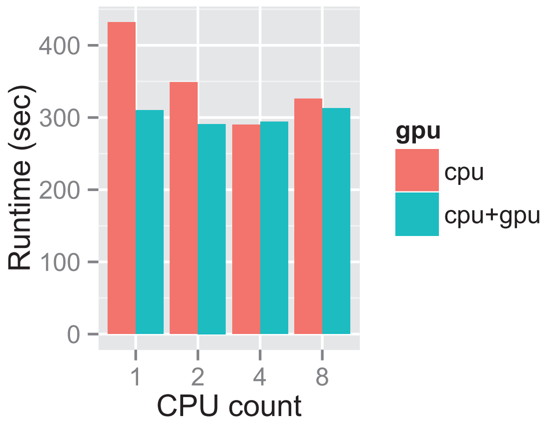

Figure 8: Six node cluster

generalizes the approach to support multiple query types. Besides, our approach is not limited to memory size. VegaGiStore [33] tries to provide a Quadtree based global partitioning and indexing, and a spatial object placement structures through Hibert-ordering with local index header and real data. The global index links to HDFS blocks where the structures are stored. Work in [3] takes a fixed grid partitioning based approach and uses sweep line algorithm for processing distributed joins on MapReduce. The work in [14] presents an approach for multi-way spatial join for rectangle based objects, with a focus on minimizing communication cost. A MapReduce based Voronoi diagram generation algorithm is presented in [8]. In our work [6], we present results on supporting multi-way spatial join queries and nearest neighbor queries for pathology image based applications. Previously, we proposed Pixelbox [29] for accelerating cross-comparison queries for pathology image analysis. In [22, 30, 31], authors discuss a GPU accelerated approach for spatial query processing. However, none of those approaches are concerned with an integrated approach which combines both MapReduce and GPU.

There are several recent efforts on task scheduling for hybrid machines [17, 19, 24, 26, 9, 27, 15, 28]. Most of the previous works deal with task mapping for applications in which operations attain similar speedups when executed on a GPU vs a CPU. On the other hand, we are exploiting performance variability to better use heterogeneous processors.

6. CONCLUSION

Big spatial data from spatial applications shares many similar requirements for high performance and scalability with enterprise data, but has its own unique characteristics – spatial data are multi-dimensional and spatial query processing comes with high computational complexity. In this paper, we present Haggis – a hybrid system that combines the benefit of scalable and cost-effective data processing with MapReduce, and the benefit of efficient spatial query processing with GPU. Our preliminary experimental results demonstrate that Haggis provides a scalable and effective solution for analytical spatial queries over large scale spatial datasets. However, there are many challenges such as effective task scheduling and assignment, and mitigating sub-optimal partition granularity for achieving better speedup. Our ongoing work includes support of 3D spatial data which requires even more intensive computation compared to 2D cases.

7. ACKNOWLEDGEMENTS

This work is supported in part by NSF IIS 1350885, by NSF ACI 1443054, by Grant Number R01LM009239 from the National Library of Medicine, and CNPq. We are also grateful for the support from Amazon AWS in Education Research Grant, and Google Summer of Code.

8. REFERENCES

[1] `http://esri.github.io/gis-tools-for-hadoop`.

[2] `http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_M2050_M2070_Apr10_LowRes.pdf`.

[3] Spatialhadoop. http://spatialhadoop.cs.umn.edu/.

[4] A. Aji. High performance spatial query processing for large scale scientific data. In *Proceedings of the on SIGMOD/PODS 2012 PhD Symposium*, pages 9–14. ACM, 2012.

[5] A. Aji, X. Sun, H. Vo, Q. Liu, R. Lee, X. Zhang, J. Saltz, and F. Wang. Demonstration of hadoop-gis: A spatial data warehousing system over mapreduce. In *SIGSPATIAL/GIS*, pages 518–521. ACM, 2013.

[6] A. Aji, F. Wang, and J. H. Saltz. Towards Building A High Performance Spatial Query System for Large Scale Medical Imaging Data. In *SIGSPATIAL/GIS*, pages 309–318. ACM, 2012.

[7] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *Proc. VLDB Endow.*, 6(11):1009–1020, Aug. 2013.

[8] A. Akdogan, U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. Voronoi-based geospatial query processing with mapreduce. In *CLOUDCOM*, pages 9–16, 2010.

[9] C. Augonnet, O. Aumage, N. Furmento, R. Namyst, and S. Thibault. StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators. In S. B. Jesper Larsson Träff and

J. Dongarra, editors, *The 19th European MPI Users' Group Meeting (EuroMPI 2012)*, volume 7490 of *LNCS*, Vienna, Autriche, 2012. Springer.

[10] S. Borkar and A. A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, 2011.

[11] A. Cary, Z. Sun, V. Hristidis, and N. Rishe. Experiences on processing spatial data with mapreduce. In *SSDBM*, pages 302–319, 2009.

[12] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[13] A. Eldawy and M. F. Mokbel. A demonstration of spatialhadoop: an efficient mapreduce framework for spatial data. *Proceedings of the VLDB Endowment*, 6(12):1230–1233, 2013.

[14] H. Gupta, B. Chawda, S. Negi, T. A. Faruquie, L. V. Subramaniam, and M. Mohania. Processing multi-way spatial joins on map-reduce. In *EDBT*, pages 113–124, 2013.

[15] T. D. R. Hartley, E. Saule, and Ü. V. Çatalyürek. Automatic dataflow application tuning for heterogeneous systems. In *International Conference on High Performance Computing (HiPC)*, pages 1–10, 2010.

[16] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars: a mapreduce framework on graphics processors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 260–269. ACM, 2008.

[17] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars: A MapReduce Framework on Graphics Processors. In *Parallel Architectures and Compilation Techniques*, 2008.

[18] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *VLDB*, pages 500–509, 1994.

[19] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng. Merge: a programming model for heterogeneous multi-core systems. *SIGPLAN Not.*, 43(3):287–296, 2008.

[20] J. Lu and R. H. Guting. Parallel secondo: Practical and efficient mobility data processing in the cloud. In *Big Data*, pages 107–25. IEEE, 2013.

[21] J. Orenstein. A comparison of spatial query processing techniques for native and parameter spaces. In *ACM SIGMOD Record*, volume 19, pages 343–352. ACM, 1990.

[22] S. Puri and S. K. Prasad. Mpi-gis: New parallel overlay algorithm and system prototype. 2014.

[23] S. Ray, B. Simion, A. D. Brown, and R. Johnson. A parallel spatial data analysis infrastructure for the cloud. In *SIGSPATIAL*, pages 274–283. ACM, 2013.

[24] C. J. Rossbach, J. Currey, M. Silberstein, B. Ray, and E. Witchel. PTask: operating system abstractions to manage GPUs as compute devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 233–248, 2011.

[25] B. Simion, S. Ray, and A. D. Brown. Surveying the landscape: an in-depth analysis of spatial database workloads. In *SIGSPATIAL*, pages 376–385. ACM, 2012.

[26] G. Teodoro, T. Hartley, U. Catalyurek, and R. Ferreira. Optimizing dataflow applications on heterogeneous environments. *Cluster Computing*, 15:125–144, 2012.

[27] G. Teodoro, R. Sachetto, O. Sertel, M. Gurcan, W. M. Jr., U. Catalyurek, and R. Ferreira. Coordinating the Use of GPU and CPU for Improving Performance of Compute Intensive Applications. In *IEEE Cluster*, pages 1–10, 2009.

[28] G. Teodoro, E. Valle, N. Mariano, R. Torres, J. Meira, Wagner, and J. Saltz. Approximate similarity search for online multimedia services on distributed CPU-GPU platforms. *The VLDB Journal*, pages 1–22, 2013.

[29] 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. *Proc. VLDB Endow.*, 5(11):1543–1554, 2012.

[30] S. You, J. Zhang, and L. Gruenwald. Parallel spatial query processing on gpus using r-trees. In *ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, BigSpatial '13, pages 23–31, 2013.

[31] 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.

[32] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu. Sjmr: Parallelizing spatial join with mapreduce on clusters. In *CLUSTER*, 2009.

[33] Y. Zhong, J. Han, T. Zhang, Z. Li, J. Fang, and G. Chen. Towards parallel spatial query processing for big spatial data. In *IPDPSW*, pages 2085–2094, 2012.