A Spatial Join Algorithm Based on a Non-uniform Grid Technique over GPGPU

(Extended Abstract)

Danial Aghajarian and Sushil K. Prasad
Computer Science Department
Georgia State University
daghajarian@cs.gsu.edu, sprasad@gsu.edu

ABSTRACT

Grid-based techniques are well-suited for spatial join algorithms over General Purpose Graphic Processing Unit (GPGPU) architectures because of their non-hierarchical structure. However, these techniques are well-established years before the existence of GPU computing. As a result, they do not fully take advantage of many-core architectures. Last year, we had introduced a spatial join GPU system based on discarding even those cross-layer pairs of polygons whose Minimum Bounding Rectangles (MBRs) intersect but their rectangular intersection does not contain edges from both layers. These MBR intersections are called *Common MBRs*. In this extended abstract, we briefly introduce *CMF-Grid*: a non-uniform GPU-based grid technique over such *Common MBRs*, that can be used in polygonal spatial join operations such as overlay, edge-intersection etc. to significantly reduce their computationally-extensive refinement phase workload. Based on our experimental results on real datasets, *CMF-Grid* can cut down the refinement phase workload by more than $30,000$ times that of all-to-all algorithms and it improves upon its predecessor, *CMF* filter, by 700 times. Our upgraded spatial join system with *ST_intersect* predicate is able to process more than $600,000$ polygons with more than 2 billions edges on a single GPU in less than a second end-to-end processing time that is 225% time improvement compared to GCMF, the state of the art GPU-based system. The system also achieves up to 200-fold end-to-end speedup versus the best optimized sequential routines of GEOS C++ library as well as PostgreSQL spatial database with PostGIS.
[*]

1 Introduction

The ever increasing volume of spatial data from various communities representing geographic location of features and boundaries, medical images or traffic on one hand and the crucial need of realtime processing of these datasets in order to extract helpful information on the other hand makes it necessary to exploit High Performance Computing (HPC) in Geographic Information System (GIS) domains [1]. Currently, the primitive-like overlay operation over two layers of spatial object, including less than *700,000* polygons using state of the art ArcGIS software takes more than 13 minutes on a single node [2]. To address these challenges, researchers have designed several distributed architectures to make HPC computing available for geospatial processing including cloud-based systems [3, 4], Message Passing Interface (MPI) systems [5, 6], and map-reduce systems [7]. Most of these systems make use of powerful and expensive computing clusters to break down the computations over several distributed nodes. Some of these applications handle tremendous volume of spatial data which requires using many nodes. Even with such parallelism, employing only CPUs in modern heterogeneous architectures, typically equipped also with GPU, one to two orders of speedup remains unharnessed [8]. One effective way of reducing the number of nodes while keeping up with the required computing power is to accelerate the computations in Graphic Processing Units (GPU). Effective employment of CPU-GPU pair is critical for real-time spatial processing.

Generally, spatial join algorithms over two layers of polygonal (vector) datasets follow a two-phase paradigm [9]:

- Filtering phase: reduces all possible pairs of cross-layer spatial objects to a smaller set of potentially intersecting candidate pairs based on some computationally light algorithms such as Minimum Bounding Rectangle (MBR) overlap test.
- Refinement phase: removes any results produced during the filtering phase that do not satisfy the join predicate, such as ST_intersect, overlap, union, overlay, and etc.

The refinement phase is significantly time consuming as it typically involves $O(n^2)$ algorithms. Our *GCMF* algorithm recently reduced the refinement phase processing time for edge-intersection test down to about 50% of the total time [10]. Older analysis of join operations on CPU show that refinement phase can take up to five times more than the rest of the operations including filtering and parsing

[*]This research is partially supported by NSF grant #1205650.

SIGSPATIAL'17 November 7–10, 2017, Los Angeles Area, CA, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5490-5/17/11.
DOI: https://doi.org/10.1145/3139958.3140056

datasets [11]. In this work, we introduce *CMF-Grid*, a non-uniform grid-based filtering technique that is specifically designed based on GPU architecture to further reduce refinement processing time. It reduces the refinement phase workload more than $30,000$ times compared to the naive all-to-all algorithm. It also improves over its predecessor, *CMF* filter, by reducing workload by 700 times [10]. Furthermore, to demonstrate the efficiency of *CMF-Grid* technique, we refine a GPU-based system for spatial join based on *ST_intersect* operation. The experimental results show that by applying *CMF-Grid*, this system is 225% faster than *GCMF* [10], which is the state of art. The proposed system can process end-to-end spatial join over more than $600,000$ polygons with over 2 billions edges in less than a second on a single *NVIDIA P100* GPU. Finally, we compare the refinement workload of our proposed grid technique with uniform grid to show efficiency of *CMF-Grid*.

The remainder of this extended abstract is organized as follows. In Section 2, we introduce the *CMF-Grid* technique and in Section 3, we briefly discuss a spatial join system with ST_intersect operation based on *CMF-Grid* to study the efficiency of this technique. Experimental results are presented in Section 4. We conclude and point out our future work plan in Section 5.

2 CMF-Grid

In this section, we briefly introduce the idea of proposed grid technique.

2.1 Problem Definition

CMF-Grid can be formulated as an adaptive grid method as follows. For every pair of polygons i, (P_{i_1}, P_{i_2}) in set C, set of MBR overlapping polygon pairs, we want to break down Common MBR of that pair, $CMBR_i$, into N_{g_i} equal-size cells (grid-cell) and identify all the edges of P_{i_1} and P_{i_2} that lie in each cell. It is worth mentioning that each edge can belong to zero to several grid-cells. We want to partition each *Common MBR* in a way that the total workload for in-cell all-to-all refinement phase becomes small enough to minimize end-to-end processing time.

2.2 General Idea

Uniform grid algorithms are not among the best techniques for datasets with unevenly distributed spatial objects to achieve the highest performance. They perform well only if objects are distributed uniformly over the universe where by applying a uniform grid we can have equal size grid-cells with almost balanced workload per cell for further refinement phase in-cell all-to-all processing. Furthermore, classic grid-based techniques are not designed for many-core parallel architectures.

To address these issues, we introduce *CMF-Grid*, a non-uniform grid-based filter applicable to *Common MBR* area of potentially overlapping pairs. Figure 1 illustrates the difference between traditional grid techniques and our adaptive approach. *CMF-Grid* can be distinguished from all the other grid-based techniques in the geospatial processing literature by following two features:

1. In *CMF-Grid*, grid-cells do not necessarily cover the whole universe (Figure 1-b).

2. In *CMF-Grid*, grid-cells may overlap with each other (in cases that *Common MBRs* of overlapping pairs overlap). Later on, we see that such overlaps are infrequent and therefore do not impact performance.

CMF-Grid that is carefully designed to be an embarrassingly parallel algorithm to exploit key aspects of many-core parallel architectures can be formulated as an adaptive grid method as follows. As mentioned earlier, the main goal of partitioning the *Common MBRs* is to optimize the total workload for in-cell all-to-all refinement phase. *CMF-Grid*does that by determining each CMBR cell-sizes independently based on following local information:

- Number of edges partially lying in Common MBR from each polygon.

- Average length of edges in each polygon inside Common MBR.

- Width and height of Common MBR.

It can be proved that for a given pair of MBR-overlapping polygons, the average in-cell all-to-all workload in this grid technique is a quadratic function of their Common MBR cell-size. As a result, by changing cell-size we can change the total workload. Furthermore, It can be shown that finer grid cells do not always lead to smaller processing time. In fact, there is a optimal cell size that minimize processing time and it larger than the cell size that minimize the workload.

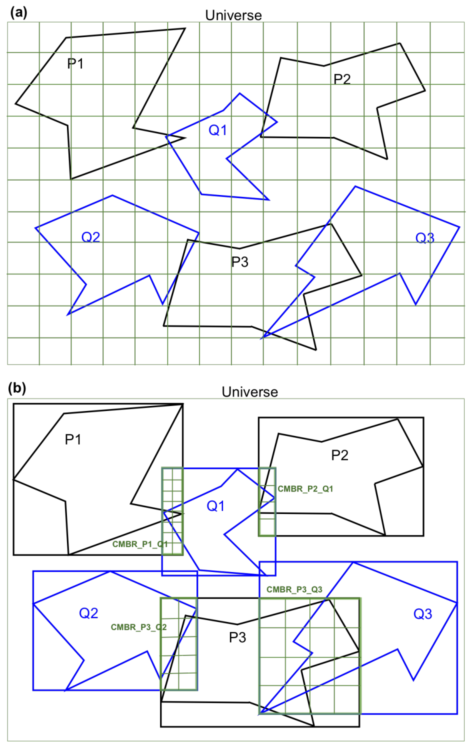

Figure 1: (a) uniform grid technique, and (b) *CMF-Grid*: In *CMF-Grid*, grid-cells are not of the same size and may not cover the whole universe.

3 ST_intersect Spatial Join

In this section, we demonstrate the efficiency of *CMF-Grid* by applying this filter to upgrade our previous GPU-based spatial join system referred to as *GCMF* [10]. To the best of our knowledge, *GCMF* is the state of art for *ST_intersect* join operation over GPU. For any given pair of polygons (P_1, P_2), *ST_intersect* is true if and only if 1) one polygon lies inside another one, or 2) there exists a pair of edges $E_{P1}(i)$ and $E_{P2}(j)$ such that they intersect, or overlap. In other words, *ST_intersect* is true if and only if two polygons share any space.

In the following subsections, we explain the system architecture and point out the changes we needed to apply in *GCMF* to make it functional with *CMF-Grid*.

GCMF introduced *CMF* filter to reduce *edge-intersection* refinement phase workload by filtering out those edges that lie outside of their *Common MBR* for pairs of polygons [10].

Building on our previous work in *GCMF*, the improved system has two upgraded components compared to *GCMF* as follows:

- We replace *CMF* filter with *CMF-Grid* technique with optimized grid cell parameters.

- A new grid-based *edge-intersection* test component substitutes all-to-all *Edge-intersection* test used in *GCMF*.

Similar to its predecessor (*CMF* [10]), *CMF-Grid* classifies MBR-overlapping polygon pairs into three groups as follows:

1. *Within candidate set*: set of all the MBR-overlapping polygon pairs (i_1, i_2) such that $MBR_{i_1 \cap i_2}$ is either equal to MBR_{i_1} or MBR_{i_2}.

2. *Intersecting-edge candidate set*: set of all MBR-overlapping polygon pairs (i_1, i_2) such that (i_1, i_2) is not in (1) and also there exists a non-empty cell in $CMBR_i$.

3. *Disjoint set*: Remaining MBR-overlapping polygon pairs that are neither in (1) nor (2).

Applying this grid technique, grid-based *edge-intersection* algorithm can be developed in a load-balanced way with less workload compared to its predecessor.

4 Results

We use two real datasets (*Urban*, *Water*) each including two sets of polygonal objects from [10] for evaluation of our algorithms. Urban dataset is a small dataset with about $18,000$ polygons. Water dataset, however, is a large dataset with more than $600,000$ polygons and 2 billion edges.

We carried out all the GPU experiments on Bridges cluster located at Pittsburgh Supercomputing Center (PSC), one of the XSEDE (Extreme Science and Engineering Discovery Environment) resources supported by the National Science Foundation (NSF) cyberinfrastructure program. We used a Regular Shared Memory node (RSM-GPU) with 128 GB of RAM memory equipped with two *NVIDIA Tesla P100*, the latest GPUs from *NVIDIA* with 16 *GB* of the main memory. For spatial join use case, we used results of PostgreSQL version 9.4 with PostGIS version 2.2 and *GEOS* library version 3.4.2 presented in [10] conducted on a *2.6 GHz Intel Xeon E5-2660v3* processor, as a sequential baseline. However for GPU baseline, we ran *GCMF* experiments on the *P100* GPU to make a fair comparison.

4.1 CMF vs. CMF-grid

To compare *CMF-Grid* with *CMF* in terms of workload reduction, we define *edge-reduction* factor in *CMF-Grid* for a MBR-overlapping pair similar to the one in [10] for *CMF* as:

$$R_{E_{grid-CMF}} = \frac{\sum_{i:(i_1,i_2)\in \mathbf{C}} |E_{i_1}| + |E_{i_2}|}{\sum_{i:(i_1,i_2)\in \mathbf{C}} \sum_{k=1}^{N_{g_i}} |\hat{E}_{i_1}^k| + |\hat{E}_{i_2}^k|} \qquad (1)$$

where N_{g_i} is the number of grid-cells in $CMBR_i$ and $|\hat{E}_{i_1}^k|$ is number of edges of polygon i_1 partially lying inside cell k. For $N_{g_i} = 1$ (one grid-cell covering the whole $CMBR_i$) *CMF-Grid* and *CMF* are technically equivalent. Table 1 shows timing and workload of *edge-intersection* test for naive all-to-all, *CMF*, and *CMF-Grid* algorithms for Water dataset. Both *CMF* and *CMF-Grid* eliminate almost two-third of pairs. *CMF edge reduction* factor is more than 43 times while *CMF-Grid edge* reduction factor is about 13 (third row in Table 1). Moreover, $\frac{Total\ number\ of\ edges\ in\ cells}{Total\ number\ of\ edges\ in\ CMBR}$ is about 3.17 that implies each edge passes through almost three grid-cells on average (edge duplication). On the other hand, number of active edge-cells in *CMF-Grid* is less than that of edges in *CMF* by a factor of 3. As a result, total workload of *edge-intersection* test after applying *CMF-Grid* is more than $30,000$ times smaller than all-to-all and also more than 730 times smaller than *CMF* filter. This tremendous decrease in workload leads to more than 10-fold speedup of *grid-based edge-intersection* test versus the one used in *GCMF* that brings down $\frac{Edge-intersection\ time}{Total\ time}$ ratio from 58% in *GCMF* to 12% in new system. It also achieves more than 800-fold speedup versus sequential *all-to-all* test.

Finally, Table 2 shows running time for two sequential PostGIS and GEOS optimized library systems reported in [10]. We also provide running time for our new system based on *CMF-Grid* and the current state of the art, GCMF. New end-to-end running time of GCMF is more than 4 times faster than its original results reported in [10] that is because of *P100 GPU* used in this experiment. The end-to-end running time for our new system with *CMF-Grid* shows a significantly impressive improvement of 225% compared to GCMF. It also shows up to 200-fold speedup gain compared to the best sequential system.

Table 1: Refinement workload reduction of Water dataset for all-to-all, *CMF* and *CMF-Grid*.

	All-to-all	*CMF* [10]	***CMF-grid***
Time (ms)	120,751	981	**89**
Pairs $\in I$	566,656	198,142	198,125
Edges(all)	2,002,910,863	46,421,188	**147,002,513**
Edges(active)	2,002,910,863	46,421,188	**15,043,183**
Workload	$535,108,085,968$	12,794,606,592	**17,370,352**

4.2 CMF-grid vs. uniform grid

Finally, figure 2 shows the actual total refinement workloads of *CMF-Grid* and uniform grid for a wide range of grid-cell sizes (coarse to fine) for Urban dataset. The refinement workload in the case of *CMF-Grid* seems to be converging to the minimum value much faster than uniform grid. In other word, workload of uniform grid for the coarser grid-cells (smaller number of grid-cells) is much larger than *CMF-Grid*. However, as the number of grid-cells increases (finer grid) it converges toward *CMF-Grid*. Given that for a larger

Table 2: End-to-end running time of spatial join with ST_intersect operation for four different systems.

Dataset	Running Time (ms)				CMF-grid Speedup	
	Sequential		Parallel		Best Sequential	Parallel ($\frac{GCMF}{CMF-grid}$)
	PostGIS	GEOS	GCMF [10]	**CMF-grid**		
Urban	3,120	5,770	52	31	101 (PostGIS)	1.68
Water	232,122	148,040	1,663	739	200 (GEOS)	2.25

number of grid-cells more time is needed to process the whole grid and more memory is needed to keep its data structure, the efficiency of our method is significantly better than that of uniform grid. In fact, if we want to lower the workload of uniform grid down to optimal (converged point) we need to have almost three times more grid-cells (finer cells) than *CMF-Grid*.

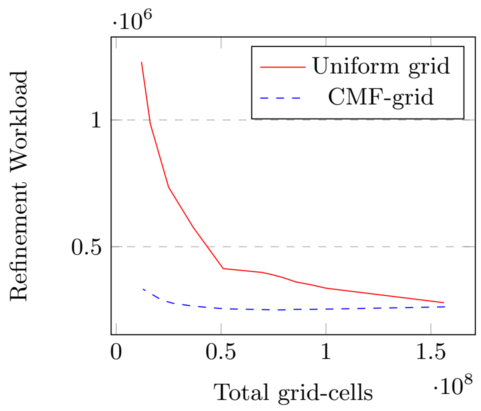

Figure 2: Workload of CMF-Grid versus uniform grid for a wide range of grid-cells in Urban dataset.

5 Conclusion

In this extended abstract, we have introduced *CMF-Grid*, a new non-uniform grid technique that is designed with consideration of many-core GPU architectures. We also used this filter to design a system for spatial join with ST_intersect operation as a case study that improved the end-to-end running time of the current state of the art GPU-based system (GCMF) by 225%. The system was upgraded in two ways, *CMF-Grid* component was able to reduce the edges for refinement phase by a factor of 800 compared to GCMF and a new grid-based edge-intersection algorithm was able to achieve more than 10-fold speedup compared to the one in GCMF.

Our future work plan is to scale the system up by exploiting MPI to distribute the workload across nodes making it possible to process much larger datasets among the distributed GPU nodes.

6 References

[1] S. K. Prasad, D. Aghajarian, M. McDermott, D. Shah, M. Mokbel, S. Puri, S. J. Rey, S. Shekhar, Y. Xe, R. R. Vatsavai, *et al.*, "Parallel processing over spatial-temporal datasets from geo, bio, climate and social science communities: A research roadmap," in *Big Data (BigData Congress), 2017 IEEE International Congress on*, pp. 232–250, IEEE, 2017.

[2] S. Puri and S. K. Prasad, "A parallel algorithm for clipping polygons with improved bounds and a distributed overlay processing system using mpi," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pp. 576–585, IEEE, 2015.

[3] S. You, J. Zhang, and L. Gruenwald, "Large-scale spatial join query processing in cloud," in *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*, pp. 34–41, IEEE, 2015.

[4] 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*, pp. 284–293, ACM, 2013.

[5] S. Puri, D. Aghajarian, and S. Prasad, "MPI-GIS : High Performance Computing and IO for Spatial Overlay and Join," *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC-16)*, 2016 (Research Poster).

[6] S. Puri and S. K. Prasad, "MPI-GIS: New parallel overlay algorithm and system prototype," 2014.

[7] A. Eldawy and M. F. Mokbel, "A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data," *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1230–1233, 2013.

[8] S. K. Prasad, M. McDermott, X. He, and S. Puri, "Gpu-based parallel r-tree construction and querying," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pp. 618–627, IEEE, 2015.

[9] E. H. Jacox and H. Samet, "Spatial join techniques," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 1, p. 7, 2007.

[10] D. Aghajarian, S. Puri, and S. K. Prasad, "GCMF: An efficient end-to-end spatial join system over large polygonal datasets on gpgpu platform," *SIGSPATIAL*, 2016.

[11] A. Aji, G. Teodoro, and F. Wang, "Haggis: Turbocharge a mapreduce based spatial data warehousing system with gpu engine," in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, pp. 15–20, ACM, 2014.