Incremental Refining Grid-File (IRGF) based Spatial Filtering in CudaGIS

(Initial Design & Implementation)

Jianting Zhang
Department of Computer Science, the City College of New York

While quadtree and R-Tree indexing can be used to speed up spatial joins, we are more interested in non-indexed spatial joins on GPUs. This is primarily because pre-built quadtrees may use different grid cells (which requires implicit and explicit conversions) and synchronizing traversals on pre-built R-Trees can be inefficient even on CPUs. Previously we have developed a spatial join technique based on a simple Single-Level Grid-File (SLGF) structure for spatial filtering and applied to a few spatial join applications (Fig. 1). The approach is closely related to the classic Partition Based Spatial-Merge-Join (PBSM) technique [1] in the sense that both input datasets are decomposed into regular tiles (or grid cells) for filtering. PBSM relies on a tile-to-partition mapping scheme to group tiles into partitions to reduce skewness and increase filtering power, which can be used for load balancing in a parallel setting on CPUs. Our approach essentially implemented the tile-to-partition mapping scheme on GPUs (using 1D parallel primitives) which is much easier than implementing plane-sweeping based filtering that have been extensively used for parallel spatial joins on CPUs [2] which is hard to parallelize. However, our previous results on using single-level grid-file for spatial joins have experienced difficulties in choosing grid cell sizes in reducing memory footprint in the filtering phase which has limited its applicability to a certain extent for large-scale data on GPUs with limited memory.

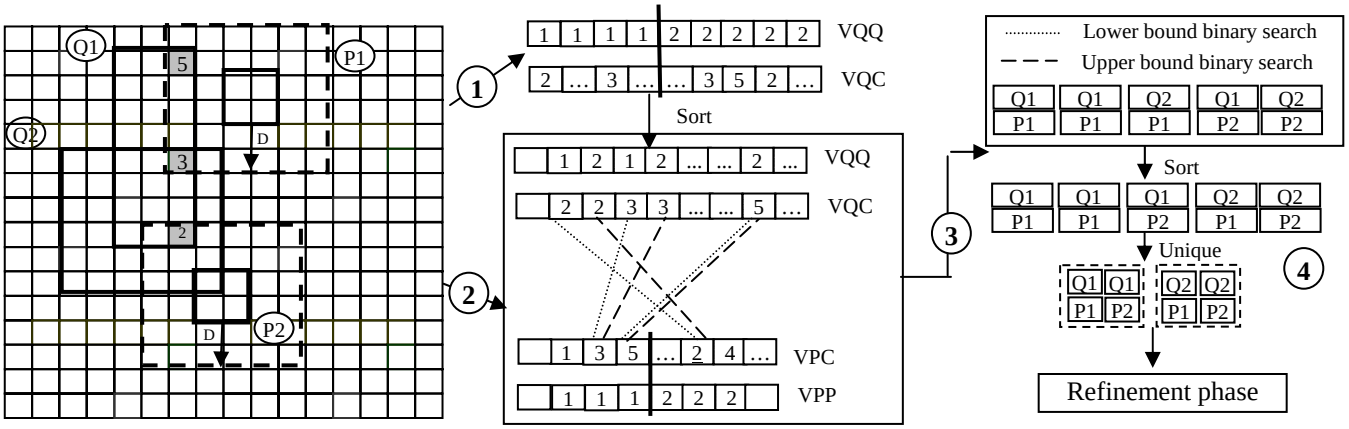
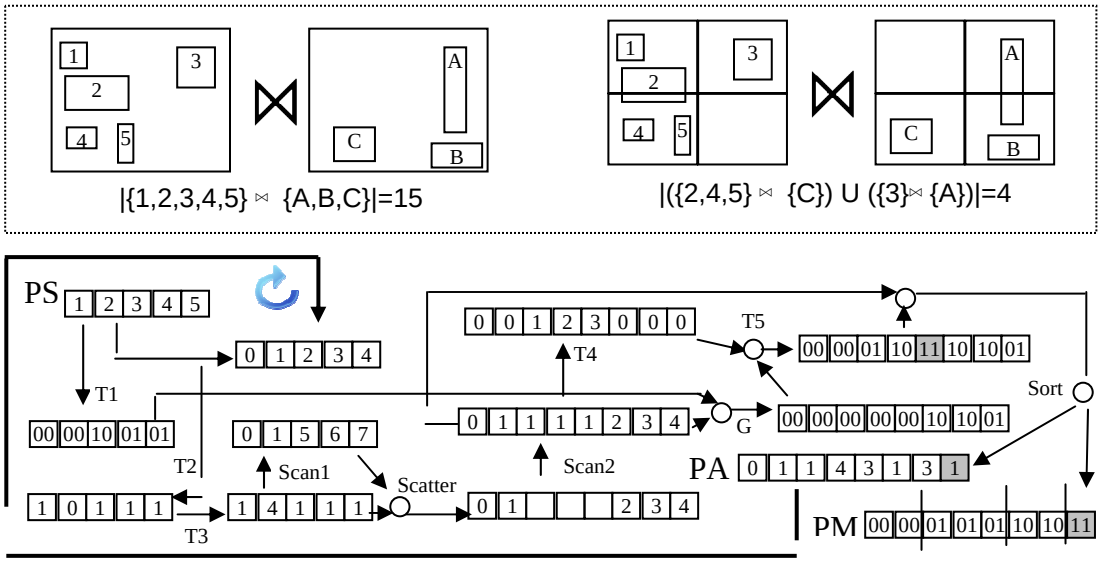


Fig. 1 Illustration of Single-Level Grid-File (SLGF) for Spatial Filtering

The issue can be explained as following. If a large cell size is chosen, then more MBRs from both input datasets will be associated with non-empty grid cell. As the design requires pairing all the MBRs from both datasets, very often a large number of

pairs, i.e., $\sum_{i=1}^{N} |Ai| * |Bi|$ (where N is the number of non-empty cell and $|A_i|$ and $|B_i|$ are the numbers of MBRs associated non-empty cell i), need to be output before unique pairs can be computed and used in the refinement phase (Fig. 1). Although the number of unique pairs might be small, the number of intermediate pairs can be too large to be fit in the GPU memory. On the other hand, if a small cell size is chosen, while $|A_i|$ and $|B_i|$ are likely to be smaller, N usually grow quadratically which may also incur large numbers of intermediate pairs. Intuitively, for small cell sizes, large MBRs will be associated with multiple grid cells (proportional to their area sizes) and the resulting pairs will grow quadratically. As using smaller grid cells will improve filtering power, there is a tendency to use small grid cells in practices, however, large member footprint remains to be an issue.

While the large memory footprint issue has not been yet addressed extensively in CPU based serial and parallel spatial joins (possibly due to the reason that the majority of spatial join algorithms are designed for external storage), as our goal is to develop high-performance spatial join techniques on GPUs which are essentially memory resident, it is imperative to reduce memory footprint in spatial joins to allow processing larger datasets without resorting to external storage techniques at the cost of signficant performance degradation. The design of Incremental Refining Grid-File (IRGF) structure for spatial filtering is illustrated in Fig. 2. Our idea is to build multiple grid files using different cell sizes, from coarse resolution at the top (level 1) to fine resolution at the bottom (level K). As the example shown in Fig. 2, the number of the candidate pairs to be sent for the next step (Step 4 in Fig.1) for duplication removal (the last step of the filtering phase) is reduced from 15 to 4. Note that MBR 2 has been replicated in the both quadrants that it intersects at the finer level grid.



T1: transformation-generate MBR Morton; T2-transformation- generate MBR level; T3-transformation – assign level
T4: transformation – generate intra-node sequence (using thread identifier); T5-transformaiton – addition
Scan1:exclusive prefix sum to compute starting position; Scan2: max scan to propagate values to the right; G: Gather to copy Morton codes based on the position array; Sort: stable sort to re-arrange the position array and Morton codes (also can be regenerated from the position array and original MBR)

Fig. 2 Illustration of Incremental-Refining Grid-File (IRGF) for Spatial Filtering and its Implementation Using Parallel-Primitives

While the idea is quite intuitive and its serial implementation on CPUs is also straightforward, the design and implementation on GPUs is not as simple as it appears due to the complexities in fine-grained thread coordination. We further note that while pairing cells that have same spatial extent takes just two lookups using sparse matrix representation (2D) of a grid file, it is much more complex if a compact vector representation (1D) is needed, which is necessary to reduce memory footprint. We first compute the Morton codes of all the coordinates of MBRs and the grid levels that they should be associated with, in a way similar to what we have proposed for quadtree indexing in Section 5.1, for both input MBR sets (assuming PS and QS, respectively). To subdivide level k cells into level k+1 cells in parallel for PS, a sequence of (sum-)scan-reduce-scatter-(max-)scan-gather-recue(by key) primitives can be applied as shown in the lower part of Fig 2. Similarly the subdivision can be applied to QS. Subsequently the MBRs associated with the cells of the two input datasets can be paired with the same Morton codes. The resulting PA and PM vectors shown in the lower part of Fig. 2 are essentially the same as VPP and VPC in Fig. However, PA and PM can be dynamically expanded level by level as necessary rather than being fixed based on a predefined grid cell resolution.

The design also allow setting flexible policies in deciding whether to user finer resolutions, such as available memory, numbers of MBRs associated with grid cells or their combinations. Our approaches share quite some similarities with the design of splitting of a PMR-quadtree node using a different set of parallel primitives [3]. The clone primitive used in [3] essentially is the combinations of T3, Scan1, Scatter and Scan2. The unshuffle primitive in [3] is also functionally equivalent to the combination of T4, Gather, T5 and Sort. Despite the similarities, our approach originates from a single-level grid-file structure and the multi-level grid-file is intended to be used as a light-weighted structure for spatial joins on non-index datasets rather than building a full-fledged indexing tree which usually requires much more sophisticated design and parameter tuning. We consider our approach a hybrid of grid-files and quadtrees: it is similar to quadtrees as space is dynamically partitioned into regular quadrants but the hierarchy is not maintained; it is similar to grid-files as only a flat structure is used but the cell resolutions can be incrementally refined.

We note that the proposed approach may introduce false positives during the cell subdivisions as illustrated in the right part of Fig. 2 where vector elements correspond to the false positives are highlight in gray. We could have eliminated the false positive by subdividing a MBR along the two dimensions separately. We consider it a design option and plan to model the behavior in the project. Another design option is to dynamically expand 1D vectors to 2D matrixes for simpler cell pairing and compress the results back to 1D vectors for storage efficiency. This is possible due to the data parallelism in the design where the grid cells can be batch processed without incurring signficant overheads. The option is especially beneficial for dense cells at the finer grid levels where memory requirement for processing a few grid cells is small. Similar to spatial indexing, we plan to build cost models for the basic design and its options to quantify the performance and evaluate them empirically.

Reference:

[1] J. M. Patel and D. J. DeWitt (1996). Partition based spatial-merge join. Proceedings of the International Conference on Management of Data, SIGMOD, 259-270.

[2] L. Arge, O. Procopiuc, S. Ramaswamy, et al. (1998). Scalable Sweeping-Based Spatial Join. Proceedings of the 24rd International Conference on Very Large Data Bases, 570 - 581

[3] E. G. Hoel and H. Samet (1995). Data-parallel primitives for spatial operations using PM quadtrees. Proceedings of Computer Architectures for Machine Perception, 266-273.