Heterogeneous Parallel Programming COMP4901D

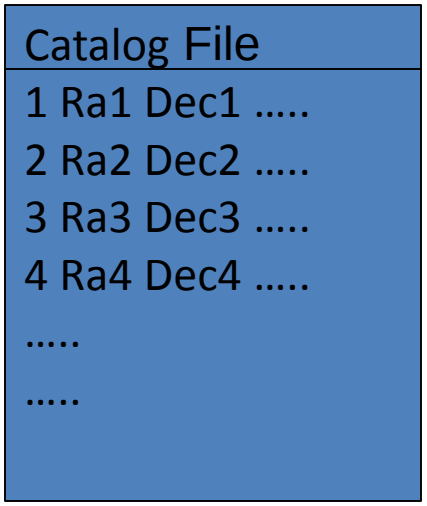
Cross Match of Astronomical Catalogs on the GPU

Overview

- Cross match of astronomical catalogs
- Existing CPU-based methods
- Our GPU-based methods

Astronomical Catalogs

- Each record in a catalog contains information of an astronomical object, e.g., position, brightness, and so on.

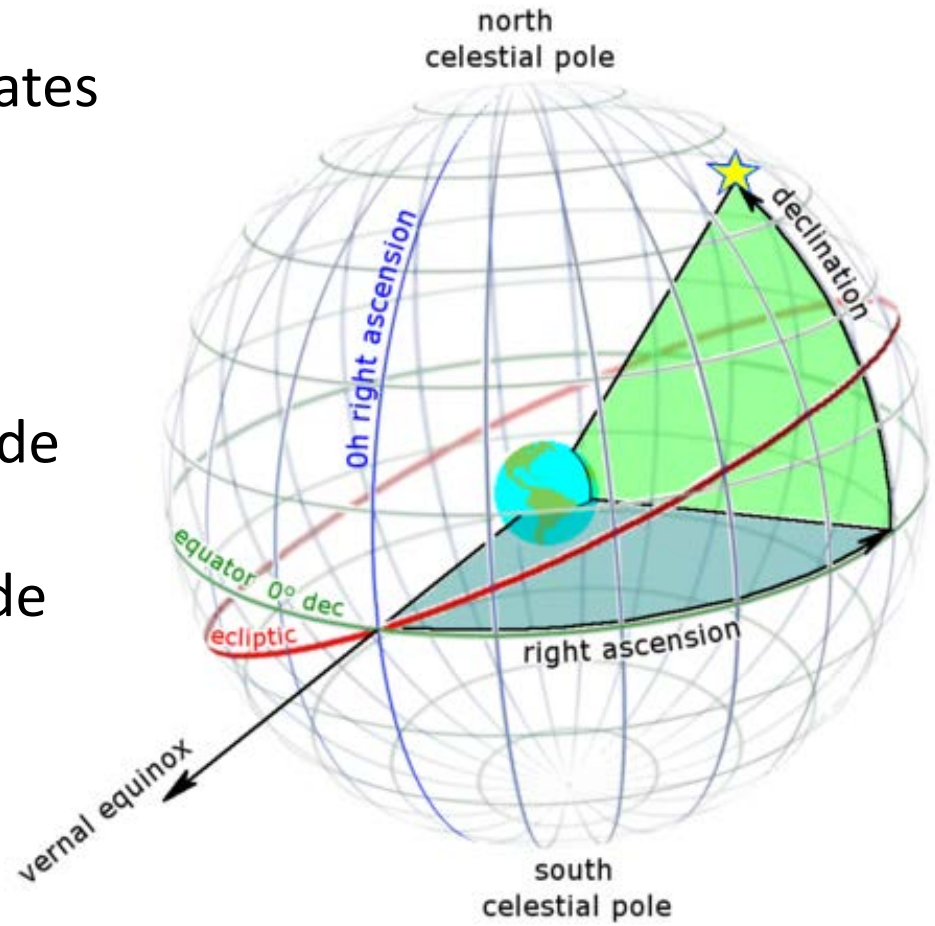


Celestial Coordinate System

Right ascension (Ra) and declination(Dec) are coordinates on the celestial sphere.

The position of a star is represented as (Ra, Dec):

- Ra is similar to longitude (0 ~360 degree).
- Dec is similar to latitude (-90~+90 degree).



Distance in the Spherical Space

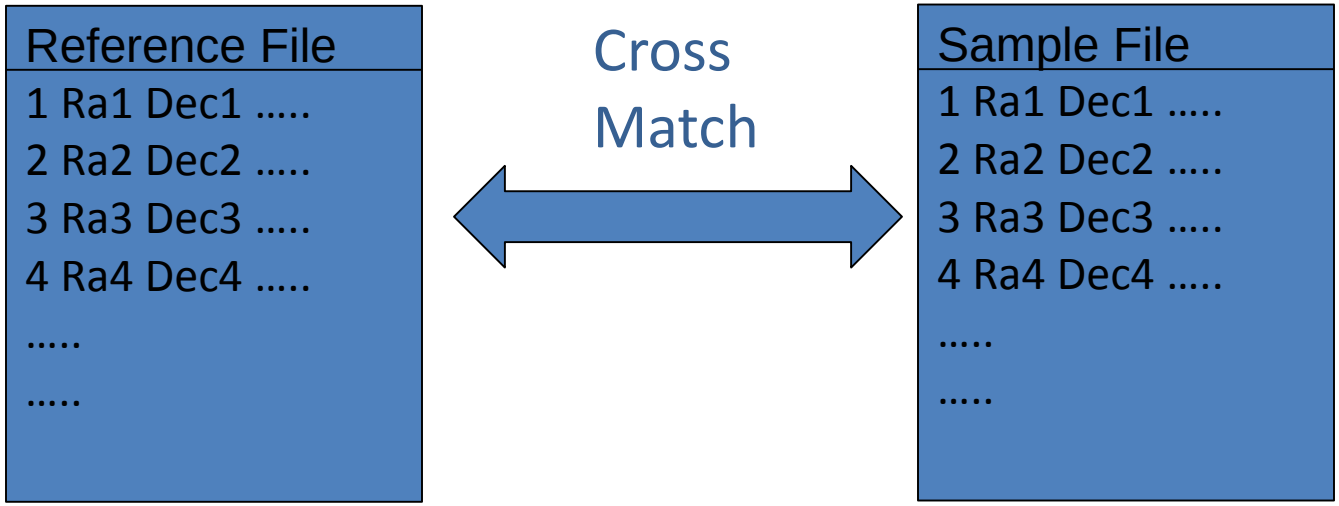
The *central angle* is used to represent the distance:

$Distance(O_1,O_2) = arccos(sin(dec1)\ sin(dec2) + cos(dec1)\ cos(dec2) cos\ (|ra1-ra2|)\ )$

If the distance between two points is less than a distance threshold *r*, they are considered the same point (object).

Cross Match between Catalogs

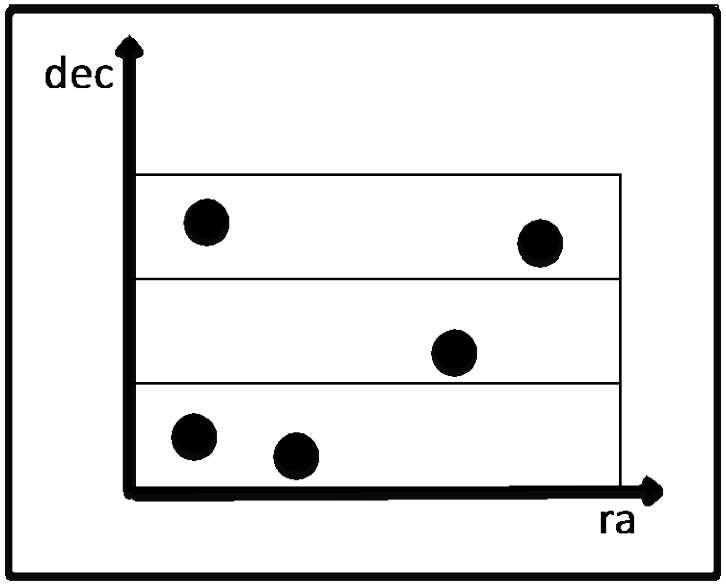
- For each point in the sample file, find its matching points in the reference file.
- Find the abnormal points in the sample file that have no matching points in reference file.



Challenges

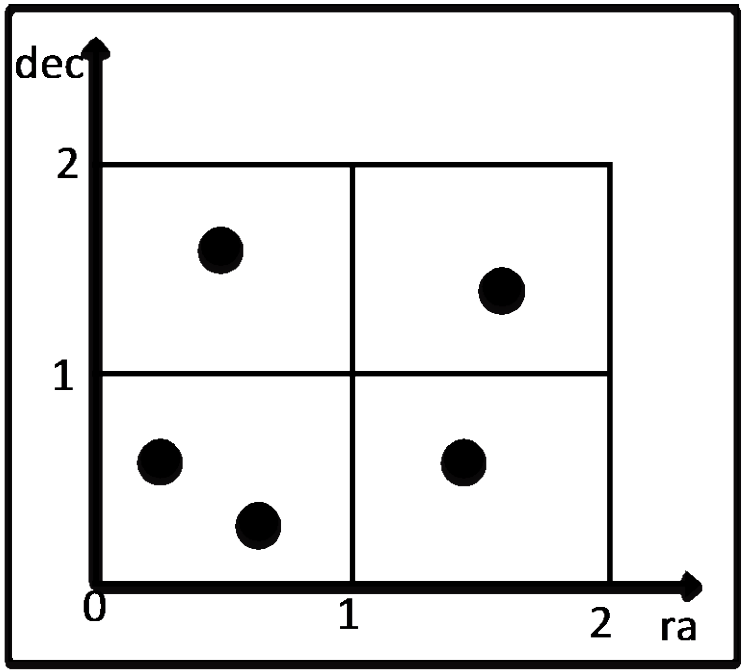
- There are a great number of objects in each catalog, from millions to billions.
- Fast response time is desired in interactive environments and is mandatory in online sky surveys for transient detection and other purposes.

The Zones Algorithm



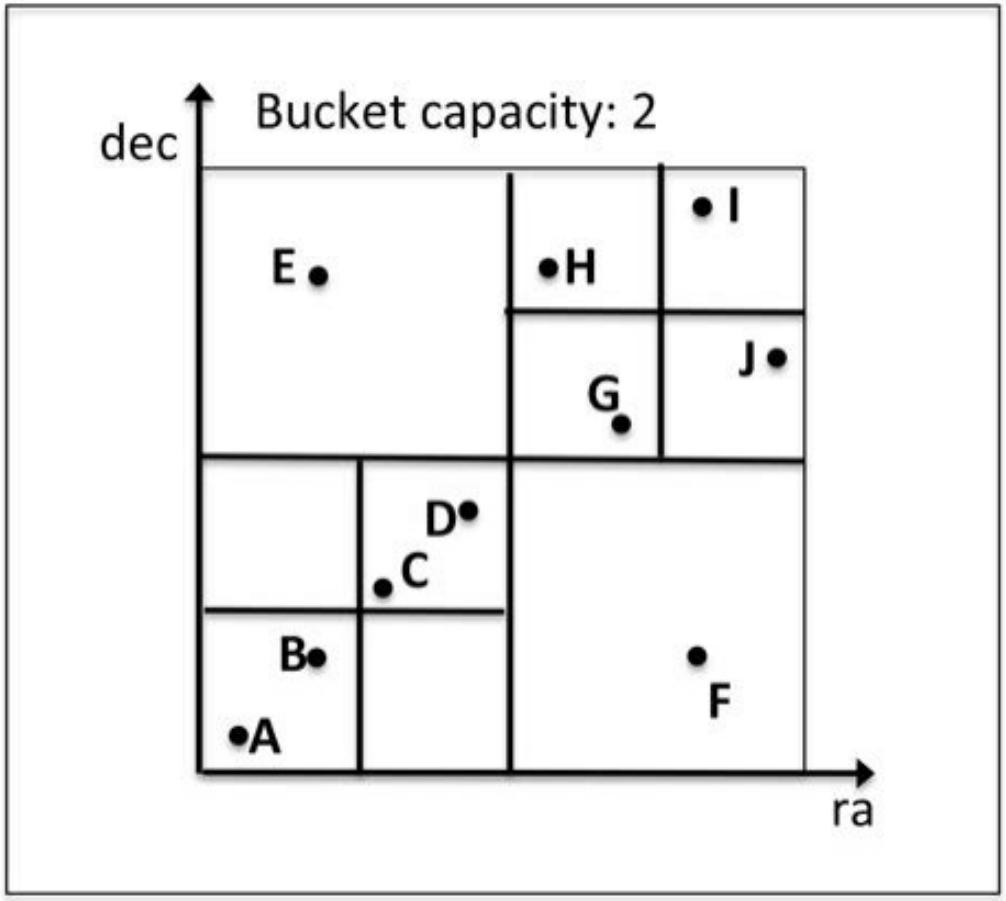
Jim Gray. The zones algorithm for finding points-near-a-point or cross-matching spatial datasets. CoRR,abs/cs/0701171, 2007.

The Grid File



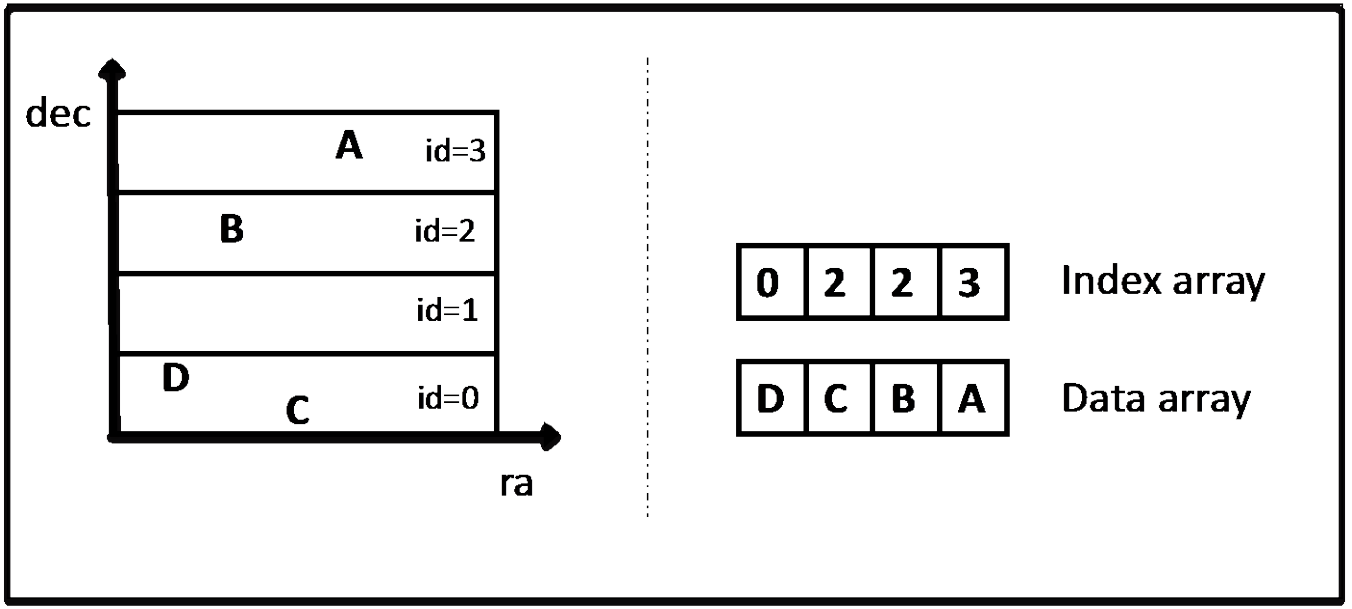
J. Nievergelt and H. Hinterberger. The grid file: An adaptable, symmetric multikey file structure. ACM Trans. Database Syst.,9(1):38–71, 1984.

The Quad Tree



Hanan Samet. The Quadtree and Related Hierarchical Data Structures. ACM Computing Surveys. Volume 16 Issue 2, June 1984

GPU-Based Zone Index Construction



Step 1: each thread takes one point and computes its zone
Step 2: sort points based on zone id and *ra*
Step 3: compute the starting offset of each zone

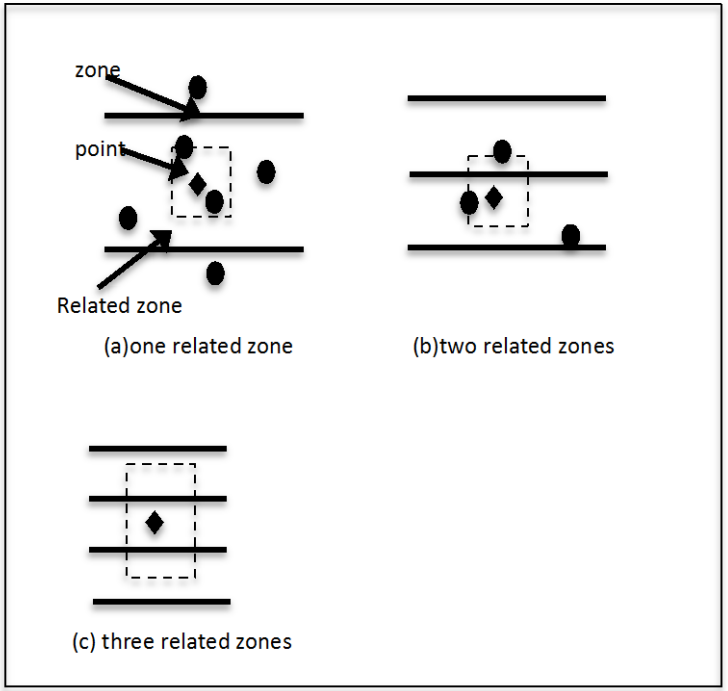
GPU-Based zoneMatch

Step 1: for $p_s(ra,dec)$, compute the query box and the related zones

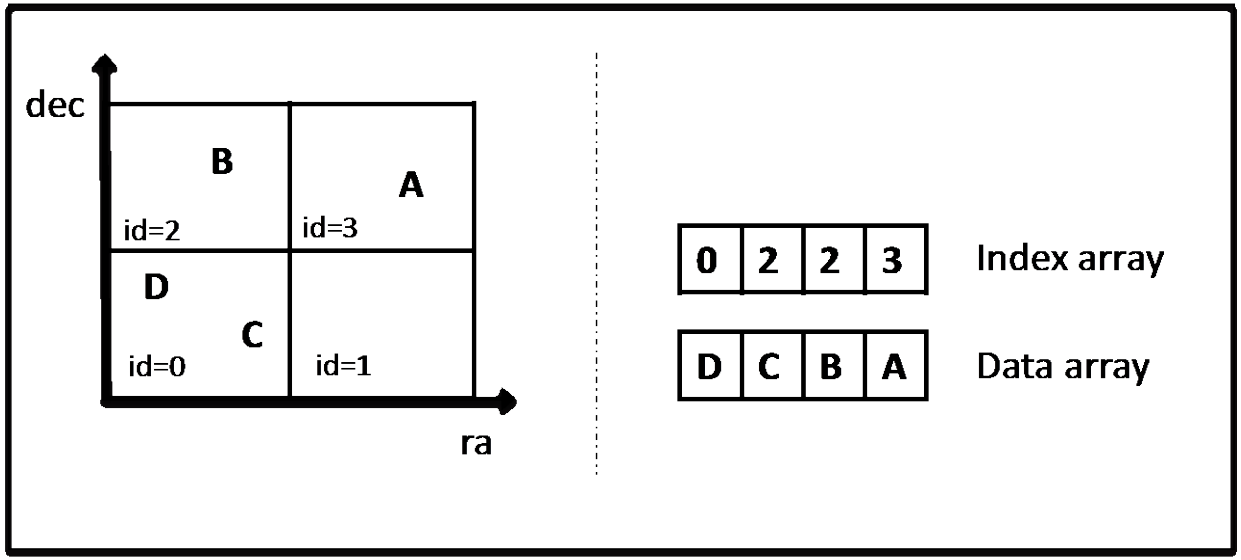
In each related zone

Step 2: perform a binary search to find the starting and ending offsets of candidate points

Step 3: if the *dec* falls within the query box, compute the precise distance for candidate points and find out the matching points



GPU-Based Grid Index Construction

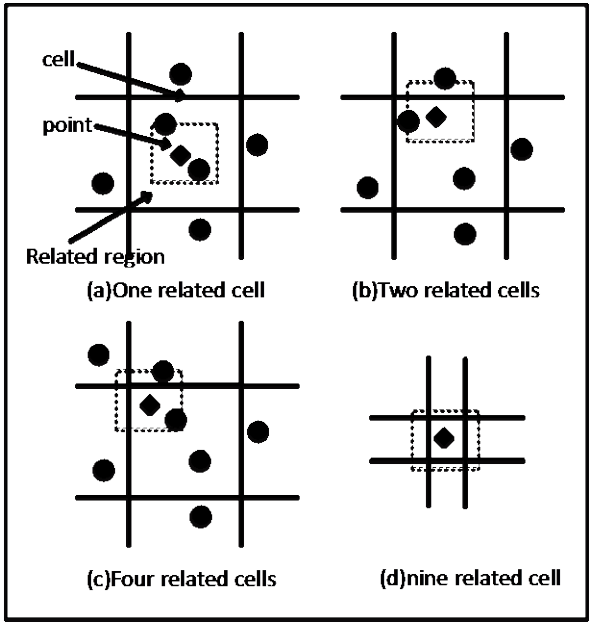


Step 1: each thread takes one point and computed the cell it belongs to
Step 2: sort points based on cell *id*
Step 3: compute the starting offset of each cell

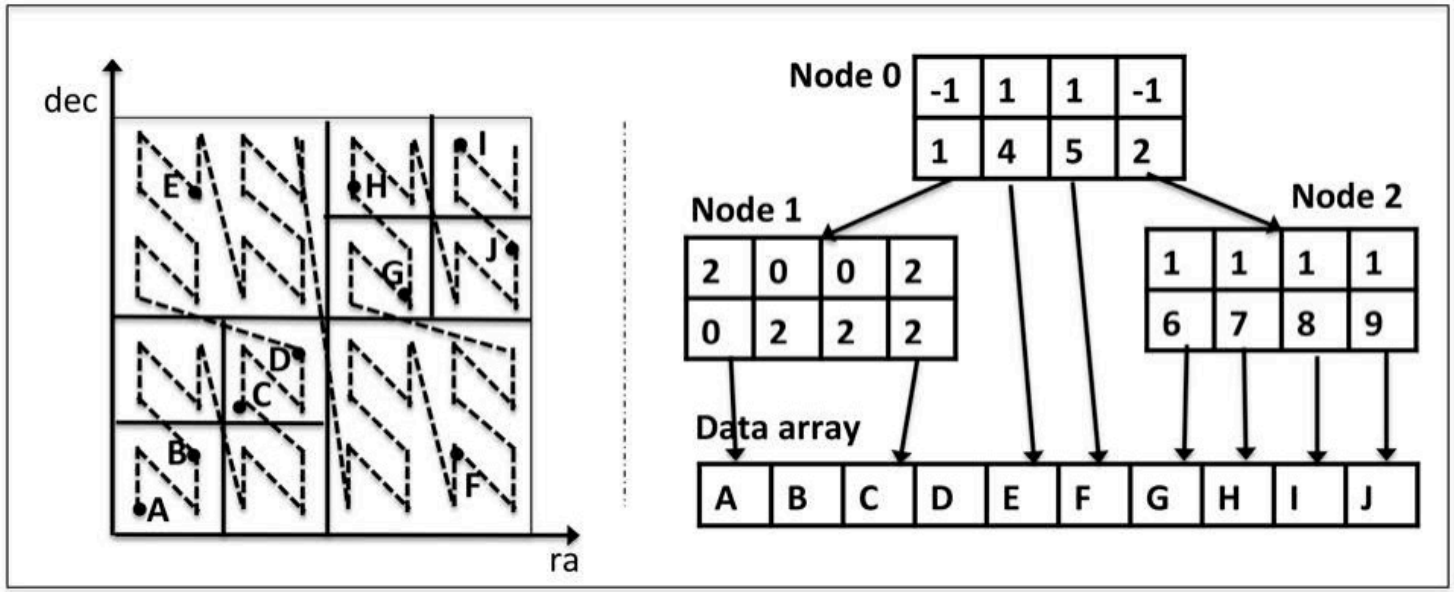
GPU-Based gridMatch

Step 1: for $p_s(ra,dec)$, the thread computes the query box around p_s and the boundary of the related cells on the reference grid

Step 2: check the reference points that fall in the query box and compute the exact distance



GPU-Based Quad Tree Construction



Step 1: compute the Z-value for all reference points on the GPU
Step 2: sort the reference points by Z-value on the GPU
Step 3: pre-compute the size of quadtree on the CPU
Step 4: compute the first *N* nodes on the CPU
Step 5: each thread takes one node and continue building quad-tree on the GPU

GPU-Based quadtreeMatch

Step 1: each thread computes the query box for each sample point p_s

Step 2: Use depth-first search to find the leaf quadrants that intersect the query box. Then compute the distance between the points in quadrants and p_s to find the matching points

Experimental Setup

- Two Intel Xeon E5520 2.27GHz Quad-Core CPUs, 32 GB main memory
- NVIDIA M2090 GPU with 6 GB global memory
- The maximum number of threads per block: 1024
- The size of shared memory per block: 48KB
- Fedora release 14, OpenMP 3.0 and CUDA 5

Data Sets

- Synthetic datasets: randomly generated 12,759,064 objects, with dec and ra ranging between 0 and 10 degrees
- Real-world datasets: SDSS* catalog, where dec and ra range from 0 to 10 degrees and there are 2,448,790 objects
- Distance threshold r is set to 0.0056 degrees as suggested by astronomers

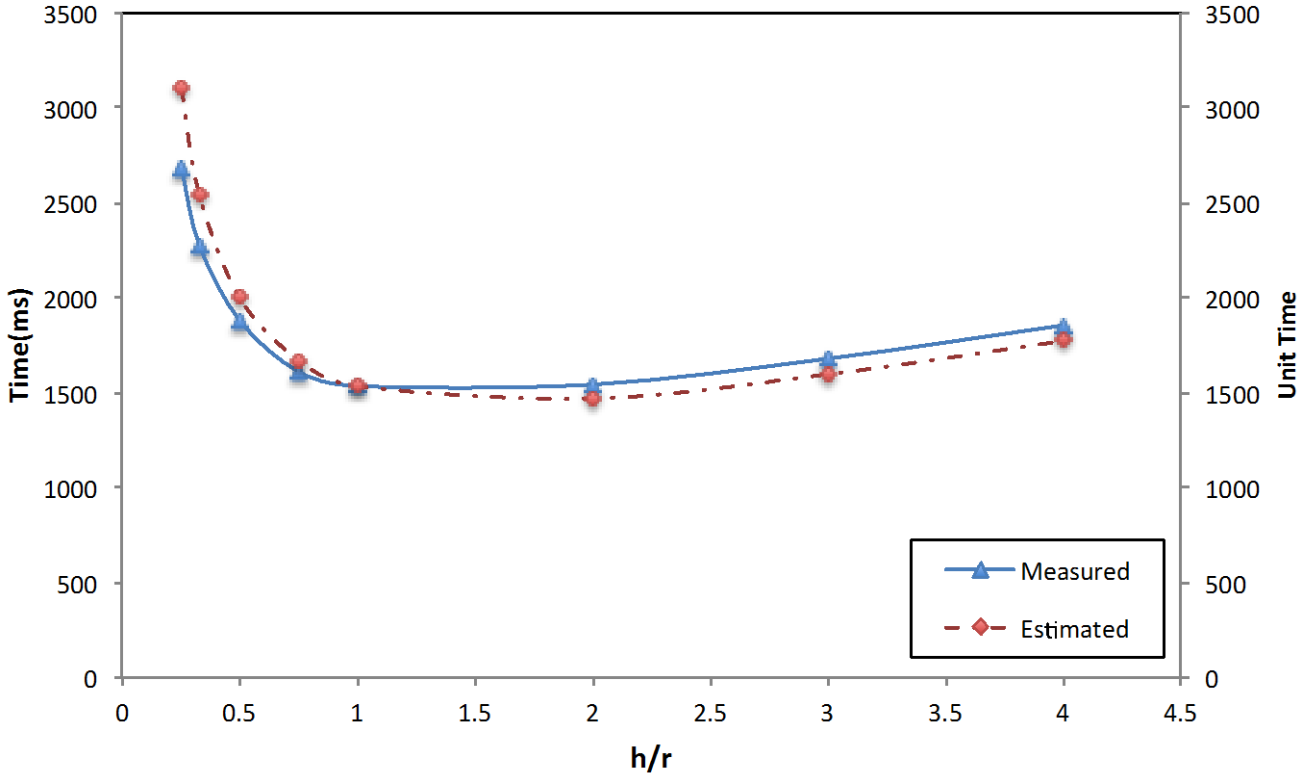
*SDSS: Sloan Digital Sky Survey

CPU-Based Implementations

- Index Construction on the CPU
  - In the sequential version, the reference points are processed one by one. Sequential radix sort, prefix sum and Z-value computation are used correspondingly.
  - In the parallel implementation, openMP is used to parallelize the radix sort, prefix sum and Z-value computation.
- Cross Match on the CPU
  - In the sequential version, sample points are processed sequentially.
  - In the parallel version, openMP is used to launch multiple threads to calculate the matching points with one thread in charge of one sample point

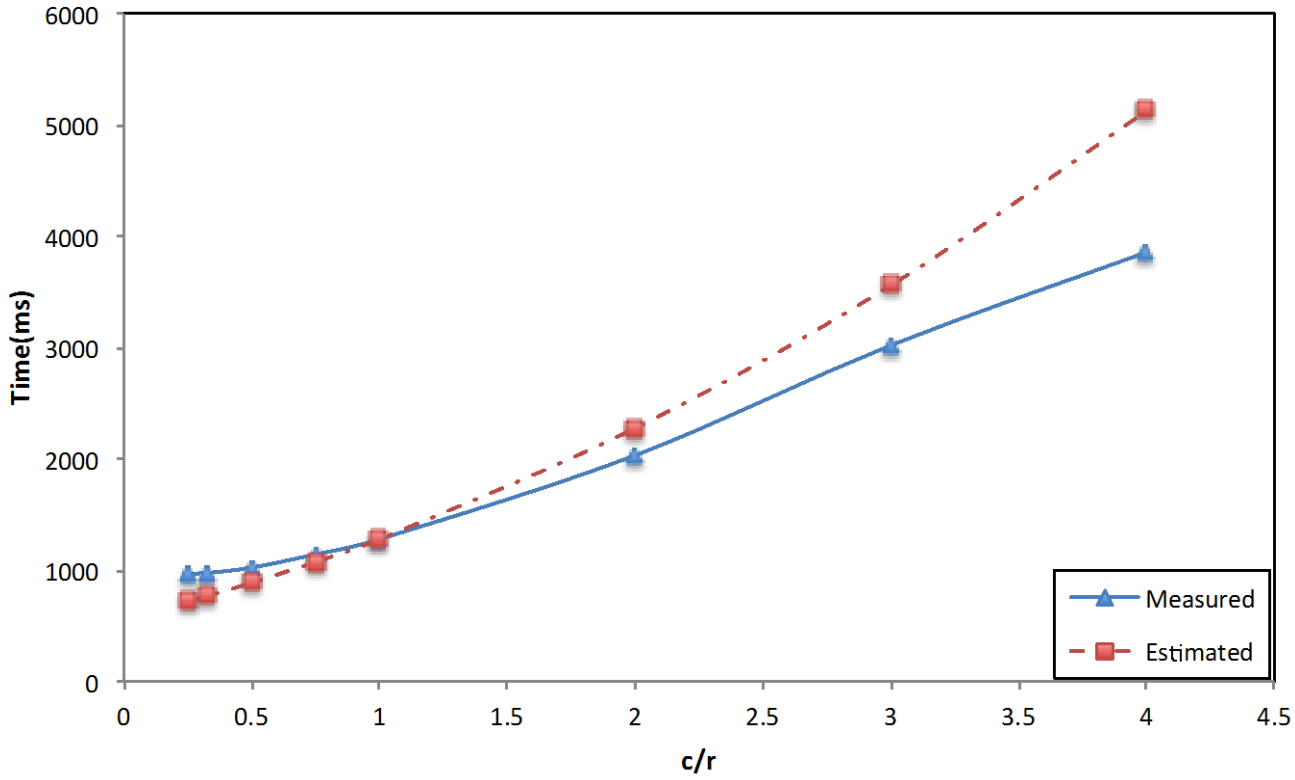
zoneMatch on Synthetic Data

- Best zone height equals distance threshold.



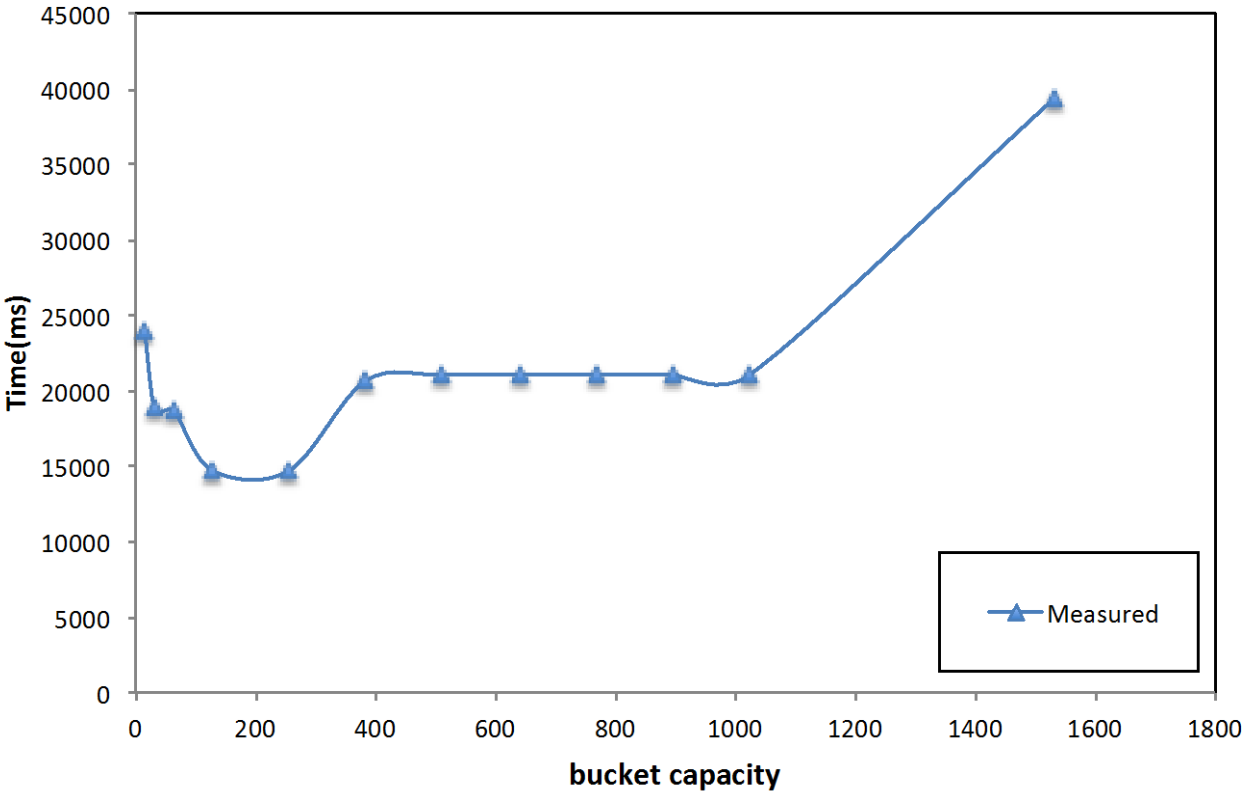
gridMatch on Synthetic Data

- Best cell size is around a quarter of the distance threshold.



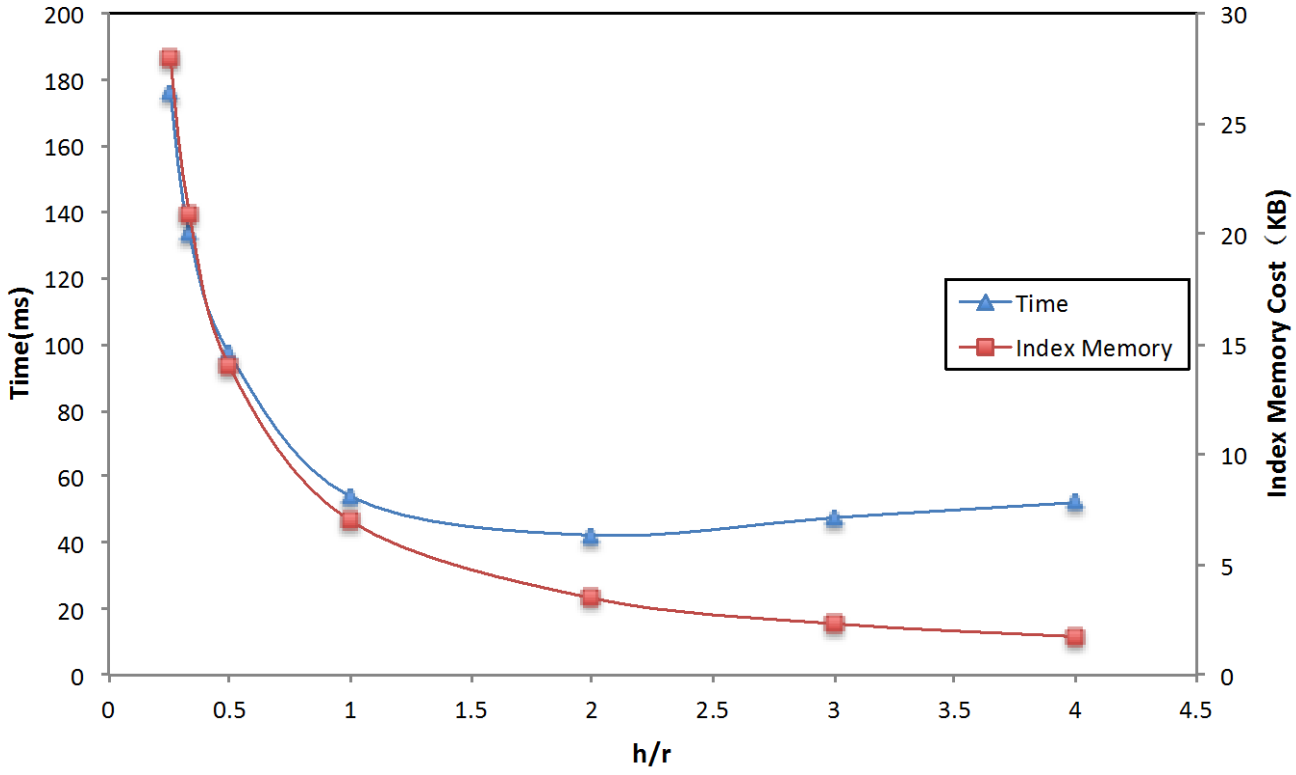
quadtreeMatch on Synthetic Data

- Best tree node size is about 256 points.



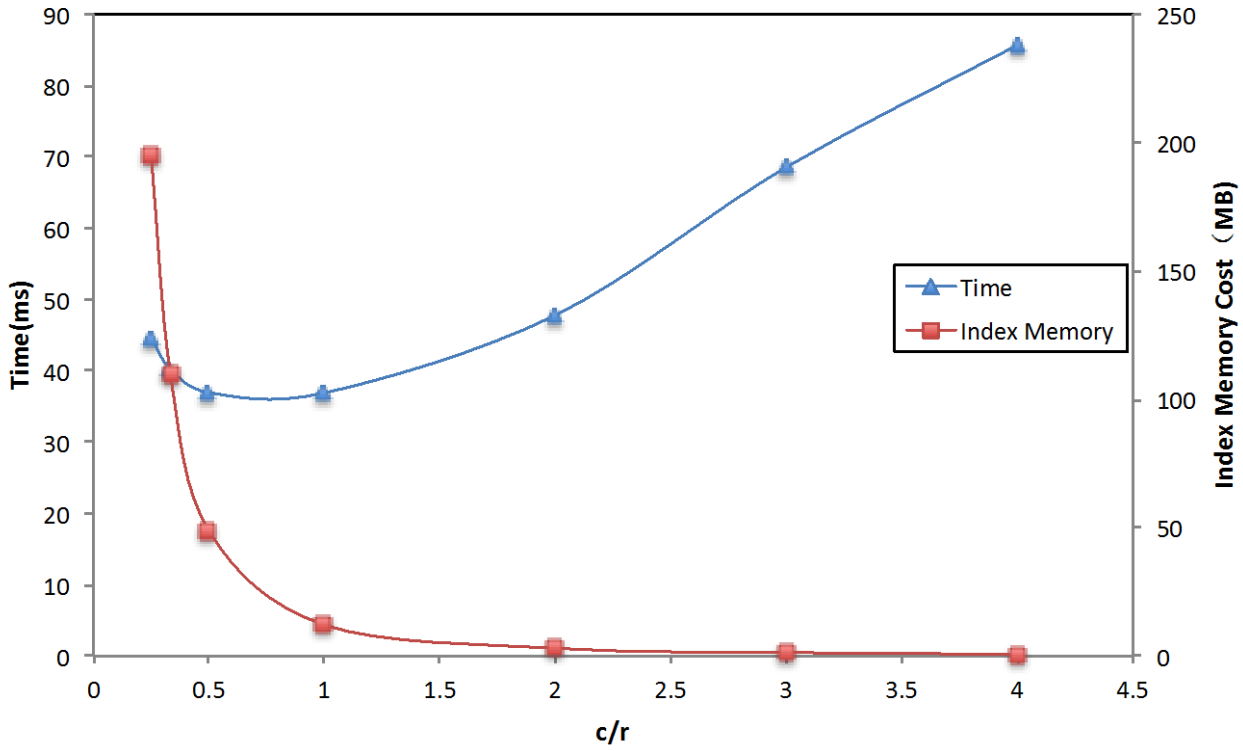
zoneMatch on SDSS Data

- Best Time: 42.29 ms, index: 3.5 KB, h=2r



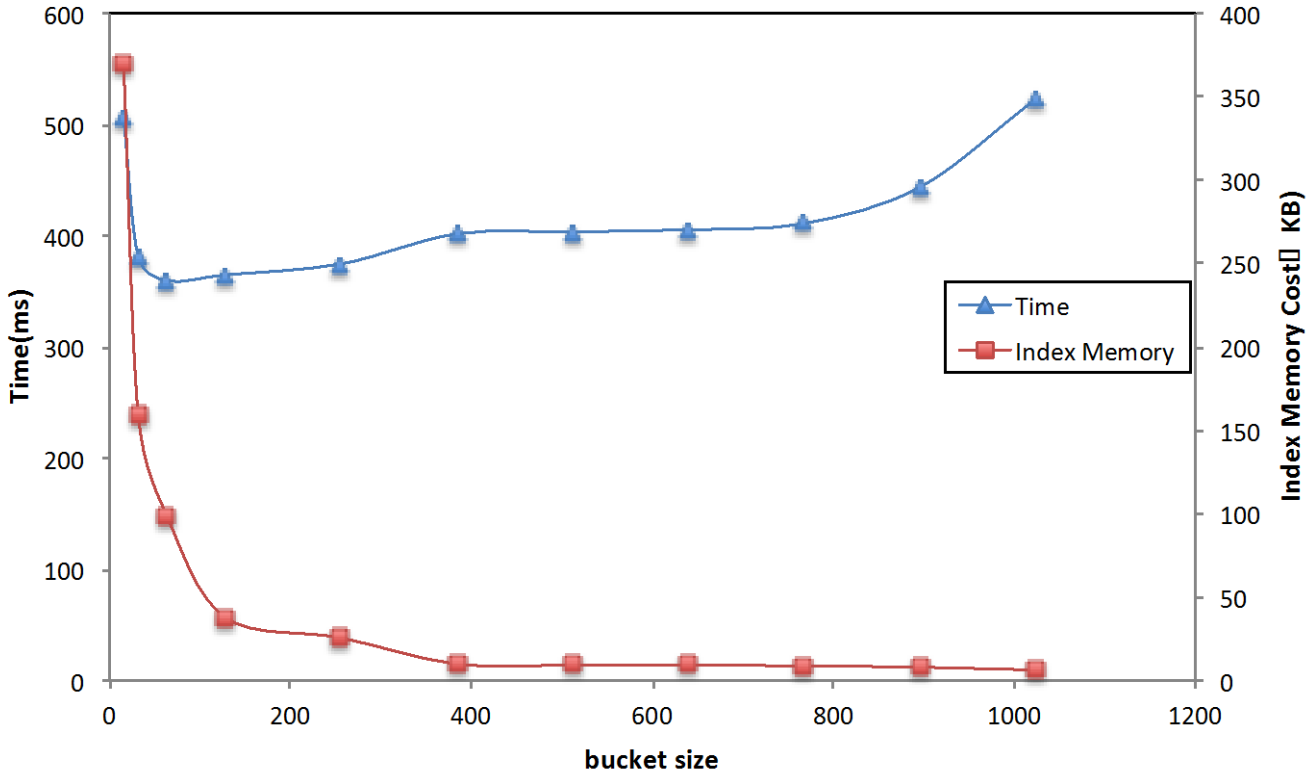
gridMatch on SDSS Data

- Best Time: 36.9 ms, index: 48.6 MB, c=0.5r

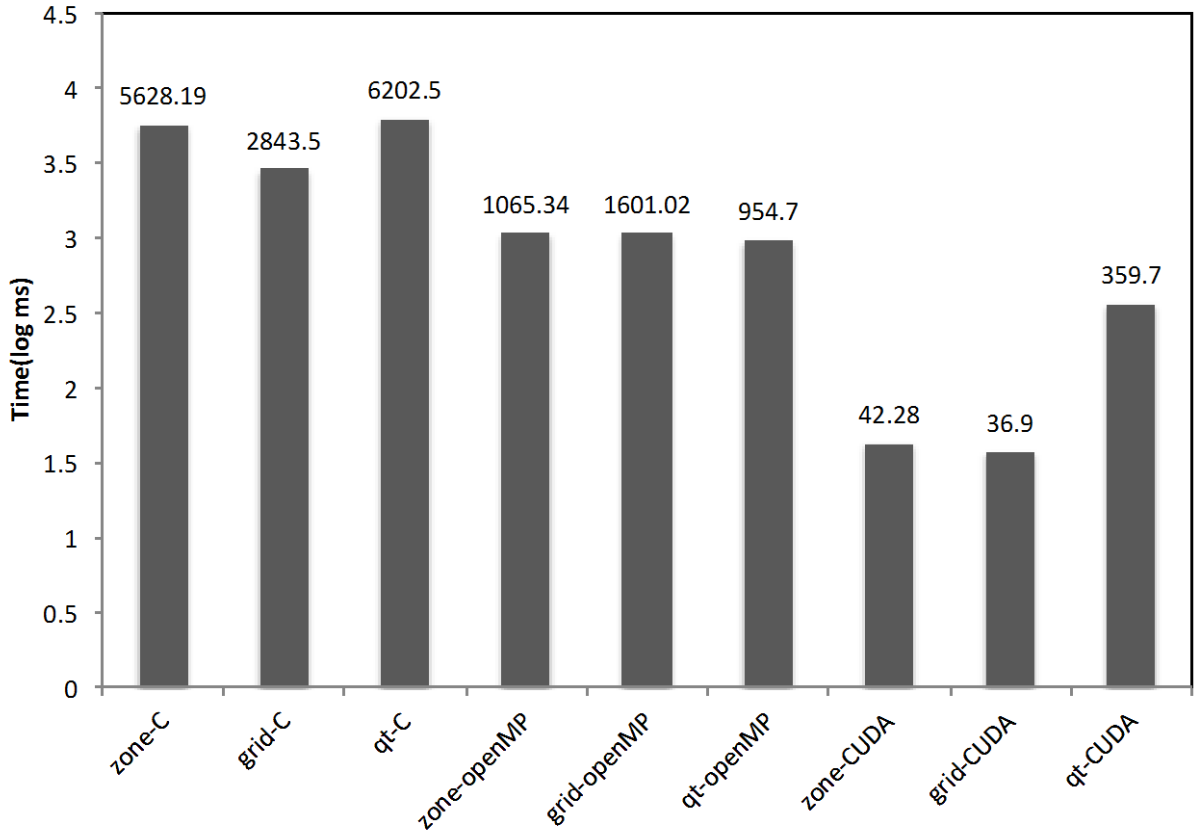


quadtreeMatch on SDSS Data

- Best time: 359.7 ms, index: 98.3 KB, b=64



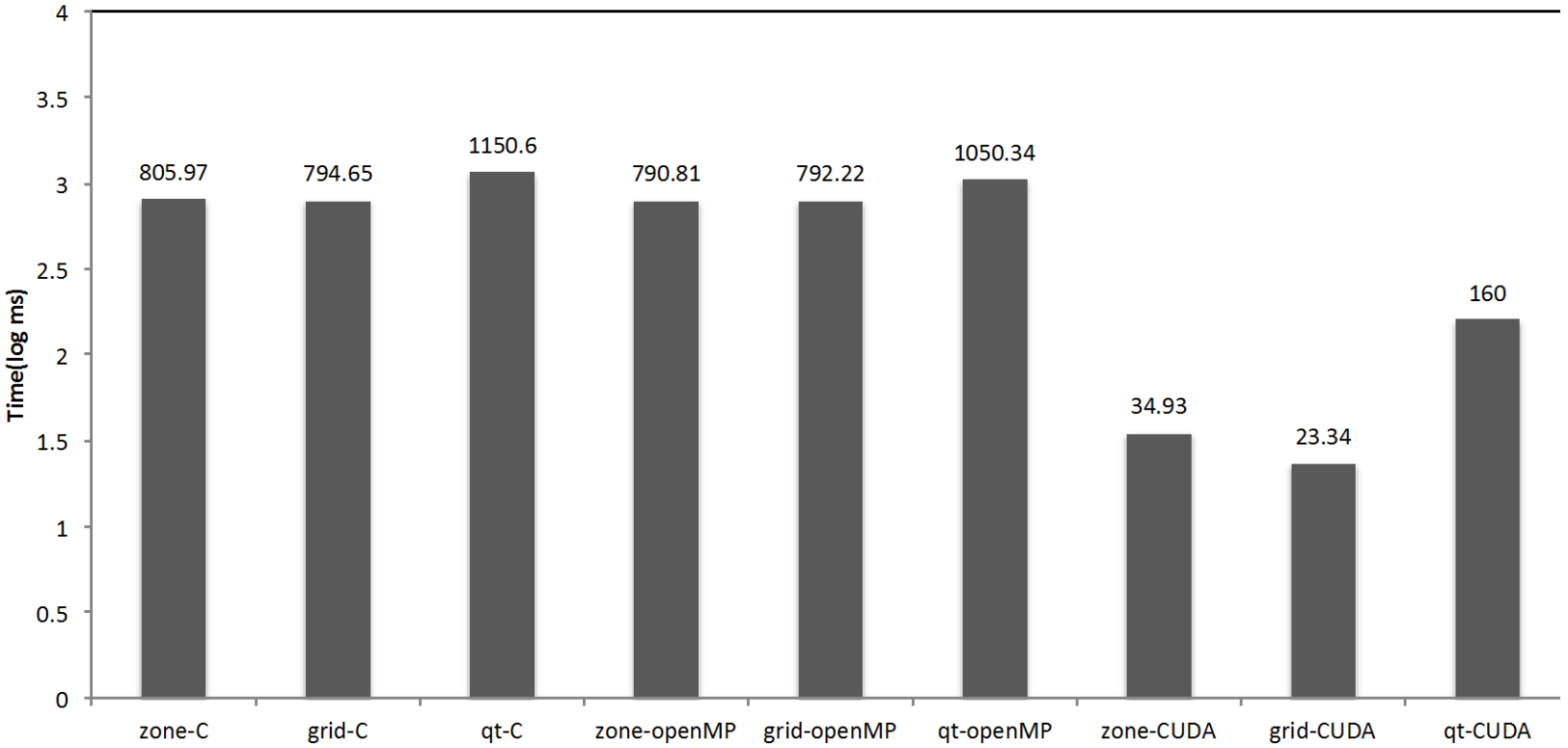
Query Performance Comparison



On SDSS Data

Algorithm	Speedup of GPU over openMP	Speedup of GPU over sequential
zoneMatch	25.2X	133.1X
gridMatch	28.8X	77X
quadtreeMatch	2.7X	17.2X

Index Construction Time



Summary

- On the GPU, zoneMatch is slightly slower than gridMatch; both are 8-9 times faster than quadtree Match.
- With 2.5million SDSS objects, grid index takes only 3.5KB, quad tree 98.3KB, and zone 48.6MB to achieve the best performance.
- All GPU-based algorithms perform much faster than their CPU parallel and CPU sequential counterparts.

http://www.cse.ust.hk/starflow/