Fast Cross-matching of Astronomical Catalogs on GPUs

Matthias A Lee & Tamás Budavári
The Johns Hopkins University

Kernel Optimization

For each zone in Segment A, the worker thread launches as many kernels as there are zones within range in Segment B. Every thread of each of these kernels calculates the vector distance between a unique pair of object, one object from each segment. If the computed distance is below a certain threshold, the result is marked “found”.

Many of the calculated distances will be greater than the threshold, resulting in a sparsely populated results array. We utilize this fact to minimize to data we need to return. In our current implementation we compact the “found” results after every kernel run into a global array using a parallel prefix scan as described in “Parallel Prefix Sum (Scan) with CUDA” by Mark Harris[3].

Results

Our results show an incredible 45x speedup over the previous best implementation based on multi CPU performance on SQL Server. Our speedups are slightly sub-linear, but that can be expected as the overhead of controlling worker threads grows, speedups decrease.

The larger the segment size we can keep in memory the better we perform on a same size problem. This is due to the overhead of launching sorters and passing jobs back and forth. Theoretically Nvidia c2050s support 2 segments of 50 million objects, but we ran into glitches with our kernels and the overhead required by the Thrust library for sorting and copying segments. This is a work-in-progress.

In the future we would like to attempt adding dynamic segment size scaling based on the memory of available GPUs. We also aim to have a public release of version 1.0 in the near future. Please email us if you are interested.

References


[2] “The Zones Algorithm for Finding Points Near-a Point or Cross-Matching Spatial Datasets”[2] to map the catalogs into horizontal zones of a height zoneHeight, where zoneHeight is measured in arcseconds. (see Figure 5)

If we aim to find all matches within a radius 6 of object O, we find the min. and max. zone enclosing our search radius in the range of declination:

\[
\text{minZone} = \left[\text{[deg] - 6} / \text{zoneHeight}\right]
\]

\[
\text{maxZone} = \left[\text{[deg] + 6} / \text{zoneHeight}\right]
\]

In Figure 5 minZone and maxZone would be Zone 1 and Zone 3. We also compute the search radius bounds in the range of RA(right ascension). Given the min and max Zones as well as the RA range, we have successfully narrowed our search field.

Parallel on Multiple GPUs

From here on we move to the parallel implementation. We start by dividing the catalog into multiple segments so that 2 segments can fit on 1 GPU at a time. We then utilize the Thrust library to sort each segment by declination (see Figure 6) to allow us to break each segment into zones.

Next we create Jobs. Each Job is a unique match between one segment of catalog A and one segment of catalog B, such that all combinations are covered. Ex: SegA_0 x SegB_0 | SegA_0 x SegB_1 | and so on.

These Jobs are passed off to the Job manager which stores them in a Job Queue. The Job Manager is the entity which provides jobs to each processing thread. When a processing thread finishes a Job it requests a new one. To optimize memory transfers, between the host and the device, the Job Manager prefers to give out Jobs based on which segments are left in memory from the previous Job, therefore cutting down duplicate transfers.

Contact us:

Matthias A Lee
School of Arts, Humanities, and Social Sciences
The Johns Hopkins University

Tamás Budavári
Department of Neuroscience and Physiology
The Johns Hopkins University

email us at idies@jhu.edu if you are interested.