Compute the Spatial Distance Histogram of a Set of 2D Points using Parallel Computing

Yu Liang, Master Student in CSE
Jiabo Liang, Master Student in CSE
Jiayi Wang, Master Student in CSE

Abstract — *as parallel computing has evolved, all areas of computing are faced with rebuilding to get faster processing speed. This article focuses on one important issue in scientific simulation data analysis: the spatial distance histogram (SDH). When calculating the spatial distance from the histogram when the calculation is very huge. Usually these calculations should be kept for a short period of time, in the event of failure, but also to start again. We propose a more efficient algorithm that uses parallel computing and prove that this algorithm has a good fault tolerance. The core of our algorithm is to decompose the space of a particle into a quadtree-based structure and then obtain the number of particles in each node of the tree to determine the spatial distribution of the particle. Our research shows how we can implement these algorithms in a graphics processing unit (GPU) to achieve a certain level of fault tolerance. The efficiency of our proposed algorithm will have a wide range of effects on the data produced by the actual simulation study*

Index Terms — spatial distance histogram, parallel computing, 2D Points

I. INTRODUCTION

Every year through computer simulation experiments will produce a large amount of data. Despite the use of a database management system to process this data, data management software is not sufficient to handle such a large amount of data [1-3]. Traditional database management system is established for business applications, not suitable for managing big data. So we need parallel computing to redesign the data management system. These programs usually require parallelism methods that use large numbers of GPUs for computations to significantly improve operational efficiency [4]. In this way, these scientific big data can be effectively used for scientific simulation experiments.

In order to analyze the data of 2D points, traditionally, scientists have to do a lot of complex calculations to create a spatial distance histogram [5-7]. In general, the queries used in this analysis are based on the location of each particle: the function of all m-tuple subsets involving data is called the m-body correlation function. One such analysis query discussed in this article is the so-called Spatial Distance Histogram (SDH). SDH is a histogram of the distances between all pairs of particles in the system, which represents a continuous probability distribution of distances (called the radial distribution function (RDF).) This type of query is very important in an MS database as one of the basic building blocks to describe a series of key quantities (such as total pressure and energy) required by physical systems.

With the development of GPU-based parallel computing in recent years, GPU parallel computing has become a fantastic choice when the CPU cannot handle large amounts of social networking data quickly[8, 9]. Through the GPU's large-scale processing power, can quickly construct SDH relationship between notes. In this study, we use parallel computing to construct SDH relationship in 2D points.

II. METHOD

The SDH problem can be formally described as follows: given the coordinates of N particles and a user-defined distance w, we need to compute the number of particle-to-particle distances falling into a series of ranges (named buckets) of width w: [0, w), [w, 2w), …, [(l − 1)w, lw]. Essentially, the SDH provides an ordered list of non-negative integers H = (h0, h1, …, hl−1), where each hi(0 ⩽ i < l) is the number of distances falling into the bucket [iw, (i+1)w). We also use H[i] to denote hi in this paper.

Clearly, the bucket width w is the only parameter of this type of problem.

To capture the variations of system states over time, there is a need to compute SDH for a large number of consecutive frames. We denote the count in bucket i at frame j as Hj [i].

In this project, we use a computer to generate a random number of points. Then we find the two points farthest away to determine the maximum boundary and use it as the root node of the quad-tree

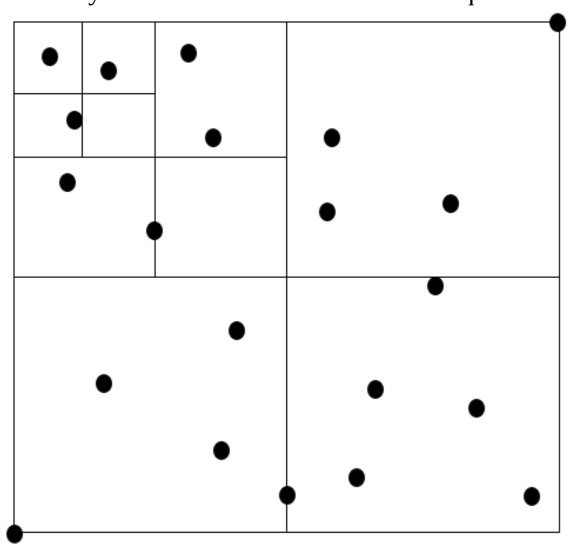

Fig 1. Data points distribution

Each node has 4 child nodes, down until it reaches a threshold, the last layer is the leaf node. The nodes above the leaf nodes need only store the number of internal points of the Cell (each square is a Cell). The leaf nodes not only store the number but also store the index of all the points in the cell, so as to directly calculate the distance between the points distance.

Down through the tree layer by layer until you find that level diagonal less than the width of the bucket to stop. This level is the work level (start level). The distances between points in each cell on this level will be less than the width of the bucket by adding all of them to the first histogram (histogram [0]). Then calculate the maximum distance and the minimum distance between the two cells with the cell of the same level, and if the maximum distance and the minimum distance are in the same bucket, then the mutual distance between the two cell middle points is in this bucket. If the maximum distance and the minimum distance are not in the same bucket, compute their child node until it meets this condition or calculate the distance between the leaf nodes directly.

If you can not find the diagonal less than the width of the bucket level, the direct calculation of the distance between the leaves and the point.

III. RESULT

All the points we randomly generate falls between 0 and 23,000. First, we generate 10,000 points and build SDH with bucket width 500. The results for CPU (Figure 1) and GPU (Figure 2) are as below.

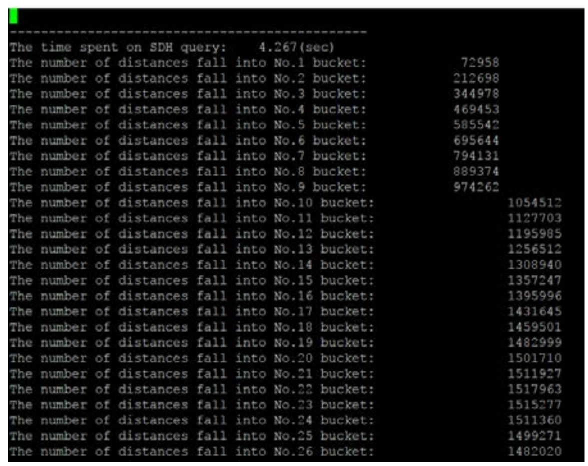

Fig 2. Time cost 4.267s for 10,000 points with bucket width 500 in CPU

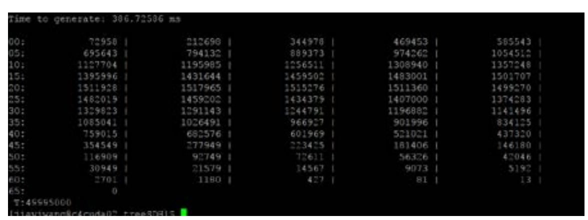

Fig 3. Time cost 0.386s for 10,000 points with bucket width 500 in GPU

Second, we generate 100,000 points and build SDH with bucket width 1,000. The results for CPU (Figure 4) and GPU (Figure 5) are as below.

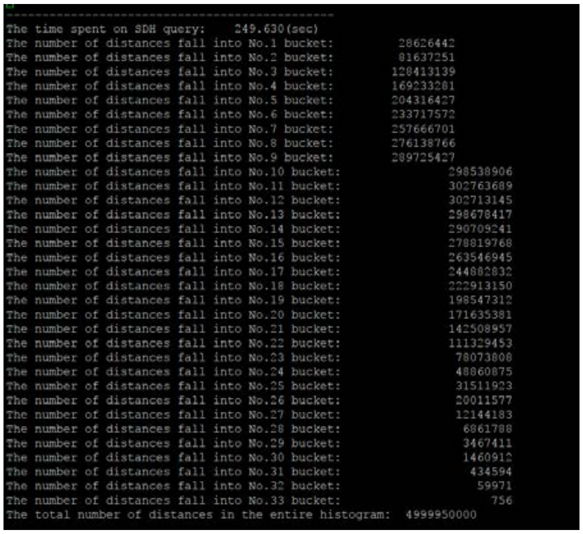

Fig 4. Time cost 249s for 100,000 points with bucket width 1,000 in CPU

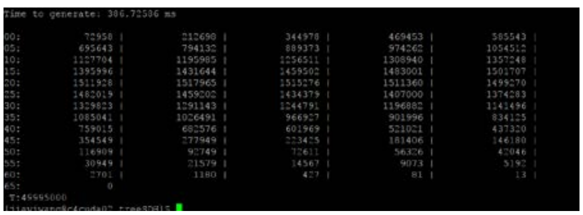

Fig 5. Time cost 12.894s for 100,000 points with bucket width 1,000 in GPU

Third, we generate 100,000 points and build SDH with bucket width 500. The results for CPU (Figure 6) and GPU (Figure 7) are as below.

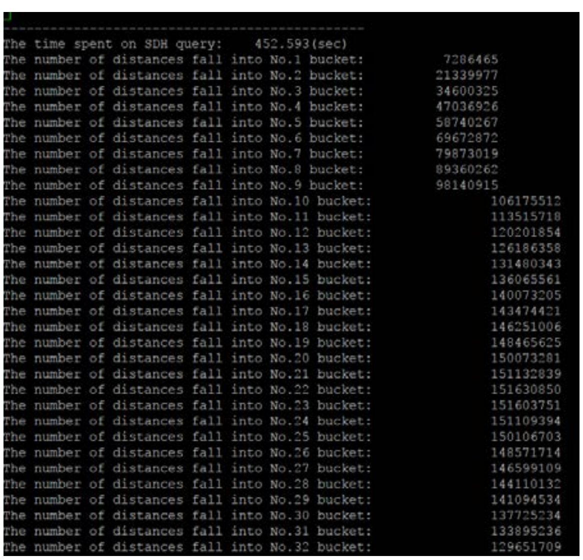

Fig 6. Time cost 452s for 100,000 points with bucket width 500 in CPU

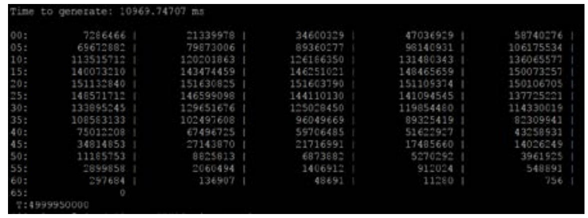

Fig 7. Time cost 10.969s for 100,000 points with bucket width 500 in GPU

Finally, we generate 100,000 points and build SDH with bucket width 5,000. The results for CPU (Figure 8) and GPU (Figure 9) are as below.

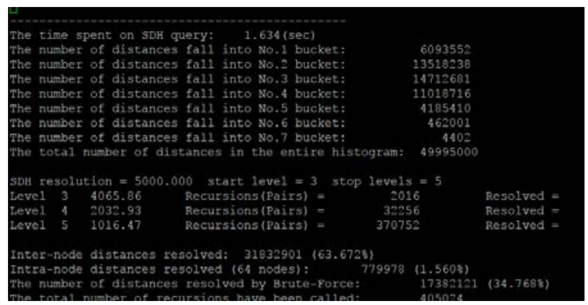

Fig 8. Time cost 1.634s for 100,000 points with bucket width 5,000 in CPU

Fig 9. Time cost 3.911s for 100,000 points with bucket width 5,000 in CPU

Points; bucket width	CPU	GPU
10,000; 500	4.267s	0.386s
100,000; 1000	249s	12.894s
100,000; 500	452s	10.969s
10,000; 5,000	1.634s	3.911s

Table 1. Overall result

III. CONCLUSION

We found that for 10,000 dots, the GPU performed better when the width was small. When the width is large, the CPU performs better. For

100000 points, the GPU behaves better than the CPU, regardless of the width.

III. REFERENCE

[1] R. Bhatti, A. Samuel, M. Y. Eltabakh, H. Amjad, and A. Ghafoor, "Engineering a policy-based system for federated healthcare databases," (in English), Ieee Transactions on Knowledge and Data Engineering, vol. 19, no. 9, pp. 1288-1304, Sep 2007.

[2] M. Gray and J. Kalpers, "Ranger based monitoring in the Virunga-Bwindi region of east-central Africa: A simple data collection tool for park management," (in English), Biodiversity and Conservation, vol. 14, no. 11, pp. 2723-2741, Oct 2005.

[3] M. H. Ng et al., "BioSimGrid: Grid-enabled biomolecular simulation data storage and analysis," (in English), Future Generation Computer Systems-the International Journal of Grid Computing Theory Methods and Applications, vol. 22, no. 6, pp. 657-664, May 2006.

[4] A. Kumar, V. Grupcev, Y. K. Yuan, J. Huang, Y. C. Tu, and G. Shen, "Computing Spatial Distance Histograms for Large Scientific Data Sets On-the-Fly," (in English), Ieee Transactions on Knowledge and Data Engineering, vol. 26, no. 10, pp. 2410-2424, Oct 2014.

[5] V. Grupcev et al., "Approximate Algorithms for Computing Spatial Distance Histograms with Accuracy Guarantees," (in English), Ieee Transactions on Knowledge and Data Engineering, vol. 25, no. 9, pp. 1982-1996, Sep 2013.

[6] J. Narwade and B. Kumar, "Comparison of Spatial Color Histograms Using Quadratic Distance Measure," (in English), 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), pp. 995-998, 2015.

[7] S. P. Chen, Y. C. Tu, and Y. N. Xia, "Performance analysis of a dual-tree algorithm for computing spatial distance histograms," (in English), Vldb Journal, vol. 20, no. 4, pp. 471-494, Aug 2011.

[8] C. Chen, K. L. Li, A. J. Ouyang, Z. Tang, and K. Q. Li, "GPU-Accelerated Parallel Hierarchical Extreme Learning Machine on Flink for Big Data," (in English), Ieee Transactions on Systems Man Cybernetics-Systems, vol. 47, no. 10, pp. 2740-2753, Oct 2017.

[9] F. Han and S. Z. Sun, "Petroleum Geoscience Big Data and GPU Parallel Computing," (in English), 2015 1st Ieee International Conference on Multimedia Big Data (Bigmm), pp. 292-293, 2015.