PARALLEL IMPLEMENTATION OF DBSCAN ALGORITHM USING MULTIPLE GRAPHICS ACCELERATORS

Sándor Szénási[1]
[1] szenasi.sandor@nik.uni-obuda.hu, Óbuda University - Hungary

ABSTRACT

Road accident hotspot (also called black spot) identification is one of the most important tasks of road safety experts to avoid further accidents. One of the available methods is based on the GPS coordinates of accidents and uses the data-mining method called DBSCAN. The DBSCAN method is well parallelizable because we can run multiple searches from different starting points of the search space. This paper presents an NVIDIA CUDA implementation of the algorithm which uses multiple graphics accelerators to decrease the necessary runtime. As the results show, the accuracy of the method is the same as of the sequential one, but the runtime is significantly lower.

Keywords: GPS, DBSCAN, graphics accelerators, CUDA, multi-GPU

INTRODUCTION

There are several methods to find accident black spots [1],[2] but most of them are based on the traditional road number + road section location identification system [3],[4]. Nowadays, where accident scene investigators use GPS (Global Positioning System) devices to record the accident location, these methods are outdated. To use the old methods we have to convert the GPS coordinates to road number + road section pairs, which can decrease the accuracy. Another issue is that in the case of built-up areas, this conversion cannot be used because several streets exist without any road number.

To solve this problem, we can use spatial methods based on the GPS coordinates directly. Finding an accident black spot is very similar to the general clustering tasks of data-mining. The goal is very similar: to find an area where the density of the elements is higher than outside the area. In the case of accidents: we have to find some parts (road sections, junctions, etc.) of the public road network, where the density of accidents (the number of accidents divided by the area) is significantly higher than outside the area.

RELATED WORK

One of the most promising density-based data mining methods is the DBSCAN (Density-Based Clustering of Application with Noise) algorithm. It is capable of finding clusters of any shape, and it is well usable in noisy environments as well. We can adopt this method for black spot searching: we start the DBSCAN algorithm using the locations of accidents, and the result will be the collection of accident black spot candidates.

We have implemented this method as a web-based application. The input parameters of the method were the followings [5]:

- *ε:* a radius-type value
- *MinPts:* a lower limit for accident numbers
- *MinDensity:* a lower limit for accident density

To use DBSCAN in any clustering tasks, we have to define the concept of distance between two elements of the search space. In the case of accident prediction, it is easy: accidents (elements) are identified by two-dimensional coordinates (GPS latitude and longitude values), and the distance between them is the standard Euclidean distance between these two points.

The DBSCAN algorithm can calculate the “transitive closed domain” of directly dense accessibilities (the maximum domain of densely accessible elements from a starting point). To calculate this, we have to select one point of the search space (in our case, this is typically a location of a randomly selected accident). In the next step, we have to count the number of accidents in the ε-environment of the starting one. If this number is greater than MinPts, we have to start the iterative search process:

1. We choose the nearest new accident and extend the cluster candidate (the set of accidents) with this.
2. We investigate the ε-environment of the last selected accident, and if there are any acceptable new points, they are also added to the cluster.
3. If there are not any new accidents, the previous iteration ends.

The result of the clustering algorithm is the accident black spot candidate.

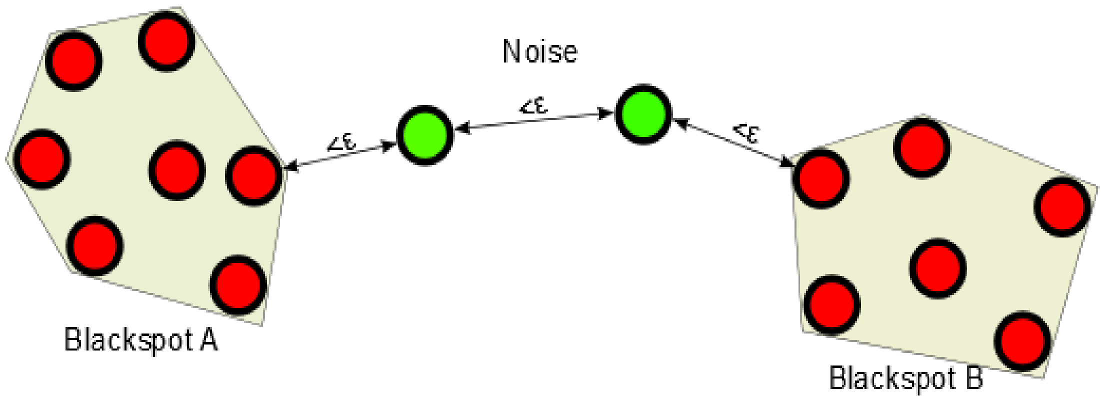

Figure 1. Two independent black spots (red circles) and some noise (green circles).

DATA-PARALLEL DBSCAN

The DBSCAN algorithm is a very resource-intensive method.

- In the growing phase, we have to check the distance between lots of elements (accidents). It is possible to speed this up by spatial indexes, but in the case of accidents, the distribution of the elements is uneven. This makes it hard to develop the appropriate spatial indexing method.
- We have to start the growing phase from several locations of the search space. We can drastically decrease the required number of growing iterations with a simplification: we start the growing process only from the locations of already existing accidents. Nevertheless, in the case of a large number of accidents, the full process remains very resource-intensive.

As a previous work [6] we have implemented the data parallel DBSCAN algorithm using graphics accelerators. The parallelization is based on the observation that there is not any communication need between the different growing processes. Therefore, we can run these in any order or in a parallel fashion. Using this idea, we can fully utilize the processing power of all processor cores (CPU and GPU cores) without unnecessary waiting.

The graphics accelerator based implementation was very impressive. It was about ten times faster than the sequential CPU implementation. However, it is worth noting that the speed up depends on the number of accidents. If the search space contains only a few accidents, the CPU can be faster than the graphics card.

DBSCAN USING MULTIPLE GPU-S

As known, it is possible to use multiple graphics accelerators in one personal computer (and of course, it is also available in the case of server machines). In this case, CUDA based applications can distribute work across multiple Graphics Processing Units. But this is not done automatically; the developer has to control the distribution process. It is necessary to decompose the problem into two or more independent parts that can be run in parallel (one on each GPU in the same time).

The CUDA programming language has native support for multiple graphics cards. The host thread can set the device it operates on at any time by a particular function call. After that, all device memory allocations and kernel launches are made on the last selected device (except we use streams to attach operations to a given GPU).

Splitting the accidents into two equal sets and loading these into one of the GPUs can lead to incorrect results, because:

- Estimating the size of the expected clusters is hard. Considering an extreme example, all of the accidents can be in the same black spot candidate. In this case, it is obvious, that none of the graphics cards will find the entire black spot.
- Several accidents do not have enough neighbouring accidents in the ε-environment. It is not necessary to start a DBSCAN from these accidents, and in unfortunate cases, one of the GPUs has a lot of accidents of this kind so it will finish the DBSCAN method early. In the meantime, another GPU has to do

several growing processes. This situation leads to an unbalanced resource allocation.

Based on these considerations, the distribution of the accidents is as follows:

1. Load accident data from the RDBMS (relational database management system) to the main memory.
2. Count the number of accidents acceptable as DBSCAN starting elements (N).
3. Upload all of these accidents to all graphics cards.
4. Execute N/G DBSCAN growing processes in each graphics cards.
5. Download the results from all graphics cards.
6. Merge the results into one set of black spot candidates.
7. Display the results to the operator.

It is worth noting that it is possible that the result sets of different GPUs contain overlapping black spot candidates. Therefore, it is important to find these overlapping items and keep only one cluster/merge these into one.

EVALUATION OF THE ALGORITHM

The main reason for the multi-GPU implementation was to speed-up the search process. To check the results, we run a lot of tests using different number of accidents in the following configurations:

- Configuration 1 – only CPU
    - Processor: Intel® Core™ i7-2600
    - Architecture: Sandy Bridge
    - Number of cores: 4
    - Memory: 16GB DDR2
- Configuration 2 – one GPU
    - Graphics accelerator: NVIDIA GTX Titan Black
    - Architecture: Kepler
    - Number of CUDA cores: 2880
    - SMX Count: 15
    - Memory: 6GB GDDR5
    - Host: the same as the CPU configuration
- Configuration 3 – multi-GPU
    - Graphics accelerators: 2 x NVIDIA GTX Titan Black
    - Host: the same as the CPU configuration

Table 1 shows the execution time values. The first column illustrates the number of accidents. In the case of CPU implementation, we use only one thread to process all accidents; in the case one GPU implementation, the number of threads equals to the number of accidents; in the case of multi-GPU implementation, the number of threads is half of the total number of accidents.

The subsequent columns show the runtime values of all implementations. The second column contains the CPU implementation; the third column contains the one GPU implementation; the fourth column contains the multi-GPU implementation.

Table 1: runtime values for each configuration according to the number of accidents

Number of accidents	CPU runtime (μs)	GPU runtime (μs)	Dual-GPU runtime (μs)
100	6551	124	163
150	6742	170	170
200	6301	219	211
250	11388	282	260
300	8215	388	298
350	9976	486	311
400	12360	576	371
450	12023	747	445
500	16697	976	496
550	21882	1051	597
600	19087	1274	722
650	19914	1552	814
700	22827	1537	909
750	29544	1759	1034
800	28165	2045	1098
850	29808	2385	1284
900	33663	2735	1436
950	33617	3146	1599
1000	41446	3592	1885

Fig. 2. Shows the same values. It is clearly visible that both GPU implementations are faster than the CPU one. It is also visible that the multi-GPU implementation is about twice faster than the single-GPU version.

CONCLUSIONS

Our goal was to implement an accident black spot searching method using multiple graphics accelerators. We have developed a new algorithm based on DBSCAN data-mining method and implemented it as a CUDA C application. The processing speed increases linearly with the number of GPUs. We can assume, that this statement is true in the case of more GPUs, while the number of accidents is high enough to utilize all streaming multiprocessors of all graphics accelerators.

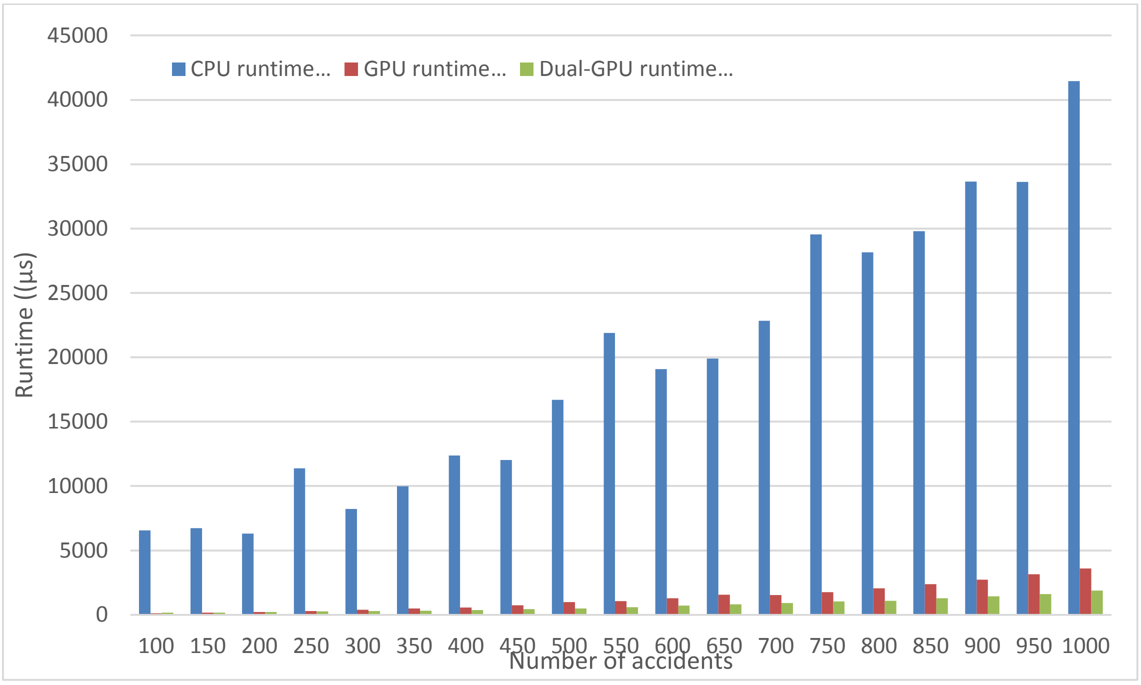

Figure 2: CPU (blue), one-GPU (red) and multi-GPU (green) runtimes

It is worth noting that we use only one thread of the CPU; however, it is possible to run a multithreaded application to utilize all processing cores. But the expected speed-up cannot be larger than 400% (in the case of four cores). And it is also worth noting that we use a naïve implementation in the GPU, using shared memory could significantly decrease the required runtime.

Acknowledgement

The authors would like to thank NVIDIA Corporation for providing graphics hardware for the GPU benchmarks through the CUDA Teaching Center/GPU Education Center program. The authors also wish to thank the members of the CUDA Teaching Center at Óbuda University for their constructive comments and suggestions. The content of the paper, however, does not necessarily express the views of these companies and persons, the authors take full responsibility for any errors or omissions.

REFERENCES

[1] A. Bogardi-Meszoly, A. Rovid, H. Ishikawa, S. Yokoyama, and Z. Vamossy, "Tag and topic recommendation systems," in Acta Polytechnica Hungarica, vol. 10, no. 6, pp. 171–191, 2013.

[2] Road Safety Manual. PIARC Technical Committee on Road Safety. 2003

[3] I. Lizamol, A. Shibu, M. S. Saran, Evaluation and treatment of accident black spots using Geographic Information System, International Journal of Innovative Research in Science Engineering and Technology, Vol. 2., No. 8, 2013, pp. 3866-3873.

[4] K. Geurts, G. Wets, Black Spot Analysis Methods: Literature Review, Steunpunt Verkeersveiligheid bij Stijgende Mobiliteit, 2003, pp. 1-30.

[5] S. Sergyán, L. Csink, Automatic Parameterization of Region Finding Algorithms in Gray Images, in 4th International Symposium on Applied Computational Intelligence and Informatics, Timisoara, 2007, pp. 199-202.

[6] S. Szénási, P. Csiba, „Clustering Algorithm in Order to Find Accident Black Spots Identified By GPS Coordinates", SGEM 2014, Bulgaria, 17-26 June 2014, pp.497-503