GPU IMPLEMENTATION OF DBSCAN ALGORITHM FOR SEARCHING MULTIPLE ACCIDENT BLACK SPOTS

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

ABSTRACT

Nowadays, graphics cards are not only used for processing video data but also suitable for other general purpose operations too. We can use these graphics accelerators in the field of road accident black spot searching. There are several black spot searching methods and most of them are very time and resource consuming; therefore, it is worth implementing them as parallel GPU codes. This paper presents a naive implementation of the DBSCAN algorithm to speed-up accident black spot localization. We can run multiple DBSCAN searches from different points parallel to find clusters of accidents with high accident density.

Keywords: GPU, CUDA, DBSCAN, GPS, accidents, hot spots

INTRODUCTION

Road accident hotspot (also called black spot) identification and cataloguing [1] is one of the most important tasks of road safety experts. Its purpose is to identify locations of the national public road network (both inside built-up and outside built-up areas), where the accident density is higher than expected. There are several papers about hot spot locator techniques [2,3,4], but none of them give us complete and unquestionably accurate results. Therefore, experts use many different procedures and it is still interesting to find some alternative solutions (e.g. fuzzy based approaches [5]).

In our previous paper [6] we have presented a new black spot searching algorithm using data mining methods. This new method uses the well-known DBSCAN algorithm, and the results are very promising. It can find accident black spots in the public road network, especially in built-up areas. However, the DBSCAN algorithm is a very time and resource-consuming method. Therefore, its runtime is not satisfactory.

The DBSCAN algorithm itself is a sequential algorithm: it is hard to parallelise the environment processing part. However in most cases, we have to run several DBSCAN iterations from different starting points, and these can be considered as independent searches. Therefore these searches are executable in any order and this leads to the idea that we can run these in a parallel fashion.

For this reason, it is worth examining the DBSCAN algorithm implemented for multi-core systems, especially graphics accelerators. In this paper, we discuss the runtime of a massively parallel implementation using NVIDIA graphical processing units.

RELATED WORK

One of the most known density-based data mining methods is the DBSCAN (Density-Based Clustering of Application with Noise) algorithm. It is capable of recognising clusters (in our case, black spots) of any shape, and can be used in noisy environments as well (in our case, noise refers to accidents not belonging to any black spots).

The main parameters [7] of the DBSCAN algorithm are the following:

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

Furthermore, we have to define the concept of distance between two elements. In our case, it is very simple: accident locations are identified by GPS coordinates, and the distance between them is the Euclidean distance between the two coordinates.

The goal of the DBSCAN algorithm is to calculate the “transitive closed domain” of directly dense accessibilities, which means the maximum domain of densely accessible elements from a given starting point. We can calculate that this starts from each point of the database (accidents). For this, we have to check all neighbouring points in the ε-environment of the starting accident. If there are any, we choose the nearest of them and extend the cluster with this. In the next iteration, we have to investigate the ε-environment of this new element and if there are any acceptable points, they are also added to the cluster. If there are not any selectable points, the iteration ends. The final cluster is the result of the clustering algorithm.

Some consequential definitions: 1) *ε-environment*: the space within a radius of ε of an element; 2) *internal element*: if the ε-environment of an element contains at least MinPts number of elements; 3) *directly densely accessibility:* for a given domain of elements, one element is directly densely accessible from other internal elements if it is the first element’s ε-environment; 4) *dense reachability:* it is similar as it is directly densely accessible, but for one element it is permitted to be accessible from another only through a chain of directly densely accessible elements; 5) *densely connected elements:* if there is an element from which both are densely reachable with the given parameters; 6) d*ensity-based cluster:* a domain of densely connected elements that shows maximum accessibility of density.

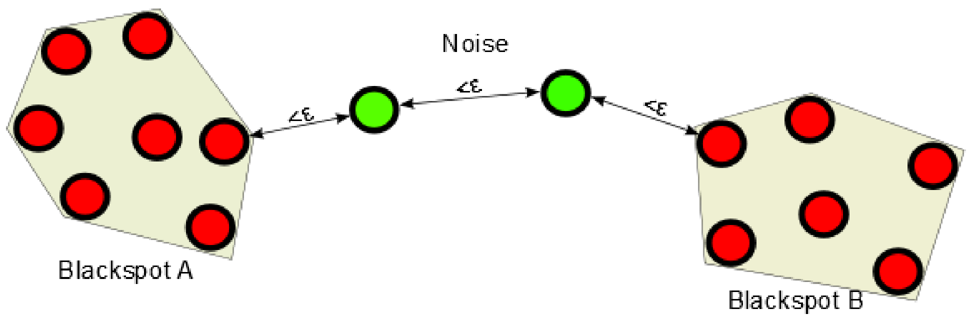

Figure 1. Two independent black spots (red circles) and some noise (green circles).

Our goal is to find sets of accidents in which all elements are densely connected and no further expansion is possible. We call these sets as black spot candidates, because these need some further investigation by road-safety experts (are these real black spots? what are the reasons? etc.).

DATA-PARALLEL DBSCAN

As it is visible, we can run more than one DBSCAN iteration from different starting points, these will not interfere with each other. In the case of multi-core CPUs, we can assume linear speed-up if we use more than one core. There is no communication between threads: therefore, we can fully utilise the processing power of all cores without unnecessary waiting.

Based on this, it is worth implementing the whole algorithm to a graphic accelerator too. All threads must execute the same iterations using different data: this is the main attribute of the embarrassingly parallelisable problems (in parallel computing, we call a problem embarrassingly parallel, if little or no effort is required to separate it into several parallel tasks. This is the same case where there is no dependency or communication between those separate sub-problems). In case of newer graphics cards, there are thousands of processing units (for example, our Tesla K40 graphics accelerator has 2880 CUDA cores in 15 streaming multiprocessors). Therefore, we need a lot of threads to fully utilise the processing power of a GPU. We can easily fulfil this statement, because we have to start the iteration from each accident and there are thousands of accidents.

The main steps of the GPU implementation are the following:

1. Load accident data from the relational database management system (RDBMS) to the main memory.
2. Split accident location data and copy this to the GPU memory.
3. Run the DBSCAN function on the GPU using as many threads (called kernels in GPU terminology) as the number of accidents.
4. Copy the result (list of black spot candidates) back to the main memory.
5. Post-process the results (filtering the black spot candidates).
6. Display the results to the operator.

Step 3 uses the traditional DBSCAN algorithm implemented with the CUDA C language to make it capable to run in the GPU. We use a very simple (naive) implementation, where each thread runs separate DBSCAN iterations starting from a given accident.

EVALUATION OF THE DATA-PARALLEL ALGORITHM

The main reason of the GPU implementation was to speed-up the calculations. To check the results, we run a lot of tests using different databases (using different number of accidents) and compare the CPU and GPU runtimes.

We measure only the kernel runtimes not including the host to device and device to host memory copies.

We used the following configurations for the tests:

- CPU configuration
  - Processor: Intel® Core™ i7-2600
  - Architecture: Sandy Bridge
  - Number of cores: 4
  - Memory: 16GB DDR2
- GPU configuration
  - Graphics accelerator: NVIDIA Tesla K40c
  - Architecture: Kepler (GK110B)
  - Number of shaders: 2880
  - SMX Count: 15
  - Memory: 12GB GDDR5
  - Host: the same as the CPU configuration

Table shows the runtime values. The first column contains the number of accidents (it equals to the number of threads in the case of the GPU implementation), the second column shows the CPU runtime. In the case of the GPU, there is an additional step, we have to copy all accident data to the GPU. The cost of this method is visible in the third column, and the runtime of the GPU based DBSCAN in the fourth column. The last column shows the full runtime of the GPU implementation.

Table 1: CPU and GPU runtimes

Number of accidents	CPU runtime (	GPU runtime (µs)
100	683	193
200	887	163
300	811	82
400	1079	85
500	1271	154
600	1627	105
700	2181	172
800	2355	207
900	2986	254
1000	3417	240

As it is visible in Fig. 2, the GPU implementation is usually faster than that of the CPU. The speed-up depends on the number of accidents, more accidents mean better utilisation of the graphics accelerator.

The need for a high number of accidents is not a real drawback. We have checked the national road accident database in Hungary, and the average annual number of accidents is ~1500. A typical black spot searching process usually takes accidents of the last 3-5 years, therefore we can utilize the full processing power of the GPUs.

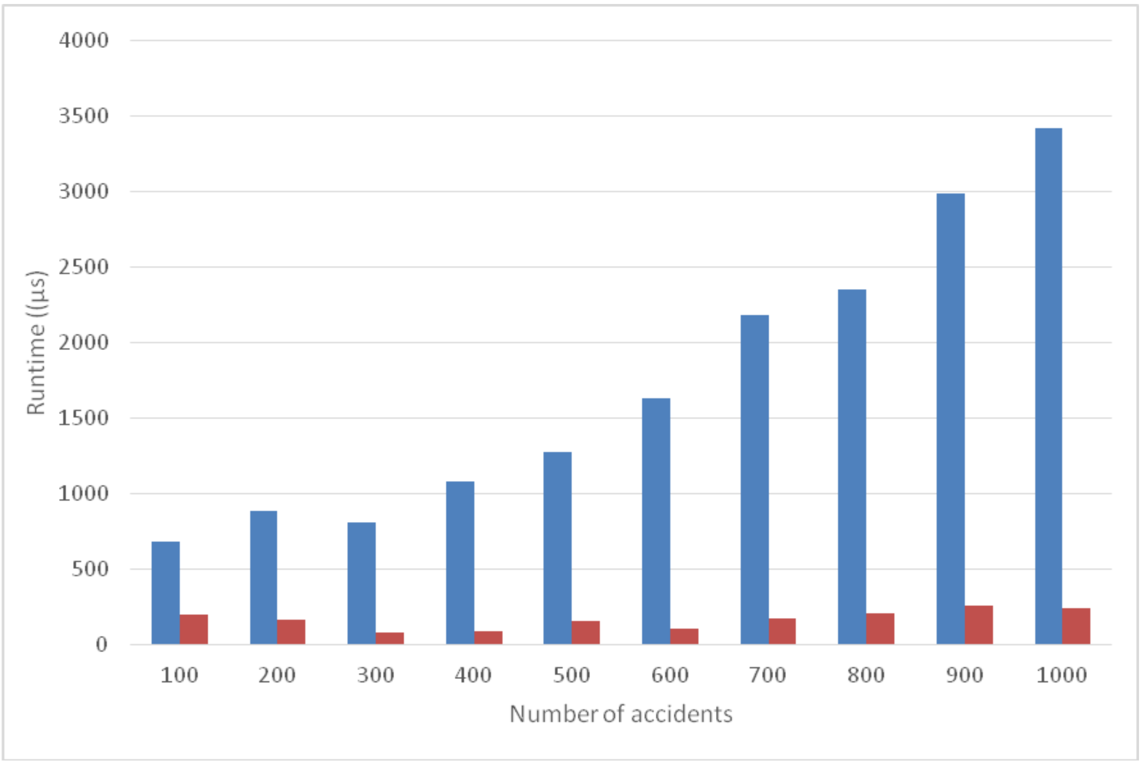

Figure 2: CPU (blue) and GPU (red) runtimes

CONCLUSIONS

Our objective was to implement a new accident black spot searching method in the GPU using the DBSCAN algorithm. We have developed a new algorithm for this purpose and implemented it as CUDA C application. After the implementation, we ran several tests to check the usability of the new algorithm (based on the runtime and on the accuracy of the results).

As it is visible the GPU implementation is significantly faster than the CPU one. The results are better in the case of a large number of accidents. However, it is obvious that it is worth using the graphics accelerator based parallel method in the case of small accidents numbers too.

The results of the new method are exactly the same as the old values. This means that the accuracy of the new parallel and the old sequential algorithm is exactly the same. Therefore, we can use this method in the future without any drawbacks.

ACKNOWLEDGEMENTS

The authors would like to thank NVIDIA Corporation for providing graphics hardware for the GPU benchmarks through the CUDA Teaching 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] E. Tóth-Laufer, M. Takács, and I. J. Rudas, “Neuro-fuzzy risk calculation model for physiological processes,” in IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics (SISY), pp. 255–258, 2012

[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

[7] 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.