GPU ACCELERATED COMPUTING IN RADAR INFORMATION PROCESSING

Cao Viet Linh*, Vu Tuan Anh, Duong Tuan Viet

Abstract: *Position synchronization on maps is a critical problem in displaying radar information. Previously, this process was done in a sequential manner in the central processing (CPU). This often causes some bottlenecks leading to a long processing time because of the limited CPU architecture. To mitigate this problem, we proposed an algorithm of parallel processing to take advantage of the multi-core architecture of GPU to quickly reproject coordinates in radar information processing, improving the information display performance. Practical tests were done on different data sets to demonstrate the performance of this method, and the result showed that the improvement speeds are between 10 to 100 times better in each test.*

Keywords: Radar information processing; Position synchronization; Parallel processing; GPU processing.

1. INTRODUCTION

Representing the Earth's surface in 2D is always a critical problem in position synchronizing for radar information processing, tracking, and navigation. Although there is much research into this problem, the reprojection of a 3D sphere into flat surface is inevitably accompanied by deformations. Many methods were proposed and researched, and each causes a different deformation (in angles, areas, distances, etc.). Because of that, each map is typically generated by a specific method, depended on the viewing area and its intended purpose.

In essence, radar information can be considered as a digital map of target positions, noise, geo-features, and the synchronization between them is to reproject all radar information into the same reprojection currently using in the geographic map. To correctly reproject information from one spatial reference to another requires a heavily computing processing pixel by pixel, as such takes a lot of time. This in turn causes a delay in the radar system, reduces the responsiveness of the radar. It is unacceptable in radar applications, especially in military radars, where the reaction time is a critical requirement.

To alleviate this problem, scientists and engineers have developed many supercomputers with very high-speed CPU. However, using these computers is costly and requires too many resources. Furthermore, increasing CPU speed can only be done so much as the limitation in the number of transistors. Another simpler and more cost-effective method is using parallel processing in the Graphics Processing Unit (GPU) to alleviate the burden of the CPU. This method takes advantage of the fact that the reprojection can be done pixel by pixel independently with each other.

In this paper, we will briefly discuss the differences between CPU and GPU architectures, then present a method of parallel processing for position synchronization and evaluate its performance against the traditional CPU utilizing method.

2. THE DIFFERENCES BETWEEN CPU AND GPU ARCHITECTURES

CPU is the central unit of a computer, designed to control the data processing

provided by other devices. They are built with a complex architecture and focus on executing more general controls and a large number of instructions as quickly as possible. Thus, CPUs are designed for quick responses (low latency) by large buffers and complex arithmetic logic units (ALUs). This architecture has a drawback for the CPU, which limits the number of cores that can exist in the same chip. To ensure rapid processing, CPUs are equipped with high-speed buffers with different levels to reduce the main memory accessing time.

GPUs are first designed for graphic processing, which contains a large number of smaller processing units to quickly calculate the position of each pixel independently. This allows GPU manufacturers to eliminate some logic control units and buffers to produce simplified cores. It is because GPUs contain many more cores than CPUs. Another difference is that GPUs have a much smaller buffer than CPU. Thus, GPUs are limited in the number of tasks and the complexity of the tasks they can process.

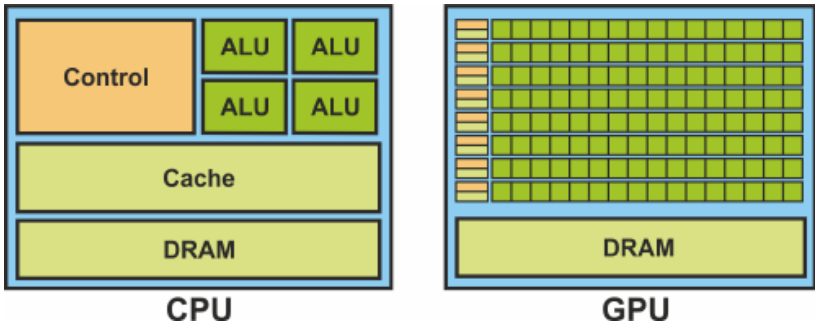


Figure 1. *The differences between CPU and GPU architectures.*

In summary, CPU architecture is designed to maximize the performance of individual tasks but can process various types of tasks. In contrast, GPU is for parallel processing, focuses on processing a large number of tasks simultaneously, but only for limited types of tasks.

For reprojection between spatial references, coordinates of radar information are independent of each other. As such, they are ideal for parallel processing using GPU.

3. PARALLEL PROCESSING IN COORDINATES REPROJECTION USING GPU

3.1. Coordinates reprojection in sequential processing

All information (plots, tracks, clutter maps, etc.) output by the radar's signal processor and tracker is in range and azimuth with reference to the radar position, which is enough to use if we don't want to place them on a map. Using them with a map, however, requires us to convert them from range, azimuth to pixel position. To map them correctly to their pixel positions in the geographical map, we first have to convert them to their real-world position, i.e., with latitude and longitude positions. After that, using the map's spatial reference, project them into their pixel coordinate to place them accurately on the screen. This must be done for each input coordinate and typically done sequentially in practice. The general method for sequentially reprojecting coordinates comprises 4 main steps, described below:

Step 1: Read the input data. In this step, the algorithm will unpack the data set from the output of the signal processor and tracker, thenread the input data set.

Step 2:Calculate their geodesic coordinates. Calculate their real position on the Earth. This is done by solving the direct geodesic problem: Finding the position of point P given the coordinate of the radar position O (λ_0, ϕ_0), azimuth θ, range *r* to point P, see figure 2.

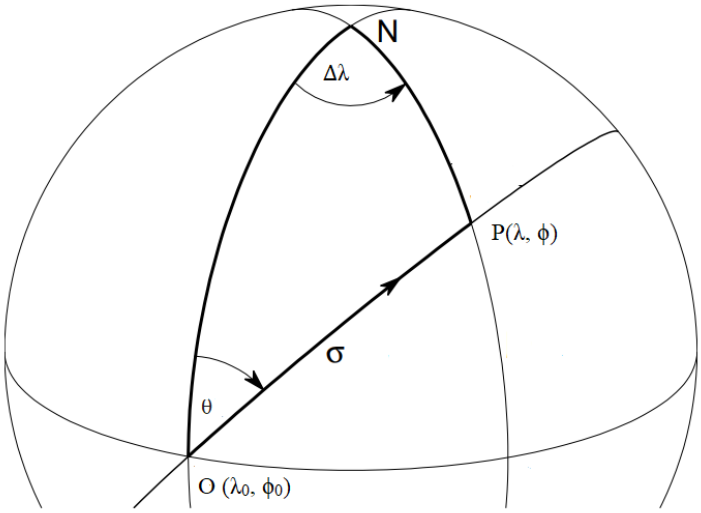


Figure 2. *Direct geodesic problem.*

This is solved by solving the formulas for a spherical triangle, as follows:

First, we calculate the angular distance between two points O and Pin the great circle by:

$$\sigma = r / R \tag{1}$$

where, R is the Earth's radius.

Then, solving the spherical triangle, we get:

$$\phi = \arcsin(\sin\phi_0 \cdot \cos\sigma + \cos\phi_0 \cdot \sin\sigma \cdot \cos\theta) \tag{2}$$

The longitude difference $\Delta\lambda$ is calculated by the following formula:

$$\Delta\lambda = \arctan \frac{\sin\sigma \cdot \sin\theta}{\cos\phi_0 \cdot \cos\sigma - \sin\phi_0 \cdot \sin\sigma \cdot \cos\theta} \tag{3}$$

Thus, the longitude of point P_2 is given by:

$$\lambda_2 = \lambda_0 + \Delta\lambda \tag{4}$$

Step 3: Reproject the pixels

After getting the real positions on Earth of each point, their coordinates will be reprojected into the spatial reference using in the geographic map. For the purpose of demonstration, we will use the Web Mercator projection, a de facto standard for Web mapping applications. This a variant of the Mercator projection, used by virtually all major online map providers, including Google Maps, Bing Maps, OpenStreetMap, etc. Its official EPSG identifier is EPSG:3857.

The formulas for the Web Mercator are fundamentally the same as for the standard spherical Mercator, but before applying zoom, the world coordinates are adjusted such that the upper left corner is (0, 0) and the lower right corner is (256, 256):

$$x = \frac{256}{2\pi} \cdot 2^{zoom} \cdot (\lambda + \pi) \tag{5}$$

$$y = \frac{256}{2\pi} \cdot 2^{zoom} \left(\pi - \ln\left[\tan\left(\frac{\pi}{4} + \frac{\phi}{2} \right) \right] \right) \tag{6}$$

where *zoom* is the detail level of the maps, it depends on the map resolution (calculated by meters per pixel), x and y are the pixel coordinate values.

Step 4: Combine the reprojection and generate the output. In this last step, the pixel values from the previous step are used to generate a reprojected image.

3.2. Design and execute parallelly

Because the output pixels are independent of each other, we can apply parallel processing to increase the projection speed. Figure 3 illustrates the projection process using parallel processing in the GPU. This process is similar to the 4 steps method described above. However, to process parallelly, each thread of the GPU processes a pixel or a set of pixels in the output image simultaneously, not sequentially like in CPU. (Note: depending on the selected spatial reference, the coordinate reprojection uses different formulas; however, the parallel process has similar steps as in sequential processing).

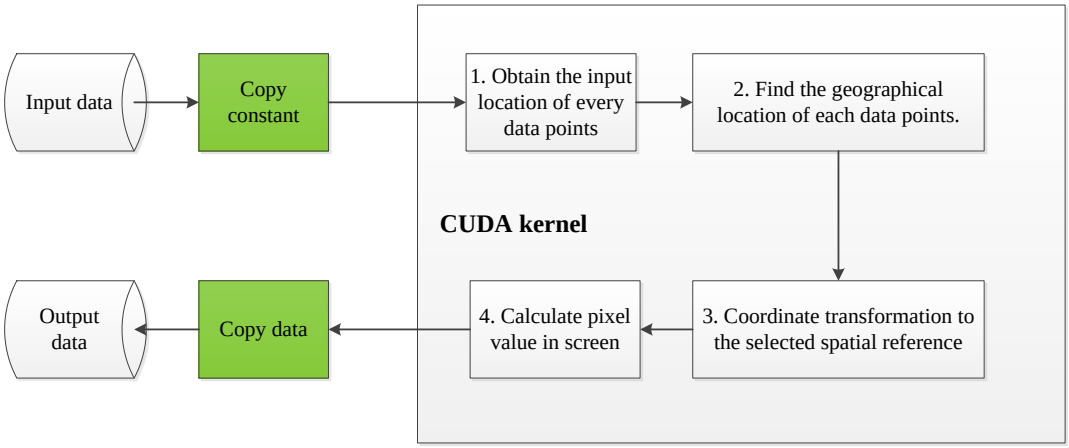


Figure 3. *Coordinate reprojection in parallel processing.*

Compare to sequential processing, this method has some additional differences. Firstly, the parallel processing includes 2 more steps for transferring data between CPU and GPU (2 green rectangles in figure 3). This transferring process can cause some delay because of the limited memory of GPU. Secondly, the pixel calculations (equivalent to the 2nd and 3rd steps of the method above) are executed in parallel by GPU's cores.

In parallel processing, particularly with a large data set, the data processing process is divided into smaller tasks. Because of the limited memory of GPU, the data partitioning is critical to ensure the correct reprojection without overflowing

the memory. GPU's memory is much smaller than the CPU's memory. When the data set's size exceeds the GPU's memory, it is partitioned into smaller chunks. A chunk is defined as a part of the data set using in parallel processing, determined by the number of pixels/cell in 1 dimension (horizontal or vertical). Each chunk is processed as one grid. A grid is a containerof several blocks, created when a kernel (i.e., CUDA function) is launched. Each block contains a number of threads. Each thread will process a pixel (or a subset of pixels).

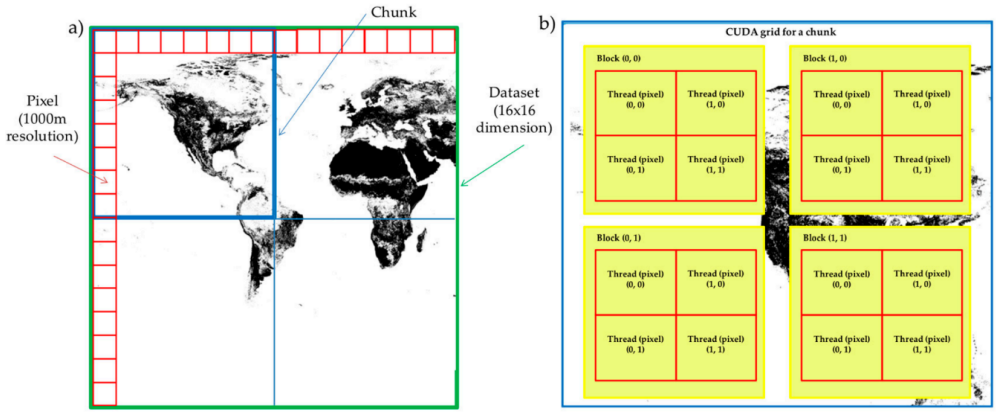


Figure 4. *Divide the processing into groups: (a) Divide the data set into groups and pixels, (b) Thread architecture in each block of a grid.*

To illustrate the process of generating chunks. Figure 4 shows a decomposition example. In figure 4a, the data set has a grid size of 16 rows by 16 columns, or 16 pixels in the x-direction and 16 pixels in the y-direction. The spatial resolution is 1000 meters. This dataset is divided into four chunks. The chunk comprises 8 columns by 8 rows. Figure 4b shows the block and thread composition for a grid containing a chunk from figure 4a.

Figure 5 presents the algorithm diagram using for data partitioning. It starts with the provision of a dataset as input and configurations of the projected out. Additionally, the size of the chunks for breaking the output image should be specified. Based on these provided grid sizes for chunks, the algorithm generates the divisions so that every chunk is indexed with a pair of row and column numbers. Given the grid size information, the algorithm only retrieves the pixels in the input image required by each of the chunks on the fly. As a result, instead of loading the complete input, only a portion of the dataset is loaded at a time for GPU processing to avoid memory overflow. Depending on how users may want to access the reprojected results, the datasets can be organized through different data indexing methods and load individually.

One crucial step in data partitioning is to determine the equivalent pixel in the input and output. To do that, first we calculate all the bounding pixels in the output space (i.e., the pixel in the bound of that areas), then reproject back to find the equivalent data points in the input space and build the polygons to partition the input. All data points in this area will be reprojected to the output space. To increase the processing speed, instead of using random polygons, we create a minimum

rectangular bound contains all the pixels. This is also executed parallelly.

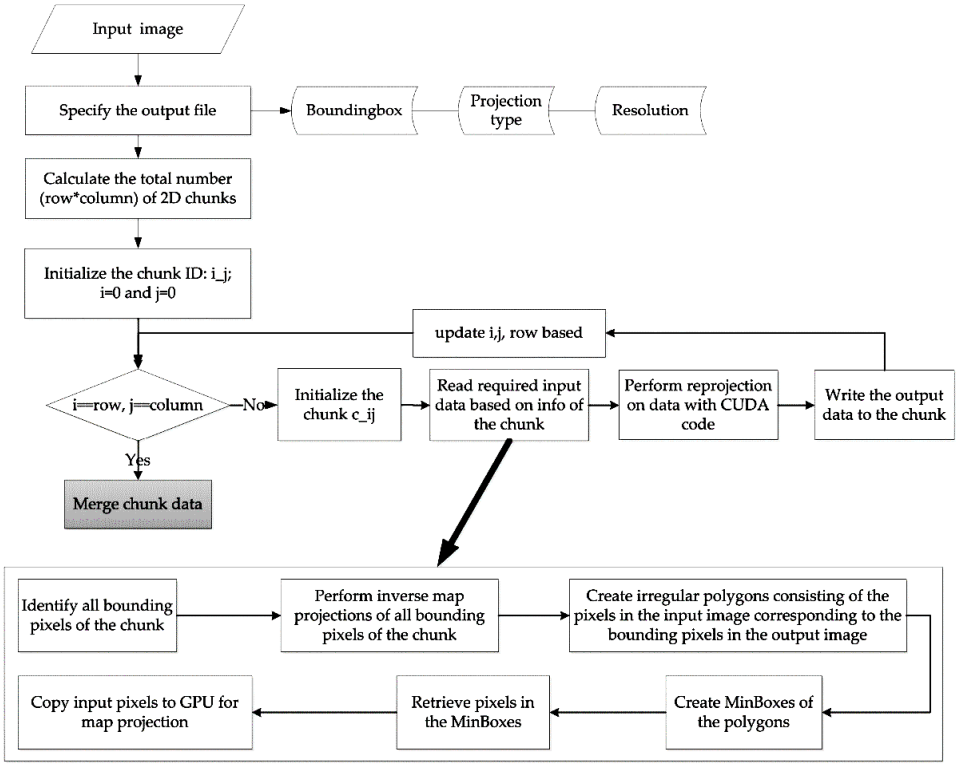


Figure 5. *Algorithm diagram for partitioning data.*

This architecture design helps take advantage of GPU parallel processing capability, using its numerous cores to process data quickly. The advantage of this method will be proven through empirical results presented below.

4. EXPERIMENT'S RESULTS

4.1. Testing environment

To evaluate the performance of GPU parallel processing, we have done some tests in some geographic data sets. For the purpose of demonstration, we use Web Mercator projection to test the performance of this technique. This spatial reference is chosen because it is typically used in generating maps and virtually used in all online map services.

Table 1. *System configuration of test equipment.*

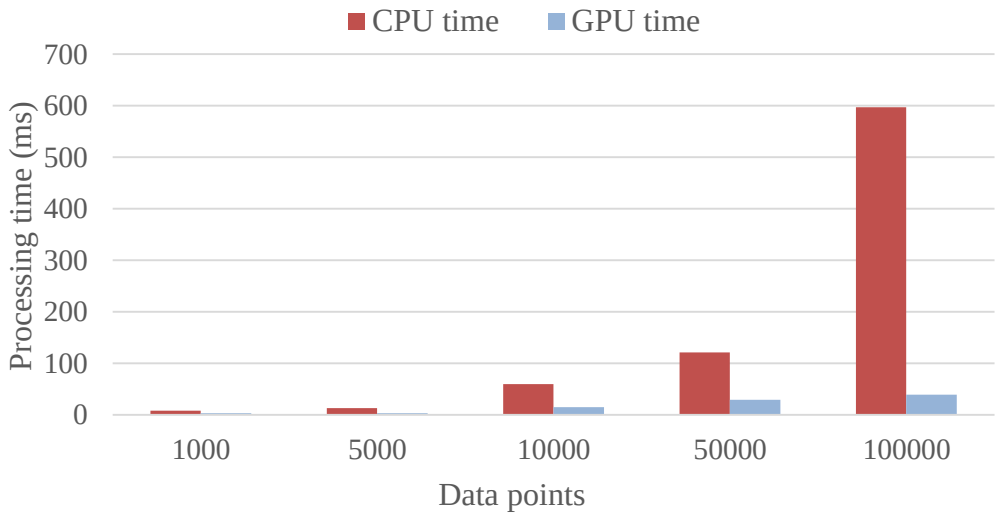
Device information	Computer 1
CPU	Intel Quad-core i7 2,40 GHz
Memory	16 G
GPU	Nvidia Geforce GTX 860M, 640 cores, 2 GB memory

Tests are done one in a mid-end computer presented in table 1. We don't use high-end computers to evaluate the feasibility of this method in normal working

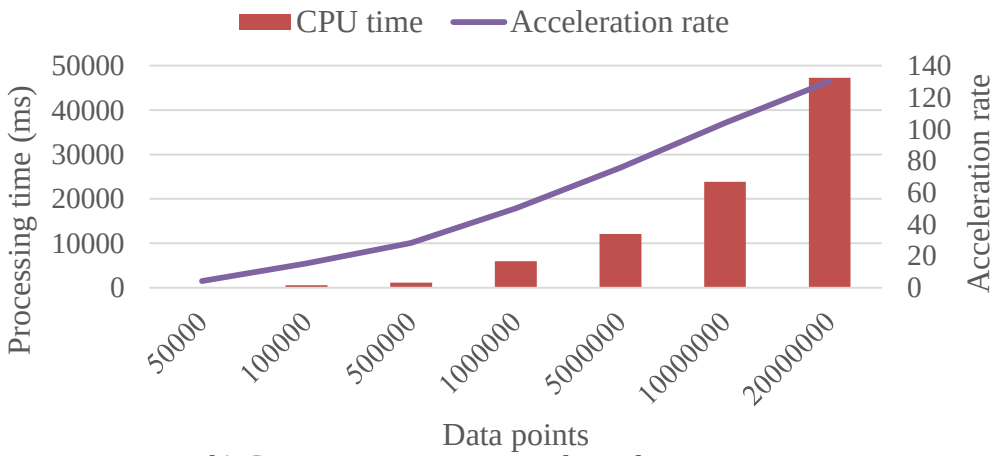
conditions. The source codes are written in NVIDIA CUDA TOOLKIT 10 and Visual Studio 2017.

The tests will be done for 1000, 5000, 10000, 50 000, 100000, 500000, 1000000, 5000000, 10000000, 20000000, 40000000 points to compare the processing power of CPU and GPU.

4.2. Results



a) Comparision between CPU and GPU processing time



b) CPU processing time and acceleration rates

Figure 6. *Time to reproject when projecting using GPU and CPU (a), and their acceleration rates (b).*

First, we consider the duration cost to execute the same reprojection with CPU and GPU individually. Then, the results are comparing between two cases to ensure the calculation is correct in both cases. In the case of CPU, the output pixels are processed sequentially. In the case of GPU, for smaller data sets, each pixel is processed by one thread to avoid the imbalance in GPU processing. For big data set, it is required to divide the data set into smaller chunks to fit in with the GPU.

Total time from the starting moment to read data, reproject, and generate output, is recorded to compare with each other. Parallel processing is only applied for reprojecting pixel coordinates. Test results are presented in the graphs in figure 6. For visualization, the data points only display up to 100000 points in figure 6a. In fig. 6b, on the left is the processing time in milliseconds, on the right is the acceleration rates when comparing between GPU and CPU. The complete results are presented in Table 2.

As shown in the graphs and the table, in all cases, the parallel technique reduces the processing time several times compared to CPU technique. Acceleration rates are between 1 to 150 times. And the bigger the data size, the better the acceleration rate.

Table 2. *Test results.*

Data points	CPU time (ms)	GPU time (ms)	Acceleration rate
1000	1	1	1.00
5000	8	3	2.67
10000	13	3	4.33
50000	60	15	4.00
100000	121	29	4.17
500000	597	39	15.31
1000000	1189	42	28.31
5000000	6010	120	50.08
10000000	12096	160	75.60
20000000	23882	230	103.83
40000000	47259	363	130.19

When analyzing the acceleration rates, we can see that there are some abnormal in the acceleration rate. With a small number of data points, there is no acceleration between GPU and CPU. Also, theoretically, this should be the case because the complexity of this algorithm is O(n). But the acceleration rate is linearly proportional to the data size. That is due to many factors: First, the threads are synchronized to generate the output data; however, the load imbalance can cause degradation in performance (this doesn't happen in sequential processing). Thirdly, the abnormal in grid partitioning and block size (e.g., the grid size of 100 x 100, block size of 32 x 32 threads) causes rounding errors and may create a block in which some threads don't process any pixel). This also causes imbalanced load distribution. Third, GPU technique has two additional steps: transfer data from CPU to GPU, and vice versa. The transfer time varies depending on data size, group position in the input and output. The optimizing problem for this parallel method is not in the scope of this paper. Further researches need to be done to evaluate and find a more optimized method.

5. CONCLUSION

Testing and evaluation show that GPU parallel processing is ideal for the projection of coordinates. In all cases, it is shown that using GPU significantly reduces the time duration for coordinates transformation compared to sequential

processing in CPU. The tests show that processing in GPU yields better performance than in CPU, and the bigger the data size, the better the acceleration rates. With 1000 data points, the speed is the same in two methods. Increasing it to 5000, it started to show some acceleration in GPU, the processing time is 2,67 times better than in CPU, At 40000000 data points, the acceleration rate is 130,19, which is significantly better than CPU. This proves the advantage of parallel processing in mid and low-end computers, allow them to process complex computations with limited resources.

REFERENCES

[1]. Finn, M.P; Steinwand, D.R; Buehler, R.A: Mattli, D.; Yamamoto, K.H. *"A program for handling map reprojections of small scale geospatial raster data"*. Cartog. Perspect. 2012, 71, 53-67.

[2]. M.J. Flynn, *"Very high-speed computing systems,"* in Proc. The IEEE 54.12, 1966, pp. 1901-1909.

[3]. NVIDA Corporation. (February 2019). *"CUDA C Programming Guide"*, Design Guide PG-02829-001, Version 10.x.

[4]. Bakhoda, A.; Yuan, G.L; Fung, W.W.L; Wong, H.; Aamodt, T.M. *"Analyzing CUDA workload using a detailed GPU Simulator"*. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA, USA, 26-28 April 2009; pp. 163-174.

TÓM TẮT

ÁP DỤNG CÔNG NGHỆ TĂNG TỐC ĐỒ HỌA TRONG XỬ LÝ THÔNG TIN RA ĐA

Quá trình đồng bộ vị trí trên bản đồ là một vấn đề quan trọng trong hiển thị thông tin ra đa. Trước đây, quá trình này được thực hiện tuần tự trong bộ xử lý trung tâm (CPU). Điều này thường dẫn tới tắc nghẽn và làm tăng thời gian xử lý do cấu trúc hạn chế của CPU. Để giải quyết vấn đề này, nhóm tác giả đều xuất một thuật toán xử lý song song, tận dụng ưu điểm của cấu trúc đa lõi trong GPU để biến đổi tọa độ thông tin ra đa khi xử lý, làm tăng hiệu suất hiển thị. Các thử nghiệm thực tế đã được thực hiện trên nhiều bộ dữ liệu khác nhau để chứng minh cho hiệu quả của phương pháp này. Kết quả thử nghiệm cho thấy tốc độ xử lý tăng lên từ 1 đến 150 lần trong các bài thử.

Từ khóa: Thông tin ra đa; Tính toán địa lý; Xử lý song song; Xử lý GPU.

Received 18th June 2019
Revised 19th August 2020
Accepted 16th November 2020

Author affiliations:
Viện Ra đa – Viện Khoa học và Công nghệ quân sự.
*Corresponding author:linhviet306@gmail.com.