CUDA-Based Algorithm for Lidar Position Determination in Mobile Robotics

Lesia Mochurad, Ostap-Vasyl Matviiv, Halyna Lema, Roksolana Vilhutska

Lviv Polytechnic National University, 12 Bandera street, Lviv, 79013, Ukraine

Abstract

Accurate assessment of the Lidar position has always been a fundamental task in any navigation system: localization, dynamic map construction, and path planning. In mobile robotics, it is usually essential for the robot to know where it is in a known or unknown environment. However, processing the large amount of data obtained from LiDAR requires a lot of time and large computing resources. Therefore, optimizing the Lidar localization problem to obtain a high-quality real-time solution is becoming increasingly relevant. The paper proposes an algorithm for optimizing the computational process of determining the Lidar position. The method is based on CUDA technology and uses graphics processors. This approach is evaluated in comparison with the use of OpenMP technology. Hence, based on the proposed algorithm, it was possible to obtain an acceleration of 19.8. It is four times higher than the acceleration obtained using OpenMP technology. At the same time, the error in determining the Lidar position was 0.001.

Keywords

Particle Filter algorithm, optimization, parallelization, robotics problem.

1. Introduction

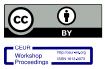
Autonomous localization and environment perception are essential for a mobile robot to perform high-level tasks. As for robot localization, odometry, and inertial sensors cannot avoid the problem of error accumulation. Therefore, exteroceptive sensors must be used to achieve effective localization and mapping. Among various external sensors, two-dimensional laser sensors can provide accurate and reliable environmental information with a wide viewing angle. They are widely used in perceptual tasks, including localization [1-2], mapping, and place recognition [3].

The problem of determining the Lidar position with optimal accuracy and in an acceptable time is relevant in various fields of application. Examples include Smart House (robot vacuum cleaners), vehicle route planning, dynamic map construction, collision avoidance, atmospheric research, cartography, economic efficiency management [4], etc. As known [5, 6], the Particle Filter algorithm solves the problem above. However, the number of reading iterations after each movement is significant if the Lidar moves many times in space. Accordingly, the execution time of this algorithm increases several times. That is, it is impossible to decide in real-time. One of the options for solving this problem is the development of a parallel algorithm [7] and its optimization using CUDA technology [8] and the graphics processors of the NVIDIA video card. Massively parallel hardware can perform significantly more operations per second than a CPU at a fairly similar cost, resulting in a performance increase of 50 or more in situations that allow it. CUDA is an SMT parallelism paradigm. It uses state-of-the-art GPU architecture to provide parallelism. A GPU contains (blocks (set of cores)) that operate on the same block-step instruction (this is similar to the SIMD model). But

MoMLeT+DS 2023: 5th International Workshop on Modern Machine Learning Technologies and Data Science, June 3, 2023, Lviv, Ukraine.

EMAIL: lesia.i.mochurad@lpnu.ua (L. Mochurad); ostap20021906@gmail.com (O.-V. Matviiv); halyna.v.mykhailiak@lpnu.ua (H. Lema)); roksoliana.b.vilhutska@lpnu.ua (R. Vilhutska)

ORCID: 0000-0002-4957-1512 (L. Mochurad); 0000-0002-6929-2537 (O.-V. Matviiv); 0000-0001-5298-7693 (H. Lema); 0000-0002-9291-8606(R. Vilhutska)



the amount of shared memory and global memory in the GPU is limited. One of the advantages of CUDA is that a general-purpose language is available, rather than having to use pixel and vertex shaders to emulate general-purpose computers.

The essence of the Lidar localization problem is that we may not be given its initial coordinate, but only points that characterize successive corners of our room. Also, with each step, the measurements of the Lidar to the room's walls (the Lidar rotates with a certain degree) and the vector of its movement after the performed actions are specified. In general terms, this algorithm is a successor of Monte Carlo methods [9, 10].

Therefore, the aim of this work is to study and optimize the Particle Filter algorithm using CUDA technologies.

2. Analysis of literary sources

At the moment, there is a significant number of literary sources dedicated to our problem. A review of such sources is an important stage of research, as it allows you to familiarize yourself with the state of scientific research in this area and determine possible ways of further development and improvement of algorithms.

Nowadays, various issues in robotics are increasingly being investigated [11-13]. One of the most interesting problem is determining the current Lidar position. In production and everyday life, this task can be widely used as a model in many fields, such as vehicle route planning, machine learning, and economic efficiency management.

The article [14] is devoted to developing an obstacle detection algorithm based on 2D LiDAR sensor data. The authors proposed a parallelization method using thread-safe collections and an efficient multi-threaded DBSCAN algorithm.

The article [15] provides a detailed analysis of the proposed parallelization algorithm for solving the Lidar localization problem using a genetic algorithm. The OpenMP parallel computing technology is used to optimize the computing process in order to speed it up and provide real-time results. Also, the computational complexity of the sequential algorithm is significantly reduced. Many numerical experiments were conducted using the multi-core architecture of modern computers. As a result, it was possible to speed up the computing process by approximately eight times and achieve an efficiency of 0.97. The notable difference in the execution time of the sequential and parallel algorithms allows increasing the number of Lidar measurements and iterations, which is relevant when modeling various robotics tasks. The obtained results can be significantly improved by choosing a computing system with more than eight cores.

Mochurad et al. proposed a parallelization method to speed up an obstacle detection algorithm based on 2D LiDAR sensor data while processing big data [16]. The result is an algorithm that finds obstacles and objects with high accuracy and speed. So, Cluster Homogeneity = 86% and Cluster Completeness = 91%. The authors also investigated the relationship between the selected hyperparameters' values and the algorithm's efficiency.

Articles [17-20] consider the comparison of different parallel programming frameworks, such as MPI, OpenMP, OpenCL, and CUDA. In particular, the article [16] studied parallel programming on computers with distributed and shared memory. The research is conducted using test cases that show the need for different approaches to parallel programming. Test cases are implemented in Chapel and Julia, and MPI and OpenMP are added to C. It is shown that both languages, Chapel and Julia, represent a viable alternative to Fortran and C/C++, supplemented by parallel programming frameworks. With their use, the programmer's efficiency is significantly improved, and the speed of the programs is not considerably affected.

The authors in [21] consider the possibility of autonomous navigation, where the particle filter algorithm was applied to obtain the most accurate position of the car. Here, GPS and Lidar were used for localization. The proposed system provides autonomous control in an unfamiliar environment. It increases localization accuracy by solving the error accumulation problem in an unconstrained environment. The particle filter algorithm showed excellent accuracy, with a trajectory error of about 0.094 cm.

The authors of the article [22] show the use of this algorithm for better space orientation of a robot engaged in household chores. It was equipped with a 2D Lidar and an RGB camera for environmental perception.

In [23], a new method of genetic algorithm parallelization for solving the Traveling Salesman Problem (TSP) using the CUDA algorithm is presented. The solution provides route information, excluding all services autonomous vehicles require in the cloud implementation of intelligent transportation systems.

The authors of the article [24] presented an effective implementation of the extended Kalman filter for the simultaneous localization and mapping algorithm on a multiprocessor architecture. The overall accuracy of the algorithm depends on the number of reference points in the state vector and the matched observations.

After conducting a detailed analysis of the problem considered in our work, it was noted that no one has optimized the Particle Filter algorithm using CUDA technologies.

3. Methods and means of research

In general, the problem of Lidar localization can be described as follows. A car in the room can move in any direction with a specific deviation and take measurements of the surrounding environment. Each subsequent measurement differs in angle from the previous one. It is crucial to determine the Lidar position in the room in a few steps and with optimal accuracy.

Articles [25-298] have shown in detail how this algorithm works, where it can be applied, and how it can be improved by increasing the weight of particles by adding Kalman-filtered global navigation satellite system (GNSS) information [30].

Let's consider the algorithm in more detail. The Lidar successively performs two actions. It scans the space around itself and moves in space on a vector $\{x, y\}$ with a certain error.

It is also important whether the initial position of the Lidar is specified. If so, then the array of all particles can be filled with these position coordinates. If not, we need to generate some given number of particles that will be placed randomly in the room. This is done using a Ray Tracing algorithm. The idea of the algorithm is to count the number of intersections of the room walls with a ray emitted from a point horizontally to the right. If their number is even, then the point does not lie in this space. The point is suitable if the intersection number is odd (see Figure 1).

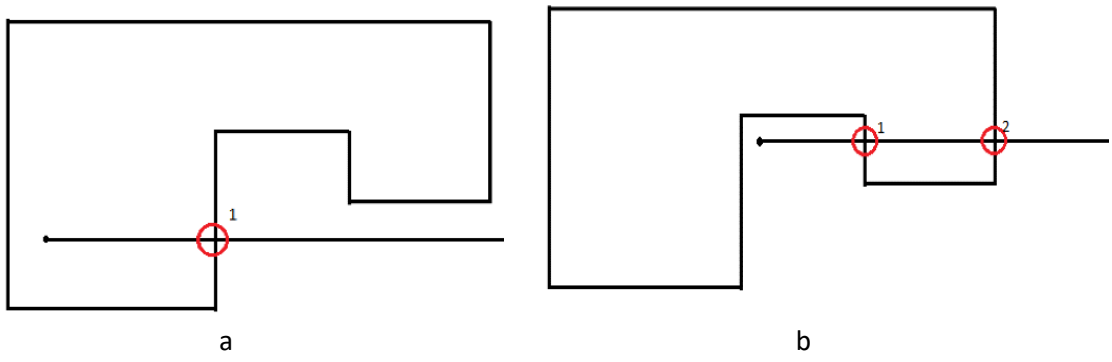


Figure 1: a) – example of a point that lies in the room but the number of intersections is odd; b) – example of a point that does not lie in the room, and the number of intersections is even

First, we set the coordinates of the vertices of our room. After that, depending on whether the initial position is specified, we fill the array with random particles or particles with the initial position coordinates.

The next step will be a Lidar scan of the surrounding space (see Figure 2). The scan takes place k times, starting the movement horizontally and to the right, and each measurement differs from the previous one by an angle that changes by $360/k$ degrees. This way, we get an array with the distances from the Lidar to the room's walls with a given ray launch angle.

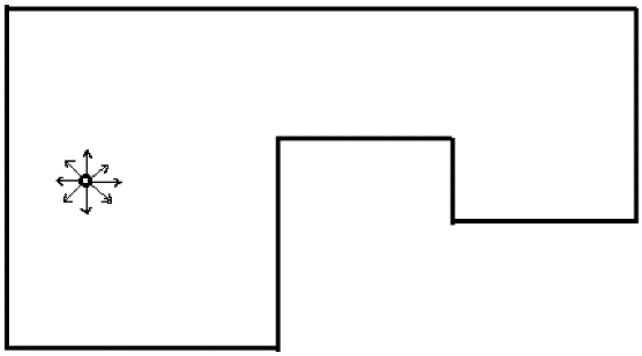


Figure 2: An example of the rays emitted by the Lidar, scanning the space around it

Next, we calculate the weights for each particle. The calculation is as follows:

1. Take the first particle from the array.
2. Calculate the angle relative to the horizontal, the final coordinate of the end of the segment, which starts in the particle coordinates and has the ray length from the array.

$x_1 = x + l\cos\alpha$, $y_1 = y + l\sin\alpha$, where α – ray angle, and l – ray length.

3. Calculate the perpendicular length lowered from the end of the segment to the nearest room wall.
4. Add the square of the length of the perpendicular to the temporary variable.
5. Repeat steps 2-4 for each ray in the array.
6. The weight of the particle is calculated according to the normal distribution formula (1).

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (1)$$

where μ – mathematical expectation, σ – standard deviation that are set by Lidar parameters.

7. Repeat steps 1-6 for each particle.

The next step is to repopulate the particles based on their weights. This procedure is divided into four stages:

The first stage is to add up all the values of particle weights.

The second stage is to fill the new array of weights with values equal to $NewW[i] = \frac{W[i]}{s}$, where s – sum of all values. Thus, we normalize them so that their sum is equal to 1.

The third stage is to fill in an array that has a value $prefW[i] = NewW[i] + prefW[i-1]$. In this way, we will get an array of weight intervals.

The fourth stage is the generation of random value from 0 to 1. Then we need to determine which range in the weight array the number belongs to. The index of that range would correspond to the particle that should be created.

The complexity of the algorithm described above can be calculated as *O(n*(k+r)*q)*, where

- n – the number of Lidar movement iterations;
- k – the number of measurements at every step;
- r – the number of walls in the room;
- q – the number of generated particles.

As a result, it grows very rapidly. One of the methods of optimizing this algorithm can be the parallelization of certain particles. For the most part, the most time is spent on the calculation of weights and repopulation, so these stages of the algorithm are parallelized. Thus, the complexity of the algorithm should be significantly reduced.

To reduce the implementation time and increase the algorithm's efficiency, we will compare the algorithm execution time to a different number of processor threads using OpenMP technology. We will also make time measurements on the graphics processor using CUDA technology [31].

As a result, the #pragma omp parallel method was used. It tells the compiler that the code below uses inter-thread transmission and provides safe thread processing, if threads exchange information. The schedule(type[, chunk]) method was also used, specifying how the cycle iterations are distributed between threads. Synchronization in this case is implicit, and is performed in two situations:

1. At the end of the parallel region. OpenMP relies on fork-join. When the program starts, one thread (main thread) is created. When we create a parallel section using #pragma omp parallel, several threads (fork) are created. These threads will work in parallel and, at the end of the parallel section, will be destroyed (join). So, at the end of the parallel section, we have synchronization and know precisely the status of all threads (they have finished their work).
2. There is an implicit barrier at the end of some OpenMP designs, such as #pragma omp for. Thread can only continue to work once all the threads reach the barrier. It is essential to know exactly what work different threads have done.

For CUDA technology, some weight calculation function is defined as __global__, which indicates that this code passage will be performed on the graphics processor. Additionally, calcweights() method will cause auxiliary functions, which will be defined as __device__. It means that only a graphics processor can cause and perform such functions. Numerous experiments are conducted on a graphics card with four multi-processor blocks that can perform 512 threads. The function for determining the block size is cudaOccupancyMaxPotentialBlockSize(), which in a heuristic manner calculates the block size that reaches maximum occupancy. The thread group is called the CUDA block. Each CUDA block is performed by one multi-processor (MP) and cannot be transferred to other multi-processors in GPU. One multi-processor can perform several simultaneous CUDA blocks, depending on the resources required by CUDA blocks. Each core is performed on one device, and CUDA supports several cores on one device at a time. In this way, we call calcweights() function, passing gridSize and blockSize as parameters. Here blockSize is the number of threads on the block determined by the cudaOccupancyMaxPotentialBlockSize() function. A gridSize is the number of blocks we calculate by the formula (2).

$$gridSize = (N + blockSize - 1)/blockSize, \quad (2)$$

where N – the size of the array (in our case it is the number of particles we will generate).

4. Numerous experiments

This section will feature tables and figures showing the program's time, speed, and efficiency with the different number of threads. The input was taken from Algotester [32]. The program was tested for four different input variations [33], with varying room shapes, the number of walls, iterations of Lidar, and the number of measurements at every step.

3000 (number of particles) was selected for the most accurate localization q. Table 1 presents the execution time of a sequential and proposed parallel algorithm based on OpenMP technology in a variation of the threads number (two, four, and eight) and the values obtained by the acceleration and efficiency of our parallelization without the distribution between threads. k/n/m – the input data described above. t_s – the time of performing a sequential algorithm, t_{p1} – the time of execution of the parallel algorithm based on OpenMP technology, S – acceleration, and E – efficiency. Table 2 presents the same results on the basis of CUDA technology and a graphics processor. Here t_{p2} – the time of the parallel algorithm based on CUDA technology.

Table 1
Results of program execution based on OpenMP technology

k/n/m	t_s , ms	2 threads			4 threads			8 threads		
		t_{p1}, ms	S	E	t_{p1}, ms	S	E	t_{p1}, ms	S	E
36/6/4	273.9	139.4	1.96	0.98	106.3	2.57	0.64	90.8	3.021	0.37

36/15/8	960.6	491.3	1.95	0.97	327.1	2.93	0.74	293.2	3.28	0.40
100/15/8	2102.4	1055.4	1.99	0.99	735.2	2.85	0.71	541.8	3.88	0.48
180/29/27	29414.8	15040.4	1.95	0.97	8231.9	3.57	0.89	5245	5.60	0.70

Analyzing the results of Table 1, we see that when performing an algorithm with two threads, acceleration was obtained close to two, and efficiency also goes to one. Since numerous experiments were conducted on a four-core processor, we have received reliable results. Acceleration with four threads was approximately 3.5, and efficiency – 0.89. When using eight threads, it was possible to improve the acceleration and bring it closer to 5.6. However, the efficiency decreased significantly and was 0.7. All this is because the number of threads in the last two cases exceeds the number of cores in the multi-core computer system. The results can be significantly improved by choosing computers with more cores. Still, the results obtained should have the same trend between the number of cores and the variation of the threads number.

Table 2

Results of program execution based on CUDA technology

k/n/m	t_s, ms	Graphics processor	
		t_{p2}, ms	t_{p2}, ms
36/6/4	273.9	63.1	4.340729
36/15/8	960.6	94.8	10.13291
100/15/8	2102.4	113.2	18.57244
180/29/27	29414.8	1485.6	19.79995

In Table 2, acceleration is close to 20. It is known that when using CUDA technology, the performance is not calculated. Therefore, the proposed parallel algorithm has improved about four times the acceleration based on CUDA technology compared to OpenMP technology. The results of Tables 1 and Table 2 for visual analysis are presented in Figure 1 and diagrams (see Figure 2 and Figure 3).

Therefore, to objectively evaluate the results, you need to consider the computer's capabilities on which the experiments were conducted. In particular, in our case, the program was launched on the computer with the characteristics specified in Table 3.

Table 3

The configuration of the computer on which the numerical experiments were performed

Processor	Intel Core i5-8250
Base frequency of the processo	1.8 GHz
RAM	32 GB
Kernels	4
Logical processors	8
System type	64-bit)
Video card type	Discrete NVIDIA(940MX)

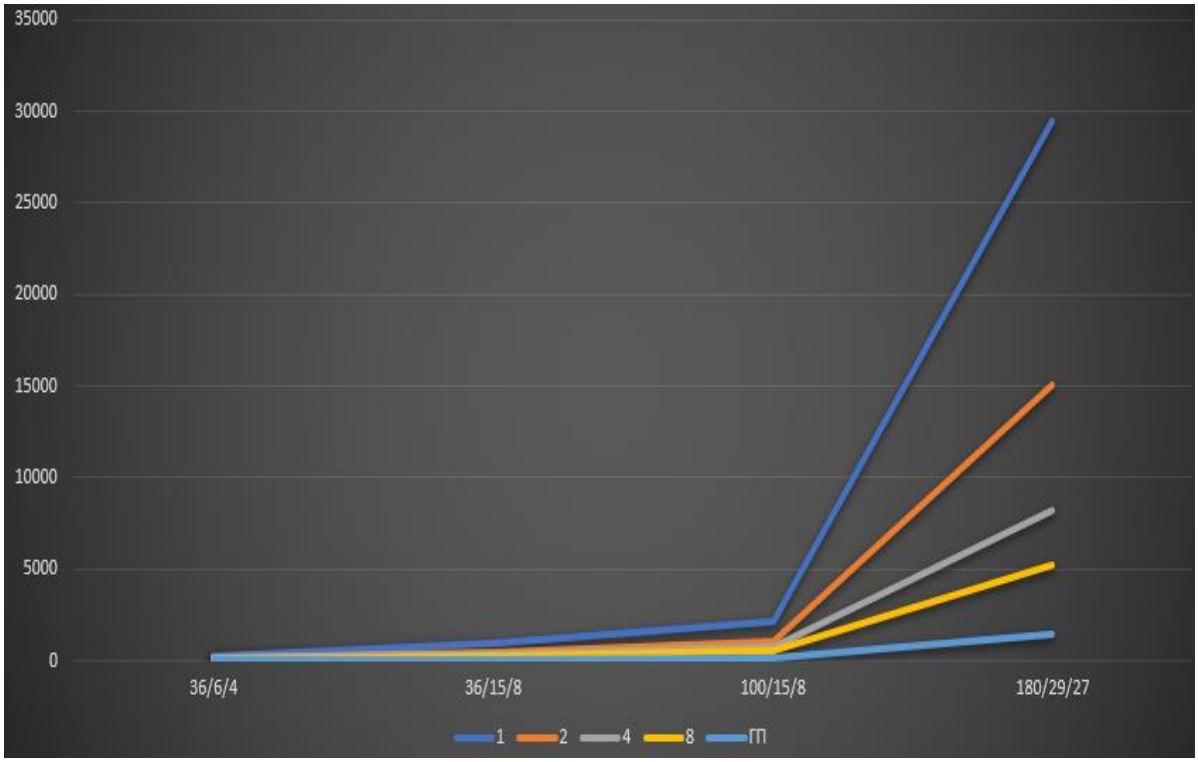


Figure 1: Execution time of the sequential algorithm (1), parallel based on OpenMP technology for different number of threads (2, 4 and 8), and parallel based on CUDA technology (m)

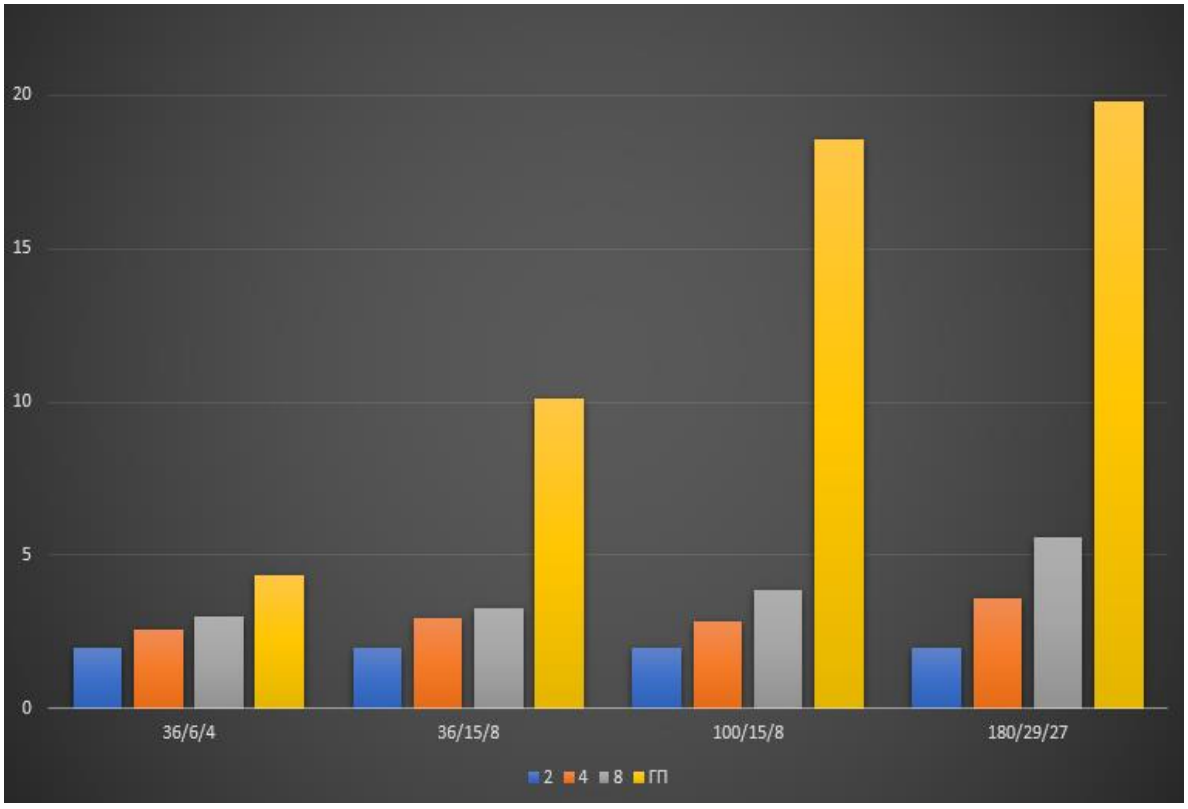


Figure 2: Acceleration obtained based on OpenMP technology for different number of threads (2, 4 and 8) and based on CUDA technology (m)

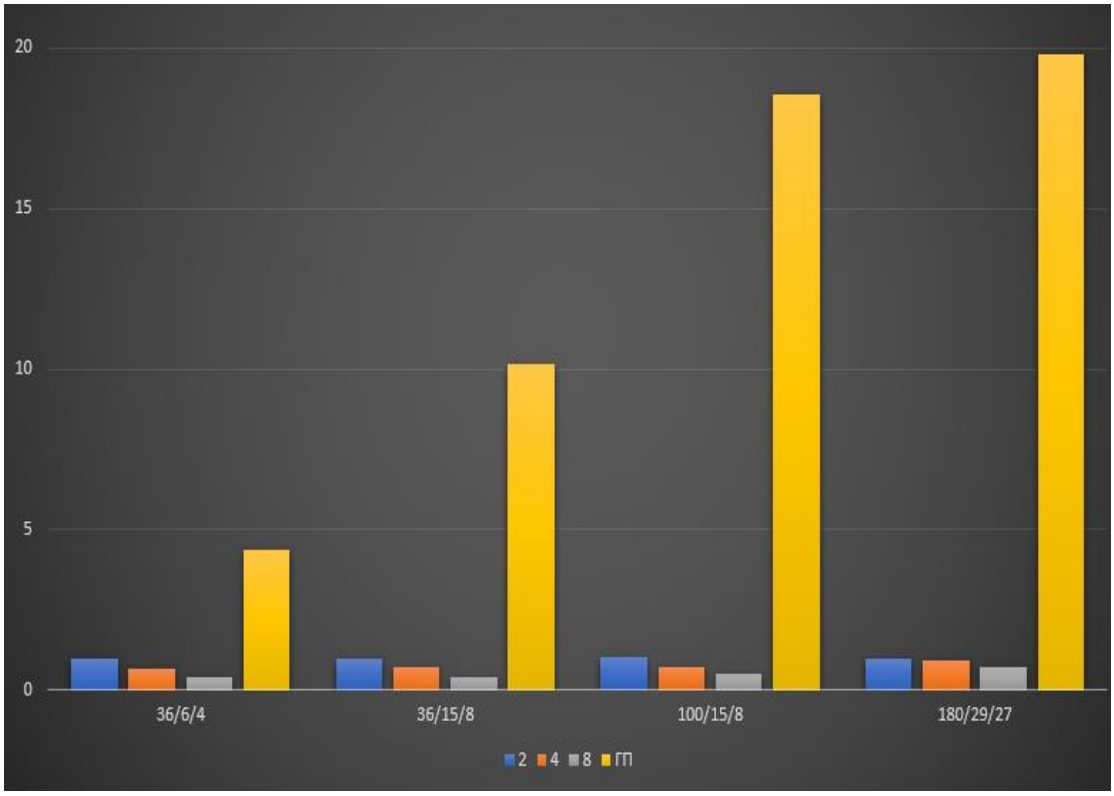


Figure 3: Efficiency obtained based on OpenMP technology for different number of threads (2, 4 and 8) and based on CUDA technology (m)

The OpenMP library was used to implement a parallel algorithm for determining the current Lidar position and CUDA technology to execute a specific part of the code on the graphics processor. After analyzing the results, the following conclusions can be made. The time with the small amount of input data is not significantly different.

Acceleration becomes more noticeable with large input data. Looking at the results, it is noticeable that the time of the sequential algorithm and the time obtained by paralleling the algorithm is exceptionally distinct. As a result, it was achieved an efficiency of ~ 0.99 (when accelerating the algorithm using two threads) and an acceleration of ~ 19.8 (when performing the code on the graphics processor). The error of determining the position was 0.001.

From the results, we can see that the speed of the algorithm parallelized on the GPU is higher than that of the usual sequential one. As the parametric complexity of the problem increased, the difference in execution time grew rapidly, due to the reduction of the total complexity of the algorithm (taking into account the synchronization time) and the parallel use of CUDA microkernels for computational operations, as expected.

5. Conclusions

Determining the current Lidar position is a task that requires taking into account many factors, and, therefore, requires a lot of time to execute the algorithm. In our work, this problem was addressed by parallelizing calculations on a graphics processor using CUDA technology, thanks to which we managed to consistently obtain significantly better execution time indicators (with an increase in the number of computational loads) than in the sequential implementation of the algorithm on a central processor.

So, this paper proposes and investigates the parallel algorithm for determining the current Lidar position using CUDA technologies. The results of using OpenMP technology are also compared. As a result, it was possible to speed up the process about twice times and achieve an efficiency of 0.99. The algorithm is accelerated almost four times, and an efficiency of 0.98 was obtained. The acceleration reached approximately 19.8 by using CUDA technology for parallel programming. Better

results can be achieved in the future based on the proposed parallel algorithm using more powerful computing systems.

Prospects for the further development of this study are the application of the algorithm proposed in the work in the case of 3D Lidar [34, 35].

6. References

[1] M. U. Khan, S. A. A. Zaidi, A. Ishtiaq, S. U. R. Bukhari, S. Samer and A. Farman. "A Comparative Survey of LiDAR-SLAM and LiDAR based Sensor Technologies," 2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC), Karachi, Pakistan, 2021, pp. 1-8, doi: 10.1109/MAJICC53071.2021.9526266.

[2] S. Chen, B. Zhou, C. Jiang, W. Xue, Q. Li. "A LiDAR/Visual SLAM Backend with Loop Closure Detection and Graph Optimization". Remote Sensing. 2021; 13(14):2720. doi:10.3390/rs13142720.

[3] Wolfgang Hess, Damon Kohler, Holger Rapp, Daniel Andor. "Real-Time Loop Closure in 2D LIDAR SLAM." 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1271-1278.

[4] I. Oleksiv, H. Lema, V. Kharchuk, T. Lisovych, O. Dluhopolskyi and T. Dluhopolska, "Identification of Stakeholders Importance for the Company's Social Responsibility using the Analytic Hierarchy Process", 10th International Conference on Advanced Computer Information Technologies (ACIT), 2020, pp.573-576

[5] J. Elfring, E. Torta, R. van de Molengraft. "Particle Filters: A Hands-On Tutorial". Sensors, 2021, 21, 438. doi:10.3390/s21020438.

[6] Jin Yanga , Xuerong Cuib, Juan Lia, Shibao Lib, Jianhang Liua, Haihua Chena. "Particle filter algorithm optimized by genetic algorithm combined with particle swarm optimization." Procedia Computer Science, 187, 2021, pp. 206–211.

[7] L. Mochurad, H. Lema, R. Vilhutska. "Parallel Algorithms Assessment Usage of Image's Segmentation Quality in Medicine". CEUR Workshop Proceedings, 2022, 3171, pp. 1509–1519.

[8] R. S. Dehal, C. Munjal, A. A. Ansari and A. S. Kushwaha, "GPU Computing Revolution: CUDA," 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Greater Noida, India, 2018, pp. 197-201, doi: 10.1109/ICACCCN.2018.8748495.

[9] G. Xie. "A novel Monte Carlo simulation procedure for modelling COVID-19 spread over time". Sci Rep 10, 13120, 2020. doi:10.1038/s41598-020-70091-1.

[10] P. Andreo. "Monte Carlo simulations in radiotherapy dosimetry". Radiat Oncol 13, 121, 2018. doi:10.1186/s13014-018-1065-3.

[11] M. B. Alatise, G. P. Hancke. "A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods", in *IEEE Access, 2020, Volume 8, pp. 39830*-39846, doi: 10.1109/ACCESS.2020.2975643.

[12] K. Groves, E. Hernandez, A. West, T. Wright, B. Lennox. "Robotic Exploration of an Unknown Nuclear Environment Using Radiation Informed Autonomous Navigation". Robotics, 2021, 10(2), 78, doi:10.3390/robotics10020078.

[13] C. Pang, X. Zhong, H. Hu, J. Tian, X. Peng, J. Zeng. "Adaptive Obstacle Detection for Mobile Robots in Urban Environments Using Downward-Looking 2D LiDAR", Sensors, 2018, 18, 1749, doi:10.3390/s18061749.

[14] L. Mochurad, Y. Hladun, R. Tkachenko. An Obstacle-Finding Approach for Autonomous Mobile Robots Using 2D LiDAR Data. Big Data Cogn. Comput. 2023, 7, 43, doi:10.3390/ bdcc7010043.

[15] L. Mochurad, N. Kryvinska. "Parallelization of Finding the Current Coordinates of the Lidar Based on the Genetic Algorithm and OpenMP Technology". Symmetry 2021, 13, 666. doi:10.3390/sym13040666.

[16] L. Mochurad, Y. Hladun, R. Tkachenko R. “An Obstacle-Finding Approach for Autonomous Mobile Robots Using 2D LiDAR Data”. Big Data and Cognitive Computing. 2023; 7(1):43, doi:10.3390/bdcc7010043.
[17] R. Novosel, B. Slivnik. “Beyond Classical Parallel Programming Frameworks: Chapel vs Julia”. 8th Symposium on Languages, Applications and Technologies (SLATE 2019), 2019, doi: 10.4230/OASIcs.SLATE.2019.12.
[18] A. Shterenlikht, L. Cebamanos. “MPI vs Fortran coarrays beyond 100k cores: 3D cellular automata”. Parallel Computing, Volume 84, May 2019, Pages 37-49, doi:10.1016/j.parco.2019.03.002.
[19] Baldomero Imbernón, Javier Prades, Domingo Giménez, José M. Cecilia, Federico Silla. Enhancing large-scale docking simulation on heterogeneous systems: An MPI vs rCUDA study, Future Generation Computer Systems, Volume 79, Part 1, 2018, 26-37, doi.org/10.1016/j.future.2017.08.050.
[20] M. Breyer, A. Van Craen, D. Pflüger. “A Comparison of SYCL, OpenCL, CUDA, and OpenMP for Massively Parallel Support Vector Machine Classification on Multi-Vendor Hardware”. In International Workshop on OpenCL (IWOCL'22). Association for Computing Machinery, New York, NY, USA, 2022, Article 2, 1–12. doi: 10.1145/3529538.3529980.
[21] J. Patoliya, H. Mewada, M. Hassaballah, M.A. Khan, S. Kadry. “A robust autonomous navigation and mapping system based on GPS and LiDAR data for unconstraint environment”, Earth Science Informatics, vol. 15, no. 4, pp. 2703–2715, 2022. doi:10.1007/s12145-022-00791-x.
[22] L. Zhi-jie, C. Ming-yu, Z. Zhi-ming and Y. You-ling, "Design and Implementation of Home Service Robot," 2020 Chinese Automation Congress (CAC), Shanghai, China, 2020, pp. 3541-3546, doi: 10.1109/CAC51589.2020.9327151.
[23] M. Abbasi, M. Rafiee, M. R. Khosravi et al. “An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems”. J Cloud Comp 9, 6, 2020, doi:10.1186/s13677-020-0157-4.
[24] N.B. Amor, M. Baklouti, K. Barhoumi, M. Jallouli. “Efficient embedded software implementation of a low cost robot localization system”. In Proceedings of the 2019 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS), Gammarth, Tunisia, 28 April–1 May 2019; pp. 1–5.
[25] KyungJae Ahn, Yeonsik Kang. "A Particle Filter Localization Method Using 2D Laser Sensor Measurements and Road Features for Autonomous Vehicle", Journal of Advanced Transportation, vol. 2019, Article ID 3680181, 11 pages, 2019. doi:10.1155/2019/3680181.
[26] D. Talwar and S. Jung, "Particle Filter-based Localization of a Mobile Robot by Using a Single Lidar Sensor under SLAM in ROS Environment," 2019 19th International Conference on Control, Automation and Systems (ICCAS), Jeju, Korea (South), 2019, pp. 1112-1115, doi: 10.23919/ICCAS47443.2019.8971555.
[27] M.Á. de Miguel, F. García, J.M. Armingol. Improved LiDAR Probabilistic Localization for Autonomous Vehicles Using GNSS. Sensors 2020, 20, 3145. doi:10.3390/s20113145.
[28] Li Lin, Rong Wei, Su Fei, Xing Xiaoyu. “An improved detection method based on morphology and profile analysis for bridge extraction from Lidar”. Optics & Laser Technology, Volume 121, January 2020, 105790. doi: 10.1016/j.optlastec.2019.105790.
[29] Xieyuanli Chen, Ignacio Vizzo, Thomas Labe, Jens Behley, Cyrill Stachniss. “Range Image-based LiDAR Localization for Autonomous Vehicles”. Computer Science, Robotics, 2021, doi.org/10.48550/arXiv.2105.12121.
[30] Y. Kim and H. Bang, “Introduction to Kalman Filter and Its Applications,” Introduction and Implementations of the Kalman Filter, May 2019, doi: 10.5772/intechopen.80600.
[31] Martin Heller. “What is CUDA? Parallel programming for GPUs”, 2018. https:// www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html.
[32] “Lidar robot localization 1”. https://algotester.com/uk/ContestProblem/DisplayWithEditor/ 83112 (accessed March 12, 2023)
[33] “Data Lidar”. https://www.researchgate.net/publications/create? publicationType=dataset. (accessed March 12, 2023).

[34] Y. Liu, C. Wang, H. Wu, Y. Wei, M. Ren, C. Zhao. Improved LiDAR Localization Method for Mobile Robots Based on Multi-Sensing. Remote Sens. 2022, 14, 6133, doi:10.3390/rs14236133.

[35] J. Gómez, O. Aycard, J. Baber. Efficient Detection and Tracking of Human Using 3D LiDAR Sensor. Sensors (Basel). 2023 May 12;23(10):4720. doi: 10.3390/s23104720. PMCID: PMC10222621.