ISSN 2308-5916. Mathematical and computer modelling.
Series: Technical sciences. 2026. Issue 29. P. 39-48.

UDC 004.8:004.93:004.42
DOI: 10.32626/2308-5916.2026-29.39-48

Hentosh L.

ORCID: 0000-0002-4957-1512,
Ph.D. in Engineering, Lviv Polytechnic
National University, Lviv, Ukraine,
E-mail: lesia.i.mochurad@lpnu.ua

A PARALLEL TRAJECTORY PLANNING METHOD IN THE STATE SPACE WITH EXACT SORTING-BASED NEIGHBOR SEARCH

The trajectory planning problem in a state space with obstacles and feasibility constraints has been studied. The rapidly exploring random tree method has been considered, in which each iteration has performed random sampling, selection of the nearest tree vertex, construction of a candidate extension, feasibility checking, and insertion of a new vertex into the tree. In parallel implementations on a graphics processing unit, nearest-vertex selection has become the latency-limiting stage because exhaustive distance evaluation has been followed by a global minimum reduction that has required multi-stage synchronization and frequent access to global memory, resulting in a large synchronization-induced hidden time constant. An improved parallel planning method has been proposed that has combined batch-based macrosteps for tree expansion with replacement of the standard nearest-vertex kernel by an exact sorting-based nearest-neighbor search optimized for a graphics processing unit. The batch organization has overlapped independent candidate generation and feasibility checks and has applied a coordinated tree update after all subtasks have completed, while the sorting-based search has ordered vertices by projections onto a leading direction and has performed guaranteed candidate-set shrinking so that exact distance evaluation has been executed only for a small subset. Numerical experiments on the nearest-vertex stage have confirmed a pronounced reduction of the normalized hidden constant, and an end-to-end analysis based on Amdahl's law has indicated increasing overall speedup as nearest-vertex selection has become dominant, particularly for large trees and higher-dimensional representations. Additional end-to-end tests in two-dimensional environments of size 100×100 and 200×200 with varying obstacle counts have demonstrated consistent runtime improvements, with the relative benefit tending to grow for more cluttered scenes.

Стаття надійшла до редакції: 26.02.2026
Рекомендовано до друку: 07.04.2026
Оприлюднено (online): 15.05.2026

Key words: *trajectory planning, rapidly exploring random tree, parallel computing, graphics processor, nearest-node selection, sorting-based exact neighbor search, hidden constant, scalability.*

Introduction. Trajectory planning problems in obstacle-constrained state spaces are fundamental for robotics, autonomous navigation, control of unmanned platforms, and other intelligent systems where a solution must be obtained under strict time constraints [1-3]. One of the most widely used trajectory planning approaches is based on constructing a rapidly exploring random tree, in which the space is explored by iteratively expanding the set of reachable vertices [4-5].

At the same time, in complex three-dimensional environments or in high-dimensional state spaces, the cost of finding the tree node nearest to a random sample increases sharply [6-7]. In parallel GPU (Graphics Processing Unit) implementations, this operation includes not only massively parallel distance computations, but also a coordinated reduction to the minimum, which introduces significant synchronization overhead and a large hidden time constant [8-9]. As a result, practical scalability degrades, and the parallel speedup saturates as the tree grows and the environment becomes more complex [10].

This paper proposes and evaluates an improved parallel trajectory planning method, in which the algorithmic latency is reduced through two complementary solutions: a batch-based organization of tree expansion and the replacement of the nearest-node search with an ordered exact neighbor search that guarantees a reduced candidate set and is implemented on a GPU.

Formulation of the problem. Let X be the state space of the mobile object, $X_{free} \subset X$ be the subset of admissible states with no collisions with obstacles, and O be the set of obstacles. The initial state $x_{init} \in X_{free}$ and the goal state $x_{goal} \in X_{free}$ are given. The task is to find a continuous path

$$\rho : [0,1] \to X_{free}, \tag{1}$$

such that $\rho(0) = x_{init}$, $\rho(1) = x_{goal}$, and the path does not intersect any obstacles.

Basic structure of the rapidly exploring random tree and identification of the dominant stage. Let, at step k, the tree be defined as

$$T_k = (V_k, E_k),\ V_k = \{x_i\}_{i=1}^{N_k} \subset X_{free},\ N_k = |V_k|. \tag{2}$$

One iteration of tree construction is given by a sequence of interdependent operations: generating a random sample $x_{rand}^{(k)} \in X_{free}$; finding the nearest-node

$$x_{near}^{(k)} = \arg\min_{x_i \in V_k} x_i - x_{rand\,2}^{(k)}; \quad (3)$$

constructing a candidate node $x_{new}^{(k)}$ as a directed step from $x_{near}^{(k)}$ toward $x_{rand}^{(k)}$; checking the feasibility of the segment $\left[x_{near}^{(k)}, x_{new}^{(k)}\right] \subset X_{free}$ with respect to the obstacle set O; and consistently attaching $x_{new}^{(k)}$ to the tree by updating V_{k+1} and E_{k+1}.

Since $x_{new}^{(k)}$ is defined based on $x_{near}^{(k)}$, and the feasibility check is performed for the edge $\left(x_{near}^{(k)}, x_{new}^{(k)}\right)$, operation (3) becomes the latency limiting component of each iteration. Therefore, the key factor in reducing the total solution construction time is the implementation mechanism of operation (3).

We represent the execution time of operation (3) as

$$\tau_{near}\left(N_k, d\right) = \tau_{dist}\left(N_k, d\right) + \tau_{min}\left(N_k\right), \quad (4)$$

where τ_{dist} is the time to compute the distances $x_i - x_{rand}$ for $i = 1,\ldots,N_k$, and τ_{min} is the time to find the minimum among N_k values. On a GPU, τ_{dist} is massively parallel, whereas $\tau_{min}\left(N_k\right)$ is implemented as a multi-stage reduction of intermediate minima and involves significant synchronization overhead.

Improved parallel trajectory planning method. The proposed method combines two components: an algorithmic component and a local component. In the algorithmic component, a batch-based organization of tree expansion is introduced, where weakly dependent subtasks for generating candidate extensions and performing feasibility checks are executed with temporal overlap. This shortens the dominant part of the iterative process and reduces the algorithmic component of latency. In the local component, the limiting nearest-node operation (3) is implemented via an ordered exact neighbor search based on projection ordering and a guaranteed reduction of the candidate set. As a result, the global minimum reduction is performed not over N_k elements, but over a candidate subset $m_k \ll N_k$, which reduces the synchronization induced hidden constant.

Let us introduce a macro-step s, in which a batch of $B \geq 1$ samples is processed:

$$\left\{x_{rand}^{(s,\,b)}\right\}_{b=1}^{B} \subset X_{free}. \quad (5)$$

For a fixed tree $T^{(s)} = \left(V^{(s)}, E^{(s)}\right)$, all B expansion subtasks are formed independently up to the attachment stage. For each b, $x_{near}^{(s,\,b)}$ is de-

termined according to (3); a candidate $x_{new}^{(s,\,b)}$ is constructed; and the feasibility of the edge $(x_{near}^{(s,\,b)}, x_{new}^{(s,\,b)})$ is checked. Next, a coordinated attachment to the tree is performed, during which a subset of feasible candidates is selected such that the coordination rules and access constraints to shared data structures are not violated. The duration of one macro-step is expressed as

$$T_{step}^{(s)} \approx \max_{b=1..B}\left(\tau_{gen}^{(s,\,b)} + \tau_{near}^{(s,\,b)} + \tau_{steer}^{(s,\,b)} + \tau_{col}^{(s,\,b)}\right) + T_{sync}^{(s)}, \quad (6)$$

where s is the macro-step index; B is the batch size; $b = 1,...,B$ is the batch element index; $\tau_{gen}^{(s,\,b)}$ is the time to generate the sample $x_{rand}^{(s,\,b)}$; $\tau_{near}^{(s,\,b)}$ is the time to determine $\tau_{near}^{(s,\,b)}$; $\tau_{steer}^{(s,\,b)}$ is the time to construct the candidate node $\tau_{new}^{(s,\,b)}$; $\tau_{col}^{(s,\,b)}$ is the feasibility checking time; and $T_{sync}^{(s)}$ denotes coordination overhead, including synchronization of subtask completion and controlled updates of the tree structure.

Let the sequential iteration have the average duration

$$\tau_{it}(N_k, d) = \tau_{gen}(k) + \tau_{near}(N_k, d) + \tau_{steer}(k) + \tau_{col}(k) + \tau_{add}(k), \quad (7)$$

where τ_{add} denotes the overhead of updating the structures V_{k+1}, E_{k+1}. The sequential time to grow the tree to size N is estimated as

$$T_{alg}^{seq}(N) \approx \sum_{k=1}^{N-1} \tau_{it}(N_k, d). \quad (8)$$

In the batch organization, during one macro-step s we form B candidates, but only $0 \le B_{acc}^{(s)} \le B$. Let us introduce the average acceptance ratio

$$\eta = E\left[\frac{B_{acc}^{(s)}}{B}\right] \in (0,1]. \quad (9)$$

Then, to grow the tree to size N, approximately $S_N \approx (N-1)/(\eta B)$ macro-steps are required, and the total batch construction time is estimated as

$$T_{alg}^{pack}(N) \approx \sum_{s=1}^{S_N} T_{step}^{(s)}. \quad (10)$$

Using the average $T_{step}^{(s)} \approx \bar{T}_{step}$, we obtain

$$T_{alg}^{pack}(N) \approx \frac{N-1}{\eta B}\bar{T}_{step}. \quad (11)$$

Hence, the theoretical speedup of the batch organization relative to the sequential one is estimated as

$$S_{pack}(N) = \frac{T_{alg}^{seq}(N)}{T_{alg}^{pack}(N)} \approx \frac{\eta B \bar{\tau}_{it}}{\bar{T}_{step}}, \quad (12)$$

where $\bar{\tau}_{it} = \mathrm{E}\left[\tau_{it}(N_k, d)\right]$ is the average duration of one sequential iteration over $N_k \in [1, N]$ for a fixed dimensionality d.

Since the nearest-node subtask is executed for each batch element and appears inside the $\max_{b=1..B}$, operator in (6), even under ideal overlap of the other subtasks the following lower bound holds:

$$\bar{T}_{step} \geq \mathrm{E}\left[\tau_{near}(N_k, d)\right] + \mathrm{E}\left[T_{sync}^{(s)}\right], \quad (13)$$

which implies a limitation on the growth of $S_{pack}(N)$ as N_k becomes dominant $\tau_{near}(N_k, d)$.

Ordered exact search as a local kernel and reduction of the hidden constant. Assume that the GPU implementation of (3) in the baseline case relies on a full distance scan followed by a coordinated reduction to the minimum, whose synchronization-driven time can be described parametrically as

$$\tau_{min}(N_k) \approx k_{min} L \log_2 N_k, \quad (14)$$

where L is the characteristic latency of one coordination step, and k_{min} is a hidden constant that summarizes the costs of barrier synchronizations, exchange of intermediate minima, and contention for global memory access.

In the proposed method, the full scan and reduction over N_k elements are replaced by ordering via projections with a guaranteed narrowing of the candidate set to $m_k \ll N_k$, after which the exact minimum is computed only over the candidates. Then

$$\tau_{min}(m_k) \approx k_{min} L \log_2 m_k, \; m_k \ll N_k, \quad (15)$$

which reduces the synchronization contribution in (4). In addition, the key global component is shifted to the ordering stage, which has more regular memory access patterns and typically a smaller and more stable hidden constant than a barrier based reduction to the minimum.

Modeling and results. The numerical experiments are aimed at verifying the key improvement of the proposed method, namely replacing the standard nearest-node procedure in the rapidly exploring random tree with an ordered exact neighbor search with a guaranteed reduction of the candidate set. Since, in GPU-parallel implementations, the nearest-node operation includes a synchronization-heavy reduction-to-minimum stage after the massive distance computation, the quantitative analysis is performed at the level of the local nearest-neighbor search operation. This approach

makes it possible to isolate the effect of the minimum-selection mechanism and directly assess the reduction of the hidden constant formed by barrier synchronizations and global memory accesses [11-12]. In addition, a model-based estimate of the expected overall planning speedup is provided as a function of the time fraction spent on the nearest-node operation in the baseline algorithm.

To quantitatively illustrate the reduction of the synchronization-induced hidden constant, experiments on the local nearest-neighbor search were conducted in two variants:

1. Full distance scan followed by reduction to the minimum;
2. Ordered exact neighbor search based on ordering and guaranteed narrowing of the candidate set.

The measured processing time per query τ was obtained using the protocol and hardware configuration described in [5]: an AMD Ryzen 3 1200 CPU, an NVIDIA GeForce GTX 1050 Ti GPU, and double-precision arithmetic. Scalable databases of size $N \in \{13233, 26466, 52932, 105864\}$ were considered, generated by duplicating vectors with added small Gaussian noise, as well as two state-space dimensions $d = 512$ and $d = 4096$. For each N and d, the average per-query processing time was recorded for both implementation variants.

As a normalized indicator of the hidden constant, we use

$$\tilde{k}(N) = \frac{\tau(N)}{\log_2 N}, \tag{16}$$

which characterizes the effective time contribution per logarithmic coordination level, accounting for hardware overhead, and enables comparison across different N.

Tables 1 and 2 report the values of τ and the corresponding estimates of $\tilde{k}(N)$ for $d = 512$ and $d = 4096$, respectively, as well as the ratio $\tilde{k}_{bf} / \tilde{k}_{SNN}$, which is interpreted as a quantitative measure of the reduction of the hidden constant when replacing the brute-force mechanism with the ordered exact search.

Table 1

Estimated normalized hidden constant for $d = 512$

N	τ_{SNN}, c	τ_{bf}, c	$\tilde{k}_{SNN}$, c	$\tilde{k}_{bf}$, c	$\tilde{k}_{bf} / \tilde{k}_{SNN}$
13233	0.002000	0.010980	0.000146	0.000802	5.4900000
26466	0.003000	0.019010	0.000204	0.001294	6.336667
52932	0.003990	0.030000	0.000254	0.001912	7.518797
105864	0.006000	0.051010	0.000359	0.003055	8.501667

Table 2

Estimated normalized hidden constant for $d = 4096$

N	τ_{SNN}, c	τ_{bf}, c	$\tilde{k}_{SNN}$, c	$\tilde{k}_{bf}$, c	$\tilde{k}_{bf} / \tilde{k}_{SNN}$
13233	0.005010	0.062010	0.000366	0.004529	12.377246
26466	0.009000	0.130010	0.000613	0.008849	14.445556
52932	0.018990	0.244990	0.001210	0.015613	12.901001
105864	0.024980	0.389400	0.001497	0.023329	15.588471

In Tables 1 and 2, τ_{bf} denotes the query processing time for the brute-force distance scan followed by reduction to the minimum; τ_{SNN} denotes the query processing time for the ordered exact search; N is the database size; and d is the dimensionality of the vector representation.

Tables 1 and 2 confirm that the normalized indicator $\tilde{k}(N)$ for the sorting-based nearest-neighbor (SNN) search variant is significantly smaller than for the brute-force variant. For the high dimensional case d = 4096 the ratio $\tilde{k}_{bf} / \tilde{k}_{SNN}$ is approximately 12 to 16, and for $d = 512$ it is approximately 5 to 9. This is consistent with the fact that SNN replaces the global minimum reduction over a large set of size N with ordering and exact selection over a narrowed candidate set, thereby reducing synchronization-induced overhead.

Since the nearest-node operation (3) is repeated at each iteration of tree construction, the expected end-to-end speedup of planning due to replacing brute-force search with SNN is determined by the fraction of iteration time spent on this operation. Let $f \in (0,1)$ be the fraction of iteration time spent on nearest-node search, and let $g > 1$ be the speedup of this component when switching to SNN. Then, the expected overall planning speedup can be estimated by Amdahl's law [13-14]:

$$S_{total} = \frac{1}{(1-f) + \frac{f}{g}}. \tag{17}$$

Table 3 reports the values of S_{total} for two representative levels of local-kernel speedup g, consistent with the ratios $\tilde{k}_{bf} / \tilde{k}_{SNN}$ in Tables 1 and 2.

Table 3

Expected overall planning speedup as a function of f *and* g

f	S_{total} for $g = 15.6$	S_{total} for $g = 8.5$
0.300	1.390	1.360
0.500	1.879	1.790
0.700	2.899	2.615
0.850	4.889	4.000

Table 3 indicates that even at a moderate parallelizable fraction $f = 0.5$, the expected speedup is about 1.8-1.9. When the nearest-node search dominates ($f \geq 0.7$), the predicted end-to-end speedup becomes multiple-fold. This result provides a formal justification for why reducing the synchronization-induced hidden time constant in the iteration's bottleneck is essential in complex environments, for large trees, and in high-dimensional state spaces.

Conclusions. This paper develops and substantiates an improved parallel trajectory planning method based on a rapidly exploring random tree, where the key improvement is replacing the standard nearest-node procedure with an ordered exact neighbor search designed for GPU execution. It is quantitatively confirmed that this replacement of the nearest-node kernel reduces the synchronization-induced hidden time constant: for high dimensionality, the normalized indicator decreases by approximately 12-16 times, and for lower dimensionality by approximately 5-9 times. This is consistent with narrowing the candidate set and shifting from a global minimum reduction to ordering and selection over a small set. Model-based estimates of the overall impact show that, for a moderate fraction of time spent on nearest-node search, the expected speedup of the entire planning process is about 1.8-1.9 times, and when this stage dominates it can increase to 2.6-4.9 times depending on environment complexity and tree size. Future work will focus on profiling the contribution of the nearest-node stage for different classes of environments, extending the method to three-dimensional scenes, and adaptively selecting the expansion batch size based on the candidate rejection rate and obstacle structure.

References:

1. Karaman S., Frazzoli E. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*. 2011. Vol. 30. № 7. P. 846-894. DOI: 10.1177/0278364911406761.
2. Decentralized Drone Swarm Coordination Methods Using Reinforcement Learning and Parallel Computing for Optimized Management Systems. *IJIES*. 2025. Vol. 18. № 5. P. 83-99. DOI: 10.22266/ijies2025.0630.07.
3. Mochurad L. Implementation and analysis of a parallel kalman filter algorithm for lidar localization based on CUDA technology. *Front. Robot. AI*. 2024. Vol. 11. P. 1341689. DOI: 10.3389/frobt.2024.1341689.
4. Kuffner J. J., LaValle S. M. RRT-connect: An efficient approach to single-query path planning. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. San Francisco, CA, USA: IEEE, 2000. P. 995-1001. DOI: 10.1109/ROBOT.2000.844730.
5. Mochurad L., Davidekova M., Mitoulis S.-A. Parallel rapidly exploring random tree method for unmanned aerial vehicles autopilot development using

graphics processing unit processing. *IJ-AI.* 2025. Vol. 14. № 1. P. 712. DOI: 10.11591/ijai.v14.i1.pp712-723.
6. Otte M., Frazzoli E. RRTX : Asymptotically optimal single-query sampling-based motion planning with quick replanning, *The International Journal of Robotics Research.* 2016. Vol. 35. № 7. P. 797-822. DOI: 10.1177/0278364915594679.
7. Ichnowski J., Alterovitz R. Scalable Multicore Motion Planning Using Lock-Free Concurrency. *IEEE Trans. Robot*. 2014. Vol. 30. № 5. P. 1123-1136. DOI: 10.1109/TRO.2014.2331091.
8. Wang K., Fussell D., Lin C. Fast Fine-Grained Global Synchronization on GPUs. *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, Providence RI USA: ACM, Apr.* 2019. P. 793-806. DOI: 10.1145/3297858.3304055.
9. Li Y., Schwiebert L., Hailat E., Mick J., Potoff J. Improving performance of GPU code using novel features of the NVIDIA kepler architecture. *Concurrency and Computation.* 2016. Vol. 28. № 13. P. 3586-3605. DOI: 10.1002/cpe.3744.
10. Pérez-Hurtado I., Martínez-del-Amor M. Á., Zhang G., Neri F., Pérez-Jiménez M. J. *A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. Integrated Computer-Aided Engineering.* 2020. Vol. 27. № 2. P. 121-138. DOI: 10.3233/ICA-190616.
11. Barbero S., Bellini E., Sanna C., Verbel J. Practical complexities of probabilistic algorithms for solving Boolean polynomial systems. *Discrete Applied Mathematics.* 2022. Vol. 309. P. 13-31. DOI: 10.1016/j.dam.2021.11.014.
12. Kneis J., Langer A. A Practical Approach to Courcelle's Theorem. *Electronic Notes in Theoretical Computer Science*. 2009. Vol. 251. P. 65-81. DOI: 10.1016/j.entcs.2009.08.028.
13. Al-hayanni M. A. N., Xia F., Rafiev A., Romanovsky A., Shafik R., Yakovlev A. Amdahl's law in the context of heterogeneous many-core systems – a survey. *IET Computers & Digital Tech.* 2020. Vol. 14. № 4. P. 133-148. DOI: 10.1049/iet-cdt.2018.5220.
14. Hill M. D., Marty M. R. Amdahl's Law in the Multicore Era. *Computer*. 2008. Vol. 41. № 7. P. 33-38. DOI: 10.1109/MC.2008.209.

ПАРАЛЕЛЬНИЙ МЕТОД ПЛАНУВАННЯ ТРАЄКТОРІЙ У ПРОСТОРІ СТАНІВ ІЗ ВПОРЯДКОВАНИМ ТОЧНИМ ПОШУКОМ СУСІДІВ

Досліджено задачу планування траєкторій у просторі станів з перешкодами та обмеженнями на допустимість переходів. Розглянуто метод дерева швидкого випадкового пошуку, у якому на кожній ітерації виконуються формування випадкового зразка, вибір найближчого вузла дерева, побудова кандидатного розширення, перевірка допустимості та приєднання нового вузла до дерева. Показано, що у паралельних реалізаціях із використанням графічного процесора операція вибору найближчого вузла стає обмежувальною за затримкою, оскільки після повного перебору відстаней необхідне глобальне зведення до мінімуму з багатоетапною синхронізацією та частими зверненнями

до глобальної пам’яті, що формує значну синхронізаційно зумовлену приховану константу часу. Запропоновано удосконалений паралельний метод планування, який поєднує пакетну організацію макрокроків розширення дерева із заміною стандартного ядра визначення найближчого вузла на впорядкований точний пошук найближчих сусідів, оптимізований для графічного процесора. Пакетування забезпечує накладання незалежних операцій формування кандидатів і перевірок допустимості та узгоджене оновлення дерева після завершення всіх підзадач, тоді як впорядкований пошук виконує впорядкування вузлів за проєкціями на провідний напрямок і гарантоване звуження множини кандидатів, тому точний мінімум відстані обчислюється лише для невеликої підмножини. Чисельні експерименти на етапі визначення найближчого вузла підтвердили істотне зменшення нормованої прихованої константи, а аналітична оцінка наскрізного прискорення показала зростання виграшу за умов домінування цієї операції, що є характерним для великих дерев і високорозмірних подань. Додаткові наскрізні тести у двовимірних середовищах розміром 100×100 і 200×200 з різною кількістю перешкод продемонстрували стабільне скорочення часу виконання, причому відносний виграш, як правило, є більшим для більш захаращених конфігурацій.

Ключові слова: *планування траєкторій, дерево швидкого випадкового пошуку, паралельні обчислення, графічний процесор, визначення найближчого вузла, впорядкований точний пошук сусідів, прихована константа, масштабованість.*