pAORRTC: GPU-Parallel Almost-Surely Asymptotically Optimal Planning

Chih H. Huang[1], Brian Plancher[2], and Zachary Kingston[3]

***Abstract*— Finding globally optimal paths remains a fundamental challenge in motion planning. Almost-surely asymptotically optimal (*a.s.a.o*) planners provide probabilistic guarantees of convergence to globally optimal solutions, but their computational cost limits their real-time applicability. Recent efforts have explored CPU-based parallelism for the asymptotically optimal AO-x meta-algorithm to achieve both probabilistic convergence guarantees and strong computational performance. However, little has been done to utilize the massive parallelism available on modern GPUs. In this work, we propose pAORRTC, a GPU-based *a.s.a.o* planner based on RRT-Connect and the AO-x meta-algorithm. We redesigned pAORRTC for GPU-acceleration throughout, with SIMT-parallelism for running hundreds of planning iterations simultaneously along with parallel informed sampling, collision checking, and path simplification. We evaluate pAORRTC on the MotionBenchMaker dataset using the 8-DoF Fetch against state-of-the-art *a.s.a.o* planners. pAORRTC was able to find initial solutions in milliseconds for challenging manipulation tasks and optimize to low-cost paths faster.**

I. Introduction

Motion planning aims to find low-cost, collision-free paths through a robot's configuration space [1–3]. The problem can then be separated into two stages: constructing the initial path and optimizing for a better solution. While sampling-based motion planners (SBMPs) such as RRT and its bidirectional variant RRT-Connect can efficiently find initial paths [4, 5], the subsequent optimization of the resulting path cost toward the global optimum is generally not feasible in real-time.

An *a.s.a.o* planner address this optimization problem by continuously improving solution quality with the probabilistic guarantee of finding the global optimum under infinite samples. Anytime *a.s.a.o* planners, e.g., RRT* and Batch Informed Trees [6, 7], iteratively refine solutions until a user-set termination condition is met. However, the computational overhead of this iterative optimization process remains a key barrier to achieving real-time, high-quality solutions in high-dimensional configuration spaces.

Recent efforts have been made to accelerate *a.s.a.o* planners by utilizing the asymptotically optimal x (AO-x) meta-algorithm [8]. AO-x achieves the *a.s.a.o* guarantees and has been accelerated with CPU-based SIMD parallelism for collision checking [9]. However, CPUs offer limited parallelism as compared to modern GPUs, which are capable of running thousands of threads simultaneously via SIMT parallelism.

As such, in this paper, we propose pAORRTC, a GPU-based *a.s.a.o* planner based on the AO-x meta-algorithm and RRT-Connect planner, parallelized across all levels of the planning process. We evaluate pAORRTC against state of the art *a.s.a.o* planners using the 8-DoF Fetch on the MotionBenchMaker dataset, with pAORRTC demonstrating a 6x speedup in finding initial solutions and faster convergence to optimal solutions [10]. Our result shows that SIMT parallelism is effective in optimizing for path cost with the probabilistic guarantee of converging to global optimality.

II. Background and Related Work

A. The AORRTC Planning Algorithm

The *a.s.a.o* planning algorithms have been extensively studied due to their probabilistic guarantee of convergence to the globally best-cost path, with many being modifications of SBMP algorithms that were initially designed for finding any collision-free paths. A widely used example is RRT*, which continuously runs an RRT-style algorithm of sampling, nearest neighbor search, steering, and collision checking, while also carrying out the additional steps of selecting for a neighbor with the lowest cost-to-come and rewiring to a lower-cost parent [6]. RRT* is preferred for its *a.s.a.o* guarantee along with monotone convergence. Many improvements to RRT* have since been proposed, such as informed sampling where the samples are more likely to improve path cost [11, 12]. However, RRT* has faced challenges in achieving real-time performance due to its collision-checking intensive rewiring operations and unidirectional tree construction.

The AO-x meta-algorithm was proposed as an alternative to the RRT* approach for its flexibility in use with existing planners to achieve *a.s.a.o* guarantees [8]. Given an n-dimensional problem, the AO-x meta-algorithm augments an additional dimension describing the cost-to-come, leading to a $(n+1)$-dimensional system. The algorithm then attempts to generate trajectories with an increasingly lower cost bound. The AO-x meta-algorithm has been shown to be *a.s.a.o* with any feasible inner planner. AORRTC combines the AO-x meta-algorithm with the RRT-Connect planner and has shown orders of magnitude speedup compared to other planners in finding paths of lower suboptimality factor [9].

B. Parallel Acceleration of Planning Algorithms

Parallelism has been shown to provide great benefits in achieving real-time planning performance. Parallelism can be

This material is based upon work supported by the National Science Foundation (under Award 2411369). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding organizations.

[1]Columbia University chih.h@columbia.edu
[2]Dartmouth College plancher@dartmouth.edu
[3]Purdue University zkingston@purdue.edu

divided into three levels: high, medium, and low [13]. High-level parallelism refers to running multiple instances of the planning algorithms simultaneously; mid-level parallelism is running multiple iterations of the planning algorithm in parallel; low level parallelism is when primitive operations, such as collision checking, have been further parallelized.

Much work has been done in accelerating planning performance though CPU parallelism. For high-level parallelism, C-Forest runs several instances of the planning algorithm simultaneously and achieves superlinear speedup [14]. In terms of mid-level parallelism, PRRT takes a multi-thread approach with each thread independently running a RRT iteration. For low-level parallelism, VAMP redesigns sampling, forward kinematics tracing, and collision checking to utilize SIMD parallelism and achieves microsecond-scale planning. Notably, AORRTC uses VAMP for accelerating primitive operations and achieves orders of magnitude speedup vs. state of the art *a.s.a.o* planners [9] in path cost optimization.

With emerging GPU SIMT parallelism, some efforts have been made to utilize GPU's massive parallelism for real-time planning performance. Ways to utilize high-level parallelism has been demonstrated in Quad-RRT, which grows four trees in parallel and offers 10x speedup compared to other RRT-based methods [15]. For mid-level parallelism, Kino-PAX expands multiple nodes simultaneously [16]. In terms of low-level parallelism, cuRobo redesigns kinematics computation and collision checking to be carried out by multiple threads in parallel [17]. pRRTC is a GPU-based SBMP planner that utilizes all three levels of parallelism, and it achieves 5-10x speedup compared to existing CPU and GPU parallel approaches in finding initial solutions [13].

Despite these advances, to the best of the author's knowledge there has been no attempt at paralleizing *a.s.a.o* planners for all three levels of parallelism on the GPU. Inspired by the performance of AORRTC and pRRTC, we are interested in applying the all-level GPU parallelism approach to AORRTC and achieve real-time *a.s.a.o* planning performance for high degrees of freedom systems.

III. pAORRTC Algorithm

The pAORRTC planner retains similar structure as the CPU-based AORRTC algorithm, with GPU parallelism utilized at the high, medium, and low level, as described in Algs. 1 to 3. The RRT-Connect component of pAORRTC is as described in [13]. Since AORRTC has been proven to be *a.s.a.o*, pAORRTC also achieves the *a.s.a.o* guarantee.

On the high level, pAORRTC builds two search trees in parallel. For mid-level parallelism, pAORRTC runs hundreds of AORRTC iterations simultaneously, with each iteration running independently without the need to synchronize. This is achieved by mapping each AORRTC iteration to a single block on the GPU grid. On the low-level, pAORRTC utilizes multi-thread parallelism within each block for nearest neighbor search, forward kinematics tracing, and collision checking as described in [13]. The cost resampling step, as shown in Alg. 3, is used to accelerate convergence towards a lower-cost path and also benefits from parallelized collision checking. pAORRTC has been coupled with informed sampling (Alg. 1, Ln.8), which increases the probability of generating samples that could improve path cost [12]. pAORRTC's informed sampler utilizes multi-thread parallelism for acceleration, as we later discuss in further detail. After every cycle of pAORRTC, the paths are further simplified through B-spline smoothing and path shortcutting [18–20].

Algorithm 1: pAORRTC($c_{start}, c_{goal}, \delta, \phi, best_cost$)

1: T_a.init(), T_b.init()
2: T_a.add(c_{start}), T_b.add(c_{goal})
3: $d \leftarrow$ DISTANCE(c_{start}, c_{goal})
4: **for block index** $i = 0 \ldots N_{samples}$ **do in parallel**
5: $\quad c^i_{bound} \leftarrow$ RAND($d, best_cost$)
6: $\quad$ **for** iteration $iter \leftarrow 0 \ldots$ max_iters **do**
7: $\quad\quad T^i_s, T^i_o \leftarrow$ argmin($|T_a|, |T_b|$), argmax($|T_a|, |T_b|$)
8: $\quad\quad c^i_{rand} \leftarrow$ RANDOM_CONFIG()
9: $\quad\quad c^i_{nns} \leftarrow$ NN(c^i_{rand}, T^i_s, c^i_{bound})
10: $\quad\quad c^i_{new}$, valid^i $\leftarrow$ EXTEND(c^i_{rand}, c^i_{nns}, δ)
11: $\quad\quad$ **if** valid^i **then** // Add and Greedily extend to T^i_o
12: $\quad\quad\quad T^i_s$.add(c^i_{new})
13: $\quad\quad\quad c^i_{cost} \leftarrow$ COST(c^i_{new})
14: $\quad\quad\quad$ COST_RESAMPLE(c^i_{new}, T^i_o, c^i_{cost},
15: $\quad\quad\quad c^i_{nns}, \phi, d, \delta$)
16: $\quad\quad\quad c^i_{nno} \leftarrow$ NN(c^i_{new}, T^i_o, c^i_{bound})
17: $\quad\quad\quad n_{ext} \leftarrow$ DISTANCE(c^i_{new}, c^i_{nno}) / δ
18: $\quad\quad\quad v_{ext} \leftarrow (c^i_{nno} - c^i_{new})/\delta$
19: $\quad\quad\quad$ **for** $i_{ext} \leftarrow 1 \ldots n_{ext}$ **do**
20: $\quad\quad\quad\quad c^i_{ext} \leftarrow c^i_{new} + v_{ext}$
21: $\quad\quad\quad\quad c^i_{new}$, valid^i $\leftarrow$ EXTEND(c^i_{new}, c^i_{ext}, δ)
22: $\quad\quad\quad\quad$ **if** valid^i **then** T^i_s.add(c^i_{new})
23: $\quad\quad\quad\quad$ **else break**
24: $\quad\quad\quad$ **if** $i_{ext} = n_{ext}$ **then**
25: $\quad\quad\quad\quad$ **if** PATH_COST(T_a, T_b)< $best_cost$ **then**
26: $\quad\quad\quad\quad\quad best_cost \leftarrow$ PATH_COST(T_a, T_b)
27: $\quad\quad\quad\quad\quad$ **return** PATH(T_a, T_b), $best_cost$
28: $\quad\quad\quad\quad$ **else return**
29: $\quad$ **return** Failed

Algorithm 2: EXTEND(c_{rand}, c_{nn}, δ)

1: valid $\leftarrow$ True
2: $c_{new} \leftarrow$ INTERPOLATE(c_{rand}, c_{nn}, δ)
3: **for** $i \leftarrow 1 \ldots N_{cc}$ **do**
4: $\quad c_{check} \leftarrow$ INTERPOLATE($c_{rand}, c_{new}, i/N_{cc} * \delta$)
5: $\quad$ valid & = COLLISION_CHECK(c_{check})
6: **return** c_{new}, valid

Algorithm 3: COST_RESAMPLE($c_{new}, T, c_{cost}, c_{parent}, \phi, d, \delta$)

for $i \leftarrow 1 \ldots \phi$ **do**
  $c_{new_bound} \leftarrow$RAND(d, c_{cost})
  $c_{nns} \leftarrow$ NN$(c^i_{new}, T, c_{new_bound})$
  $c_{check} \leftarrow$ INTERPOLATE$(c_{rand}, c_{new}, i/N_{cc} * \delta)$
  valid $\&=$ COLLISION_CHECK(c_{check})
  if valid **then**
    $c_{new}.parent \leftarrow c_{nns}$
    break
return

The simplification process also benefits from SIMT parallelism, which we later discuss. The operations that benefit from multi-thread parallelism have been highlighted in blue.

The complete algorithm is shown in Alg. 1, with c_{start} and c_{goal} denoting the start and goal configurations. δ describes the extension range of RRT-Connect, while ϕ is the limit on cost resampling attempts per function call. As the name suggests, $best_cost$ denotes the current best cost of the found paths. During the first run of pAORRTC to construct an initial solution, $best_cost$ is set to $+\infty$.

A. Informed Sampling

The overall idea of informed sampling is to sample from a shrinking prolate hyperspheroid as the best-found path cost decreases, and therefore the informed sampler needs to transform a n-dimensional robot configuration space into a n-dimensional prolate hyperspheroid [12]. This requires the construction of a $4x4$ transformation matrix for each dimension. First, the handling of each dimension is independent, and therefore pAORRTC has n-thread per block constructing n transformation matrices in parallel. Second, informed sampling requires solving the wahba problem for constructing the rotation matrix. This is done to align the direction between the start and goal configurations with the major axis of the prolate hyperspheroid. The original informed sampling method recommends solving the wahba problem through singular value decomposition (SVD). However, this is not suitable for pAORRTC, which is a lightweight GPU-based planner with the aim of millisecond-scale performance, and most GPU-based SVD solvers are designed for matrices of hundreds of dimension which amortizes the solver's longer warmup time [21]. Since informed sampling requires transformation to a n-dimensional prolate hyperspheroid, meaning the hyperspheroid has one axis of differing length while the $(n-1)$ axis are identical, the wahba problem can be simplified into a Householder reflection problem. Constructing the Householder matrix is then a lightweight operation suitable for a single GPU thread [22].

B. Path Simplification

For B-Spline Smoothing, the simplification result of each edge is independent [18]. Hence, each edge of the path is assigned to a single block for smoothing, with all blocks carrying out the process in parallel. Within each block, the collision checking process also benefits from multi-thread parallelism as described in [13].

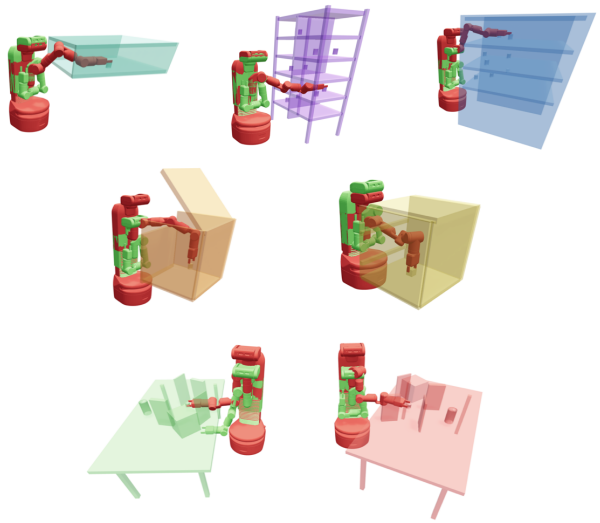

Fig. 1: The MotionBenchMaker scenes provide diverse manipulation problems for the 8 DoF Fetch. The problems include counter top manipulation, accessing shelves, and constrained reaching. Fetch has 7 scenes with 100 problems each. The green represents the start and red the goal configuration for a sample problem of the corresponding scene. From left to right, top to bottom the scenes are: *bookshelf_thin*, *bookshelf_small*, *bookshelf_tall*, *box*, *cage*, *table_under_pick*, *table_pick*

In terms of path shortcutting, the full shortcutting procedure is carried out on the path to achieve optimal results. This stands in contrast with planners like AORRTC which utilizes greedy or random shortcutting to reduce simplification time [9, 23]. The standard shortcutting process has time $O(N^2 \times Collision\ Checking)$, with N representing the number of nodes in the path. In the outer iteration, the program loops through all nodes of the path from start to goal. Within the inner iteration for each node n_t, the program checks the subsequent nodes $n_{t+1} \ldots n_N$ to determine whether it can form a collision-free edge with node n_t. If n_i is checked to have a collision-free edge with n_t, the cost and parent of node n_i is updated. In our case, the inner loop has been parallelized for multi-block parallelism. For block $b_1, b_2 \ldots b_k$, they each carry out the collision checks for the edge between node n_t and $n_{t+1} \ldots n_{t+k}$. Since GPUs are capable of running hundreds of blocks in parallel when in most cases the path has no more than a few dozen configurations, the inner loop ends up taking the time of one collision check. Again, within each block the collision checking further benefits from multi-thread parallelism as in [13].

IV. Experiments

A. Methodology

All experiments were conducted on an x86-based desktop computer with an AMD Ryzen Threadripper PRO 5965WX 24-Core CPU and an NVIDIA GeForce RTX 4090 GPU. We evaluate pAORRTC against vector-accelerated AORRTC. AORRTC has shown orders of magnitude speedup while

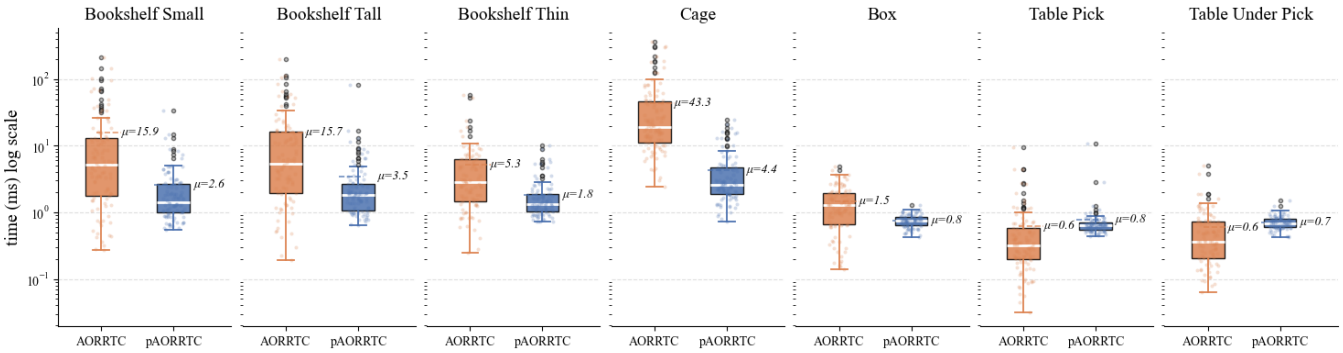

Fig. 2: Planning time for initial solution on the MotionBenchMaker using Fetch (8 DoF). All times are shown on a **logarithmic** scale. On average, pAORRTC achieves a 6× speedup in initial planning time and a 7.7x decrease in the standard deviation of planning time.

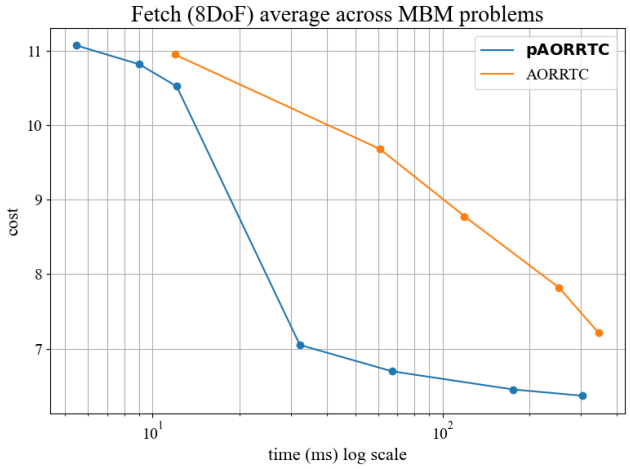

Fig. 3: MotionBenchMaker planning time and cost for all problems on Fetch (8 DoF). All times are shown on a **logarithmic** scale. pAORRTC achieves a 6× speedup in initial solution time, while consistently finds lower cost path in the following optimizing iterations.

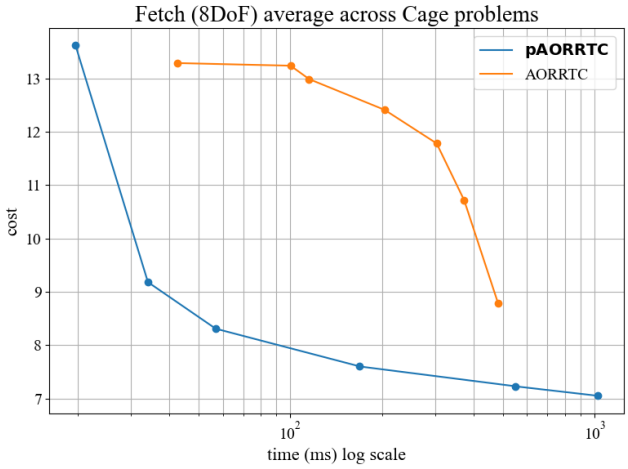

Fig. 4: *Cage* planning time and cost by problem on Fetch (8 DoF). All times are shown on a **logarithmic** scale. Not only does pAORRTC find a collision-free path faster, but it shows a more significant speedup in optimizing for lower-cost paths compared to averaging across all MBM problems as in Fig. 3.

demonstrating higher success rate in finding initial solution compared to other state of the art planners, such as RRT*, BIT*, and AIT* from the Open Motion Planning Library [9, 24]. The hyperparameters were controlled across all experiments and were selected so they achieve a 100% solve rate, as it was possible for both pAORRTC and AORRTC. We benchmarked the planners on the MotionBenchMaker (MBM) dataset, which is a collection of realistic and challenging manipulation problems involving counter top manipulation, shelf operation, and constrained reaching [10]. The robot used was the 8Dof Fetch, which comes with 7 nominal scenes each containing 100 manipulation problems, as depicted in Fig. 1.

B. Software Benchmarks

In terms of time to initial solution (Fig. 2), pAORRTC on average finds a collision-free path 6x faster than AORRTC, with a 7.7x decrease in the standard deviation of planning time. AORRTC solved a portion of the MBM problems faster due to its low setup time for CPU parallelism, while pAORRTC has a set GPU kernel launch overhead. Afterwards, pAORRTC was able to find paths of lower cost faster than AORRTC as shown in Fig. 3. For problems of increased difficulty, such as *Cage* which involves constrained reaching, the benefit of pAORRTC becomes increasingly clear as shown in Fig. 4. This is due to GPU's operation overhead being amortized for problems that require longer planning time.

V. Conclusion and Future Work

In this work, we present pAORRTC, a GPU-based *a.s.a.o* planner coupling the AO-x meta-algorithm with RRT-Connect. pAORRTC is the first GPU-accelerated *a.s.a.o* planner utilizing high, medium, and low-level parallelism. Through SIMT parallelism, pAORRTC is able to construct multiple search trees in parallel, run hundreds of AORRTC iterations simultaneously, while utilizing multi-thread parallelism for primitive operations like informed sampling.

For future work, we want to extend pAORRTC for problems of higher complexity. Potential directions include constrained planning [25] and multi-robot collaboration [26].

REFERENCES

[1] S. M. LaValle. *Planning Algorithms*. Cambridge university press, 2006.

[2] L. E. Kavraki and S. M. LaValle. "Motion Planning". In: *Springer Handbook of Robotics*. Springer, 2016, pp. 139–162.

[3] A. Orthey, C. Chamzas, and L. E. Kavraki. "Sampling-based motion planning: A comparative review". In: *Annual Review of Control, Robotics, and Autonomous Systems* 7 (2023).

[4] S. M. LaValle and J. J. Kuffner. "Rapidly-Exploring Random Trees: Progress and Prospects". In: *Algorithmic and computational robotics: new directions* 5 (2001), pp. 303–307.

[5] J. J. Kuffner and S. M. LaValle. "RRT-Connect: An Efficient Approach to Single-Query Path Planning". In: *IEEE International Conference on Robotics and Automation*. Vol. 2. 2000, pp. 995–1001.

[6] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[7] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 3067–3074.

[8] K. Hauser and Y. Zhou. "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1431–1443.

[9] T. S. Wilson, W. Thomason, Z. Kingston, and J. D. Gammell. "AORRTC: Almost-surely asymptotically optimal planning with RRT-Connect". In: *IEEE Robotics and Automation Letters* (2025).

[10] C. Chamzas, C. Quintero-Peña, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki. "MotionBenchMaker: A Tool to Generate and Benchmark Motion Planning Datasets". In: *IEEE Robotics and Automation Letters* 7.2 (2021), pp. 882–889.

[11] R. A. Knepper and M. T. Mason. "Real-time informed path sampling for motion planning search". In: *The International Journal of Robotics Research* 31.11 (2012), pp. 1231–1250.

[12] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *2014 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2014, pp. 2997–3004.

[13] C. H. Huang, P. Jadhav, B. Plancher, and Z. Kingston. "prrtc: Gpu-parallel rrt-connect for fast, consistent, and low-cost motion planning". In: *arXiv preprint arXiv:2503.06757* (2025).

[14] M. Otte and N. Correll. "C-FOREST: Parallel Shortest Path Planning with Superlinear Speedup". In: *Transactions on Robotics* 29.3 (2013), pp. 798–806.

[15] A. Hidalgo-Paniagua, J. P. Bandera, M. Ruiz-de-Quintanilla, and A. Bandera. "Quad-RRT: A Real-Time GPU-Based Global Path Planner in Large-Scale Real Environments". In: *Expert Systems with Applications* 99 (2018), pp. 141–154.

[16] N. Perrault, Q. H. Ho, and M. Lahijanian. "Kino-PAX: Highly Parallel Kinodynamic Sampling-Based Planner". In: *arXiv preprint arXiv:2409.06807* (2024).

[17] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, et al. "Curobo: Parallelized Collision-Free Robot Motion Generation". In: *IEEE International Conference on Robotics and Automation*. 2023, pp. 8112–8119.

[18] J. Pan, L. Zhang, and D. Manocha. "Collision-free and smooth trajectory computation in cluttered environments". In: *The International Journal of Robotics Research* 31.10 (2012), pp. 1155–1175.

[19] R. Geraerts and M. H. Overmars. "Creating high-quality paths for motion planning". In: *The international journal of robotics research* 26.8 (2007), pp. 845–863.

[20] K. Hauser and V. Ng-Thow-Hing. "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts". In: *2010 IEEE international conference on robotics and automation*. IEEE. 2010, pp. 2493–2498.

[21] A. Abdelfattah and M. Fasi. "An Efficient Batch Solver for the Singular Value Decomposition on GPUs". In: *arXiv preprint arXiv:2601.17979* (2026).

[22] G. H. Golub and C. F. Van Loan. *Matrix computations*. JHU press, 2013.

[23] W. Thomason, Z. Kingston, and L. E. Kavraki. "Motions in Microseconds via Vectorized Sampling-Based Planning". In: *IEEE International Conference on Robotics and Automation*. 2024, pp. 8749–8756.

[24] I. A. Şucan, M. Moll, and L. E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics and Automation Magazine* 19.4 (2012). https://ompl.kavrakilab.org, pp. 72–82.

[25] J. Hu, J. Wang, and H. Christensen. *cpRRTC: GPU-Parallel RRT-Connect for Constrained Motion Planning*. 2025.

[26] P. Huang, C. Gao, and J. Li. *VAMP-MR: Vector-Accelerated Motion Planning and Execution for Multi-Robot-Arms*. https://github.com/vamp-mr/vamp-mr. AAAI 2026 Workshop on Multi-Agent Path Finding (WoMAPF), project website. 2026.