VAMP-MR: Vector-Accelerated Motion Planning and Execution for Multi-Robot-Arms

Philip Huang[1], **Chenrui Gao**[2], **Jiaoyang Li**[1]

[1] Carnegie Mellon University
[2] University of Michigan
philiphuang@cmu.edu, crgao@umich.edu, jiaoyangli@cmu.edu

Abstract

Multi-robot-arm motion planning is a key challenge in deploying multiple manipulators for industrial tasks such as manufacturing. Existing search-based and sampling-based solvers often require significant computation time to produce collision-free, high-quality motions suitable for safe execution. In this work, we introduce a new suite of multi-robot-arm motion planners capable of near real-time motion generation, combining classical planning algorithms with state-of-the-art vectorized collision-checking techniques. Based on CPU SIMD instructions, our new planners remove motion validation as the primary bottleneck and achieve up to two orders of magnitude speedup in both motion planning and execution postprocessing for multi-arm manipulation tasks. We also release the implementation of our vector-accelerated multi-robot planning and execution algorithms, and we believe this will lower the barrier for research and development of multi-robot-arm planning and manipulation problems. Our code is available at https://vamp-mr.github.io/vamp-mr/.

1 Introduction

Robotic systems have the potential to transform industries such as construction and manufacturing by automating hazardous and physically demanding tasks like welding, polishing, assembly, and handling heavy objects. In theory, deploying multiple robots within a shared workcell can greatly enhance efficiency and throughput through collaboration and parallelization. In practice, however, this vision remains elusive: coordinating teams of robot arms to operate safely, efficiently, and robustly is highly challenging, and most multi-robot setups still require extensive manual programming. This approach is costly, time-consuming, and difficult to adapt when production needs change. Consequently, there is a demand for fast and reliable multi-robot-arm motion planning (M-RAMP) algorithms for pushing forward industrial automation.

In this work, we target a key bottleneck shared by nearly all multi-robot-arm motion planning methods—collision checking and motion validation. Our approach builds upon Vector-Accelerated Motion Planning (VAMP) (Thomason, Kingston, and Kavraki 2024), a high-performance single-robot-arm motion planner featuring CPU-SIMD-vectorized forward kinematics and collision checking, and extends it to multi-robot settings (Fig. 1). SIMD (Single Instruction, Multiple Data) increases the throughput of processing robot geometry data by parallelizing vector operations over multiple robot configurations. As a central contribution, we introduce a generalized collision-checking module that supports multiple robot arms with flexible input definitions for relative transformations, robot count, attachments, obstacles, and configurations. This design allows our system to replace existing frameworks such as FCL (Pan, Chitta, and Manocha 2012) and Bullet (Coumans and Bai 2021), achieving up to 100× speedups in both motion planning and postprocessing—without requiring any other algorithmic changes. Our experiments demonstrate that standard planning and post-processing pipelines can efficiently generate the motion for four robot arms in under a second on average, and that existing multi-agent pathfinding (MAPF) algorithms perform remarkably well even without adaptation to articulated robots. We believe this work lowers the barrier to research in multi-robot-arm manipulation, including optimal planning, task and motion planning, and workcell layout optimization.

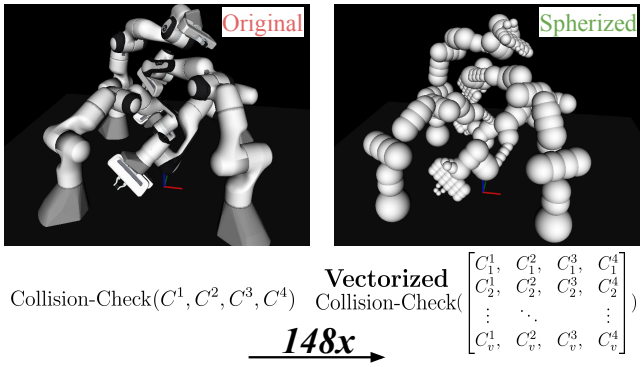


Figure 1: Our multi-robot-arm collision checking primitive achieves two orders of magnitude speedup for vectorized motion validation with spherized robot geometry and CPU SIMD instructions.

2 Background

2.1 Multi-Robot-Arm Motion Planning

Multi-Robot-Arm Motion Planning (M-RAMP) is an emerging research area focused on centrally coordinating

the motion of multiple robot arms operating in tight and cluttered environments. We define the M-RAMP problem as follows. Given a set of N robot arms, each with a fixed base transform ${}^{w}\mathbf{T}_i \in SE(3)$ and a configuration space $\mathcal{C}^i \subseteq \mathbb{R}^{\text{DoF}^i}$, where DoF^i denotes the number of degrees of freedom of robot i, all robots operate in a shared environment with obstacles ${}^{w}\mathcal{O}$. The objective is to find a set of collision-free joint space trajectories $\tau = \{\tau^1, \tau^2, \dots \tau^N\}$ from a pair of start configuration $[C^1_{\text{start}}, \dots, C^N_{\text{start}}]$ to a goal configuration $[C^1_{\text{goal}}, \dots, C^N_{\text{goal}}]$ for each robot.

A straightforward approach is to model all robots as a single composite system with the combined DoF and apply a single-agent planner such as RRT-Connect (Kuffner and LaValle 2000), RRT* (Karaman and Frazzoli 2011), or graph-of-convex-sets planning (Marcucci et al. 2022). However, the dimensionality of such joint configuration spaces grows rapidly with the number of robots, making these methods computationally prohibitive.

To address scalability, researchers have extended multi-agent pathfinding (MAPF) techniques—originally developed for 2D discrete grid worlds—to continuous, high-dimensional configuration spaces. For instance, Solis et al. (2021), Shaoul et al. (2024a), and Shaoul et al. (2024b) adapt variants of conflict-based search (CBS) (Sharon et al. 2015) to incrementally plan motions for individual robot arms while resolving inter-robot collisions through a high-level tree search with motion constraints. Prioritized planning methods have also been applied, generating trajectories sequentially for each robot arm (Hartmann et al. 2023; Chen et al. 2022), although these approaches lack the theoretical guarantees of CBS-style solvers. For executing a multi-robot-arm plan in the real world, uncertainties due to controller and sensor delay must be addressed. One solution is to use a temporal plan graph (TPG) (Huang et al. 2025), a partially ordered graph, to identify all potential collisions and coordinate each robot based on their precomputed actions and precedences.

Beyond traditional planning-based techniques, learning-based approaches such as (Lai et al. 2025) train centralized neural networks to control the synchronized motion of multiple robot arms using reinforcement learning in randomized environments. Our proposed method is complementary to all these works: it can accelerate motion validation in both sampling- and search-based planners and can also speed up feature and reward evaluations in learning-based controllers.

2.2 Accelerated Motion Planning

For most planning- and control-based motion generation methods, forward kinematics (FK) and collision checking (CC) are among the most computationally expensive operations (Bialkowski, Karaman, and Frazzoli 2011). Other notable costs include nearest-neighbor searches in sampling-based planners and priority-queue management in search-based methods.

Numerous strategies have been proposed to reduce the computational burden of collision checking. Lazy evaluation delays collision checking until necessary (e.g., LazyPRM (Bohlin and Kavraki 2000)), while experience-based methods exploit past search data to accelerate planning (Shaoul et al. 2024a). Learning-based approaches have trained approximate collision detectors (Das and Yip 2020) for sampling-based planning or learned signed distance fields (Koptev, Figueroa, and Billard 2023) for gradient-based control. Other work amortizes the cost of motion generation via neural motion planners, such as MPNet (Qureshi et al. 2021), which produce collision-free trajectories directly from sensory input after large-scale training. In parallel, several efforts have explored hardware or parallelization strategies, including multi-core CPU implementations of RRT (Ichnowski and Alterovitz 2014).

Our work aligns with approaches that directly parallelize collision checking itself. Bialkowski, Karaman, and Frazzoli (2011) parallelize collision checking on GPUs, and Murray et al. (2016) implement robot-specific collision detection circuitry on FPGAs. However, these methods often incur significant communication overhead, limiting their efficiency in complex settings such as task and motion planning. Alternatively, we build upon the approach introduced by Thomason, Kingston, and Kavraki (2024), which batches multiple configurations into a vector to evaluate forward kinematics and collisions with SIMD instructions.

2.3 Comparison of Traditional and Vector Accelerated Collision Checking (VAMP)

Traditional collision checkers such as FCL (Pan, Chitta, and Manocha 2012) and Bullet (Coumans and Bai 2021) are primarily designed for single-threaded execution and typically follow a common pipeline. Each collision object—such as a primitive, mesh, or point cloud—is represented by a hierarchical data structure that encodes both its bounding volumes (e.g., axis-aligned or oriented bounding boxes) and its underlying geometric primitives (e.g., triangles or points). During collision checking, the system updates the world transforms of all bounding volumes based on kinematics and recursively traverses the corresponding bounding volume hierarchies (BVHs). The process is generally divided into two stages: a broad-phase filter that identifies potentially overlapping bounding volumes, followed by a narrow-phase test (e.g., GJK (Gilbert, Johnson, and Keerthi 1988)) to determine exact intersections. While this approach is widely adopted, it is inherently difficult to parallelize. The recursive BVH traversal introduces significant conditional branching and irregular workloads, complicating efficient task distribution across multiple threads. Moreover, the high memory bandwidth required to access BVH transforms and geometry data often becomes a limiting factor without careful data layout and caching strategies.

VAMP overcomes these challenges by fusing forward kinematics (FK) and collision checking (CC) into a single optimized kernel implemented as a C++ header. It reorganizes batches of joint-space configurations into a struct-of-arrays memory layout, which compactly represents a batch of robot configurations for data parallelism and reduces memory overhead. Robot geometries are approximated by sets of spheres (Coumar et al. 2025), while obstacles are represented as simple primitives such as spheres, cubes, cylinders, or capsules. Forward kinematics is unrolled via

a custom tracing compiler that computes the positions of these spheres directly, minimizing branching and data dependencies. Self-collisions are interleaved with the FK computation, allowing early terminations if some link collides. Computing forward kinematics and collision checking of each sphere now becomes fully parallelizable for a batch of configurations, and if any one collides, the entire batch is rejected. With batching, VAMP discards traditional broad-phase collision checking and instead opts for a "rake" strategy. It evaluates a set of uniformly distanced configurations along an edge within a batch, enabling faster detection of collisions along those edges. VAMP's vectorized FK and CC routines can be seamlessly integrated with search- or sampling-based planners, achieving millisecond-level motion planning.

While VAMP demonstrates the substantial impact of fast collision checking on single-robot motion planning, the following sections extend these ideas to VAMP-MR, a multi-robot-arm planning system that vectorizes multi-robot collision checking and accelerates multiple stages of the pipeline, including motion planning, trajectory shortcutting, and safe execution.

3 Vectorized Multi-Robot Collision Checking

3.1 Method

Generating safe and high-quality motion for real-world multi-robot-arm systems involves several key stages—from task assignment and motion planning to postprocessing for execution. In many current multi-arm systems, such as those used for assembly (Huang et al. 2025; Chen et al. 2022), collision checking remains the dominant computational bottleneck, e.g., in roadmap construction, task allocation, and motion validation.

We overcome this bottleneck by accelerating forward kinematics and collision checking through approximate robot modeling and vectorized computation. This significantly reduces runtime overhead across the entire multi-robot planning and execution stack. Below, we describe the design of our vectorized collision-checking framework and its extensions for multi-robot coordination.

To adapt VAMP for multi-robot planning, we introduce several key modifications. A naive approach would be to merge all DoFs across robots into a single composite system and generate a corresponding VAMP kernel. However, many multi-robot planning algorithms require distinguishing between different types of collisions—for example, counting inter-robot collisions in CBS-style motion planning, computing pairwise collisions between selected robot pairs in TPG construction, or separating self-, robot-environment-, and robot-robot collisions for single-robot roadmap generation. We also aim to simplify practical multi-robot-arm manipulation tasks, such as assembly, where it is useful to easily calibrate inter-robot transformations ${}^{w}\mathbf{T}_i$, adjust attachments $\mathcal{A}^i$, and specify intentional contacts between robots and shared environment obstacles $\mathcal{W}_{\text{allow}}$.

To support these capabilities, we design a new routine: FK_CC_MULTI outlined in Algorithm 1. Given a batch of v multi-robot configurations $(C_j^1, \ldots, C_j^n)$ for n robots—represented as a $v \times n$ matrix as illustrated in Fig. 1, FK_CC_MULTI computes forward kinematics, checks collisions, and returns a boolean validity flag if and only if all v sets of multi-robot configurations are collision-free. Each robot first passes through the single-robot FK then CC routine to perform self-collision checks. If a collision is detected, the configuration is immediately rejected. Environment obstacles ${}^{w}\mathcal{O}$ environment obstacles transformed to the robot's base frame and checked against each robot while ignoring any allowed environment collisions $\mathcal{W}_{\text{allow}}$. In practice, we cache transformed environment obstacles in each robot's base frame since base transforms and obstacles often remain fixed during planning, which allows robot-environment collisions to be detected quickly. Then, each robot's spheres are transformed to the world frame based on its base transform ${}^{w}\mathbf{T}_i$. Subsequently, the sphere representations of all robots are compared pairwise to detect robot-robot collisions. This modular design is extremely flexible and still enables efficient, vectorized collision evaluation for complex multi-robot-arm setups.

Algorithm 1 FK_CC_MULTI: Vectorized FK and Collision Checking for Multiple Robots

Require: For each robot i: base transform ${}^{w}\mathbf{T}_i$, batch of configurations $\{C_j^i\}_{j=1}^{v}$, optional attachments $\mathcal{A}^i$; environment obstacles ${}^{w}\mathcal{O}$; allowed contacts $\mathcal{W}_{allow}$

for each robot i **do**
  ${}^{i}\mathbf{S}_{1:v} \leftarrow \text{FK}(C_{1:v}^i, \mathcal{A}^i)$ ▷ spheres in robot-i frame
  if SELFCC$({}^{i}\mathbf{S}_{1:v})$ detects collision **then**
    return Invalid
  end if
  ${}^{i}\mathcal{O} \leftarrow ({}^{w}\mathbf{T}_i)^{-1} \circ {}^{w}\mathcal{O}$ ▷ obstacles in robot-i frame
  if ENVCC$({}^{i}\mathbf{S}_{1:v}, {}^{i}\mathcal{O}, \mathcal{W}_{allow})$ detects collision **then**
    return Invalid
  end if
  ${}^{w}\mathbf{S}_{1:v}^i \leftarrow {}^{w}\mathbf{T}_i \circ {}^{i}\mathbf{S}_{1:v}$ ▷ spheres in world frame
end for
for each robot pair $(i, k),\ i < k$ **do**
  if INTERCC$({}^{w}\mathbf{S}_{1:v}^i, {}^{w}\mathbf{S}_{1:v}^k)$ detects collision **then**
    return Invalid
  end if
end for
return Valid ▷ iff all v batch lanes are collision-free

3.2 Evaluation of Collision Checking and Motion Validation

To evaluate the runtime improvement of our new collision checking method, we test on three challenging multi-robot-arm environments—Panda Two Rod, Panda Four, and Panda Four Bins (see Fig. 2)—introduced in Huang, Shaoul, and Li (2025). Each Panda arm has 7 DoF, and the 59-sphere approximation in Fig. 1 is generated with the tool in Coumar et al. (2025). These environments also involve attachments in Panda Two Rod and obstacles in Panda Four Bins.

We first assess the speedup achieved by our vectorized collision-checking primitives using randomly sampled robot configurations. Table 1 reports results over 10000 random

Figure 2: Multi-robot motion planning environments.

samples for both (1) collision checking of single sets of multi-robot configurations and (2) motion validation between pairs of configurations. The collision checking resolution, defined as L_1 distance between two consecutive interpolated configurations along the motion, is set to 0.1 radian. To ensure consistent modeling, we use the same simplified spherized robot geometry representation in FCL. All experiments are conducted on an AMD 7840HS Laptop CPU. We use C++ compiled with GCC 9 and 256-bit AVX2 instructions (e.g., -march=native, -mavx2, -O3). This allows a SIMD batch size of 8 with 32-bit single-precision floating-point for vectorized collision checking.

Our method achieves 11-27x speedup for single-configuration collision checking, and up to 148x speedup for motion validation between two random configurations due to vectorization. The compiler optimization of the robot-specific forward kinematics and collision checking kernel makes our FK_CC_MULTI routine significantly faster for spherized robot collision checking. Empirically, we notice that our VAMP-MR can significantly reduce the cache miss rate by 59% to 86% compared to FCL when profiled with the Linux tool PERF. For motion validation, the runtime speedup sometimes even exceeds the single-check speedup by more than 8 times (i.e., the number of SIMD lanes). We believe this shows the combined effect of vectorization, good cache utilization, and the "rake"-style scan when evaluating discretized configurations during motion validation. We also observe less variance across different environments compared to FCL. This is because randomly sampled motions between a pair of configurations are very likely to collide in the obstacle-rich Panda Four Bins environment, and FCL can invalidate a motion much more quickly than in other environments.

Table 1: Average collision checking runtime and speedup over 10000 random samples. Our method is compared against FCL with the same approximated spherized geometries to ensure consistency.

Check	Environment	FCL (µs)	Ours (µs)	**Speedup**
Single	Panda 2-Rod	100.99	8.60	**11.7x**
	Panda 4	206.53	15.01	**13.7x**
	Panda 4-Bins	382.63	13.70	**27.9x**
Motion	Panda 2-Rod	3936.13	36.03	**109.2x**
	Panda 4	4836.50	32.61	**148.3x**
	Panda 4-Bins	930.03	14.23	**65.4x**

4 Applications of Vector-Accelerated Collision Checking

4.1 Vectorized Multi-Robot-Arm Planning

Building on our vectorized multi-robot collision checking framework, we integrate it into two multi-robot-arm motion planners to enable fast and scalable multi-robot-arm motion planning. Our first planner is composite RRT-Connect (Kuffner and LaValle 2000), which combines the DoFs from all robots and treats them as a single robot for the RRT-Connect algorithm. We integrate our FK_CC_MULTI routine in RRT-Connect to accelerate collision checking for validating randomly sampled configurations and motions during tree expansion.

Our second planner is based on CBS-MP (Solis et al. 2021), a conflict-based search (CBS) motion planner originally designed for multi-agent systems operating on roadmaps. In CBS-MP, each agent (in our case, a robot arm) constructs its own probabilistic roadmap (PRM) (Kavraki et al. 1996). The planner then incrementally searches for collision-free trajectories, resolving inter-robot collisions through a high-level CBS tree that imposes avoidance constraints. However, the original avoidance constraint used CBS-MP is incomplete, as discussed in Shaoul et al. (2024a). To ensure theoretical completeness of CBS on the roadmaps, we adopt an asymmetric constraint scheme inspired by Li et al. (2019). Specifically, when a collision occurs at time t between robot i and robot j, we create two mutually exclusive constraints during high-level CBS expansion: one forbidding robot i from occupying any volume of robot j's volume at time t, and the other forbidding robot j from taking the same configuration at that time. With our SIMD-accelerated FK_CC_MULTI routine, we can significantly reduce the computational overhead in roadmap construction, constraint evaluation, and collision checking within CBS-MP.

Results To evaluate how VAMP-MR can improve planning time and analyze its algorithmic implications, we create 110, 132, and 132 unique pairs of start and goal poses as different planning instances in Panda Two Rod, Panda Four, and Panda Four Bins, as in Huang, Shaoul, and Li (2025).

Next, we evaluate both the composite RRT-Connect and CBS-MP motion planner. We sample 5000 nodes for each robot's probabilistic roadmap before planning in CBS-MP and run both planners with a one-minute timeout, excluding roadmap construction time.

Fig. 3 shows the planning time before and after vector acceleration for all three environments. Our results show that vectorization provides at least an order-of-magnitude speedup for both planners, with even greater improvement for CBS-MP. Before vector acceleration, the Panda Two Rod environment is generally the easiest to solve, while the Panda Four Bins environment proves most challenging due to numerous environmental and in-hand obstacles.

For RRT-Connect, the runtime acceleration correlates closely with the motion validation speedup reported in Table 1, as motion validation dominates the computational cost. However, CBS-MP exhibits the largest performance gain in Panda Four Bins, reaching a 100% success rate, whereas some instances in Panda Four remain unsolved. The difference results from the coordination requirement of the tasks.

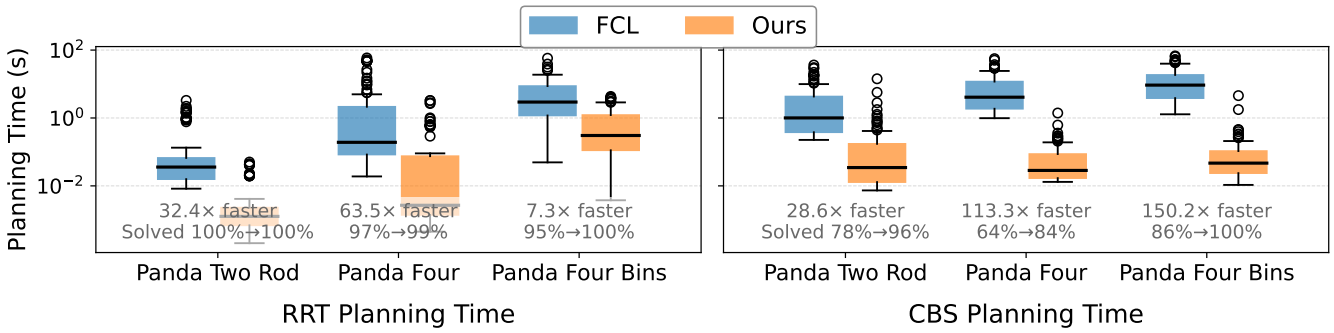


Figure 3: Planning time comparison of our vectorized multi-robot RRT-Connect and CBS planner on three environments. The median time of both RRT and CBS planning of solved instances in each environment is up to 150x faster with vectorization. Reported speedups are averaged over all planning instances in each environment.

In Panda Four, several target poses (e.g., the one in Fig. 1) require precise sequencing between robot arms, which leads to thousands of high-level CBS expansions necessary. Conversely, Panda Four Bins requires less coordination, as long as each robot can navigate around obstacles independently. Thus, CBS outperforms RRT as it can quickly deconflict agents and find collision-free solutions. We also noticed that the number of expanded conflict tree (CT) nodes in CBS increased significantly. For example, our vectorized CBS can solve an instance with 5303 expanded CT nodes on Panda Four, compared to a maximum of 53 expanded CT nodes among solved Panda Four instances with FCL.

These results suggest that our vectorized framework effectively removes collision checking as the primary bottleneck, enabling CBS-based methods to scale to more complex setups. Moreover, this highlights the potential for applying advanced CBS or MAPF techniques to multi-robot-arm systems, such as symmetry reasoning (Li et al. 2021), enhanced bounded-suboptimal search (Li, Ruml, and Koenig 2021), and even fast suboptimal planners (Li et al. 2022), to address the coordination challenge.

4.2 Vectorized Multi-Robot Shortcutting

Shortcutting is a widely used postprocessing technique for improving the smoothness and optimality of planned trajectories, particularly in multi-robot-arm motion planning (Shaoul et al. 2024a; Hartmann et al. 2023). This is because motion planners for multiple robot arms are not optimal. Sampling-based planners are only probabilistically complete and asymptotically optimal, and CBS is only resolution-complete and resolution-optimal given the roadmap. In practice, performance can usually be improved more with postprocessing.

Collision checking represents a major computational bottleneck for shortcutting: when optimizing one robot arm's trajectory, it must be verified that the new trajectory after shortcutting does not introduce collisions with other robots or the environment. This property makes shortcutting highly compatible with our vectorized collision-checking routine.

We adopt the Dynamic Thompson Shortcutting (DTS) algorithm from Huang, Shaoul, and Li (2025), an anytime shortcutting framework that adaptively combines three complementary strategies—composite, prioritized, and path shortcutting. On a high-level, DTS intelligently chooses one strategy, randomly samples a shortcut using the selected strategy, and update the trajectory if the shortcut is collision-free. This process is then repeated until a time limit is reached. Composite shortcutting jointly changes all robot trajectories in the composite configuration space, providing rapid early-stage improvement. Prioritized or path shortcutting modifies individual robot trajectories and can converge to higher-quality solutions. DTS formulates the choice among these strategies as a multi-armed bandit problem, using Thompson sampling to balance an efficient trade-off between early convergence speed and final trajectory quality. For each sampled shortcut, we evaluate if it is collision-free using our vectorized motion validation primitive FK_CC_MULTI. This collision checking step is the biggest bottleneck, and by accelerating this with vectorization, we achieve over 10× faster convergence compared to conventional shortcutting methods. Moreover, because DTS is an anytime algorithm, trajectory quality can be improved rapidly within a tenth of a second—making it suitable for online applications.

Results We compare the performance improvement of the Dynamic Thompson Sampling (DTS) algorithm for multi-robot-arm shortcutting, as shown in Fig. 4, using both RRT and CBS-generated initial trajectories. Without any other modifications, integrating a vectorized collision checking routine reduces the time required to converge to final shortcut trajectories up to 50x.

The vector-accelerated shortcutter achieves substantial improvement even within 0.1s, and all trajectories converge within 1s. When combined with either an RRT or CBS motion planner, this unlocks online-capable and high-quality motion generation for multi-robot-arm systems.

4.3 Vectorized Safe Execution Framework

For many multi-robot-arm manipulation tasks, such as object rearrangement or collaborative assembly, it is often necessary to execute a sequence of coordinated multi-robot motions derived from a higher-level task plan. However, safe execution in real-world environments must account for execution uncertainties—such as kinematic inaccuracies and

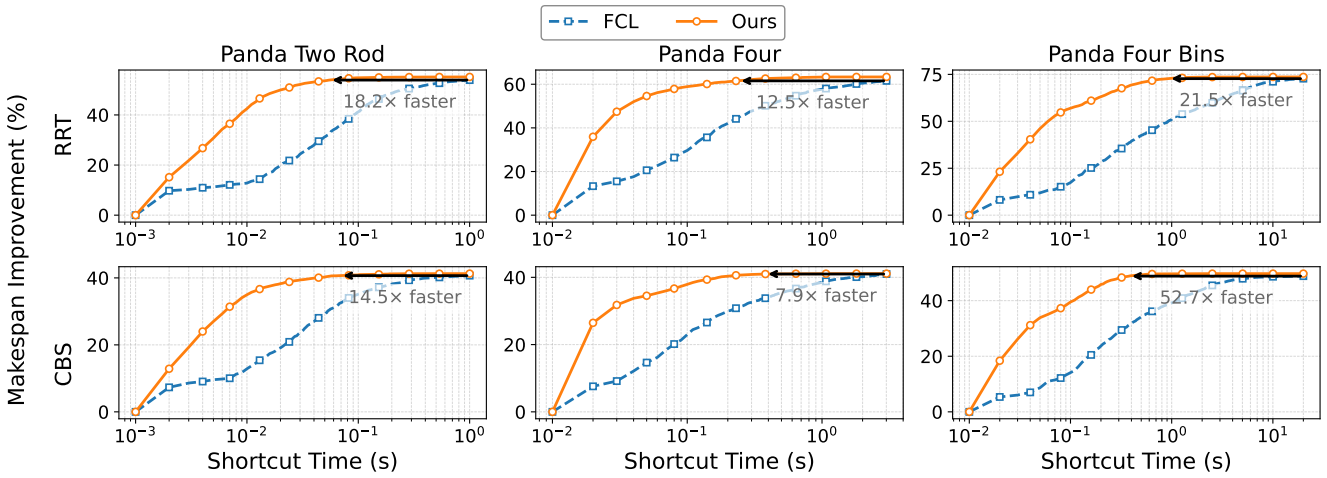


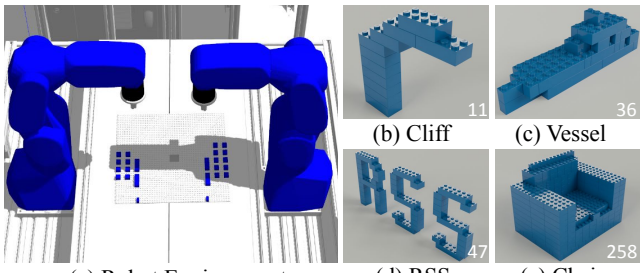
Figure 4: Average makespan improvement over time of multi-robot shortcutting on three environments, using Dynamic Thompson Shortcutting. The performance-runtime curves are averaged over all instances in each environment. Our method converges to the same level of makespan improvement up to 50 times faster, reducing the convergence time to under one second.

unpredictable physical interactions with the environment. Beyond modeling challenges, many manipulation tasks also use closed-loop control or learned policies that make it impossible to predict accurate execution time in planning.

We build upon the multi-modal Temporal Plan Graph (TPG) framework introduced by Huang et al. (2025), which systematically postprocesses a given multi-robot task and motion plan to enable safe asynchronous execution under such uncertainties. A multi-modal TPG $G = (V, E)$ is a partially ordered graph that captures kinematic transitions, changes in the environments due to objects being moved, and inter-robot task and motion dependencies. Each node v_n^i is either a pose node that corresponds to a target configuration C_n^i for robot i or a skill node that corresponds to a manipulation skill. A manipulation skill is defined as a set of object-centric motions executed by a feedback controller. We assume that each skill has a reference robot path for the purpose of TPG computation. Each edge $v_n^i \rightarrow v_{n'}^{i'}$ encodes a precedence constraint between two nodes. Edges between nodes of the same robot are added between every consecutive node, and inter-robot edges can be added for task dependencies or motion dependencies to prevent executing two spatially colliding nodes simultaneously.

During execution, each node can be safely executed if all incoming nodes are completed. Each robot maintains its own action queue and the TPG serves as a central scheduler that sends nodes that can be safely executed to the action queues. Each robot's controller executes nodes in its action queue and updates node completion status back to the TPG server, allowing new nodes to be scheduled. The partial-order structure naturally tolerates execution delays, since each node can take an arbitrary amount of time to complete.

Constructing a TPG involves converting a sequential or synchronous motion plan into a temporally dependent multi-robot schedule. This process requires finding all intra-robot motion dependencies with extensive pairwise collision checking across all robot trajectories, which scales quadratically with the number of robots N and the number of discrete trajectory waypoints. While Huang et al. (2025) used CPU multithreading to parallelize this computation, our vectorized collision-checking routine is even better suited to accelerate TPG construction with less communication overhead. However, the original TPG formulation requires identifying all colliding pairs of configurations, whereas our vectorized collision checker FK_CC_MULTI terminates early if any configuration in the batch collides. To effectively leverage SIMD parallelism within this framework, we propose a modified grouping strategy.



(a) Robot Environment (b) Cliff (c) Vessel (d) RSS (e) Chair

Figure 5: Dual-arm LEGO assembly environment in (a) and assembly tasks in (b)-(e). Numbers in (b)-(e) indicate the number of assembly steps (LEGO bricks).

For a group of k consecutive pose nodes $[v_n^i, \dots v_{n+k-1}^i]$ of the TPG, we merge them to a larger transit node $\mathbf{v}_n^i$ that represents a short path segment $[C_n^i, \dots C_{n+k-1}^i]$. We then perform collision checks between every pair of transit nodes from different robots. If any part of the path segment of a transit node $\mathbf{v}_n^i$ collides with that of another transit node $\mathbf{v}_{n'}^{i'}$, we insert an inter-robot edge $\mathbf{v}_n^i \rightarrow \mathbf{v}_{n'}^{i'}$ from the earlier node to the later node ($n < n'$) that prevents collision in execution. In this formulation, we can ignore which exact configurations between two transit nodes $\mathbf{v}_n^i$ and $\mathbf{v}_{n'}^{i'}$ col-

Table 2: Runtime and makespan of dual-arm LEGO assembly plans generated following the procedure in APEX-MR (Huang et al. 2025), with and without vector acceleration. Results are averaged over 4 random seeds.

Metric	Cliff	Vessel	RSS	Chair
TPG Shortcut Time (s)	1.0	1.0	5.0	5.0
FCL-Based				
Task Assignment (s)	0.60	4.90	14.5	13.0
Motion Planning (s)	0.63	0.76	8.99	9.98
TPG Construction (s)	9.88	25.6	43.0	817.2
Total Time (s)	**12.1**	**32.2**	**71.5**	**845.2**
Final Makespan (s)	**185**	**397**	**741**	**2180**
Ours				
Task Assignment (s)	0.55	4.68	13.6	10.2
Motion Planning (s)	0.42	0.15	7.27	0.88
TPG Construction (s)	1.42	2.55	5.34	43.22
Total Time (s)	**3.39**	**8.38**	**31.2**	**59.3**
Final Makespan (s)	**159**	**340**	**542**	**2146**
# of Bricks	11	36	47	258

lide and use vectorized collision checking to determine if any part of the large transit collides and insert an edge if necessary. Although this reduces the number of edges and is more conservative than the old TPG formulation, empirically the execution makespan difference is often negligible. As a result, our approach achieves much more efficient TPG construction for complex multi-robot assembly tasks.

Results We evaluate four long-horizon dual-arm LEGO assembly tasks (see Fig. 5) from APEX-MR (Huang et al. 2025), which involve up to 258 assembled parts. These tasks provide more realistic and complex test cases for multi-robot-arm systems. For each task, we measure the runtime across all stages of the pipeline, including task assignment, motion planning, and TPG construction. We follow the methodologies and experimental setup of (Huang et al. 2025). Given a sequential assembly plan, we perform task planning and assign robot, grasping, and support poses targets for each assembly step using an integer-linear program. The motion of each transit task and each manipulation skill are planned sequentially with single-agent RRT-Connect. RRT-Connect trajectories can be jerky and suboptimal, so the resulting motion is passed to a vectorized single-agent shortcutter for 0.1s (Choset et al. 2005). The TPG execution framework then converts the sequential task and motion plan to an asynchronous, parallelized execution order. The makespan of TPG is further optimized with randomized TPG shortcutting as in Huang et al. (2025).

We implement the RRT-Connect algorithm without using MoveIt for sequential motion planning for both FCL and our vector-accelerated planner. FCL uses the original robot mesh for collision checking since we found it is faster than using spheres. We also found that our custom implementation provides worse solutions, but is significantly faster than the result reported in Huang et al. (2025).

Table 2 summarizes the runtime of each planning step and the final makespan after shortcutting. Motion planning is accelerated between 1.25x to 11.3x, and TPG construction is accelerated between 6.9x to 18.9x. In particular, the FCL-based TPG construction uses 16 CPU threads for parallelization, whereas our modified TPG construction with vector acceleration uses a single thread. We find that multi-core parallelization is largely unnecessary with vectorization due to additional communication overhead. Our accelerated planner also produces higher-quality solutions after TPG shortcutting at the same runtime. Shortcutting is particularly effective for the RSS assembly, reducing the makespan 1.37x as the RRT motion planner struggles to find efficient trajectories quickly. However, the task assignment component has little improvement, as our acceleration affects only the verification of collision-free grasp and support poses. The integer-linear program itself can be a significant bottleneck now in the overall task and motion planning pipeline.

5 Conclusion

We present a suite of superfast multi-robot-arm planning algorithms that achieves state-of-the-art performance without requiring major algorithmic changes. This is primarily driven by our proposed vector-accelerated collision checker for multiple robot arms, delivering 10 to 100x speedup across diverse planning, shortcutting, and execution framework. With collision checking no longer the dominant computational bottleneck, new challenges emerge, such as the effectiveness of the high-level search in CBS, and optimizing multi-robot task assignment. We believe this work is a crucial step towards lowering the barrier to developing efficient multi-robot-arm planning algorithms and opens new opportunities for integrating MAPF techniques into manipulation and assembly settings.

Acknowledgement

The author would like to thank Yorai Shaoul, Itamar Mishani, and Muhammad Suhail Saleem for early discussion and feedback. This work is in part supported by the National Science Foundation (NSF) under grant numbers 2328671 and 2441629, as well as a gift from Amazon.

References

Bialkowski, J.; Karaman, S.; and Frazzoli, E. 2011. Massively parallelizing the RRT and the RRT*. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 3513–3518.

Bohlin, R.; and Kavraki, L. E. 2000. Path planning using lazy PRM. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 521–528.

Chen, J.; Li, J.; Huang, Y.; Garrett, C.; Sun, D.; Fan, C.; Hofmann, A.; Mueller, C.; Koenig, S.; and Williams, B. C. 2022. Cooperative Task and Motion Planning for Multi-Arm Assembly Systems. arXiv:2203.02475.

Choset, H.; Lynch, K. M.; Hutchinson, S.; Kantor, G. A.; and Burgard, W. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.

Coumans, E.; and Bai, Y. 2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org. Accessed: 2026-01-06.

Coumar, S.; Chang, G.; Kodkani, N.; and Kingston, Z. 2025. FOAM: A Tool for Spherical Approximation of Robot Geometry. arXiv:2503.13704.

Das, N.; and Yip, M. 2020. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4): 1096–1114.

Gilbert, E. G.; Johnson, D. W.; and Keerthi, S. S. 1988. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2): 193–203.

Hartmann, V. N.; Orthey, A.; Driess, D.; Oguz, O. S.; and Toussaint, M. 2023. Long-Horizon Multi-Robot Rearrangement Planning for Construction Assembly. *IEEE Transactions on Robotics*, 239–252.

Huang, P.; Liu, R.; Liu, C.; and Li, J. 2025. APEX-MR: Multi-Robot Asynchronous Planning and Execution for Cooperative Assembly. In *Proceedings of Robotics: Science and Systems*.

Huang, P.; Shaoul, Y.; and Li, J. 2025. Benchmarking Shortcutting Techniques for Multi-Robot Arm Motion Planning. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 13258–13265.

Ichnowski, J.; and Alterovitz, R. 2014. Scalable Multicore Motion Planning Using Lock-Free Concurrency. *IEEE Transactions on Robotics*, 30(5): 1123–1136.

Karaman, S.; and Frazzoli, E. 2011. Sampling-Based Algorithms For Optimal Motion Planning. *The International Journal of Robotics Research*, 30(7): 846–894.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12(4): 566–580.

Koptev, M.; Figueroa, N.; and Billard, A. 2023. Neural joint space implicit signed distance functions for reactive robot manipulator control. *IEEE Robotics and Automation Letters*, 8(2): 480–487.

Kuffner, J. J.; and LaValle, S. M. 2000. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 995–1001.

Lai, M.; Go, K.; Li, Z.; Kröger, T.; Schaal, S.; Allen, K.; and Scholz, J. 2025. RoboBallet: Planning for multirobot reaching with graph neural networks and reinforcement learning. *Science Robotics*, 10(106): eads1204.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing For Multi-Agent Path Finding Via Large Neighborhood Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 10256–10265.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021. Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence*, 301: 103574.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 12353–12362.

Li, J.; Surynek, P.; Felner, A.; Ma, H.; Kumar, T. S.; and Koenig, S. 2019. Multi-Agent Path Finding for Large Agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 7627–7634.

Marcucci, T.; Petersen, M.; von Wrangel, D.; and Tedrake, R. 2022. Motion Planning Around Obstacles With Convex Optimization. arXiv:2205.04422.

Murray, S.; Floyd-Jones, W.; Qi, Y.; Sorin, D. J.; and Konidaris, G. 2016. Robot Motion Planning on a Chip. In *Proceedings of Robotics: Science and Systems*.

Pan, J.; Chitta, S.; and Manocha, D. 2012. FCL: A General Purpose Library for Collision and Proximity Queries. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 3859–3866.

Qureshi, A. H.; Miao, Y.; Simeonov, A.; and Yip, M. C. 2021. Motion Planning Networks: Bridging the Gap Between Learning-Based and Classical Motion Planners. *IEEE Transactions on Robotics*, 37(1): 48–66.

Shaoul, Y.; Mishani, I.; Likhachev, M.; and Li, J. 2024a. Accelerating Search-Based Planning for Multi-Robot Manipulation by Leveraging Online-Generated Experiences. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 523–531.

Shaoul, Y.; Veerapaneni, R.; Likhachev, M.; and Li, J. 2024b. Unconstraining Multi-Robot Manipulation: Enabling Arbitrary Constraints in ECBS with Bounded Sub-Optimality. In *Proceedings of the International Symposium on Combinatorial Search*, 109–117.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219: 40–66.

Solis, I.; Motes, J.; Sandström, R.; and Amato, N. M. 2021. Representation-Optimal Multi-robot Motion Planning Using Conflict-based Search. *IEEE Robotics and Automation Letters*, 4608–4615.

Thomason, W.; Kingston, Z.; and Kavraki, L. E. 2024. Motions in Microseconds via Vectorized Sampling-Based Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 8749–8756.