2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

SHARP: Supercomputing for High-speed Avoidance and Reactive Planning for Robots

Kieran S. Lachmansingh, José R. González-Estrada, Jacob Chisholm, Ryan E. Grant, Matthew K. X. J. Pan
Ingenuity Labs Research Institute
Queen's University
Kingston, Ontario, Canada

***Abstract*— This paper presents SHARP (Supercomputing for High-speed Avoidance and Reactive Planning), a proof-of-concept study demonstrating how high-performance computing (HPC) can enable millisecond-scale responsiveness in robotic control. While modern robots face increasing demands for reactivity in human–robot shared workspaces, onboard processors are constrained by size, power, and cost. Offloading to HPC offers massive parallelism for trajectory planning, but its feasibility for real-time robotics remains uncertain due to network latency and jitter. We evaluate SHARP in a stress-test scenario where a 7-DOF manipulator must dodge high-speed foam projectiles. Using a hash-distributed multi-goal A* search implemented with MPI on both local and remote HPC clusters, the system achieves mean planning latencies of 22.9 ms (local) and 30.0 ms (remote, 300 km away), with avoidance success rates of 84% and 88%, respectively. These results show that when round-trip latency remains within the tens-of-milliseconds regime, HPC-side computation is no longer the bottleneck, enabling avoidance well below human reaction times. The SHARP results motivate hybrid control architectures: low-level reflexes remain onboard for safety, while bursty, high-throughput planning tasks are offloaded to HPC for scalability. By reporting per-stage timing and success rates, this study provides a reproducible template for assessing the real-time feasibility of HPC-driven robotics. Collectively, SHARP reframes HPC offloading as a viable pathway toward dependable, reactive robots in dynamic environments.**

I. Introduction

Modern robots are increasingly deployed in dynamic, human-shared environments that demand both accurate planning and millisecond-scale reactivity. To avoid unsafe interactions, robots must perceive changes and adapt within tens of milliseconds. Traditionally, this responsiveness relies on onboard or local CPUs or GPUs, constrained by size, power, and cost [1], [2]. As robots adopt compute-intensive AI and machine learning models, these platforms are reaching their limits, constraining responsive human–robot interaction.

High-performance computing (HPC) offers an alternative. By enabling large-scale parallel graph search and optimization—core to inverse kinematics, trajectory planning, and collision avoidance—HPC can, in principle, deliver solutions orders of magnitude faster than local compute [3], [4]. The challenge is latency [5]. Because HPC resources are typically accessed over networks, communication delays may offset raw compute gains. In time-critical robotics, where milliseconds matter, the central question is whether HPC's speed can compensate for network overhead and still enable real-time action [6], [7].

The authors can be reached at: {kieran.lachmansingh, 19jrg4, 21jc138, ryan.grant, matthew.pan}@queensu.ca

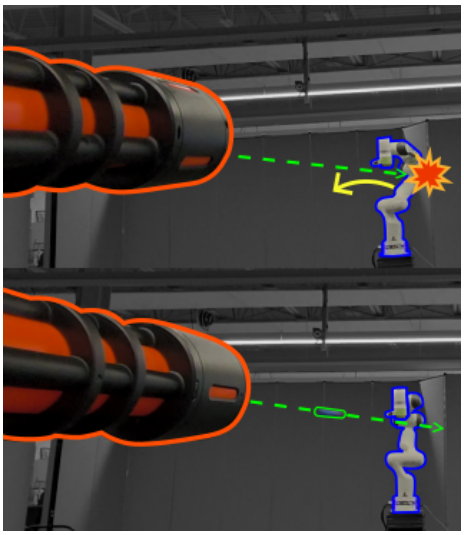

Fig. 1. Depiction of our testing setup which has a high-performance computer (HPC) plan an behaviour (represented by the yellow arrow) for a robot manipulator (outlined in blue) to avoid the predicted trajectory (dotted green line) of a projectile fired from a blaster (outlined in orange) in real-time.

We address this question through **SHARP** — Supercomputing for High-speed Avoidance and Reactive Planning — a proof-of-concept system evaluating whether HPC offloading can support real-time avoidance. To stress-test responsiveness, we use a deliberately stringent scenario: a 7-DOF manipulator dodging high-speed foam projectiles as shown in Fig. 1. This setup forces planning to operate within strict reaction budgets and serves as a proxy for broader dynamic human–robot collaboration tasks.

II. Background

A. High-Performance and Cloud Computing in Robotics

To overcome the limitations of onboard processors, robotics has increasingly explored high-performance and cloud computing [8]. Cloud robotics enables computation to be offloaded to remote clusters, providing scalable processing beyond what embedded CPUs/GPUs can support [9]. Early systems such as DAvinCi and Rapyuta demonstrated this paradigm by integrating cloud-based resources with ROS for tasks like SLAM and knowledge sharing [10], [11]. Subsequent work showed that offloading can significantly accelerate core robotic workloads [12].

However, non-local computation introduces latency, infrastructure dependence, and security concerns [13]. Frameworks such as FogROS and FogROS2 address these issues through selective offloading, allowing developers to partition workloads between local and cloud resources [6], [8]. These hybrid systems report order-of-magnitude speedups (4–31×) with moderate latency overhead (0.5–1.2 s), illustrating that cloud/HPC acceleration is viable when communication delays are acceptable.

B. HPC and Cloud-based Planning for Obstacle Avoidance

Trajectory planning and obstacle avoidance remain computationally demanding, particularly as dimensionality and obstacle density increase [14]. Parallel approaches mitigate this burden: multiple RRT/RRT* instances can run concurrently and terminate upon the first feasible solution [15]. Extending this idea, Ichnowski et al. leveraged cloud-based serverless computing to launch many parallel planners, achieving interactive motion planning for high-dimensional manipulation tasks [16].

These results demonstrate that scaling computation on demand, across threads and distributed nodes, can substantially reduce planning time. However, most prior work targets second-scale planning horizons. Whether similar offloading strategies can meet millisecond-level reaction budgets in dynamic, safety-critical settings remains largely unexplored.

III. Contributions

Despite evidence that HPC can accelerate robotic algorithms as presented in Sec. II, open questions remain about whether its raw computational speed can overcome the latency introduced by non-local access in real-time tasks. Most prior studies examine offloaded planning in offline or asynchronous contexts, with little attention to end-to-end responsiveness in dynamic, safety-critical settings. Furthermore, while latency challenges are well documented in teleoperation and cloud robotics, few frameworks investigate the feasibility of HPC for autonomous robots operating in fast, unpredictable environments.

SHARP introduces the first proof-of-concept demonstration of supercomputing applied to millisecond-scale reactive robotics. Our contributions are fourfold:

1) We design and implement a dataflow where a robot offloads avoidance requests to an HPC, receives executable trajectories, and closes the loop in real time.
2) We adapt a deterministic, voxelized hash-distributed A* search to HPC clusters via MPI, exploiting large-scale parallelism to guarantee bounded runtimes for evasive motion planning.
3) In a projectile-dodging stress test, SHARP achieves mean planning latencies of 22.9 ms (local) and 30.0 ms (remote, 300 km) with avoidance success rates of 84–88%, demonstrating viability under sub-human reaction time budgets.
4) A template for end-to-end evaluation—reporting per-stage timing and success on viable shots—that other HPC-controlled robotics systems can adopt, providing a method to compare time-sensitive offloading.

Together, these contributions provide a missing link between algorithmic parallelism and practical deployment, highlighting how HPC can extend robotic responsiveness in scenarios where timing is critical.

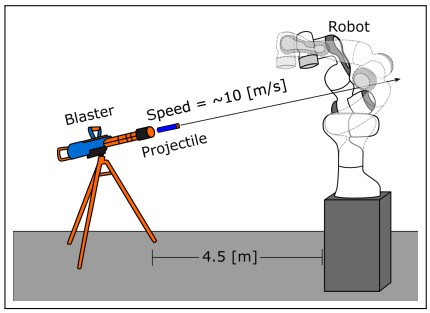

Fig. 2. Projectile-dodging benchmark: The dotted outlines denote the pre-avoidance state; the solid model shows the executed post-dodge robot configuration and the measured dart pose used for prediction.

IV. System Overview

A. Proof-of-Concept Demonstration Description

To evaluate the feasibility of HPC for time-sensitive robotics, we designed a deliberately stringent stress-test scenario: a 7-DOF manipulator tasked with dodging high-speed foam projectiles, as shown in Fig. 2. While playful on the surface, this scenario imposes strict timing constraints and serves as a proxy for broader challenges in dynamic environments and human–robot shared workspaces.

The benchmark highlights two critical aspects of reactive control. First, it forces the system to close the loop—from perception to offloading to HPC computation to robot actuation within a few tens of milliseconds. If the robot can successfully avoid fast-moving projectiles, it can also handle slower and less structured obstacles, such as humans entering its workspace. Second, the benchmark exposes the interplay between sensing fidelity, network overhead, and remote computation in a safety-critical context. Latency or jitter at any stage directly impacts whether avoidance succeeds.

We therefore adopt the projectile-dodging task as a stress test for SHARP. By pushing the system to operate under severe reaction-time requirements, we can quantify both its potential and its limitations. The results reveal not only whether HPC offloading can support reactive behaviour in principle, but also under what conditions it remains viable when milliseconds matter.

B. Hardware

1) Robot

We use a Franka Emika Panda, a 7-DOF manipulator. Its redundant configuration allows the robot to perform evasive manoeuvres while maintaining a desired position and orientation of the end-effector if allowed. The robot runs under ROS2 Humble with a real-time control frequency of 1 kHz (through the Franka Control Interface), ensuring fast and reliable command execution [17].

2) Projectile Generator

A Zuru X-Shot blaster [1] shoots foam darts at the robot. The blaster is instrumented with a Raspberry Pi Pico W, which

[1] https://zurutoys.com/brands/x-shot/insanity

provides strict timing triggers at each projectile launch over Bluetooth, ensuring synchronized avoidance requests.

3) Motion Capture

A Vicon motion capture system (12 cameras, 240 Hz) provides high-fidelity ground truth of blaster pose and projectile trajectory, isolating compute responsiveness from perception noise.

4) HPCs

Two HPC platforms are used: (i) a local 12-node Xeon cluster with sub-ms intra-cluster network RTT, and (ii) a remote +1000-node EPYC cluster located ∼300 km away, enabling evaluation under realistic latency using inter-network (WAN).

- Local cluster – a small (12) multi-node HPC located locally (on campus) equipped with 2× Intel Xeon Gold 6338 CPUs (64 cores), providing a controlled environment with minimal network overhead.
- Remote cluster – a large (+1000) multi-node HPC located ∼300 km away, featuring 2× AMD EPYC 7532 CPUs (64 cores) per compute node, accessed non-locally to evaluate the impact of real-world network latency on responsiveness.

These complementary platforms allow us to compare performance in both near-ideal (local) and practical (remote) HPC configurations.

C. System Workflow

Figure 3 summarizes the closed-loop workflow of SHARP. At runtime, the robot continuously monitors for projectile launches while maintaining a live connection to the HPC cluster. When a firing trigger is received, the system checks for potential collisions and, if needed, generates a compact voxelized representation of the predicted projectile trajectory. This representation, along with the robot's state, is transmitted to the HPC cluster.

On the HPC side, the voxel data is distributed across processes and evaluated in parallel. A hash-distributed multi-goal A* search identifies a feasible avoidance trajectory, which is then returned to the robot in compact waypoint form. The robot immediately executes this motion using a real-time controller, before resuming its monitoring state.

This loop (sense, offload, plan, and execute) repeats continuously, enabling the robot to respond to successive projectiles in real time. Each stage of the pipeline is detailed in Sec. V.

V. Methodology

This section details the end-to-end pipeline that enables SHARP to perform HPC-driven avoidance in real time. As outlined in Fig. 3, the system operates as a closed loop with four stages: (i) collision detection and environment encoding, (ii) parallel planning on the HPC cluster, (iii) trajectory generation and return, and (iv) execution on the robot. For clarity, we also describe how latency is measured for each stage.

A. Collision Detection and Encoding

When a projectile is launched, a trigger from the instrumented blaster signals the robot computer to evaluate whether its current configuration is at risk of collision. Using the most recent Vicon pose of the blaster, the predicted projectile path is approximated as a straight line, sufficient to enforce strict timing constraints.

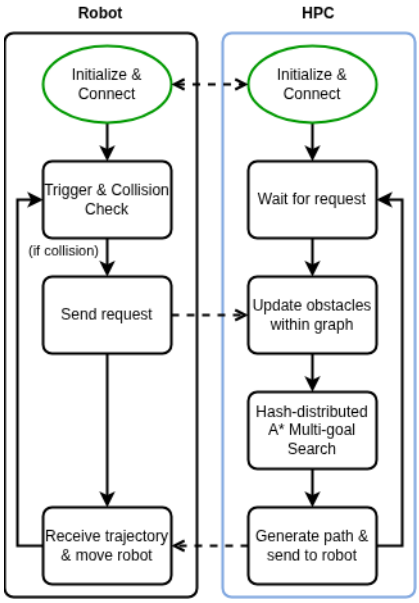

Fig. 3. General SHARP system overview.

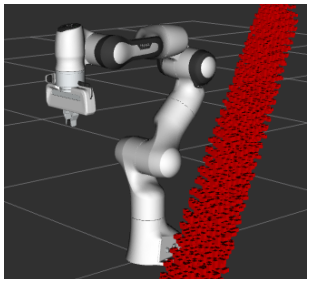

Fig. 4. Sample voxel generation displayed within RViz.

The workspace is voxelized around the predicted trajectory (Fig. 4), producing a compact obstacle representation. Voxels are arranged cylindrically by height, radius, and angular resolution, with occupied voxels flagged as obstacles.

1) Elbow Configuration Discretization

To capture the robot's redundancy, the null space is discretized using a custom analytic inverse kinematics solver. We define an elbow configuration as a distinct null-space solution for the 7-DOF manipulator. Each end-effector pose and orientation may therefore exist across multiple elbow configurations. For efficiency, the number of elbow configurations is fixed to a power of two; this allows easy scaling across different clusters, improved load balancing across all processes, and alignment with computer memory, improving overall performance.

B. HPC Planning

The encoded environment is transmitted to the HPC cluster for parallel trajectory planning.

1) Graph Formulation

The robot's discretized workspace is modelled as a four-dimensional graph (x, y, z, and elbow). Nodes represent

candidate Cartesian end-effector poses with associated elbow configurations. Edges represent feasible transitions, including Cartesian unit steps and diagonal connections, while disallowing discontinuous elbow jumps to force smooth trajectories. Specifically, for each node, there are six edges representing Cartesian connections (unit steps along a single axis) and a further 20 edges that represent the possible diagonal motions.

We store nodal information in four bit arrays: (1) node parents, (2) node distances from the starting node, (3) whether the node has been checked, and (4) whether it is invalid. As an example, discretizing the workspace into 1 cm^3 voxels with a 50 cm robot radius and 8 elbow configurations requires only $\sim$100 MB of storage. Increasing the resolution to 1 mm^3 voxels under the same conditions increases storage to $\sim$100 GB.

*2) Hash-Distributed Multi-Goal A**

We employ a hash-distributed multi-goal A* search adapted for MPI. Unlike RRT-based planners, which rely on random sampling and can require many iterations to converge [15], A* is deterministic. Additionally, traditional HDA* search has been successfully implemented on HPC resources [18]. Each process expands nodes from its local heap, distributes neighbours, and synchronizes with an MPI `Allreduce` to detect the first collision-free solution. The heuristic prioritizes minimal elbow displacement before end-effector distance, keeping evasive motions compact. Algorithm 1 summarizes the procedure. The heuristic guiding the search is defined as:

$$h(s,g) = \frac{1}{V}\Big(|e_s - e_g| + 0.3\big(|e_s - m| + |e_g - m|\big)\Big) \; + \; E(s,g) \quad (1)$$

with e_s, e_g the start/goal elbows, m the midpoint, V the elbow discretization count, and E the approximate Euclidean distance. This weighting prioritizes elbow variation over end-effector displacement, minimizing deviation from the robot's current pose.

3) Trajectory Reconstruction

Once a valid node is found, the solver reconstructs a path of waypoints including end-effector positions, elbow configurations, and joint angles. This trajectory is returned to the robot in compact form.

C. Trajectory Execution on Robot

Upon receiving waypoints, the robot interpolates them with the Ruckig library [19] to ensure smooth velocity and acceleration profiles. Execution proceeds under a 1 kHz proportional–derivative (PD) controller that converts tracking errors into motor torques. To reduce latency, the controller updates targets dynamically—advancing to the next waypoint once $\sim$20% of the current path is completed. After finishing an avoidance maneuver, the robot holds its final pose and resumes monitoring for subsequent projectiles.

D. Timing Breakdown

To evaluate responsiveness, computation is partitioned into three measurable stages:

Algorithm 1 Multi-Goal MPI A* Solver

Require: Graph G, source & target nodes s, t, Variations v
Ensure: Path from s to nearest valid node

```
Initialize invalids, checked, distances, parents
Insert s into open set with f(s) = 0
while open set not empty and path not found do
    u ← extract node with minimum f from open set
    if u is valid and better than the current best then
        Update best node and path owner
    for all neighbours n of u do
        g ← g(u) + edge weight
        if g < g(n) then
            Update parents[n] ← u
            f(n) ← g + h(n, t)
            Insert n into open set
    MPI Allreduce: Check if target is found across processes
    if a target is found then
        MPI Allreduce: Determine which process has best node
        if RECONSTRUCTPATH(best node, source) then
            return path
        else
            Reset search and continue
```

1) **Unpack:** voxel decompression and synchronization across processes.
2) **Search:** the core hash-distributed A* traversal and trajectory identification.
3) **Backtrace:** reconstruction of waypoints for execution on the robot.

This breakdown allows attribution of latency to communication overhead versus algorithmic complexity, a distinction that is critical for assessing the feasibility of HPC in real-time robotic control.

VI. RESULTS

We evaluate SHARP along three axes: (i) *responsiveness* (planning latency and its components), (ii) *avoidance success rate* on valid shots, and (iii) *sensitivity to communication latency* (local vs. remote HPC). Experiments were run on both a local (on-campus) cluster and a remote cluster (§IV-B.4). We report aggregate success across all valid projectiles and per-trial rates, and decompose compute time into *unpack*, *search*, and *backtrace* to attribute delays to communication versus algorithmic complexity (§V). We also state observed limitations arising from hardware constraints and projectile variability.

A. Experimental Parameters

1) Blaster and Timing Budget

Average exit speed of the projectile was $\sim$ 10 m/s with the blaster at $\sim$ 4.5 m from the robot, as this was the limit of the capture system, yielding a minimum time-of-flight of $\sim$ 450 ms. For context, simple human visual reaction time is $\sim$ 180–200 ms [20]. This establishes a conservative envelope within which SHARP must complete sensing, offload, planning, return, and execution.

2) Voxel Specification

The workspace was discretized at 1 cm^3 resolution with a 75 cm radius (90% of the robot's reach), and elbow redundancy discretized to the configuration count used in §V, which yielded reliable IK solutions and timely evasive motion.

3) Hardware

The robot manipulator was connected to a Dell G15 laptop (Ryzen 7 6800H) via Ethernet, allowing for use of the Franka Control Interface for 1 kHz communication [17]. For our experiments, we ran HPC computations on single-compute-node clusters, both locally and remotely (see Sec. IV-B.4), and on the local, robot-connected laptop (Dell G15) for comparison. The HPC computation can run on more compute nodes; however, for initial testing, we use only one. Resource sharing is not an issue; during setup, the required nodes are allocated for the user and cannot be used by others. Unfortunately, the laptop was unable to execute the full graph search at all due to insufficient memory and cores to run the full tests.

4) Outcome Definitions of Projectile Trajectories

Projectiles were fired at the robot and classified as *hits*, *avoidances*, or *invalids*. Hits are projectiles that contact the robot after a valid HPC request; avoidances are those that miss after a valid request; and invalids include shots without a valid request, which include those striking the immovable base of the robot, having erratic trajectories, or that are misfired. Invalids were excluded from analysis, leaving only valid shots to calculate avoidance rates. Because trials contained differing numbers of valid shots—due to reclassification during video review and occasional attempts to offset invalid base hits—the total varied across trials. This introduces a weighting effect: trials with more valid shots contribute more heavily to aggregate rates. Accordingly, we report both per-trial success rates and aggregate success across all valid projectiles (Tables I and III).

B. Local HPC Results

Table I summarizes the results of 12 trials conducted using the local (on-campus) HPC cluster. Each trial consisted of a full sequence of projectile firings from the blaster toward the robot under fixed experimental conditions (blaster orientation, Vicon tracking setup, and HPC connection). Twelve trials were chosen to provide sufficient repetitions for averaging while keeping the experiment duration manageable. Within each trial, multiple shots were fired to capture variability in projectile speed and robot response; grouping shots into trials ensured that systematic effects (e.g., bias in path finding, cluster load) were captured across repeated runs. Bias may occur because the robot uses the same starting location throughout all trials, as shown in Figure 4. Across all valid shots, the robot achieved an average avoidance success rate of **84%**, demonstrating that the system can—but not perfectly—react and avoid dynamic projectiles. Post-experiment analysis confirmed that hits occurred either due to misalignment between the real and virtual blaster or because the robot was unable to react in time due to motor speed limitations.

Performance varied across trials, largely due to inconsistent projectile speed and blaster alignment errors. Alignment errors arose from mismatches between the simulated Vicon system and the actual setup, leading to deviations from the intended aim point. In particular, L6 and L12 showed elevated invalid shot counts from misfires or poor alignment, which lowered their individual success rates. These inconsistencies made trajectory prediction more difficult and introduced additional uncertainty into the avoidance task. Another limitation stemmed from the simplified projectile model. While the system assumed a linear trajectory, the foam darts frequently followed parabolic or erratic paths, reducing prediction accuracy. This mismatch highlights the need for a more realistic flight model or real-time projectile tracking to further improve dodge consistency under variable speeds.

TABLE I

THE NUMBER OF PROJECTILE COLLISIONS WITH THE ROBOT (HITS), AVOIDANCES, INVALIDS, TOTAL PROJECTILES, AND TOTAL VALID PROJECTILES WITHIN TESTING WHILE USING LOCAL HPC.

Trial	Hits	Avoidance	Invalid	Total	Valid Total	Avoidance Rate
L1	0	11	2	13	11	100%
L2	2	12	7	21	14	86%
L3	2	13	1	16	15	87%
L4	1	12	5	18	13	92%
L5	3	10	3	16	13	77%
L6	2	4	13	19	6	67%
L7	0	11	10	21	11	100%
L8	1	8	10	19	9	89%
L9	5	13	7	25	18	72%
L10	0	13	6	19	13	100%
L11	4	15	4	23	19	79%
L12	3	4	9	16	7	57%
Total	**23**	**126**	**77**	**226**	**149**	**85%**
Avg	**1.9**	**10.5**	**6.4**	**18.8**	**12.4**	**84%**

Table II presents the compute timings for local cluster runs. Local, round-trip cluster connections used throughout the trials were measured at less than 1 ms. On average, the end-to-end compute loop required 22.9 ms, well below human visual reaction thresholds (∼200 ms) [20]. The majority of this time was spent in the search phase (∼20 ms), with voxel unpacking averaging just over 2 ms and backtracing under 0.2 ms. Only one trial (L5) experienced a notable spike (∼49.8 ms), but even this outlier remained substantially faster than human reflexes.

C. Remote HPC Results

Table III summarises projectile outcomes for tests executed on the remote cluster. Across all viable shots, the mean avoidance rate was **88%**. As with local HPC results, variability across trials was influenced by invalid shots (e.g., misfires and alignment issues), which reduced the number of viable attempts in several tests.

Timing measurements for the remote cluster (Table IV) show average round-trip latencies of 16 ms (wired) and 18 ms (wireless) to a nearby cluster (∼300 km), and 57 ms (wired) and 70 ms (wireless) to a different, long-distance cluster (∼4000 km). From this point on, the term "remote cluster" refers to the short-distance cluster. Average per-request totals were **29.98 ms**, decomposed into unpacking (**4.01 ms**), search (**24.39 ms**), and backtrace (**0.14 ms**). Compared to

TABLE II

AVERAGE TIME PER TRIAL TO UNPACK, SEARCH AND PERFORM A BACKTRACE PER REQUEST FROM THE ROBOT USING LOCAL HPC.

Trial	Avg Unpack [ms]	Avg Search [ms]	Avg Backtrace [ms]	Avg Total [ms]
L1	2.210	22.712	0.144	25.587
L2	2.209	13.127	0.111	15.940
L3	2.196	14.010	0.114	16.828
L4	2.191	12.827	0.109	15.644
L5	2.168	46.936	0.157	49.760
L6	2.251	14.523	0.117	17.419
L7	2.205	16.161	0.128	18.969
L8	2.197	13.427	0.116	16.260
L9	2.176	20.731	0.133	23.561
L10	2.203	17.859	0.144	20.717
L11	2.197	22.630	0.144	25.471
L12	2.320	24.776	0.149	27.772
Avg	**2.206**	**20.029**	**0.131**	**22.874**

TABLE III

THE NUMBER OF PROJECTILE COLLISIONS WITH THE ROBOT (HITS), AVOIDANCES, INVALIDS, TOTAL PROJECTILES, AND TOTAL VALID PROJECTILES WITHIN TESTING WHILE USING REMOTE COMPUTE.

Trial	Hits	Avoidance	Invalid	Total	Valid Total	Avoidance Rate
R1	1	14	1	16	15	93%
R2	2	17	2	21	19	89%
R3	2	8	9	19	10	80%
R4	2	7	12	21	9	78%
R5	1	8	14	23	9	89%
R6	3	11	10	24	14	79%
R7	1	11	10	22	12	92%
R8	0	16	7	23	16	100%
R9	0	9	12	21	9	100%
R10	2	10	8	20	12	83%
R11	0	13	10	23	13	100%
R12	2	7	14	23	9	78%
Total	**16**	**131**	**109**	**256**	**147**	**89%**
Avg	**1.3**	**10.9**	**9.1**	**21.3**	**12.3**	**88%**

local compute, unpacking increased by approximately 2 ms on average, the search phase increased by roughly 4–5 ms, and backtrace remained similar. Occasional longer searches were observed (e.g., R4 and R6), with average search times exceeding 40 ms.

TABLE IV

AVERAGE TIME PER TRIAL TO UNPACK, SEARCH AND PERFORM A BACKTRACE PER REQUEST FROM THE ROBOT USING REMOTE COMPUTE.

Trial	Avg Unpack [ms]	Avg Search [ms]	Avg Backtrace [ms]	Avg Total [ms]
R1	4.102	19.722	0.131	24.882
R2	4.120	20.308	0.132	27.758
R3	4.065	10.270	0.113	15.464
R4	4.095	44.672	0.156	49.906
R5	4.097	18.393	0.122	23.528
R6	3.922	48.199	0.122	53.190
R7	3.949	17.991	0.142	22.975
R8	3.764	17.466	0.138	22.271
R9	3.870	24.374	0.149	29.370
R10	3.772	32.185	0.179	40.618
R11	3.984	16.736	0.115	21.771
R12	4.288	20.952	0.142	26.329
Avg	**3.993**	**24.466**	**0.136**	**30.041**

D. Key Findings and Practical Envelope

We find that HPC planning fits comfortably within reaction budgets. Mean planning times were 22.9 ms (local) and 30.0 ms (remote), with rare spikes < 55 ms. Even accounting for short-haul RTT ($\sim$ 16 ms), SHARP leaves substantial time for robot execution relative to ~ 450 ms dart flight and ~ 200 ms human reaction.

Another key finding is that projectile avoidance success appears to be robust across local/remote HPC environments. Aggregate valid-shot success was 85% (local) vs. 89% (remote). Misses primarily reflect physical factors (dart variability, mocap alignment) and execution limits, not compute time.

Lastly, it appears that latency sensitivity is dominated by network RTT, not unpack/backtrace. Across both clusters, *search* dominates compute, while *unpack* and *backtrace* are negligible. This suggests further planner speedups yield diminishing returns unless perception or network RTT/jitter are reduced.

E. Limitations

Our evaluation focused on compute-side performance and did not capture the full end-to-end loop, which would also include sensing delays, message serialization, network jitter, and controller execution. The use of a motion capture system provided unusually clean state information compared to typical onboard sensors, reducing noise but limiting generalizability. In addition, the workspace and obstacle density were modest, keeping the problem size well within the capacity of a single HPC node. Finally, our projectile model assumed straight-line trajectories, while the foam darts often followed parabolic or erratic paths, reducing prediction accuracy.

These factors should temper over-generalization of the reported success rates, but they do not affect the central finding: HPC planning consistently completed in the tens-of-milliseconds range, enabling high avoidance success on valid shots.

VII. DISCUSSION

A. Compute Performance and Latency in Context

Across local and short-haul remote trials, HPC planning completed in the tens of milliseconds, with *search* dominating and *unpack/backtrace* negligible (Tables II, IV). Projectile avoidance success on valid shots was similar for both clusters, averaging 84% and 88%, respectively (Tables I, III). This indicates that once an avoidance request is formed, HPC-side computation fits comfortably within typical reaction budgets, even when considered alongside other delays (e.g., sensing, message (de)serialization, cluster scheduling), leaving time for execution. The main exogenous factor is network RTT: sub-ms locally vs. $\sim$16 ms short-haul (and $\sim$60 ms intercontinental for reference). Our results suggest offloading remains viable while RTT stays in the tens-of-milliseconds regime, consistent with wired industrial Ethernet and emerging low-jitter wireless (5G/6G) [21], and well within a $\sim$200 ms human reaction envelope [20].

B. Sources of Variability

Outcome variability stems primarily from the physical testbed rather than compute: dart speed fluctuations, small

mocap alignment errors, and a straight-line flight assumption that underfits parabolic/erratic darts. These factors raise invalids and make borderline cases harder, even when plans return quickly. Occasional search spikes align with hard grid regions (tight passages with many nodes at similar admissible f), not with networking or synchronization; the low variance of *unpack/backtrace* supports this view.

C. Key Insights

Based on the above discussion, we reiterate the contributions of our work. First, we demonstrate an HPC-as-a-service dataflow for robotics, implemented end-to-end with per-stage timing to localize latency. Second, we provide empirical evidence of responsiveness: compute consistently remained under 50 ms, even during worst-case spikes, while avoidance success ranged from 84–88% across both local and remote clusters. Third, we adapt a deterministic multi-goal A* to a 4D workspace with elbow redundancy, showing that bounded runtimes can be achieved through parallel search and that occasional spikes are linked to workspace complexity rather than system noise. Finally, we introduce a simple evaluation template—reporting unpack, search, and backtrace alongside success rates—that can be replicated by other HPC-driven robotic systems.

Taken together, these contributions show that: (1) HPC planning consistently completes in the tens-of-milliseconds range, leaving substantial budget for execution even under remote conditions. (2) Avoidance success rates above 80% across both local and remote trials indicate robustness despite variability in projectile flight and sensing alignment. (3) Communication, not computation, is the dominant bottleneck; once RTT exceeds tens of milliseconds, responsiveness degrades regardless of algorithmic speed. These insights provide concrete evidence that HPC offloading is not "too slow" for reactive robotics, but instead represents a viable pathway when problem structure is parallelizable, and network conditions are stable.

D. Practical Deployments

While our proof-of-concept shows that HPC can deliver sub-human reaction times in dynamic avoidance tasks, questions remain about deployment in real-world environments and HRI contexts.

At the scale of our demonstration ($\sim$3 GB), a single compute node was sufficient. The benefits of HPC become more pronounced as problem sizes grow: larger voxel grids, more elbow configurations, or fleets of collaborative robots quickly exceed onboard compute limits. Mobile platforms already encounter this during computationally intensive tasks such as SLAM, grasp planning, or high-dimensional motion planning. Compared to FogROS2 [6], which reduced motion planning to 1.2 seconds on 96 cores, our millisecond-level results on 64 cores suggest that HPC could enable real-time SLAM and similar workloads.

In more complex scenarios, HPC parallelism scales naturally, allowing multi-robot systems to coordinate trajectories in dynamic environments without sacrificing responsiveness [9]. Established programming models such as MPI make it possible to expand planning and perception workloads without redesigning algorithms, and even the smallest system on the TOP500 list provides over 8,000 cores—far beyond the requirements of typical robotics workloads [22].

Thus, the feasibility demonstrated on a single manipulator extends to broader applications. In factories, centralized clusters could coordinate fleets of cobots while reducing per-robot hardware costs. In healthcare, shared HPC services could enable safe navigation for multiple service robots operating around staff and patients. More generally, any setting that demands bursty, high-throughput planning—such as batched inverse kinematics or swarm coordination—stands to benefit from this architecture.

E. Future Work

Two directions emerge from this work. First, the HPC algorithms used in this work remain largely unoptimized; replacing global synchronizations (e.g., MPI `Allreduce`) with non-blocking communication could reduce stalls, and scaling beyond a single 64-core node will clarify how performance changes once inter-node communication dominates. Evaluating scalability at larger problem sizes is a key next step.

Second, practical deployments will require hybrid architectures: local controllers can handle low-level stability and reflexive safety, while HPC provides global planning for faster or more unpredictable obstacles. Such layered control mirrors established hierarchies in robotics but extends their reach by leveraging supercomputing resources. Exploring how best to overlap local reflexes with HPC offloading remains an open question.

F. Novelty Beyond Cloud Robotics

Prior work in cloud robotics has often concluded that offloading is too slow for real-time control, or has presented performance gains in offline or asynchronous contexts. SHARP moves this debate forward by showing that the question is not whether cloud or HPC is categorically "too slow," but rather how parallelism and network latency interact to define a quantitative envelope of reaction time. By reporting planning latencies in the tens of milliseconds, we demonstrate that HPC offloading can meet budgets well below human reaction thresholds when round-trip times remain within the same order of magnitude. Importantly, our use of a deterministic, MPI-based multi-goal A* contrasts with earlier sampling-based approaches (e.g., RRT), providing bounded runtimes and more predictable behaviour in safety-critical settings.

G. Practical Relevance to Human–Robot Interaction

Although projectile dodging may appear stylized, we use it as a stress test: if the system can react to foam darts travelling at $\sim$10 m/s, it is over-provisioned for the slower, structured motions typical of human coworkers or mobile obstacles. This directly reflects applications such as industrial cobots avoiding encroaching workers or hospital service robots navigating around staff and patients, where reactive motion is essential even at modest speeds. Robots that autonomously and transparently move around people can reduce user cognitive load while improving safety. Our results indicate that HPC offloading can enhance responsiveness in such domains, where tens of milliseconds can significantly affect both safety and user trust.

VIII. Conclusion

We have introduced SHARP, an HPC-as-a-service system that offloads time-critical planning to a compute cluster and returns executable evasive manoeuvres within tens of milliseconds. In a dynamic avoidance task, HPC computation averaged 22.9 ms (local) and 30.0 ms (remote), with an additional 16 ms RTT latency; unpack/backtrace costs were negligible, and avoidance success rates were 84% (local) and 88% (remote). These results show that, even with tens-of-milliseconds latency and jitter, HPC-based parallel planning can meet reaction budgets well below human visual response times, while a deterministic, grid-aligned multi-goal MPI A* provides bounded, parallelizable runtimes suited to discretized environments.

Practically, our results indicate that HPC offloading is viable for reactive behaviour when (i) the planning subproblem is highly parallel (e.g., batched IK/collision checks, multi-goal grid search), and (ii) network RTT/jitter are predictably low; under these conditions, compute time becomes a small slice of the end-to-end budget, leaving room for sensing and control. Architecturally, the work motivates hybrid designs that keep last-ditch reflexes at the edge while reserving HPC for bursty, high-throughput search, with communication overlapped against expansion to smooth spikes. Methodologically, we argue that deterministic search with admissible heuristics offers stable, certifiable timing envelopes that are easier to distribute and reason about than stochastic sampling in tight control loops. Scientifically, the paper provides the first (to our knowledge) quantitative envelope that ties planner speedups to concrete latency budgets for *live* control, reframing the cloud/HPC debate from a binary "too slow vs. fast enough" to a measured trade-off space defined by parallelism and network conditions. This establishes a template for end-to-end evaluation—reporting per-stage timing and success on viable shots—that other HPC-controlled robotics systems can adopt, thereby helping the community converge on comparable, auditable claims about time-sensitive offloading. Future work will quantify complete end-to-end latency under onboard perception, incorporate ballistic/drag or real-time projectile tracking, explore GPU-accelerated collision/IK, and study robustness under higher jitter and multi-robot settings. Together, these directions move HPC-driven planning from feasibility to dependable, real-time practice in shared human–robot workspaces.

Acknowledgement

We would like to thank members of MItHRIL, CAESAR and Ingenuity Labs at Queen's University for their support.

References

[1] S. W. Ali, A. Angelopoulos, D. Massey, S. Haddix, A. Georgiev, J. Goh, R. Wagle, P. Sarathy, J. H. Anderson, and R. Alterovitz, "On the Necessity of Real-Time Principles in GPU-Driven Autonomous Robots."

[2] S. M. Neuman, B. Plancher, B. P. Duisterhof, S. Krishnan, C. Banbury, M. Mazumder, S. Prakash, J. Jabbour, A. Faust, G. C. De Croon, and V. J. Reddi, "Tiny Robot Learning: Challenges and Directions for Machine Learning in Resource-Constrained Robots," *Proc. - IEEE Int. Conf. on Artif. Intell. Circuits and Syst., 2022*, pp. 296–299, 2022, iSBN: 9781665409964.

[3] D. Rakita, B. Mutlu, and M. Gleicher, "STAMPEDE: A discrete-optimization method for solving pathwise-inverse kinematics," *Proc. - IEEE Int. Conf. on Robot. and Automat.*, pp. 3507–3513, May 2019, iSBN: 9781538660263.

[4] R. Natarajan, S. Mukherjee, H. Choset, and M. Likhachev, "PINSAT: Parallelized Interleaving of Graph Search and Trajectory Optimization for Kinodynamic Motion Planning," *IEEE Int. Conf. on Intell. Robots and Syst.*, pp. 13 907–13 914, 2024, iSBN: 9798350377705.

[5] A. S. Seisa, V. N. Sankaranarayanan, G. Damigos, S. G. Satpute, and G. Nikolakopoulos, "Cloud-Assisted Remote Control for Aerial Robots: From Theory to Proof-of-Concept Implementation," *Proc. - 2025 IEEE 25th Int. Symp. on Cluster, Cloud and Internet Comput. Workshops, 2025*, pp. 171–176, 2025, iSBN: 9798331509385.

[6] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. Llontop, D. Xu, C. Buscaron, J. Kubiatowicz, I. Stoica, J. Gonzalez, and K. Goldberg, "FogROS2: An Adaptive Platform for Cloud and Fog Robotics Using ROS 2," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, May 2023, pp. 5493–5500. [Online]. Available: https://ieeexplore.ieee.org/document/10161307

[7] N. Tahir and R. Parasuraman, "Edge Computing and Its Application in Robotics: A Survey," *Journal of Sensor and Actuator Networks*, vol. 14, no. 4, p. 65, Aug. 2025. [Online]. Available: https://www.mdpi.com/2224-2708/14/4/65

[8] K. E. Chen, Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, and K. Goldberg, "FogROS: An Adaptive Framework for Automating Fog Robotics Deployment," *IEEE Int. Conf. on Automat. Sci. and Eng.*, vol. 2021-August, pp. 2035–2042, Aug. 2021, iSBN: 9781665418737.

[9] L. Camargo-Forero, P. Royo, and X. Prats, "Towards high performance robotic computing," *Robot. and Auton. Syst.*, vol. 107, pp. 167–181, Sep. 2018.

[10] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "DAvinCi: A cloud computing framework for service robots," *Proc. - IEEE Int. Conf. on Robot. and Automat.*, pp. 3084–3089, 2010, iSBN: 9781424450381.

[11] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A Cloud Robotics Platform," *IEEE Trans. on Automat. Sci. and Eng.*, vol. 12, no. 2, pp. 481–493, Apr. 2015.

[12] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, "Cloud robotics: Current status and open issues," *IEEE Access*, vol. 4, pp. 2797–2807, 2016.

[13] Z. Du, L. He, Y. Chen, Y. Xiao, P. Gao, and T. Wang, "Robot Cloud: Bridging the power of robotics and cloud computing," *Future Gener. Comput. Syst.*, vol. 74, pp. 337–348, Sep. 2017.

[14] L. Abuelsamen, M. E. Student, H.-W. Lu, W. Tang, S. Priyadarshini, and G. Gomes, "Industrial Robot Motion Planning with GPUs: Integration of cuRobo for Extended DOF Systems," Aug. 2025. [Online]. Available: https://arxiv.org/abs/2508.04146v2

[15] J. J. Kuffner and S. M. La Valle, "RRT-connect: an efficient approach to single-query path planning," *Proc. - IEEE Int. Conf. on Robot. and Automat.*, vol. 2, pp. 995–1001, 2000.

[16] J. Ichnowski, W. Lee, V. Murta, S. Paradis, R. Alterovitz, J. E. Gonzalez, I. Stoica, and K. Goldberg, "Fog Robotics Algorithms for Distributed Motion Planning Using Lambda Serverless Computing," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4232–4238, May 2020, iSBN: 9781728173955.

[17] "Franka Control Interface Documentation." [Online]. Available: https://frankarobotics.github.io/docs/

[18] A. Kishimoto, A. Fukunaga, and A. Botea, "Evaluation of a simple, scalable, parallel best-first search strategy," *Artificial Intelligence*, vol. 195, pp. 222–248, Feb. 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370212001294

[19] "Ruckig - Motion Generation for Robots and Machines." [Online]. Available: https://ruckig.com/

[20] "Reaction times to sound, light and touch - Human Homo sapiens - BNID 110800." [Online]. Available: https://bionumbers.hms.harvard.edu/bionumber.aspx?s=n&v=4&id=110800

[21] P. Li, J. Fan, and J. Wu, "Exploring the key technologies and applications of 6G wireless communication network," *iScience*, vol. 28, no. 5, p. 112281, May 2025.

[22] "TOP500 List - June 2025 | TOP500." [Online]. Available: https://top500.org/lists/top500/list/2025/06/?page=1