Collision Prediction for Robotics Accelerators

Deval Shah
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
devalshah@ece.ubc.ca

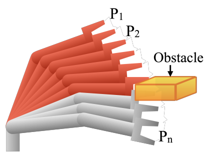
Tor M. Aamodt
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
aamodt@ece.ubc.ca

***Abstract*—Motion planning in dynamic environments is an important task for autonomous robotics. Emerging approaches employ neural networks that can learn by observing (e.g., human) experts. Such motion planners react to the environment by continually proposing candidate paths to reach a goal. Some of these candidate paths may be unsafe–i.e., cause collisions. Hence, proposed paths must be checked for safety using collision detection. We observe that $25\% - 41\%$ of the resulting collision detection queries can be eliminated if we can anticipate which queries will return an unsafe result. We leverage this observation to propose a mechanism, COORD, to predict whether a given robot position (pose) along a proposed path will result in a collision. By prioritizing the detailed evaluation of predicted collisions, COORD enables quickly eliminating invalid paths proposed by neural network and other sampling based motion planners. COORD does this by exploiting the physical spatial locality of different robot poses and using simple hashing and saturating counters. We demonstrate the potential of collision prediction on different computation platforms, including CPU, GPU, and ASIC. We further propose a hardware collision prediction unit (COPU), and integrate it with an existing collision detection accelerator. This results in an average $17.2\% - 32.1\%$ decrease in number of collision detection queries across different motion planning algorithms and robots. When applied to a state-of-the-art neural motion planner [41], COORD improves performance/watt by $1.23\times$ on average for motion planning queries of varying difficulty levels. Further, we find that the benefits of collision prediction grow as the compute complexity of motion planning queries increases and provides $1.30\times$ improvement in performance/watt in narrow passages and cluttered environments.**

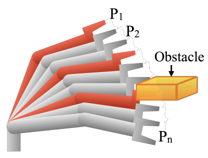
***Index Terms*—Robotics, Hardware acceleration, Motion planning, Collision detection, Collision prediction**

I. Introduction

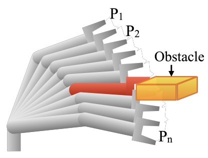
Motion planning is a fundamental computing task in autonomous robotics. Given the current state of the environment and an end goal, motion planning finds a feasible and safe path to reach the goal. Motion planning is used to enable object manipulation, full-body movement, and navigation (e.g., self-driving cars). The market for autonomous robots that can aid humans is gaining increased attention in part due to rapid progress in machine learning. For example, robots may enable improvements in health-care and assistance to older adults. In such settings these robots would be required to navigate safely in cluttered environments while reacting to dynamic obstacles in real-time [25], [32]. Collision detection is used to ensure the safety of a robot's movements and is the most



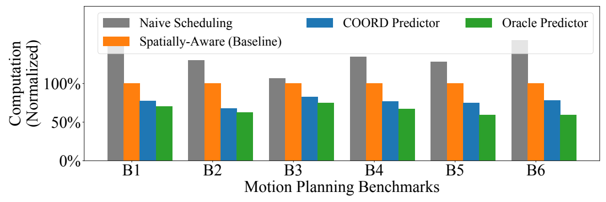
(a) Naive scheduling



(b) Spatially-aware scheduling



(c) Oracle scheduling



(d)

Fig. 1: (a)-(c) represent different scheduling orders of robot link-environment collision detection queries (CDQs) for a motion-environment collision check. Here, red-colored parts correspond to CDQs executed, and gray-colored parts correspond to CDQs skipped as a collision is found. (d) Comparison of the proposed collision predictor COORD with a naive sequential scheduler, baseline Spatially-Aware Scheduler [43], and Oracle collision predictor. The plot compares the reduction in collision detection computation across a range of motion planning problems B1-B6.

time and energy-consuming part of several motion planning approaches [5], [35], [39], [41].

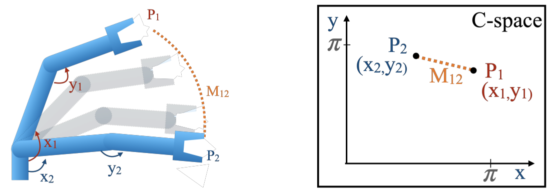
Collision detection is a computationally intensive task that must be performed in real-time to ensure safety and effectiveness of an autonomous robot. While prior work has explored CPU and GPU-based acceleration of motion planning and collision detection [2], [5], [9], [28], [39], [40], achieving real-time operation in highly dynamic environments on such hardware can be challenging. To improve robot productivity specialized hardware has been proposed to accelerate collision detection [3], [20], [31], [34], [43], [45], [48]. These works have primarily focused on accelerating an individual collision detection query. Shah et al. [43] proposed a scheduling policy to reduce redundant computation by reordering collision detection queries. We find these approaches often perform more collision detection queries than may be necessary especially in cluttered environments.

Figure 1 illustrates the source of this inefficiency and hints at how it may be overcome. In Figure 1a a motion of a robot arm is divided up into n sample poses, P_1 to P_n, checked by running repeated *Collision Detection Queries (CDQ)* starting from the beginning pose, P_1, sequentially towards the goal, P_n. The red highlighted poses indicate evaluated CDQs. Shah et al. [43] improve upon this "naive" ordering by employing a spatially-aware ordering to reduce the number of evaluated CDQs as shown in Figure 1b. In this paper we aim to further reduce the number of CDQ evaluations by predicting which of the sample points, if any, result in a collision. In the ideal case, an unsafe motion would then require only a single CDQ evaluation to rule out an unsafe motion as illustrated in Figure 1c. Our limit study, using an "oracle" predictor, shows that the number of CDQ evaluations performed during motion planning can potentially be reduced by $25.1\% - 40.7\%$ versus spatially-aware scheduling [43].

Our key insight, is that the physical spatial locality of CDQ outcomes can be leveraged to enable accurate and implementable collision prediction. We predict a pose to be in collision if a "nearby" pose was found to collide with the environment. As the robot and environment are assumed to be dynamic, collision predictions are based only on CDQ outcomes since the last environment measurement (i.e., mapping). We design a hashing function that bins physically nearby robot poses together and enables a small and densely populated collision history table. We further study the impact of environment characteristics, prediction strategies, and hash table update strategies on collision prediction precision and recall. We leverage insights from these studies to propose *COORD*, a collision prediction approach to reduce computation for robot motion planning.

We integrate COORD with CPU and GPU-based collision detection and analyze its impact on execution time, computation, and collision prediction overheads. Further, several collision detection hardware accelerators have been studied [3], [17], [34], [43] to meet the realtime requirements and energy constraints of robotic systems. We propose a hardware *Collision Prediction Unit (COPU)* that can be tightly integrated with a specialized accelerator for collision detection. Figure 1d compares the computation performed by COPU, baseline accelerator [43], and Oracle predictor (Methodology described in Section V). COPU results in $17.2\% - 32.1\%$ reduction in the collision detection queries on average across different motion planning algorithm-robot combinations. The proposed COPU increases performance/watt and performance/mm^2 of the baseline accelerator by $1.23\times$ and $1.11\times$, respectively, while providing $1.18\times$ higher collision detection throughput. Further, the proposed Collision Prediction unit results in $23.4\% - 42.9\%$ reduction in the collision detection queries for demanding benchmarks across different motion planning algorithm-robot combinations. Finally, we discuss the scope and limitations of the proposed approach and demonstrate its integration with two more accelerators for potential performance improvement. In summary, we make the following contributions in this work:

- We demonstrate the potential of collision prediction to reduce collision detection queries during motion planning.
- We propose a collision history table and explore different hashing approaches for collision prediction including *COORD*, a hashing function that groups physically nearby positions of a robot.
- We evaluate *COORD* with CPU and GPU-based execution of collision detection.
- We propose a hardware Collision Prediction Unit (COPU) suitable for integration within hardware accelerators for collision detection and evaluate it on various robots, environments, and motion planning algorithms.



(a) Robot in physical space (b) Robot in C-space

Fig. 2: (a) and (b) represents a robot with 2 DOFs in the physical and C-space, respectively. The C-space is two-dimensional as the robot has 2 DOFs. P_1 and P_2 represent the poses of the robot, and M_{12} represents the motion of the robot between these two poses. A pose P is represented using the values of its joints x and y, which corresponds to a point in the C-space with coordinates (x,y).

II. Background and Motivation

This section briefly summarizes relevant background on robot motion planning and collision detection.

A. Motion Planning

An autonomous robot's computational tasks are typically divided into a pipeline consisting of perception, motion planning, and control stages. Motion planning often dominates [4] and is used to find a collision-free and safe path for a robot from its current position to a goal position. Its complexity increases exponentially with the *Degrees of Freedom (DOFs)* of a robot. Classical motion planning algorithms (e.g., Brooks and Lozano-Perez [7]) operate in the configuration space (C-space) of the robot. The C-space of the robot has the same dimensions as its DOFs, where each dimension represents the extent of its DOF (e.g., the angle of joint for a rotational DOF). Figure 2a-2b gives an example of the physical space and C-space of a robot with two DOFs. A pose/position of the robot is specified by the values of its DOFs and is represented as a point in the C-space (e.g., P1 and P2 in Figure 2b). Similarly, a motion between two poses is represented as a line in the C-space (e.g., M12 in Figure 2b).

After decades of research, a wide range of algorithms exist for motion planning, including end-to-end learning-based, optimization-based, and sampling-based motion planning approaches. In this paper, we focus on sampling-based motion planning [6], [14], [15], [22], [26], [41], [50]. This approach is favored for robots with high DOF because it avoids the complexity of mapping environmental obstacles into C-space while providing a balance between motion planning quality and runtime.

A sampling-based motion planning algorithm is divided into two parts: sampling and collision checks. Figure 3a-3b give an example of motion planning using MPNet [41]. A

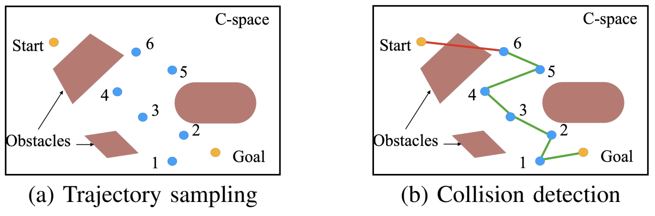


Fig. 3: (a) and (b) represent trajectory sampling and collision detection in MPNet sampling-based motion planning algorithm [41]. Collision detection is performed in the physical space, and a motion/pose from the C-space is transformed in the physical space for collision detection.

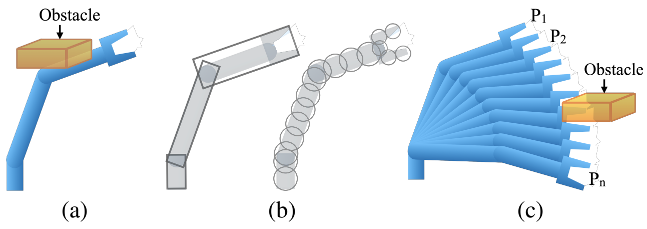


Fig. 4: (a) represents pose-environment collision detection for a robot, (b) OBB-based (left) and Sphere-based (right) representation of a robot's physical space, and (c) represents collision detection for a motion from P_1 to P_n, which is discretized into multiple poses to be checked for collision with the environment.

neural network is used for sampling and provides intermediate milestones in the C-space between the start and goal positions for determining an approximate trajectory (Figure 3a). Thus the trajectory consists of multiple short "motions" (i.e., lines in the C-space). Collision detection is performed to determine which sampled poses and motions from this trajectory are collision-free to further refine the trajectory (Figure 3b). Thus, key computation kernels for typical sampling-based motion planning algorithms include pose-environment and motion-environment collision detection queries. These queries are performed multiple times for a single motion planning query.

B. Collision Detection

Collision detection, the most time-consuming part of sampling-based motion planning [4], [27], determines whether a given pose or short motion will cause part of the robot to occupy space currently occupied by an object in the environment. Figure 4a gives an example of collision detection. Typically, the physical space occupied by a robot pose is conservatively approximated using a set of simple volumes that collectively bound the space filled by the actual robot. Examples of simple volumes include *Oriented Bounding Boxes (OBBs)* [3], [43] or spheres [47] (Figure 4b). Similarly, the environment is conservatively represented using simple volumes that bound the space actually occupied by obstacles. An individual *Collision Detection Query (CDQ)* determines whether a given bounding volume for the robot's pose intersects any bounding volume for the environment. Collision detection for a robot pose can be performed by evaluating a CDQ for each bounding volume used to represent the robot pose.

Collision detection for a motion can be performed using either continuous [8], [47] or discrete [13] approaches. In the discrete approach, illustrated in Figure 4c, collision detection for a motion is performed by uniformly dividing that motion into multiple discrete poses and performing collision detection for each pose. This approach can be sensitive to the resolution (step size) used for discretization. In contrast, continuous collision checking typically first creates a volume representing the space occupied by the robot during the motion then performing collision detection between this volume and the environment, which can be compute-intensive. Recent continuous collision checking approaches [47] using pose-environment collision queries create a dependency between these queries, and require finding the distance to closest obstacle. In this paper we explore discrete collision detection approaches as these typically provide parallelism between multiple pose-environment collision checks, and require simple Boolean pose-environment collision check result (`True` or `False`).

During operation in dynamic environments the motion planning pipeline is repeatedly executed. During a single execution obstacle occupancy is typically considered fixed. During this single execution several collision detection checks are performed for different possible poses (e.g., P_1/P_2 from Figure 2a) and motions (e.g., M_{12} from Figure 2a). An important consequence of this process that we exploit in this paper is that several CDQs are performed for different robot poses with the same occupancy information.

C. Motion Planning and Collision Detection Acceleration

Collision detection is a computationally intensive part of motion planning, and it can take more than 90% of runtime [4], [5], [35], [39], [41]. Several CPU and GPU-based acceleration approaches have been proposed for collision detection [10], [13], [24], [39]. However, the runtime and energy consumption constraints make specialized collision detection accelerators more suitable for mobile robots. Consequently, several specialized accelerators have been proposed for collision detection in motion planning [3], [31], [34], [43], [45], [48].

Figure 5 represents our baseline generic sampling-based motion planning acceleration system. During a motion planning query, the controller initiates the process of sampling ❶. Sampled poses and motions are sent to a scheduler ❷, which sends these poses and/or motions for collision check ❸. Prior works have proposed spatially-aware [43] or speculative [3] scheduling policies. A pose or a motion is converted from C-space to a set of simple geometric shapes in physical space ❹. For a robotic arm, transformation matrices for all links can be calculated using the DH parameters (4×4 matrices) of the robot and matrix multiplications [12], [38]. A transformation matrix (4×4 matrix containing rotation and translation) for a link can be further used to find the coordinates and rotation of geometric shapes (e.g., OBBs [3], [43], spheres [47]) bounding the space occupied by the link. These smaller objects are then sent to *Collision Detection Units (CDUs)* that perform collision checking with the environment (i.e., CDQs) ❺. The results from the CDUs are aggregated and sent back to the

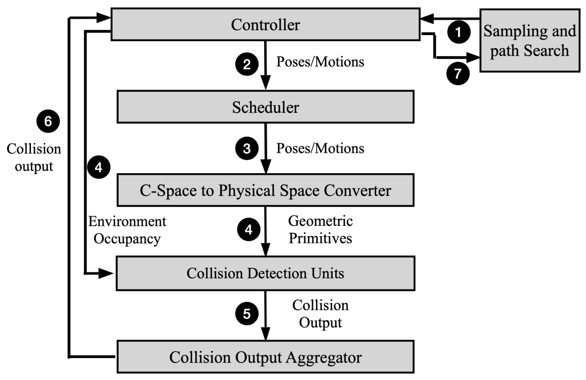


Fig. 5: Baseline sampling-based motion planning acceleration system.

controller ❻. The controller communicates collision outcomes of different poses and motions with the sampling and path search module to expand/refine the planned trajectory ❼. Note that the above describes a generic flow that can be executed on CPU, GPU, or ASIC platforms. Depending upon the motion planning algorithm, one or more blocks may be eliminated.

In this work, we discuss collision prediction integration with different compute platforms used for collision detection, including CPU, GPU, and ASIC. We propose a hardware collision prediction unit that can be tightly integrated with collision detection accelerators (Section IV).

III. Collision Prediction

This section provides a limit study to motivate collision prediction in motion planning. Further, it describes our key insight for collision prediction, motivates the use of a collision history table, and explores different hashing functions. Finally, we introduce the proposed collision prediction approach, COORD.

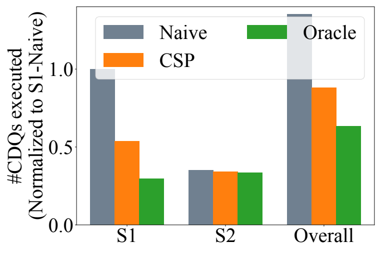
A. Collision Prediction Limit Study

As explained in Section II-B, a motion-environment or pose-environment collision check is carried out by executing multiple smaller CDQs (e.g., OBB-environment collision check) and combining the output of all CDQs using an `OR` operation. Thus, if any of these CDQs returns `True` for collision, the entire motion/pose is determined to be in a collision, and execution of subsequent CDQs is skipped. For a collision-free motion/pose, the execution order of CDQs does not matter, as all CDQs are performed before concluding that the motion/pose is collision-free. However, for colliding motions/poses, the execution order of CDQs affects the computation performed. Our profiling of 1000 motion planning queries covering different motion planning algorithms and robots shows that $52\% - 93\%$ motions checked for collision in these motion planning algorithms are colliding. Thus, there is a potential to reduce the collision checking time for these colliding motions by focusing on the execution order of CDQs.

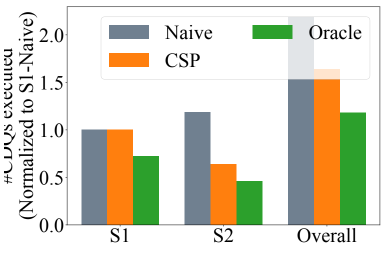
Figure 1 compares three scheduling approaches to demonstrate their impact on the number of CDQs executed. Here, each pose is represented using three OBBs. For naive ordering, all poses are checked for collision serially (Figure 1a), and 18 CDQs are executed before concluding that the motion collides with the obstacle. Shah et al. [43] proposed a **coarse-step scheduling policy (CSP)** for ordering the poses in a motion such that physically distant poses in this motion are checked first. They used a step size greater than 1 to select the order. Hence in this case, a step size of 3 results in the order P_1, P_4, P_7,..,P_2,P_5,...P_n. Thus, with CSP, 9 CDQs are executed before finding a collision, as shown in Figure 1b. However, an Oracle predictor needs to execute only one CDQ for a colliding motion/pose, as shown in Figure 1c. Thus, a predictor has significant potential to reduce the number of CDQs executed (i.e., computation).

We perform a limit study of the potential reduction in the number of CDQs performed during motion planning using three motion planning algorithm-robot combinations (methodology described in Section V). Figure 6 compares the number of CDQs executed using a naive, CSP-based, and Oracle prediction-based ordering of CDQs. We have divided each motion planning algorithm into two stages, labeled S1 and S2, based on the type of CDQs performed in these stages. For example, in S1 of MPNet, different motions are checked for collision to find a suitable and short path to the end goal. In this exploration stage, the majority of the motions checked are colliding. Whereas in S2, the trajectory (i.e., set of motions) determined by S1 is checked for feasibility. The majority of the motions checked in this stage are collision-free. Collision prediction does not reduce computation for a collision-free motion. Thus, collision prediction provides a higher reduction in the number of CDQs executed in S1 (44%) compared to S2 (0.02%). Overall, Oracle prediction results in 29.7%, 25.1%, and 40.7% reduction in the number of CDQs compared to CSP ordering. Further, we measure the potential improvement in the performance/watt on a collision detection hardware accelerator. Our evaluation using microarchitectural simulators suggests that an Oracle predictor (100% precision and recall with zero cycle latency) can provide $1.11 - 1.44\times$ increase in performance/watt for different motion planning algorithms and 7-DOF robotic arms (methodology described in Section V).

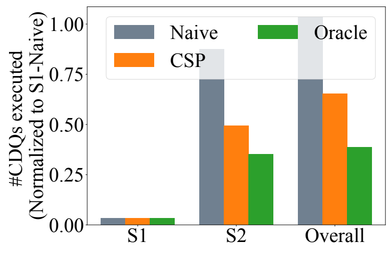
We further analyze the advantage of Oracle prediction for varying difficulty levels in motion planning. The evaluated benchmarks consist of environmental scenarios and motion planning tasks of varying difficulty levels. For cluttered environments (e.g., more obstacles) or difficult motion planning queries (e.g., passing through narrow area), motion planning typically requires more time as it needs to check multiple possible motions and poses to find a collision-free trajectory. We use the number of CDQs performed during a motion planning query to approximate its difficulty level and divide the benchmarks into five equal-size groups, G1-G5, where the difficulty level increases from G1 to G5. Figure 7 compares the number of CDQs for CSP and Oracle prediction for G1-G5 groups. The Oracle predictor achieves 9%, 14.3%, 18.7%, 28.3%, and 42.5% reduction in the number of CDQs compared to CSP, respectively. This shows a higher potential for improving motion planning for cluttered and challenging environments. Service and assistive robots suitable for



(a) MPNet [41]-Baxter robot [42]



(b) GNNMP [50]-KUKA robot [1]



(c) BIT* [14]-2D path planning

Fig. 6: Comparison of the number of CDQs executed during motion planning for different scheduling policies.

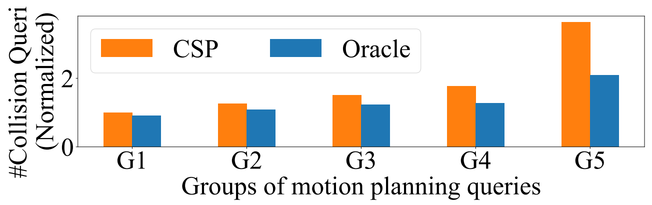
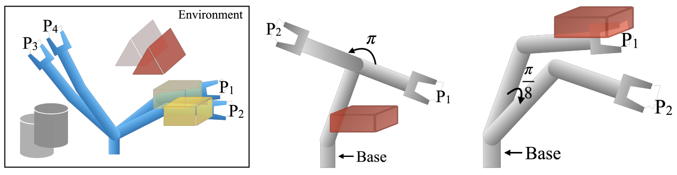


Fig. 7: Collision prediction for GNN motion planning for a 7-DOF KUKA robot. Here, G1-G5 represent difficulty levels of motion planning queries, G5 being the highest.



(a) Spatial-Temporal physical locality (b) (c)

Fig. 8: (a) shows the impact of temporal-spatial locality of obstacles on collision outputs for nearby poses to demonstrate the use of collision history. Here, faded objects represent their location in the next time frame. (b) and (c) compare the effect of change in a DOF on the space occupied by the robot. A DOF closer to the robotic arm's base has a higher impact on the space occupied by the robot, as shown in example (c).

healthcare and older adults must perform motion planning in cluttered environments in real time [25], [32]. Several works focus on improving motion planning in cluttered and highly dynamic environments as it is a challenging problem [19], [21], [30], [44]. We observe that the potential reduction in computation and runtime using collision prediction increases as the difficulty level of a motion planning query increases.

B. Collision Prediction in Robot's Configuration Space

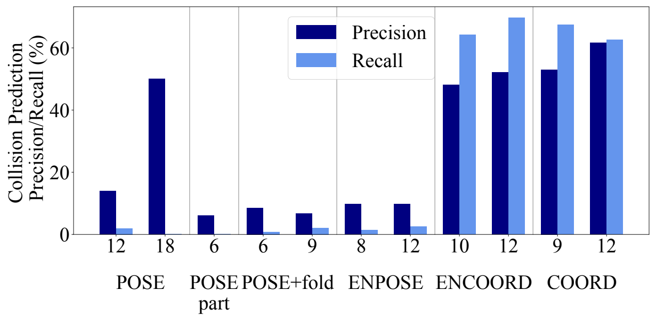
We first explore the possibility of applying a hashing function directly to the C-space representation of a robot's pose. As mentioned in Section II-C, robot motion planning is performed in a robot's C-space, and C-space representations of robot poses are converted to physical space representations (e.g., a set of geometric primitives bounding the space occupied by a pose or motion) for collision prediction using forward kinematics. Collision prediction using the C-space representation of a robot pose does not require C-space to physical space representation transformation before collision prediction, thus saving computation.

The C-space representation of an n-DOF robot's pose is an n-dimensional real-valued vector, where each value represents a DOF. Thus, the collision history of poses is very sparse due to the large space of the number of possible poses. We find that the spatial locality of the robot's poses can be exploited to use a pose's collision output to predict the collision output for other poses. Figure 8a represents four poses of a robot in an environmental scenario with three obstacles. Here, the obstacles' position for the next time frame is represented using faded objects. Here, two physically nearby poses (P_1-P_2 and P_3-P_4) are likely to have the same collision outputs. Thus, collision output for pose P_1 can be used to predict the collision outcome for pose P_2. These examples demonstrate that a collision history table can be used for collision prediction by grouping physically nearby robot poses using a hashing function. This strategy results in a small and dense collision history table. Further, depending upon the speed of obstacles and the time between two motion planning queries, temporal-spatial locality exists in the space occupied by obstacles. Thus, the collision history of a time frame can be used for the next time frame. Next, we discuss different hashing strategies for C-space representations. These hashing functions are designed with the aim of grouping physically nearby positions of a robot.

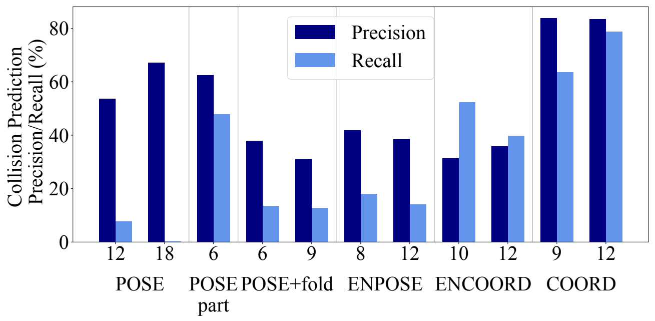
POSE: For POSE, each value of the C-space representation of a pose is quantized and converted to k bits, giving a hash code of bit-width kn for an n-DOF robot. Hash code generation can be done by taking k MSBs from the fixed-point representation of each DOF.

POSE-part: In Figure 8c, a small change (e.g., rotation) in a DOF closer to the robot's base significantly changes the space occupied by the robot. In contrast, a larger change in the DOF far from the robot's base still preserves partial physical space overlap between two poses (Figure 8b). These examples demonstrate that for a robotic arm, DOFs closer to the base have a higher impact on the physical spatial locality between two poses. In the POSE-part hashing function, we only consider the first two DOFs, and each DOF is quantized to k bits. This reduces the size of the hash table.

POSE+fold: POSE results in long hash codes (kn), which results in a large and sparse collision history table. We further explore the use of folding of the POSE hash code to reduce the size of the collision history table. In POSE+fold, a part of the POSE hash code is XORed with the other part.



(a) Low obstacle density environments



(b) High obstacle density environments

Fig. 9: (a) and (b) compare the collision prediction precision and recall of different hash functions for low- and high-clutter environments, respectively. Here, numbers on the x-axis represent the bit-width of hash codes. Random baseline precision is 2.6% for low-density and 26% for high-density environments. POSE: Hash function applied to the pose of the robot in configuration space, POSE+fold: Hash function applied to the pose with hash folding using XOR, ENPOSE: Hash function applied to the encoded pose, COORD: Hash function applied to Cartesian coordinates of the centers of individual links of the robot.

ENPOSE: One approach to reducing the size of the hash code is to use a fixed-size latent space representation of the robot's pose. We train a small encoder-decoder network on $32,768$ random poses using the loss between input poses and decoded poses. One-layer MLPs are used as the encoder and decoder to keep encoding overhead low. We explore 2 and 4-dimensional latent space representation and quantize latent space representation to generate hash code.

Figure 9 compares the precision and recall of different hashing functions for low and high-clutter environments. The collision prediction strategy is described in Section III-D. Here, collision prediction precision represents the fraction of poses in collision from poses predicted for collision. Collision prediction recall is the ratio of the number of colliding poses predicted to be in a collision and total colliding poses. We observe that even though the precision of POSE is high, its recall is very low due to a large and sparse collision history table. POSE+fold reduces the hash code size and increases the recall at the cost of precision. This is due to the folding process not preserving physical spatial similarity between two poses. In contrast, POSE+part increases the precision and recall as it preserves the physical spatial locality for links closer to the robot's base. ENPOSE results in very low precision (close to baseline using random prediction). We believe latent space representation does not preserve physical spatial locality, resulting in low precision. *In summary, C-space representation hashing does not provide sufficient precision and recall, as simple hashing strategies do not capture physical spatial locality in hash codes.*

C. Collision Prediction in Physical Space

In this section, we explore the potential of using physical space representations of a robot and environmental obstacles for collision prediction. We propose to apply a hashing function to the space represented by a robot and provide details of the proposed approach in this section. We also discuss hashing function's application to the space occupied by environmental obstacles for collision prediction in Section VII-2.

The C-space representation of the robot's pose is used to find a transformation matrix for each rigid link of the robot [43], [47]. This transformation matrix can be used to calculate geometries for each link (e.g., OBBs, spheres). This transformation matrix is a 4×4 matrix, which represents the rotation and translation of each link for a given pose [12]. Thus, this transformation process provides the Cartesian coordinates of the center of different rigid parts of the robot, which can be used to generate hash code. For example, a 2-DOF robot in Figure 8b consists of 3 rigid parts. In the proposed approach **COORD**, the hashing function is applied to the Cartesian coordinates of the center of each link. These hash codes are used to make collision predictions for each link, which can be used to prioritize execution CDQs (e.g., OBB-environment or sphere-environment collision check) corresponding to a link if a collision is predicted. We consider OBB-based [3], [43] and spheres-based [47] representation of a link in evaluation. The center of a link is represented using three 16-bit fixed point representations of its Cartesian coordinates. Figure 10 represents hash code generation from a link's center coordinates.

We further explore the application of a hashing function to the latent-space representation of the Cartesian coordinates of the center of a link. We use simple one-layer MLP to generate 2 or 4-dimensional latent space representation and quantize the latent space representation to generate a hash code. This approach is referred to as **ENCOORD**.

Figure 9 compares the precision and recall of the ENCOORD and COORD hash functions. ENCOORD achieves comparable precision and recall for low-clutter environments. However, precision and recall decrease in high-clutter environments. We believe that latent-space representation does not completely preserve the physical spatial locality. For high-clutter environments, this results in frequent inaccurate collision history updates, leading to lower precision and recall. The figure shows that COORD results in the highest collision prediction precision and recall. COORD provides 77% precision with 47% recall even for low-clutter environments.

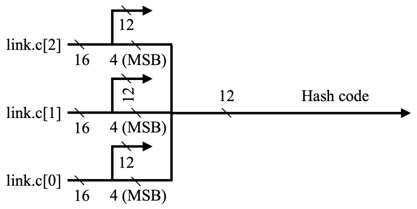


Fig. 10: Hash code generation for a robot link using the COORD hash function on its center link.c. In this example, four MSBs of each coordinate are used for hash code generation, and the rest of the bits are discarded.

D. Collision History Table Update and Prediction Strategy

For the proposed collision prediction approach, a *Collision History Table (CHT)* is maintained and updated with the collision outputs as CDQs are performed. The hash code generated for a CDQ using the proposed COORD function is used as the address for accessing the CHT. Each entry in the CHT maintains saturating counters for colliding (`COLL`) and collision-free (`NONCOLL`) queries observed in the past. We observe that updating the history table for all colliding CDQs is important for prediction precision and recall. However, we can reduce the frequency of updates for collision-free CDQs. We use a parameter U $(0 \leq U \leq 1)$ to define the update frequency for collision-free CDQs. For every N collision-free CDQs executed, $N \times U$ CDQs are chosen randomly to update the CHT. A lower value of U results in lower traffic and updates of the CHT.

The collision prediction strategy determines when a collision is predicted for a CDQ given the CHT entry with counter values `COLL` and `NONCOLL` for its hash code. The proposed collision prediction strategy predicts a query to be colliding if `COLL` $> S \times$ `NONCOLL`. Thus, the value of parameter S $(0 \leq S \leq 1)$ sets the weight of `NONCOLL` for prediction and determines the aggressiveness of the collision predictor. A lower value of S means a more aggressive predictor. Note that for $S = 0$, the CHT does not need to maintain `NONCOLL` counters, and requires only one bit per entry. Section VI-A1-VI-A2 compare the effect of different values of S and U on precision and recall.

E. Collision Prediction and Detection on CPU and GPU

Our goal is to use collision prediction to reduce the computation required for motion planning collision detection, which can be exploited to reduce runtime and improve energy efficiency by reducing dynamic power consumption. We first evaluate the impact of collision prediction for collision detection using CPU and GPU (methodology given in Section V). Algorithm 1 represents a pseudo code for motion-environment collision detection with collision prediction. A motion planning benchmark consists of several motions to be checked for collision. For the CPU-based implementation with four threads and GPU-based implementation with 64 threads, each thread executes Algorithm 1 for a group of motion. For GPU-based implementation for more than 512 threads (i.e., higher parallelism), each thread executes Algorithm 1 for a subset of poses in a motion. The hash table is shared between all threads.

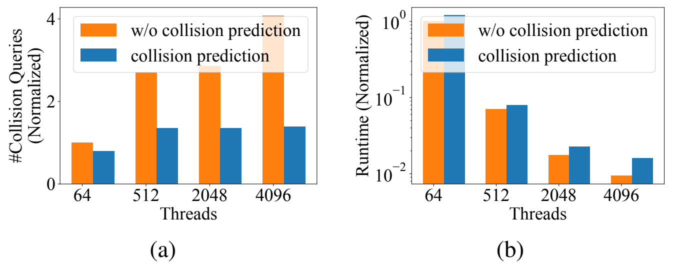


Fig. 11: Impact of collision prediction on numbers of CDQs executed (a) and execution runtime (b) for GPU-based collision detection.

For CPU-based implementation, we observe 25.3% reduction in the number of collision detection queries (i.e., computation) and 13.8% reduction in the collision detection runtime. Our profiling suggests that the CHT accesses and updates increase cache misses, which might explain the gap between computation and runtime reduction. Further, this speedup might not be sufficient to enable real-time motion planning execution on a CPU.

Algorithm 1 Motion collision detection with collision prediction

```
Input: Motion_info = [pose1, pose2,..., poseN], Hash_table, $S$ ;
Output: Collision output;
 1: Queue=[];
 2: for pose $\in$ Motion_info do
 3:     OBBs = robot_kinematics(pose)
 4:     for OBB $\in$ OBBs do
 5:         hashentry=hashcode(OBB.c)
 6:         if hashentry.COLL $>$ (hashentry.NONCOLL $\times$ S) then
 7:             CollTemp = Collision_check(OBB)
 8:             Hash_table.update(hashcode(OBB.c),CollTemp)
 9:             if CollTemp then return True
10:             end if
11:         else
12:             Queue.append(OBB)
13:         end if
14:     end for
15: end for
16: for OBB $\in$ Queue do
17:     CollTemp = Collision_check(OBB)
18:     Hash_table.update(hashcode(OBB.c),CollTemp)
19:     if CollTemp then return True
20:     end if
21: end for
22: return False
```

We further evaluate the impact of collision prediction on the number of CDQs executed and runtime for different levels of parallelism in GPU. Figure 11 represents the number of CDQs executed and collision detection runtime for MPNet motion planning with and without collision prediction. Number of CDQs and runtime are normalized with respect to 64 threads configuration without collision prediction. As the degree of parallelization increases, baseline collision detection results in higher CDQs as redundant work increases due to the parallel execution of CDQs within a motion. However, collision prediction helps with reducing redundant work by prioritizing the execution of CDQs that are likely to result in positive collision output. Further, we find that collision prediction-based execution results in 30% (2048 threads) to 70% (4096 threads) increase in the execution time. Our profiling results suggest

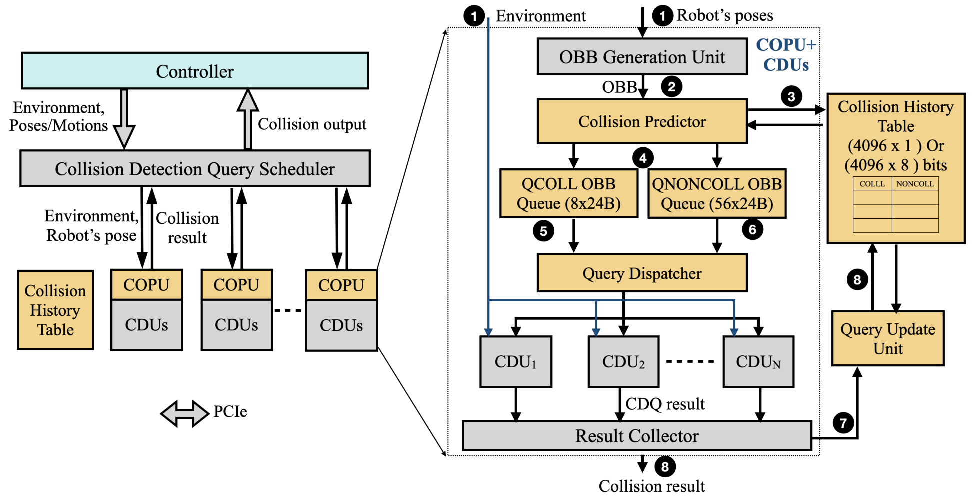


Fig. 12: Architecture of the proposed Collision Prediction Unit (COPU) and its integration with motion planning collision detection accelerator. Here, gray-colored blocks represent the baseline architecture [43], and yellow-colored blocks represent additions for collision prediction.

that software collision prediction increases warp divergence due to skipped computation and memory stalls due to hash table accesses.

The use of application-specific accelerators is desirable to meet the runtime and energy constraints of robot motion planning, especially for high-DOF robots working in dynamic environments [3], [17], [34], [43]. We find that software-collision prediction (CPU controller) is $\sim 2\times$ slower than collision detection using a specialized accelerator [3], [43], which results in an overall $\sim 2.1\times$ collision detection slowdown. This motivates the need for a collision prediction accelerator that can be integrated with existing collision detection hardware accelerators. In the next section, we propose a hardware collision prediction unit and discuss its microarchitecture.

IV. Collision Prediction Unit Architecture

This section describes the microarchitecture of the collision detection acceleration system used for evaluation. It provides details of the proposed *Collision Prediction Unit (COPU)* and its integration with a specialized accelerator for collision detection in motion planning.

Figure 12 represents the overall architecture of a motion planning collision detection accelerator (left) with collision prediction. The right side of Figure 12 represents a detailed architecture of collision prediction's integration with a *Collision Detection Unit (CDU)*. The gray-colored blocks represent the baseline architecture [43] (explained in Section II-C), and yellow-colored blocks represent additions for collision prediction. Note that the proposed COPU can be integrated with any collision detection accelerator. The collision prediction unit and OBB-generation unit is shared by multiple CDUs. Each CDU performs an intersection test between environmental obstacles and a part of the robot (e.g., OBB or Spheres). We use OBB-environment CDU proposed by Shah et al. [43] in our implementation.

The OBB Generation Unit receives a pose from the scheduler and generates OBBs bounding each link ❶. The output OBB's center (a proxy for the link's center) is used to generate the hash code ❷ and read the corresponding entry from the *Collision History Table (CHT)* ❸. The COPU sends the OBB to corresponding queues ❹, depending upon the collision prediction output. There are two queues `QCOLL` and `QNONCOLL` for storing OBBs with collision predicted and not predicted, respectively. Priority must be given to CDQs for OBBs stored in the `QCOLL` queue as these CDQs are more likely to return `True` for collision check. The Query Dispatcher sends a query from non-empty `QCOLL` queue to a free CDU ❺. However, if `QCOLL` is empty, the Query Dispatcher sends a query from `QNONCOLL` queue only if `QNONCOLL` is full or the OBB Generation Unit has received all poses from the scheduler ❻. In the latter case, queries are dispatched from `QNONCOLL` as no new entries will be added in `QCOLL` and `QNONCOLL` for this pose/motion. Thus, the Query Dispatcher prioritizes the `QCOLL` queue for collision detection and dispatches from `QNONCOLL` only if necessary. CDUs send the collision output to the Query Update Unit to update the hash table ❼. The Result Collector receives the output of all CDUs and sends the output to the scheduler ❽.

The CHT is implemented using an SRAM. Each entry of CHT consists of two 4-bit saturating counters, `COLL` and `NONCOLL`. The Collision Predictor uses OBB's center to generate CHT address using COORD and reads corresponding entry {`COLL`,`NONCOLL`}. The predictor uses comparison (`COLL`> (`NONCOLL`>>x)) to determine collision output, where x is determined using parameter S used in collision prediction strategy (Section III-D). The Query Update Unit receives the hash code and collision output for an executed CDQ. It reads the corresponding entry from CHT and updates `COLL` or `NONCOLL` (based on collision result) using a saturating adder. For collision-free CDQs, the Query Update Unit uses a random number generator and parameter U to determine whether the CHT is updated for this query. All entries (`COLL` and `NONCOLL` counters) in the CHG are reset to zero after each motion planning query, as obstacle positions might change for the following motion planning query. Thus, the most recent environment obstacle positions are considered for collision prediction.

V. Methodology

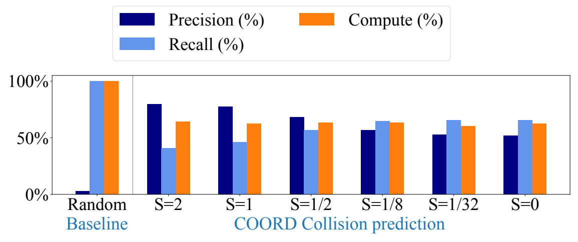
We study different hashing functions and design aspects for collision prediction using random environmental scenarios and robot poses (Section III-B, Section III-C, and Section VI-A). A 7-DOF robotic arm Kinova Jaco2 [23] is used for these benchmarks. We generate an environmental scenario for each benchmark with random placement of $5-9$ cuboid-shaped obstacle. The size of the environment is limited to the reach of the Jaco2 robot. These benchmarks are consistent with previous works on motion planning [31], [34]. 1000 random robot poses are sampled in an environment for collision prediction evaluation. For low, medium, and high obstacle density benchmarks, the size and number of obstacles are limited such that, on average, $\sim 2.5\%$, $\sim 10\%$, and $\sim 25\%$ robot poses are in collision.

We explore three motion planning algorithms to evaluate collision prediction (Section VI-B). MPNet [41] is a leaning-based motion planning algorithm and uses a neural network for trajectory sampling. We evaluate this algorithm for a 7-DOF robotic arm Baxter and 2D path planning. GNNMP [50] uses a graph neural network for sampling the C-space and path smoothing. Batch Informed Trees (BIT*) [14] is an informed-sampling-based motion planning algorithm. The above two algorithms are evaluated for a 7-DOF robotic arm KUKA [1] and 2D path planning. We use the environment scenarios and motion planning queries used in the original work [41], [50]. Each environment scenario typically consists of a work table with several objects randomly placed on the table and in the surroundings.

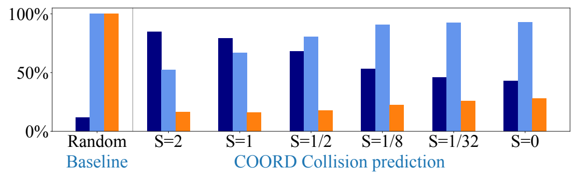
We evaluate the proposed collision prediction using a detailed microarchitectural simulator. For area and energy estimate, we use OpenRAM Memory Compiler [16] and 45nm technology using FreePDK design library [46]. Klampt [18] and PyBullet [11] robotic simulators are used to simulate robot motions and poses. The size of CHT is set to 1024 entries and 4096 entries for 2D path planning and robotic arm motion planning, respectively. We also evaluate collision prediction for CPU (Cortex A57 4-core) and GPU (NVIDIA Titan V) based collision detection systems. MPNet-Baxter benchmarks are used for this evaluation. We use C++ (with OpenMP) and CUDA programming language for implementation, and Valgrind [36] and nvprof [37] for profiling.

VI. Evaluation

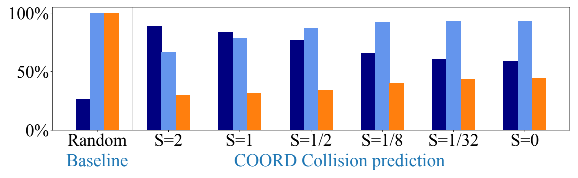
In this section, we present an evaluation of the proposed COORD collision predictor and study different design aspects' impact on prediction and computation. Next, we evaluate the motion planning runtime, computation, and performance achieved by integrating the proposed COPU with an existing motion planning collision detection accelerator.



(a) Low obstacle density environments



(b) Medium obstacle density environments



(c) High obstacle density environments

Fig. 13: (a)-(c) compare the collision prediction precision, recall, and the resultant decrease in the computation compared to a random baseline for different collision prediction strategies (parameter S introduced in Section III-D).

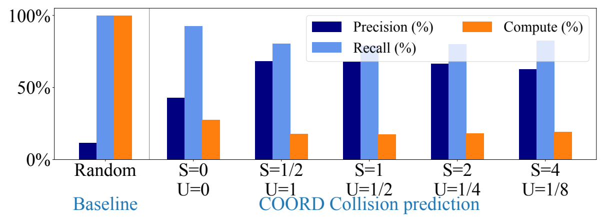


Fig. 14: Effect of the update frequency of the Collision History Table on collision prediction (parameter U introduced in Section III-D).

A. Collision Prediction Design Aspects

This section presents an evaluation of the COORD collision predictor for different environmental scenarios and design aspects.

1) Collision Prediction Strategies: First, we compare collision prediction strategies for three levels of clutterness in the environment. As described in Section III-D, the proposed predictor predicts collision if $\texttt{COLL} > S \times \texttt{NONCOLL}$, where $\texttt{COLL}$ and $\texttt{NONCOLL}$ represent the counts of colliding and collision-free CDQs observed in the past for the same hash code (i.e., CHT entries). The collision predictor becomes more aggressive with lower values of S. Here, $S = 0$ results in the most aggressive prediction strategy. Figure 13 compares the precision and recall for different values of S across different environments. We also report approximate computation reductions achieved by collision prediction using a statistical model. This statistical model considers the baseline collision probability, precision, and recall and provides the potential decrease in the number of CDQs executed for collision check of a motion consisting of 80 CDQs. As shown in Figure 13, as the value of S decreases, the predictor becomes more aggressive, resulting in higher recall and lower precision.

We observe interesting trends in computation reduction across different types of environments. For low-clutter environments, the number of colliding poses is very low, and an aggressive predictor with higher recall is needed to find these colliding poses using prediction. Hence, for low-clutter environments (Figure 13a), the recall is more important than precision, as the computation reduction is the highest for $S = 0$. Meanwhile, in highly cluttered environments, several poses are in a collision. Hence, even though recall is low, the probability that the predictor will find a colliding pose in the motion is high. Here, precision is more important to reduce redundant computation. Hence, we observe that for highly cluttered environments, the computation is the least for the highest precision ($S = 2$). For medium-clutter environments, a balance between precision and recall results in the least computation ($S = 1/2$). We observe that the computation reduction is less sensitive to S compared to precision and recall. In this work, we use a constant value of S across all environmental scenarios. However, there is scope to tune the value of S by using a heuristic to estimate environmental obstacle density (e.g., the number of voxels or the number of nodes in octree); we leave this to future work.

2) Collision History Table Updates: We find that more than 90% of the CDQs are collision-free for low or medium-cluttered environments. Here, we explore the effect of reduced update frequency (U) for collision-free CDQs. Figure 14 compares precision, recall, and computation for different combinations of S and U. Here, for all combinations, the computation decrease does not vary significantly ($\pm 1\%$), which suggests that the value of S can be adjusted to reduce the update frequency.

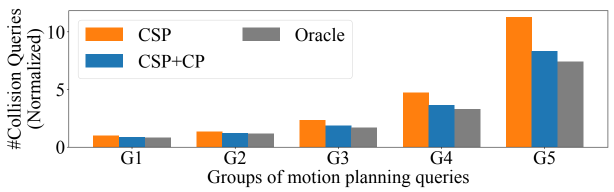
B. Collision Prediction for Motion Planning Acceleration

We integrate the proposed COPU with a collision detection unit as described in Section IV to accelerate motion planning execution. For the evaluation setup, seven CDUs, a COPU, and a Collision History Table of size 4096×8 bits (robotic arms) or 1024×8 bits (2D path planning) are used. The values of S and U are set to 1 and 0.125, respectively. We evaluate the reduction in the number of CDQs performed during motion planning for this setup. Figure 15 compares the number of CDQs executed for MPNet-Baxter, MPNet-2D planning, GNNMP-KUKA, GNNMP-2D planning, BIT*-KUKA, and BIT*-2D planning. For each motion planning algorithm-robot combination, benchmarks are grouped (G1-G5) according to the difficulty level. We use the number of CDQs performed during a motion planning query to approximate its difficulty level and divide the benchmarks into five equal-size groups, G1-G5, where the difficulty level increases from G1 to G5. Here, all numbers in a plot are normalized to #CDQs for G1 benchmarks and CSP scheduling [43].

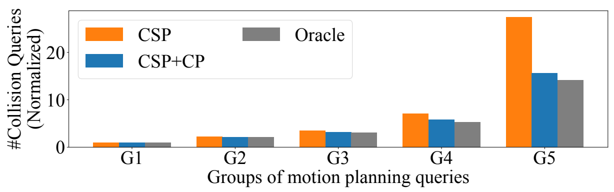
The proposed COPU provides 22.5%, 32.06%, 17.4%, 23.1%, 25.5%, and 21.6% reduction in the number of CDQs executed on average compared to CSP-based ordering across different benchmarks (Figure 15a-15f). For a more difficult problem in a cluttered scenario or narrow passage, the motion planner needs to perform more tests to find a collision-free motion. We observe that the proposed collision prediction unit provides higher benefits for such challenging scenarios. COPU provides 26.2%, 42.9%, 25.4%, 28.7%, 37.3%, and 23.4% reduction in the number of CDQs executed for group G5 across different benchmarks (Figure 15a-15f).

1) Area and Energy overhead: A Collision History Table (CHT) is added for collision prediction. Further, two queues are added per a group of CDUs (Figure 12). We estimate the energy and area overhead of these added components to a motion planning collision detection accelerator MPAccel [43] using the OpenRAM Memory compiler. For an MPAccel [43] configuration with 24 CDUs with one COPU, `QCOLL`, `QNONCOLL`, and OBB Generation Unit per 6 CDUs, a CHT of size 4096×8 bits results in 1.96% and 1.01% area and energy overheads. This overhead changes to 0.55% and 0.28% for a CHT of size 4096×1 bits. `QCOLL` and `QNONCOLL` results in 2.6% and 1.4% area and energy overheads, respectively.

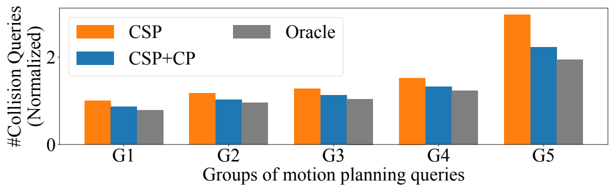
2) Performance Evaluation: We analyze the impact of collision prediction on the overall runtime, energy consumption, performance/watt, and performance/mm^2 of the proposed hardware accelerator. MPNet motion planning for the Baxter robotic arm is used for evaluation. We consider three configurations (COPU.1, COPU.4, and COPU.6) of the collision detection accelerator shown in Figure 12, where the suffix represents the number of OBB-environment CDUs. A CHT of size 4096×1 ($S = 0$, $U = 0$) is used for these experiments. The lengths of `QNONCOLL` and `QCOLL` are set to 56 and 8, respectively. For COPU.1, COORD collision prediction results in 23.4% reduction in the collision detection energy consumption with $1.29\times$ speedup (collision detection time per motion) compared to baseline (without collision prediction). For COPU.6, COORD collision prediction results in 22.4% reduction in the energy consumption with $1.10\times$ speedup compared to baseline. As described in Section IV, the Query Dispatcher proposed in this work prioritizes CDQs that are predicted to be in colliding. This dispatcher schedules a CDQ from `QNONCOLL` only if the queue is full or `QCOLL` is empty and the predictor has received all CDQs for a given motion. This results in a waiting period before executing any CDQs. In COPU.1, only one CDQ can be executed at a time. In this case, the overhead of the Query Dispatcher's waiting period is less. However, COPU.6 has significant parallelism available, and the waiting time overhead becomes prominent. This is expected as the Query Dispatcher prioritizes energy efficiency. However, the Query Dispatcher's policy can be designed to find a trade-off between runtime and energy efficiency. Note that collision prediction increases the launch-to-execute latency of a CDQ due to collision prediction latency and queuing. However, the end-to-end latency of motion collision detection reduces with collision prediction.



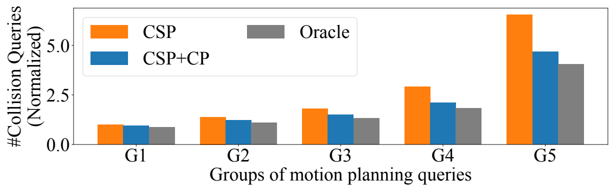
(a) MPNet [41]-Baxter



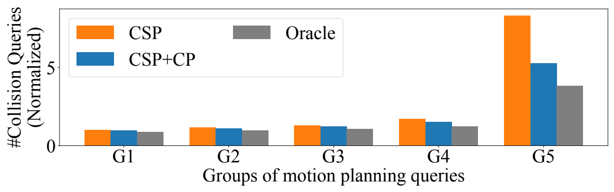
(b) MPNet [41]-2D Path planning



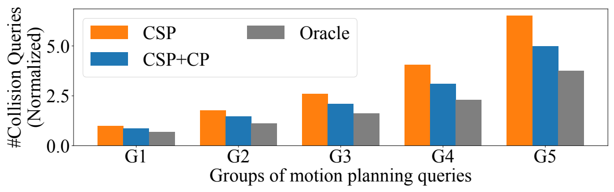
(c) GNNMP [50]-KUKA



(d) GNNMP [50]-2D Path planning



(e) BIT* [14]-KUKA



(f) BIT* [14]-2D Path planning

Fig. 15: Collision prediction for different robots and motion planning algorithms.

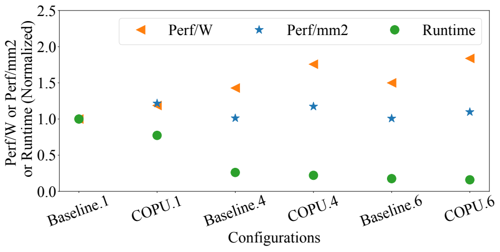
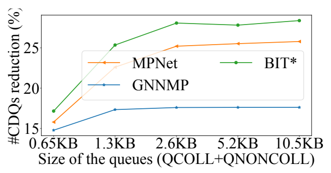
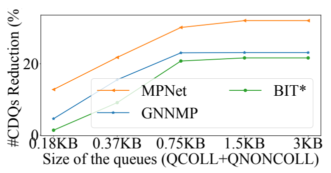


Fig. 16: Performance/mm^2, performance/watt, and latency comparison for different CDU configurations. The baseline represents a CDU without COPU. The suffix number represents the number of OBB-Environment CDUs in Figure 12.



(a) 7-DOF robots



(b) 2D motion planning

Fig. 17: CDQs reduction versus `QNONCOLL` queue size.

We further compare the runtime (average end-to-end latency for motion-environment collision check) and performance with and without collision prediction. Figure 16 compares the performance/mm^2, performance/watt, and runtime for different configurations with and without collision prediction. Here, the suffix in the configuration name represents the number of CDUs per COPU (Figure 12). For all configurations, collision prediction (COPU.x) provides lower latency and higher performance/area and performance/watt compared to the baseline (baseline.x). For example, COPU.4 increases performance/watt and performance/mm^2 by $1.23\times$ and $1.15\times$, respectively, while providing $1.18\times$ speedup.

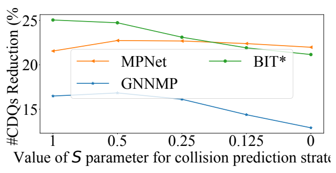
C. Sensitivity Studies

In this section, we study the impact of COPU architectural aspects on reducing executed CDQs.

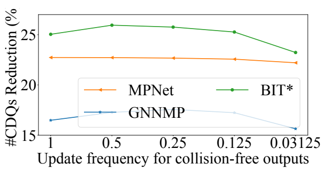
Queue Size: Two queues `QCOLL` and `QNONCOLL` are added to store the OBBs generated by the OBB Generation Unit (Figure 12). These queues are added so that the Query Dispatcher can prioritize CDQs for OBBs predicted to be colliding before performing collision detection for CDQs from the `QNONCOLL` queue. Thus, the size of `QNONCOLL` queue is crucial and should be set such that it can hold enough entries before the predictor finds a colliding OBB. Figure 17 compares the computation reduction achieved by the predictor for different `QNONCOLL` queue sizes. For all benchmarks, the runtime/energy improvement decreases significantly for a very small queue size. The improvement also saturates for larger queue sizes as the queue is not fully utilized during execution.

Collision Prediction Strategy: Figure 18a compares the improvement achieved by the proposed predictor for different collision prediction strategies determined by the values of S (Section III-D). As shown in the figure, the improvement is not highly sensitive to the prediction strategy. In several cases, $S = 0$ results in improvement within 2% of the best choice. Using $S = 0$ simplifies the design and reduces area/energy of the Collision History Table, as only one bit is needed per entry.

Collision History Table Updates: Figure 18b shows the impact of the history table update frequency for collision-free CDQs on the CDQs execution reduction. For collision-free queries, we scale the update frequency by $U < 1$ (described



(a)



(b)

Fig. 18: (a) CDQs reduction achieved for different collision prediction strategies. (b)CDQs reduction achieved for different Collision History Table update frequency for collision-free CDQs.

in Section III-D). As shown in Figure18b, the reduction in the number of executed CDQs is not highly sensitive to the update frequency; this suggests that hash table update frequency can be reduced for reduced traffic.

VII. Scope of Collision Prediction

Motion planning and collision detection are widely studied problems with a wide range of algorithms available. Section II-B provides a brief background on different types of collision detection algorithms. For collision prediction to be useful for providing computation and runtime reduction, the underlying collision detection algorithm must have certain characteristics. As mentioned in Section II, each pose-environment and motion-environment collision check consists of several CDQs, where each CDQ corresponds to an intersection test between two geometries (part of space occupied by robot and environment). Two necessary conditions for using collision prediction are: 1) There exists parallelism without data dependence between all CDQs for a pose and/or motion, and 2) Collision detection can be terminated early if a CDQ returns `True`, without affecting the correctness. Prior works have proposed continuous collision detection algorithms [8], [47], where a motion is not uniformly discretized, and the next discrete pose to be checked for collision depends upon the collision outcome of the current pose. In such cases, collision prediction can only be used to schedule CDQs from a pose. The second condition might not be true for motion planning algorithms, where collision detection is needed to provide more information, such as the separation or penetration distance between the robot and environmental obstacles. Our proposed collision prediction can be integrated with existing collision hardware accelerators that use suitable underlying motion planning and collision detection algorithms. We further explore the potential of collision prediction for different CDUs and motion planning algorithms.

1) Sphere-based Representation of Robot: Prior works have proposed the use of spheres [47] to represent the space occupied by the robot (Figure 4b). The CDUs used in this acceleration system perform collision checks between a sphere and the environment. We evaluate the potential reduction in sphere-environment CDQs using a microarchitectural simulator. In this approach, the transformation matrix for each link is first calculated, and the center of this link is used to generate hash code and perform prediction. Links are stored in `QCOLL` or `QNONCOLL`. The query dispatcher reads the transformation matrix for a link from a queue, calculates the centers and radii of spheres corresponding to this link, and sends CDQs to available CDUs. Our evaluation using the Jaco2 Robot and MPNet algorithm shows that collision prediction reduces the number of CDQs by 23.4%. Note that since we are storing link transformation matrices in the queue, buffer sizes are similar to the prior implementation (Section IV).

2) Hashing Environmental Obstacle's Physical Space: In this section, we discuss the potential of using collision prediction by applying a hashing function to the space occupied by environmental obstacles. Here, we focus on Probabilistic Roadmaps based motion planning and corresponding hardware accelerator (Dadu-P) proposed by Lian et al. [31]. In this approach, a fixed set of short motions is used for motion planning. The space occupied by each short motion is converted to an optimized octree-based representation offline and stored in memory. At runtime, each short motion (i.e., octree) is checked for collision with environmental obstacles represented using a set of voxels. Collision-free short motions are then used to construct a trajectory between start and end goals. Here, a CDQ corresponds to motion octree-voxel collision detection. In this case, a hashing function can be applied to the voxel coordinates to maintain a collision history of environmental voxels for each motion. We integrate spatially-aware scheduling policy (CSP) [43] and queue-based COPU with Dadu-P accelerator. Our evaluation shows that CSP and CSP+COPU reduce the number of CDQs by 74.3% and 81.2% for colliding motions, respectively. Our limit study shows that collision prediction can reduce the number of CDQs for colliding motions by 99% compared to naive sequential scheduling of voxels, as only one voxel needs to be checked for colliding motion. We observe that limited queue size results in lower benefits from COPU. Our evaluation demonstrates the potential of using collision prediction for this accelerator, and collision prediction integration can be re-designed to achieve computation reduction closer to the limit study.

VIII. Related Work

Prior works have focused on CPU or GPU-based acceleration of motion planning algorithms and collision detection [2], [5], [9], [28], [39], [40], [47]. Dadu-P and Dadu-CD [31], [48] proposed to use a precomputed octree representation of the space occupied by different motions of the robot. Murray et al. [33], [35] proposed using precomputed sets of axis-aligned bounding boxes to store the space occupied by robot motions. These accelerators focus on computing the space occupied by different motions offline for faster runtime collision detection. However, these accelerators are only suitable for a class of motion planning algorithms that use only a fixed set of motions for motion planning [29]. Bakhshalipour et al. [3] proposed a hardware collision detection unit based on voxelized OBB-environment collision detection. They proposed a memory access coalescing strategy. Shah et al. [43] proposed a Cascaded Early-Exit Collision Detection Unit for OBB-environment collision detection. However, prior works have not focused on collision prediction to reduce computation in motion planning. Yeh et al. [49] explored the use of branch prediction for accelerating physics simulation on a CPU using the collision history of object-object pairs. However, their main goal is to improve the branch predictor's accuracy. In contrast, this work focuses on collision prediction to skip computation in robot motion planning.

IX. Conclusion

In this work, we explore the potential of collision prediction for autonomous robots. We study different motion planning algorithms and robots and demonstrate that collision prediction can reduce the number of collision detection queries by $24\% - 40\%$ for all benchmarks and $33\% - 53\%$ for demanding benchmarks, significantly reducing energy consumption and runtime. Our proposed collision prediction approach, COORD, leverages physical spatial locality and provides $69\% - 83\%$ precision with $55\% - 75\%$ recall for environments of different levels of obstacle density. We further study different design aspects that can be adjusted to find a trade-off between precision and recall. We integrate the COORD collision predictor with CPU and GPU-based collision detection, evaluate the impact of prediction on runtime and computation, and profile overheads in software collision prediction. Finally, we integrate the proposed Collision Prediction Unit with an existing Collision Detection Unit and evaluate it across different motion planning algorithms, robots, and environments. The proposed predictor reduces the number of collision detection queries executed for motion planning by $17.2\% - 32.1\%$ on average and by $23.4\% - 42.9\%$ for more demanding benchmarks. COPU increases performance/watt and performance/mm^2 by $1.23\times$ and $1.15\times$, respectively, while providing $1.18\times$ speedup for MPNet [41] motion planning.

Acknowledgement

The authors thank the anonymous reviewers and artifact evaluators for their feedback. This research has been funded in part by the National Sciences and Engineering Research Council of Canada (NSERC) through the NSERC strategic network on Computing Hardware for Emerging Intelligent Sensory Applications (COHESA) and through an NSERC Strategic Project Grant.

References

[1] K. AG, "Industrial robots from kuka," 2023. [Online]. Available: https://www.kuka.com/en-ca/products/robotics-systems/industrial-robots

[2] N. Amato and L. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *Proceedings 1999 IEEE International Conference on Robotics and Automation*, 1999, pp. 688–694.

[3] M. Bakhshalipour, S. B. Ehsani, M. Qadri, D. Guri, M. Likhachev, and P. B. Gibbons, "Racod: Algorithm/hardware co-design for mobile robot path planning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. Association for Computing Machinery, 2022.

[4] M. Bakhshalipour and P. B. Gibbons, "Agents of autonomy: A systematic study of robotics on modern hardware," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 7, no. 3, dec 2023. [Online]. Available: https://doi.org/10.1145/3626774

[5] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT," in *International Conference on Intelligent Robots and Systems*, 2011, pp. 3513–3518.

[6] R. Bohlin and L. Kavraki, "Path planning using lazy prm," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 521–528.

[7] R. A. Brooks and T. Lozano-Perez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 2, pp. 224–233, 1985.

[8] J. Canny, "Collision detection for moving polyhedra," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 2, pp. 200–209, 1986.

[9] D. Challou, M. Gini, and V. Kumar, "Parallel search algorithms for robot motion planning," in *Proceedings IEEE International Conference on Robotics and Automation*, May 1993, pp. 46–51 vol.2.

[10] J.-W. Chang, W. Wang, and M.-S. Kim, "Efficient collision detection using a dual obb-sphere bounding volume hierarchy," *Computer-Aided Design*, vol. 42, pp. 50–57, 2010.

[11] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[12] J. Denavit and R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics*, vol. 22, no. 2, pp. 215–221, 2021.

[13] C. Ericson, *Real-Time Collision Detection*. CRC Press, Inc., 2004.

[14] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[15] 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*, 2014, pp. 2997–3004.

[16] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "OpenRAM: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2016, pp. 1–6, iSSN: 1558-2434.

[17] Y. Han, Y. Yang, X. Chen, and S. Lian, "Dadu series - fast and efficient robot accelerators," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–8.

[18] K. Hauser, *Robust Contact Generation for Robot Simulation with Unstructured Meshes*. Springer, Cham, 2016, pp. 357–373.

[19] C.-M. Huang and S.-H. Hsu, "Efficient path planning for a microrobot passing through environments with narrow passages," *Micromachines*, vol. 13, p. 1935, 11 2022.

[20] T. Jia, E.-Y. Yang, Y.-S. Hsiao, J. Cruz, D. Brooks, G.-Y. Wei, and V. J. Reddi, "Omu: A probabilistic 3d occupancy mapping accelerator for real-time octomap at the edge," in *Proceedings of the 2022 Conference Exhibition on Design Automation Test in Europe*, 2022, p. 909–914.

[21] H. T. K., A. Balachandran, and S. Shah, "Optimal whole-body motion planning of humanoids in cluttered environments," *Robotics and Autonomous Systems*, vol. 118, pp. 263–277, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889017307108

[22] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, 1996.

[23] KINOVA, "Kinova jaco assistive robot," https://www.kinovarobotics.com/sites/default/files/KINO-2018-Bro-Assistive-ZH_YUL-06-R-Web.pdf, 2018.

[24] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE Transactions on Visualization and Computer Graphics*, 1998.

[25] A. Korchut, S. Szklener, C. Abdelnour, N. Tantinya, J. Hernandez-Farigola, J. Ribes, U. Skrobas, K. Grabowska-Aleksandrowicz, D. Szczesniak-Stanczyk, and K. Rejdak, "Challenges for service robots—requirements of elderly adults with cognitive impairments," *Frontiers in Neurology*, vol. 8, 06 2017.

[26] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 1999, pp. 473–479 vol.1.

[27] S. M. LaValle, *Planning Algorithms*, 2006. [Online]. Available: http://lavalle.pl/planning/

[28] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware," *SIGGRAPH Comput. Graph.*, vol. 24, no. 4, p. 327–335, 1990.

[29] P. Leven and S. A. Hutchinson, "A framework for real-time path planning in changing environments," *The International Journal of*

Robotics Research, vol. 21, pp. 1030 – 999, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:11169688

[30] S. Li and N. T. Dantam, “Sample-driven connectivity learning for motion planning in narrow passages,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 5681–5687.

[31] S. Lian, Y. Han, X. Chen, Y. Wang, and H. Xiao, “Dadu-p: A scalable accelerator for robot motion planning in a dynamic environment,” in *Proceedings of the Annual Design Automation Conference*, ser. DAC. Association for Computing Machinery, 2018.

[32] T. Mitzner, T. Chen, C. Kemp, and W. Rogers, “Identifying the potential for robotics to assist older adults in different living environments,” *International journal of social robotics*, vol. 6, pp. 213–227, 04 2014.

[33] S. Murray, W. Floyd-jones, G. Konidaris, and D. J. Sorin, “A Programmable Architecture for Robot Motion Planning Acceleration,” in *International Conference on Application-specific Systems, Architectures and Processors*, ser. ASAP. IEEE, 2019, pp. 185–188.

[34] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. Konidaris, “Robot motion planning on a chip,” in *Robotics: Science and Systems*, 2016.

[35] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, “The microarchitecture of a real-time robot motion planning accelerator,” in *Proceedings of the ACM/IEEE International Symposium on Microarchitecture*, ser. MICRO. IEEE Press, 2016.

[36] N. Nethercote and J. Seward, “Valgrind: A framework for heavyweight dynamic binary instrumentation,” *SIGPLAN Not.*, vol. 42, no. 6, p. 89–100, jun 2007. [Online]. Available: https://doi.org/10.1145/1273442.1250746

[37] Nvprof, “command line profiling tool,” http://docs.nvidia.com/cuda/profiler-users-guide/, 2023.

[38] T. Pachidis, C. Sgouros, V. G. Kaburlasos, E. Vrochidou, T. Kalampokas, K. Tziridis, A. Nikolaou, and G. A. Papakostas, “Forward kinematic analysis of jaco2 robotic arm towards implementing a grapes harvesting robot,” in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2020.

[39] J. Pan and D. Manocha, “Gpu-based parallel collision detection for fast motion planning,” *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 187–200, 2012.

[40] E. Plaku and L. Kavraki, “Distributed sampling-based roadmap of trees for large-scale motion planning,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 3868–3873.

[41] A. Qureshi, Y. Miao, A. Simeonov, and M. Yip, “Motion planning networks: Bridging the gap between learning-based and classical motion planners,” *IEEE Transactions on Robotics*, vol. PP, pp. 1–19, 08 2020.

[42] Rethink Robotics, “Baxter,” 2013. [Online]. Available: https://robots.ieee.org/robots/baxter/

[43] D. Shah, N. Yang, and T. M. Aamodt, “Energy-efficient realtime motion planning,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA ’23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589092

[44] Z. Shiller and S. Sharma, “High speed on-line motion planning in cluttered environments,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 596–601.

[45] D. J. Sorin, G. Konidaris, W. Floyd-Jones, and S. Murray, “Motion planning for aotonomous vehicles and reconfigurable motion planning processor,” 2019, patent No. US20190163191A1, Filed June 9, 2017, Issued May 30, 2019.

[46] J. E. Stine, I. D. Castellanos, M. H. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, “FreePDK: An open-source variation-aware design kit,” in *IEEE International Conference on Microelectronic Systems Education (MSE)*, 2007, pp. 173–174.

[47] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. V. Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox, “curobo: Parallelized collision-free minimum-jerk robot motion generation,” 2023.

[48] Y. Yang, X. Chen, and Y. Han, “Dadu-cd: Fast and efficient processing-in-memory accelerator for collision detection,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[49] T. Y. Yeh, P. Faloutsos, and G. D. Reinman, “Accelerating real-time physics simulation by leveraging high-level information.” [Online]. Available: https://api.semanticscholar.org/CorpusID:802899

[50] C. Yu and S. Gao, “Reducing collision checking for sampling-based motion planning using graph neural networks,” in *Proceedings of the 35rd International Conference on Neural Information Processing Systems*, 2021.

Appendix

This paper proposes a collision prediction approach and collision prediction unit (COPU). We further integrate the proposed COPU with an existing collision detection unit (CDU). This artifact provides the complete implementation for the COPU+CDU simulator and the traces used for evaluation reported in Section VI. Section A provides basic information about the artifact. We describe the artifact description and installation procedure in Section B. Section C describes the experiment workflow for evaluation using the provided trace files.

A. Artifact Check-List (Meta-Information)

- **Program:** We provide an implementation of different collision prediction approaches and microarchitecture simulators for COPU+CDU accelerators.
- **Run-time environment:** All scripts are tested on Linux (Ubuntu 16.04) and macOS (12). It does not require root access.
- Hardware: All evaluation experiments require only CPU (no specific requirement).
- **Metrics:** Precision and Recall are reported to compare collision prediction approaches. Computation reduction, performance/watt, performance/mm^2, and throughputs are reported using microarchitectural simulators.
- **Output:** Numerical results and graphs are reported in Section III and Section VI.
- **Experiments:** README provided with instructions. Provided bash scripts to run all experiments.
- **How much disk space required (approximately)?:** 2GB.
- **How much time is needed to prepare workflow (approximately)?:** Less than 1 hour.
- **How much time is needed to complete experiments (approximately)?:** 2 hours for main evaluation using a single machine.
- **Publicly available?:** Yes.
- **Licenses (if publicly available)?:** The simulator code is available under Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
- **Archived (provide DOI)?:** 10.5281/zenodo.10982914

B. Description and Installation

1) How to access it: Our implementation is available on Zenodo: https://doi.org/10.5281/zenodo.10982914 and Github: https://github.com/ubc-aamodt-group/coll_prediction_artifact. This repository consists of the microarchitectural simulators and traces used for evaluation.

2) Hardware dependencies: There are no specific hardware requirements, a CPU system should be sufficient.

3) Software dependencies : Our artifact has been tested on Ubuntu 16.04 and macOS 12. It does not require root access. The execution requires Python 3.6.12 additional Python packages. Conda can be optionally installed to create a virtual environment.

4) Installation: Create a virtual environment and activate with python==3.6.12 (**optional to use virtual environment).

```
git clone https://github.com/ubc-aamodt-group/
    coll_prediction_artifact
cd coll_prediction_artifact
conda create -n pred_env python==3.6.12
conda activate pred_env
python -m pip install -r requirements.txt
#Download required traces
bash download.sh
```

C. Evaluation

1) Comparison of collision prediction approaches: The experiments required for Figure 9, Figure 13, and Figure 14 can be executed using the following set of scripts. All scripts should take less than 20 minutes.

```
cd coll_prediction_artifact/prediction_approaches
## Run all experiments for Figure 9
bash fig9.sh
## Run all experiments for Figure 13
bash fig13.sh
## Run all experiments for Figure 14
bash fig14.sh
```

2) Collision prediction for motion planning (COPU+CDU): The experiments required for Figure 15, Figure 16, and results reported in Section VI-B2 can be run using the following commands.

```
cd  coll_prediction_artifact/motion_planning_prediction
## Run all experiments for Figure 15
bash fig15.sh
## Run all experiments for Figure 16
bash fig16.sh
```

D. Trace Generation

We have provided required trace files for Section C1 and Section C2 in the “coll_prediction_artifact/trace_files” folder. In this section, we provide scripts to generate these traces for different robot poses, environmental scenarios, and motion planning algorithms as examples of the trace generation process.

1) Pose and Coordinate information for comparing prediction approaches: Comparison of different prediction approaches (Section C1) uses different robot poses in a given environmental scenario (i.e., placement of obstacles). The scripts provided below generate 400 environmental scenarios with 1000 robot poses sampled per environmental scenario. These scripts will store generated traces in “coll_prediction_artifact/trace_generation/scene_benchmarks” folder, which can be used for evaluation outlined in Section C1 instead of “coll_prediction_artifact/trace_files/scene_benchmarks”.

```
cd coll_prediction_artifact/trace_generation
conda deactivate
conda create -n newenv python==3.7.0
conda activate newenv
python -m pip install -r requirements.txt
bash launch_pred.sh
```

2) Motion trace generation for COPU+CDU: We provide an example of trace generation for a motion planning algorithm for the evaluation of COPU+CDU using a microarchitectural simulator. We give implementation for BIT*-KUKA motion planning, and a similar approach can be used for other motion planning algorithms. Note that the scripts below were tested on Ubuntu 18.04 and macOS 12.0.

```
cd coll_prediction_artifact/trace_generation/
    bit_planning
conda deactivate
## For Ubuntu 18.04
conda env create -f environment.yml -v
## For macOS (version 12.0)
conda env create -f environment_macos.yml -v
conda activate myenv
bash launch_bit_trace.sh
```

The above scripts will generate trace files required for collision prediction simulation using the BIT* motion planning algorithm for the KUKA robot. The above scripts will store generated traces in “trace_generation/bit_planning/logfiles_BIT_link” folder, which can be used for evaluation outlined in Section C2 for BIT*-KUKA combination instead of using provided traces in “trace_files/motion_traces/logfiles_BIT_link”.