Ray Distribution to Parallel Batching-based Updates

Youngsun Kwon[1] and Sung-eui Yoon[2]

Abstract— **We propose a novel approach to distribute rays associated with point clouds for efficiently updating tree-based occupancy maps in parallel. In this paper, we utilize threads to batch a set of cells independently, although some of those cells are overlapped among threads, for exploiting the high computational power of multi-threading. To compensate redundant updates caused by having the overlapped cells, we propose a k-d tree based method to distribute rays for minimizing the redundant updates. In addition, we define a workload that each thread processes as the number of cells to be batched. Based on the definition, our method distributes the workload to threads uniformly for efficiently performing the batching process in parallel. We test our method into a corridor benchmark, and achieve 5.2 times performance improvement on batching process using 8-threads, resulting in up to 2.0 times performance improvement on overall updates, over the state-of-the-art update method for octree-based occupancy maps.**

I. INTRODUCTION

Many robotic systems use point clouds data for sensing and understanding their environments. The point clouds consists of a large amount of points providing useful geometric information of environments, which also comes with various levels of sensor noise. However, it is difficult to use the data directly in applications such as motion planning or collision detection, because of the large amount of generated data as well as the noise.

To address these issues, various occupancy maps such as grids [1] and octrees [2] have been proposed as representing the geometric information and uncertainty of point clouds. Among those occupancy maps, tree-based maps such as quadtrees and octrees are used for reducing the memory requirement and accelerating the performance of applications such as collision detection. Unfortunately, constructing those maps out of point clouds can take a long time for maintaining the tree-based structures.

To improve the performance of building such occupancy maps, Hornung et al. [2] proposed a method to batch a set of cells to be updated before updating the map. In the batching-based updates, a set of cells that rays traverse can be found in parallel, but at least one of those cells found in threads overlaps, i.e. a cell containing the sensor origin. Thus batching the traversed cells need to be processed in serial, to merge the set of cells found in threads. While the batching process improves the overall performance of updates, the batching process can be still slow (e.g., 75% of overall update time as shown in Fig. 3-*b*)) for large-scale point clouds and be the computational bottleneck of the batching-based updates for tree-based occupancy maps.

[1]Youngsun Kwon and [2]Sung-eui Yoon are at School of Computing, KAIST, Daejeon, Korea; youngsun.kwon@kaist.ac.kr, sungeui@kaist.edu

Main contributions In this paper, we propose a novel method to parallelize the batching process for fast updating a tree-based occupancy map. For exploiting a high computational power of multi-threading on the batching process, we assign each thread to batch the traversed cells, while allowing some of those cells to be overlapped instead of imposing the locking mechanism. Furthermore, we define a workload for batching as the number of traversed cells, and then balance the workload to be assigned to each thread. We aim to efficiently consider those issues of minimizing the number of overlapped cells and balancing the workload for each thread together, and thus we propose an efficient k-d tree based method to distribute rays of point clouds to threads.

To demonstrate benefits of our methods, we test it with a public dataset for an octree-based occupancy map according to various resolutions as well as the various number of threads. We found that our method is practical enough to exploit the high computational power of multi-threading. Our method shows 5.2 times performance improvement on batching process when we use 8-threads, which results in the 1.8 times performance improvement of overall updates.

II. RELATED WORK AND BACKGROUND

When we have a point from a sensor, the point implies that the space between the sensor origin and the point is collision-free. To reflect the information on occupancy maps, we associate a ray traversing cells of the map from the sensor origin toward the point. The cell containing the end point of the ray is updated to have an occupied state, and the other traversed cells are updated to have a free state. This process of traversing cells from a ray is similar to ray tracing that has been studied in the graphics literature [3], [4].

In the robotics, a recent occupancy map, OctoMap [2], uses the 3DDDA based algorithm [3] for updating the map with point clouds. The data captured by a sensor has uncertainty from sensor noise, unlike data representations (e.g., triangles) in the graphics field. A cell of maps, therefore, has an occupancy probability representing an occupancy state of the cell such as occupied or free states, where the probability is modeled by the Bayes rule given sensor measurements from the initial time step 1 to the current time t.

Some works proposed a simple update expression using the log-odds notation for fast updates; the detailed information for the occupancy probability is available in the prior works [2], [5], [6]. Based on the model, recent work [2], [7] proposed methods to improve update performance of grid or tree-based occupancy maps.

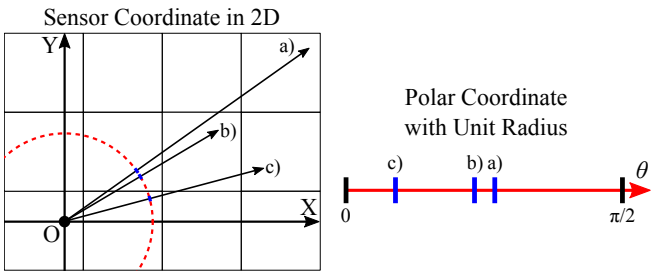


Fig. 1. This figure shows an observation that rays, associated with adjacent points in the polar coordinates, traverse a similar set of cells. The three rays – a), b), and c) – staring from the sensor origin O are mapped onto an unit circle represented by red dotted circle. In the polar coordinate, the mapped points associated with the three rays are shown as the blue lines.

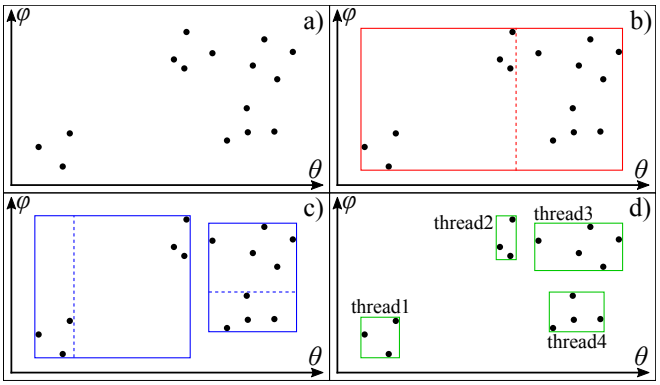


Fig. 2. These figures show an example process to cluster given points in the spherical coordinate using our criterion on workload. b) represents the first step of clustering given points shown in a). We create a bounding box of the points, and then partition the box into two boxes using our criterion. c) recursively repeats the process using the points clustered in the first step. The final output is shown in d) represented by four green boxes, where rays associated with the points within each box are assigned to each thread.

III. Real-Time Updates using Parallelization

A. Motivations

Unlike grid-based occupancy maps, tree-based maps such as quadtree and octree should update the occupancy probabilities of cells from a leaf to the root for maintaining the tree-based structure. When multiple rays try to update the same leaf cell, it results in repeated accesses from the leaf to all the parent cells and can be even an computational bottleneck of updating those maps. To avoid the duplicated accesses, prior work [2] proposed a method to batch a set of leaf cells that rays traverse before updating the occupancy maps. The batching-based method updates tree-based maps in a single time using the batched cells, improving the performance of updating tree-based occupancy maps.

Unfortunately, an overhead of batching cells can take a large portion of the total time of updating maps. The batching-based method can find cells that rays traverse in parallel. However, the same cell found in different threads, e.g., the cell containing the sensor origin, requires a lock on the cell for merging the information, i.e., counting the number of access to the cell, from those different threads. It has been well known that locking deteriorates the performance of parallel computing [8]. Ideally, we would like to design a lock-free technique for batching a set of cells in a parallel manner, to achieve high parallel performance.

B. Overview of Our Approach

For achieving high performance of updating tree-based occupancy maps, we propose an efficient, lock-free method to process the batching process in parallel. Specifically, we let each thread to have its own set of traversed cells for batching, and run different threads in a parallel manner. This approach may result in having an extra memory space on maintaining duplicate cells across different threads, but we can avoid any locks. In terms of the update process, those duplicate cells batched in multiple threads can make repeated access to parents of those cells, then giving rise to inefficient updates. Therefore, we propose a method to distribute rays of point clouds to threads using a k-d tree for minimizing the number of duplicate, overlapped cells (Sec. III-C). In addition, we define a workload as the number of traversed cells to be batched in a thread, and then balance the workloads of threads during distribution for efficient parallel batching process (Sec. III-D). As a result, our method reduces the batching time and improves the overall performance thanks to lock-free and thus efficient multi-threading.

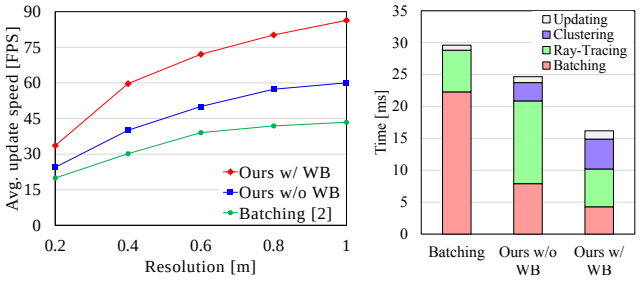
C. Ray Distribution using K-D Tree

In general, rays associated with point clouds are defined in the sensor coordinate. To minimize the number of overlapped cells to be batched in threads, we should cluster the rays that traverse a similar set of cells. As shown in Fig. 1, we observe that rays traversing similar cells are mapped onto adjacent points on the surface of unit circle for 2D case (or sphere for 3D), where the surface can be represented as the polar or spherical coordinate system with unit radius.

In the example of Fig. 1, the end point of ray b) is closer to the end point of ray c) than that of ray a). However, ray b) traverses more similar cells to ray a) than ray c). This pattern is more easily identified in the polar coordinate than the original, sensor coordinate. Note that ray b) is closer to ray a) than ray c) in the polar coordinate.

Based on our observation, we associate rays with points in the polar or spherical coordinate with unit radius. In this space, we cluster the adjacent points and assign them to a thread. In this way, we can have similar access pattern for each thread and thus have dissimilar patterns between different threads, reducing the number of overlapped cells among threads.

To cluster the points efficiently, we use a space-partitioning algorithm based on the k-d tree. As an example shown in Fig. 2, we make a bounding box from given points in the spherical coordinate, and can then partition the box along the orthogonal direction to its longest edge; the detail partitioning method is discussed in the next section. Our method recursively performs partitioning until we get the same number of boxes to the number of threads. Finally, each thread batches cells in parallel using the rays associated with the distributed points within a bounding box, as minimizing the number of overlapped cells.



(a) Update Speed (b) Time Breakdown

Fig. 3. These figures show performance of updating an octree-based occupancy maps in the dataset using 8-threads. We measure the average performance according to various resolutions for different methods in figure *a*). Also, we report the average breakdown of the total time taken for updating the map with 0.6m resolution in figure *b*). WB indicates our workload balancing method.

D. Partitioning for Workload Balancing

In the prior method, we discussed how to distribute rays based on the k-d tree. Nonetheless, we did not detail our chosen partitioning method for a box containing ray points. In a simple way, we can partition a bounding box using a criterion such as the middle point of the longest edge. However, that approach is not the best choice for our parallelization, especially in a case that one of threads has an excessive amount of heavy workload compared to others. We therefore propose a new criterion used in partitioning step for exploiting the high parallelization power.

We observe that the time to batch a set of cells that a ray traverses depends on the number of traversed cells. Using the observation, we define a workload of thread as the number of cells to be batched in the thread. By adopting the definition of workload as a criterion of partitioning, our method partitions a given box into two boxes with the same workload.

Finally, our method clusters sets of rays having the equal workloads, and then assigns threads to batch those sets, resulting in high parallelization performance.

IV. Results and Discussions

We use a machine that has 3.4GHz Intel i7-4770 CPU (maximum 32-threads) for our experiments. To compare the performance of proposed methods, we use a public dataset used in the prior work [2]. The dataset consists of 66 scans captured in a corridor and the point clouds has 89,445 points on average. We compare our workload balancing (WB) against a simple partitioning method breaking the longest edge of each bounding box through its middle point.

We compare the performance of our methods and the batching-based update method proposed in OctoMap [2]. In the tests for performance, we use octree-based occupancy maps with default parameters such as inverse sensor model provided in the OctoMap library. We report the overall performance of methods as the average Frame Per Second (FPS) on the 66 scans (frames) in Fig. 3-*a*). We also report the time distribution to perform each component of ours and the batching method in Fig. 3-*b*).

Fig. 3-*a*) shows the update speed of the methods using 8-threads. In the settings, our methods show the higher performance than the state-of-the-art method. Compared to the prior work, our method without workload balancing shows up to 1.4 times performance improvement on the overall updates. As shown in Fig. 3-*b*), we improve performance 2.8 times on batching process (from 22.3ms to 7.3ms), with a few overhead, 2.7ms, to distribute the rays.

TABLE I

This table shows the performance[FPS] of updating according to various numbers of threads.

OctoMap with 0.6m resolution					
	Overall performance [FPS]				
# of threads	2	4	8	16	32
Batching [2]	48.3	42.7	39.1	38.5	39.3
Ours w/o WB	52.8	52.5	50.1	52.6	56.7
Ours w/ WB	**54.3**	**67.8**	**72.1**	**74.6**	**78.2**

Furthermore, our method with workload balancing (the red line in the Fig. 3-*a*)) shows the best performance in all the tests. Using our workload balancing criterion, we achieve 5.2 times performance improvement on the batching process compared to the prior work, reporting the time decrease from 22.3ms to 4.3ms. As a result, our method with workload balancing provides up to 2.0 times overall performance improvement over the state-of-the-art method.

Table I shows overall performance of updates according to the various number of threads. In the all tests, we achieve the 1.3 and 1.7 times performance improvement on average, without and with our workload balancing, respectively. In this result, our method shows the high performance as the number of threads increases. In particular, our method with workload balancing outperforms other methods thanks to better exploiting the high-computational power of multi-threading.

Acknowledgment

This work was supported in part by MI/KEIT 10070171 and MSIP/IITP [R0126-17-1108].

References

[1] H Moravec, "Robot spatial perceptionby stereoscopic vision and 3d evidence grids", *Perception,(September)*, 1996.
[2] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees", *Autonomous Robots*, 2013.
[3] John Amanatides, Andrew Woo, et al., "A fast voxel traversal algorithm for ray tracing", in *Eurographics*, 1987, vol. 87, p. 10.
[4] Steven Parker, Michael Parker, Yarden Livnat, Peter-Pike Sloan, Charles Hansen, and Peter Shirley, "Interactive ray tracing for volume visualization", *Visualization and Computer Graphics, IEEE Transactions on*, vol. 5, no. 3, pp. 238–250, 1999.
[5] Alberto Elfes, "Using occupancy grids for mobile robot perception and navigation", *Computer*, vol. 22, no. 6, pp. 46–57, 1989.
[6] Hans P Moravec and Alberto Elfes, "High resolution maps from wide angle sonar", in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*. IEEE, 1985, vol. 2, pp. 116–121.
[7] Youngsun Kwon, Donghyuk Kim, and Sung-eui Yoon, "Super ray based updates for occupancy maps", in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4267–4274.
[8] Duksu Kim, Jae-Pil Heo, Jaehyuk Huh, John Kim, and Sung-Eui Yoon, "HPCCD: Hybrid parallel continuous collision detection", *Comput. Graph. Forum (Pacific Graphics)*, vol. 28, no. 7, 2009.