SpecNN: A Hardware Accelerator for k-Nearest Neighbors

Davis Sheppard, Yan Cheng Poon, Tanay Sharma, Aarti Phatke, Yunqi Zhang, Shreya Nadiger

Agenda

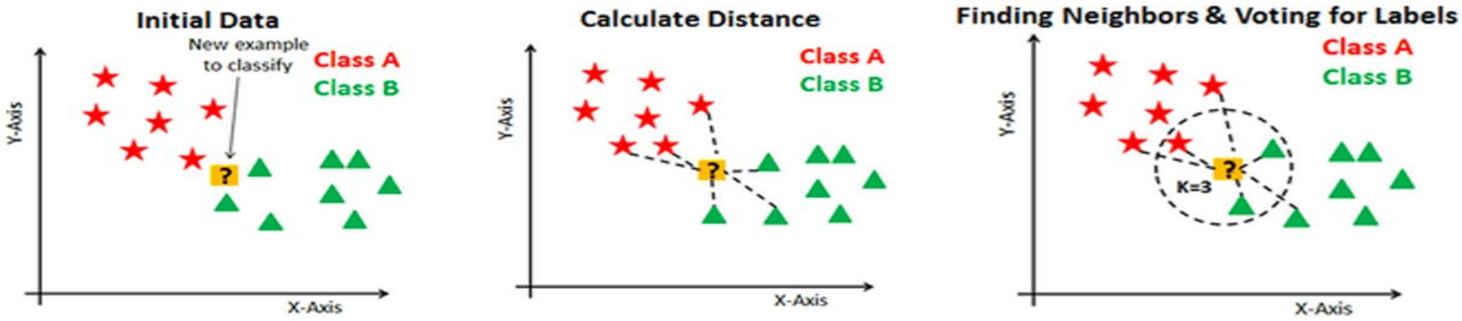
- Introduction
- Motivation & Problem Statement
- Bit-serial Computation Introduction
- Architecture Overview
- Implementation
- Optimizations
- Evaluation & Datasets
- Conclusion
- Future Work

Introduction

K Nearest Neighbors (KNN) is a simple and widely used algorithm for classification and regression.

How KNN Works

- Receive a query point
- Compute distances to all dataset points
- Select the K nearest neighbors
- Predict using majority vote (classification) or averaging (regression)



Motivation

- High Demand

  Autonomous driving & robotics (LiDAR point-cloud matching)

  Computer vision (feature matching, image retrieval)

  Recommendation systems

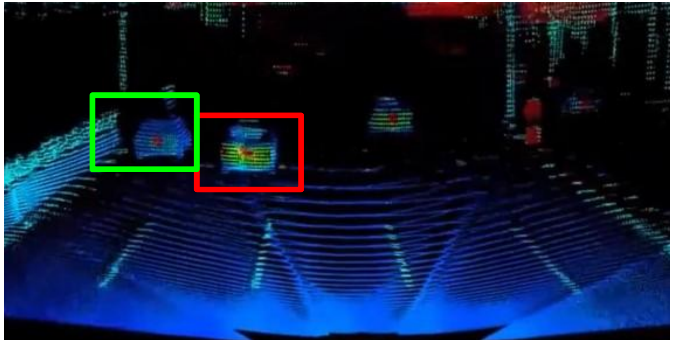
  Anomaly detection.

- No training, doesn’t need ordered data

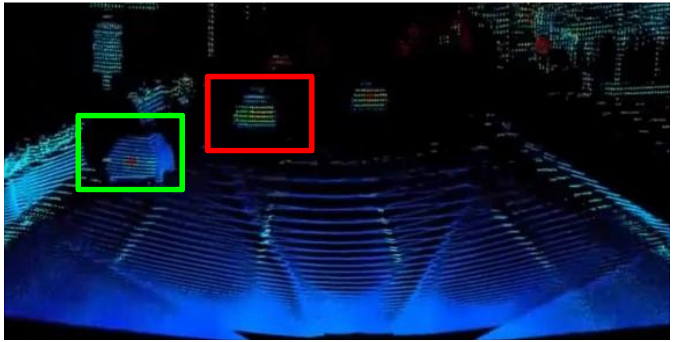
- KNN consumes up to 80% of total pipeline runtime.

Motivation

- Application example in autonomous driving:
  1. Continuously receive multiple frames of point clouds from the LiDAR
  2. Extract features in the frames using other algorithm (e.g. Convolutional Neural Network)
  3. Utilize kNN search to establish point correspondences between consecutive frames to detect vehicle motion



kNN finds correspondences between consecutive point cloud frames using distance, enabling algorithms to estimate motion.



Problem Statement

- Standard processors calculate the full Euclidean distance (32/64-bit) for every point.
- Most points are far away. Calculating full precision for "rejects" wastes energy and cycles.

Goal: Design a domain-specific accelerator for KNN that minimizes unnecessary computation and memory bandwidth/transaction.

Prev. Work: BitNN Bit-serial Distance Unit (BDU) [1]

Partial Squared Distance Formula:

$$dist_m^2(q,r) = dist_{m-1}^2(q,r) << 2 + \sum_{d=1}^{3}(q_{d,B-m} - r_{d,B-m})^2 +$$

$$\sum_{d=1}^{3}((q_{d,B-m} - r_{d,B-m}) \times f_{d,m-1}(q,r)) << 2$$

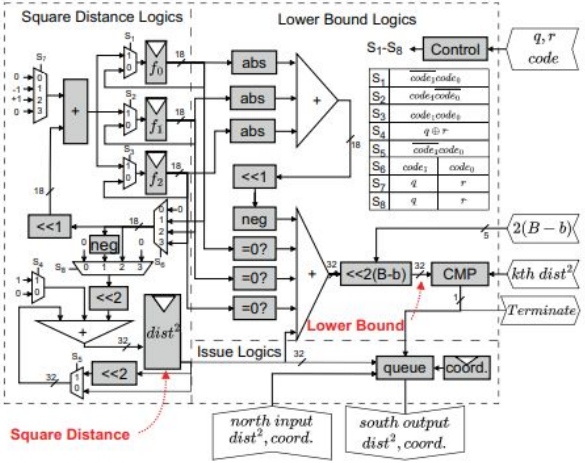
$$f_{d,m}(q,r) = f_{d,m-1}(q,r) << 1 + (q_{d,B-m} - r_{d,B-m})$$

Lower Bound Formula:

$$g_m^2(q,r) = \left(dist_m^2(q,r) - \sum_{d=1}^{3} h_{d,m}(q,r)\right) \times 2^{2(B-m)}$$

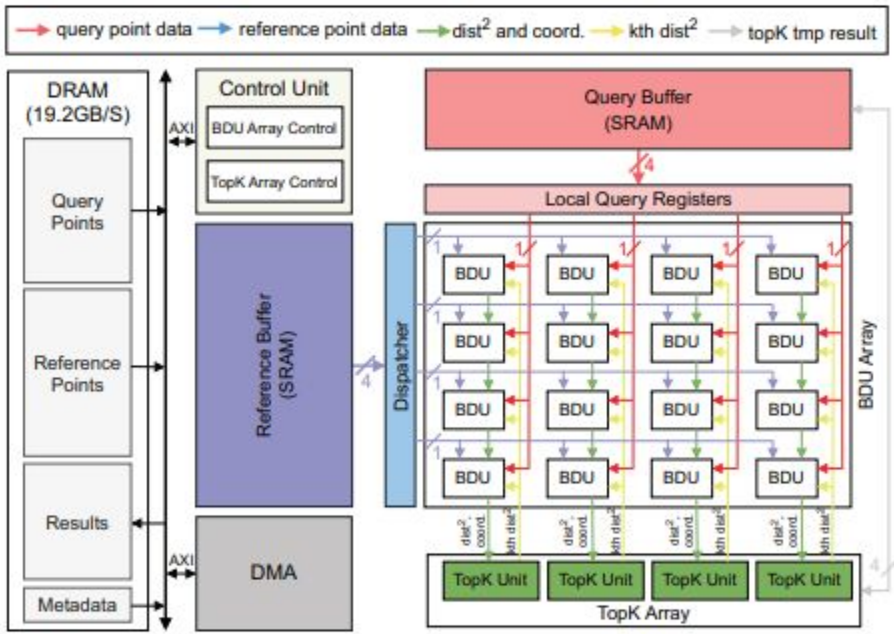
$$h_{d,m}(q,r) = \begin{cases} 2 \times |f_{d,m}(q,r)| - 1, & f_{d,m}(q,r) \neq 0 \\ 0, & f_{d,m}(q,r) = 0 \end{cases}$$

- Bit-by-bit computation of distance from MSB to LSB across all dimensions
- **Terminate if:** Lower Bound > Threshold
- **Done if:** Not Terminated by the LSB of the last dim



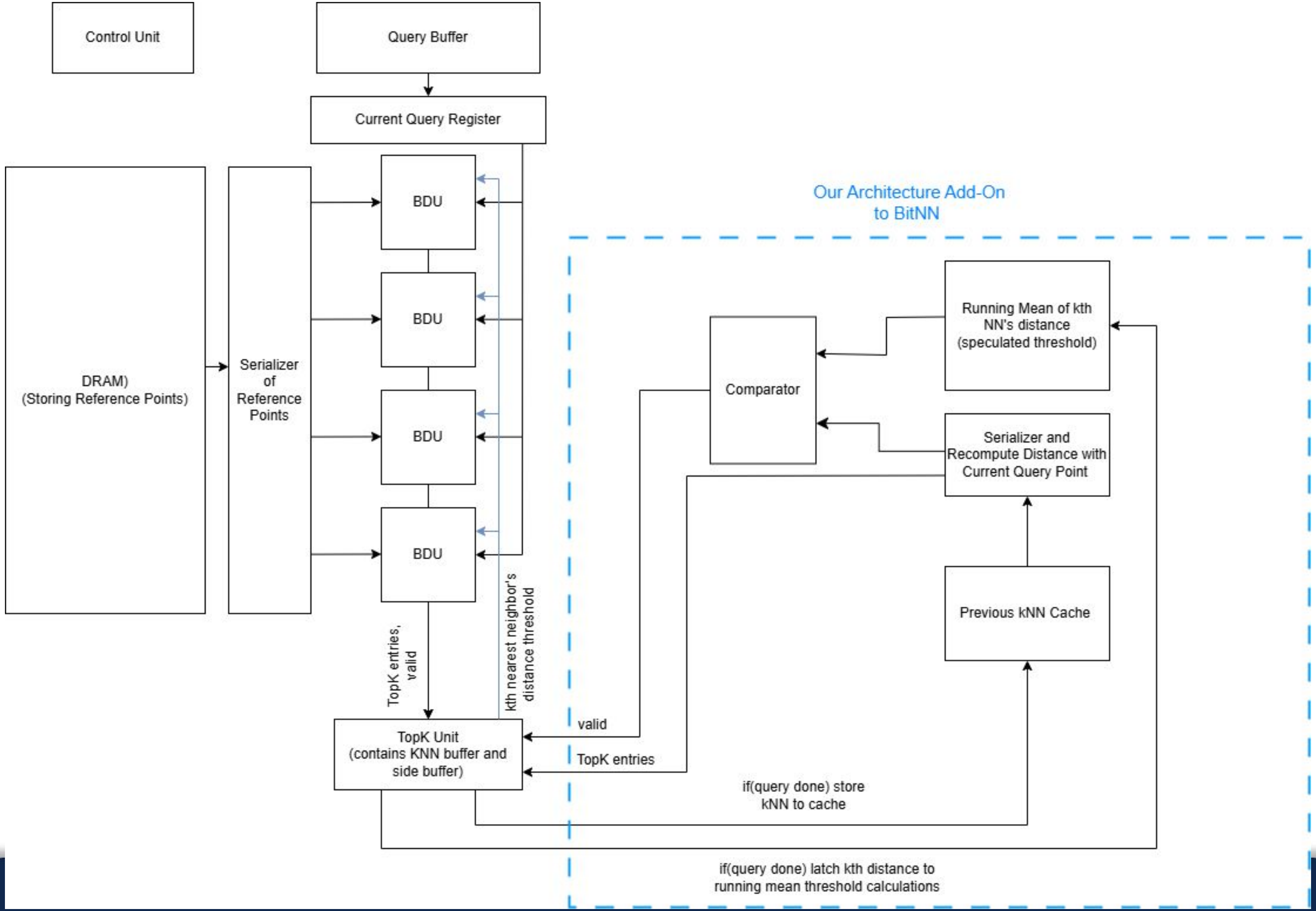
BDU Verilog Implemented by Us: https://github.com/EECS-573-KNN-Accelerator/eecs573project/blob/main/verilog/BDU.sv
Run the BDU test here: https://github.com/EECS-573-KNN-Accelerator/eecs573project/blob/main/test/BDU_test.sv

Prev. Work: BitNN Architecture



- Parallelizing reference points over an array of BDUs (rows)
- Parallelizing query point searches over array of query registers (columns)
- Storing not-terminated points to TopK unit that produces threshold based on the kth distance to Q

Architecture Overview



Optimization 1: PrevKNN Cache

- Hypothesis: **Spatial Locality of Query Points Exists** (Neighbouring Query Points will have spatial locality)
  - Autonomous Driving (KITTI Dataset) [2]
    - Average distance between consecutive frame points: < 2 meters (vehicle moves ~30 km/h = 8.3 m/s)
  - ICP Algorithm in Simultaneous Localization and Mapping (SLAM) [3]
    - In ICP registration, 85-90% of query points have their nearest neighbors within 0.5m in consecutive frames

- Idea: **Cache kNN of previous query point**, and at initialization, **compute distance between all previous kNN and the current query point**, and **use the new kth distance as a beginning threshold**

- Effect: **Reduce BDU Termination Warmup Latency** (warms the termination threshold faster, resulting in more effective and faster terminations)
  - Query points that are near to their previous query point will initialize their Top K unit with near-accurate candidates, resulting in faster terminations when looping through reference points

Optimization 2: Running Mean Threshold

- Hypothesis: **Sparsity of Reference Point Clouds Remain Largely Consistent** (kth NN distance or final threshold will be within a range across all query points)
    - While different applications have vastly different sparsity, sparsity within a reference point set presents a consistent pattern due to temporal continuity and physical constraints [4]
- Idea: **Latch the final kth distance for every query point for calculation of running mean across a window of query points, use the running mean (scaled by a multiplier) as the beginning threshold**
- Positive Effect: **Reduce BDU Termination Warmup Latency**
    - Reference point clouds with consistent sparsity will have a near-accurate termination threshold, resulting in faster terminations when looping through reference points
- Negative Effect: Valid non-terminated points might be less than k
    - Overly aggressive/tight running mean threshold might result in overly aggressive termination, resulting in less than k points collected.

Combined Algorithm Design

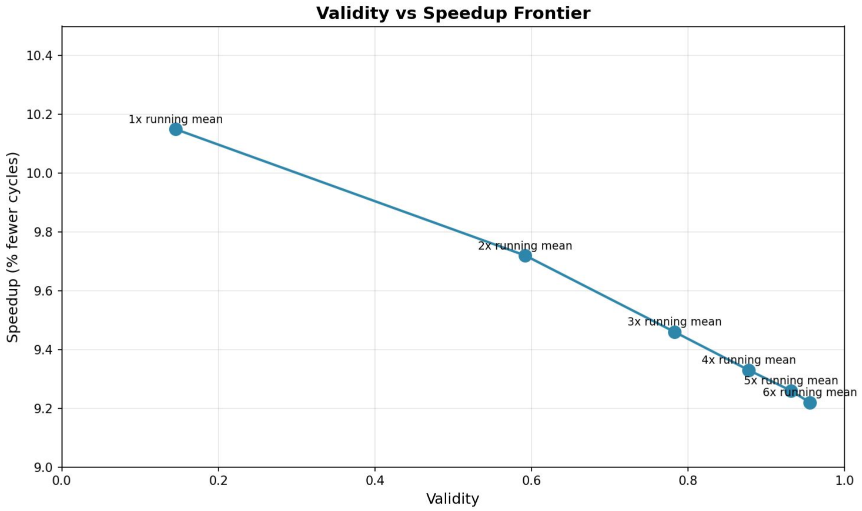
- At initialization, we run through each previous kNN points and compute distance to the current query point
- The computed distance will be compared against the current running mean threshold (scaled by multiplier)
  - If distance < threshold: point is added to current query's TopK as **valid** entry
  - Else: point is added to current query's TopK as **invalid** entry
- All reference points are shifted into TopK regardless of termination or not
  - Terminated Points are added as **invalid** entry, and **lower bound** is used as distance
  - Non-terminated points are **valid** entries, with actual distance as distance
- TopK unit **sorts entries by distance** and discard any excess
- At the end of the reference point traverse:
  - **Valid entries in TopK are accurate kNN, while invalid entries are speculated kNN** (which should be close to the actual kNN, because it is based on lower bound)
- We can adjust the scaling multiplier of running mean threshold to relax termination for more accuracy (valid entries/number of entries), but this increases cycle count (**tradeoff between accuracy and performance**)

Evaluation

- Created cycle-accurate Python simulation of BitNN and SpecNN
- Compared performance on test datasets
- Fine-tuned running mean heuristic for speed vs correctness

Evaluation Metrics

- Memory accesses
- Total cycles
- Average cycles per BDU
- Valid TopK entries



Data Sets

- Two main constraints: 3D feature space and spatial locality
- Sequential jumps from point to point in order to simulate real time sensor readings as feature vectors

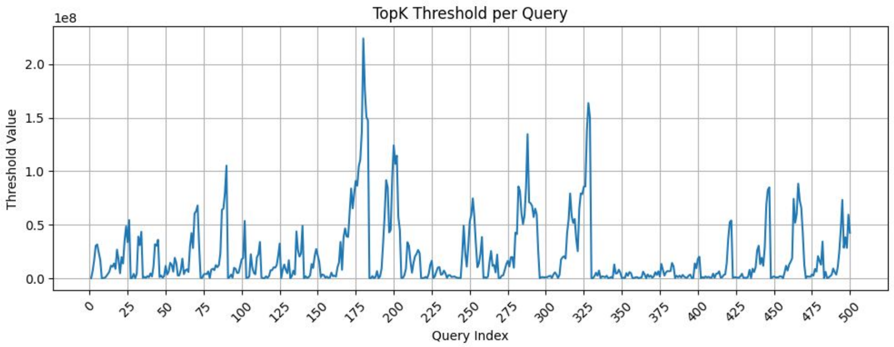
Initial Approach

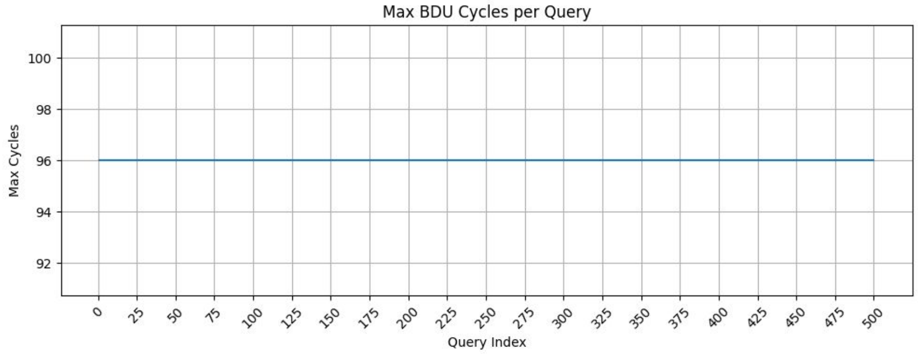
- Oxford feature embedding set
  - 4099 dimension feature set reduced to 3D feature space
  - 3D reduction did not represent full feature space

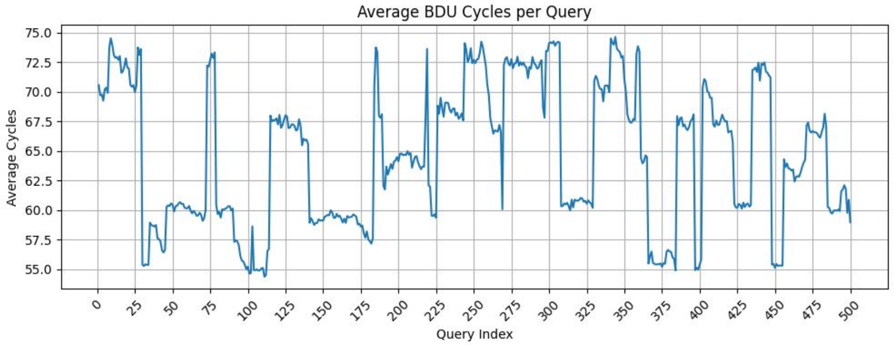
Synthetic Generation

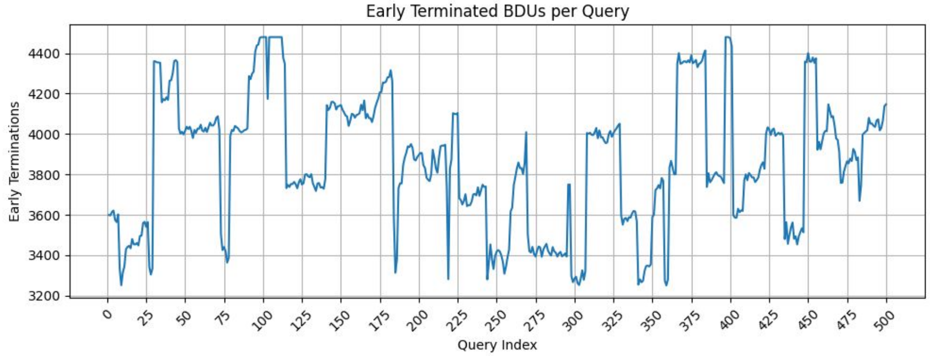
- One main drift (scale = drift_scale)
- A secondary drift with smaller variance (scale = drift_scale / 3) Mixed as:
  - 0.7 * big_drift + 0.3 * small_drift
- Convert to 64 bit blocks to represent memory
- Adequately captures real point cloud scenario

BitNN Baseline Simulation Results

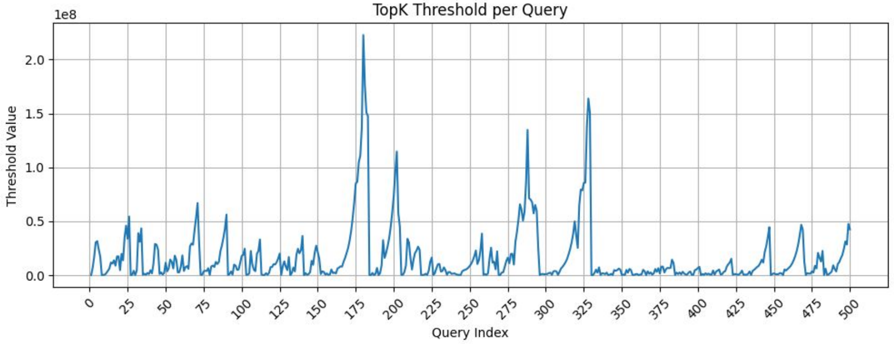


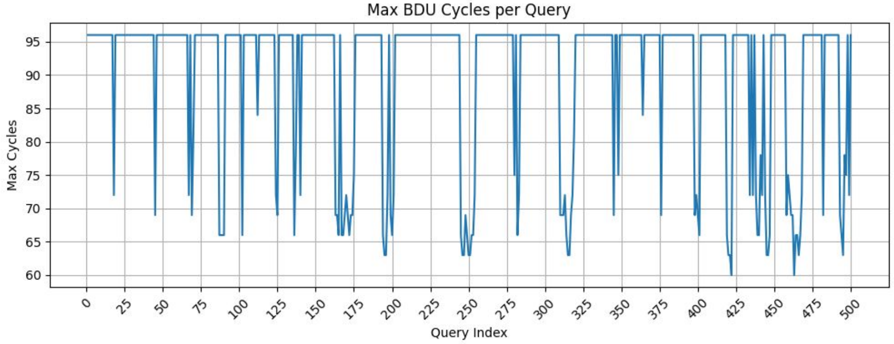


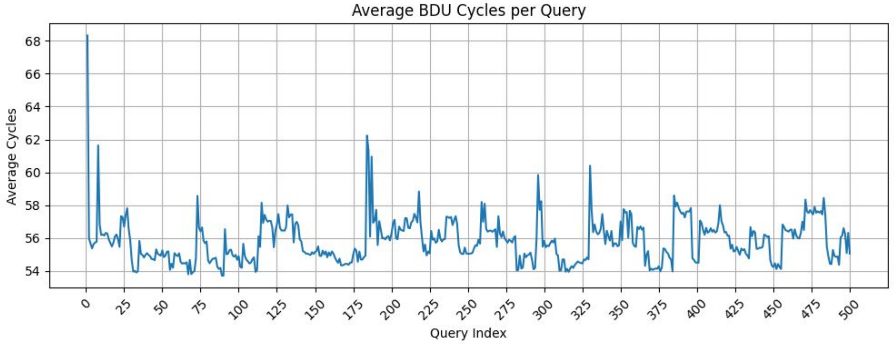


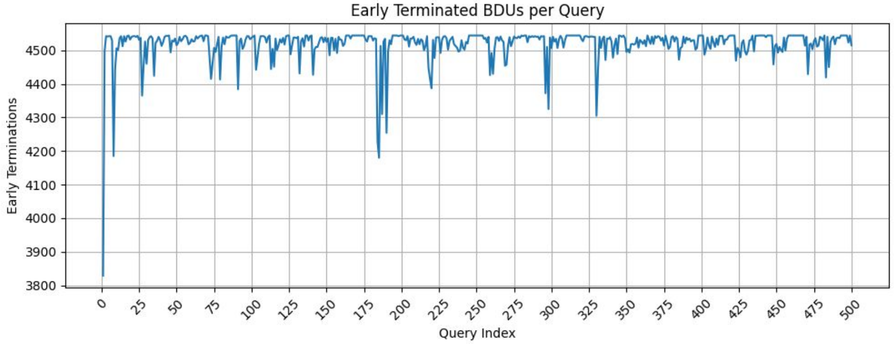


SpecNN Simulation Results









Conclusion

Goal: Leverage bit serial computation and domain specific knowledge to perform even more aggressive early termination than BitNN

Challenges:

- Amdahl's Law
- Data Set Distribution

Future Work

- Synthesis Testing
- Integration with GPU architecture
- Build out data processing pipeline for system level testing
- Testing non-lockstep BDU architecture

Q & A

References

[1] BitNN: A Bit-Serial Accelerator for K-Nearest Neighbor Search in Point Clouds,
https://ieeexplore.ieee.org/document/10609723

[2] KITTI Datatsets,
https://www.cvlibs.net/datasets/kitti/eval_odometry.php#:~:text=The%20evaluation%20table%20below%20ranks,split%20of%20the%20training%20set.

[3] ICP in Simultaneous Localization and Mapping,
https://cw.fel.cvut.cz/b192/_media/courses/aro/tutorials/icp_slam.pdf

[4] PointAcc, Efficient Point Cloud Accelerator,
https://dl.acm.org/doi/fullHtml/10.1145/3466752.3480084#:~:text=For%20instance%2C%20outdoor%20LiDAR%20point,never%20dilate%20during%20the%20computation.