Parallelized KD Tree for Parallel k-Nearest Neighbors Search

Doreen Valmyr (dvalmyr)
Neelansh Kaabra (nkaabra)

Project Github URL: https://github.com/doreenvalmyr/ParallelKDTree

SUMMARY

Our project aims to create a robust parallel kNN search algorithm by leveraging a **parallelized**, **lock free KD tree** for efficient data traversal. The parallel KD tree will be utilized to parallelize the search process, distributing the workload across multiple threads. The overarching goal is to achieve **high-performance kNN search** in large datasets within a **parallel computing environment**.

BACKGROUND

Application Overview:

The project focuses on accelerating the k-nearest neighbors (kNN) search, a compute-intensive task often used in applications such as spatial databases, machine learning, and data mining. The primary goal is to enhance the efficiency of identifying the k-nearest neighbors for a given query point within a large dataset. The accelerated algorithm leverages a parallelized KD tree, a space-partitioning data structure designed to efficiently organize multidimensional data, while also making it lock free.

Description:

The kNN search involves finding the k-nearest neighbors of a given query point in a dataset based on a distance metric, commonly Euclidean distance. The KD tree, constructed in a parallelized manner, serves as the underlying data structure for this task. The KD tree is a binary tree where each node represents a region in the data space, splitting it into two half-spaces along a chosen dimension. This hierarchical structure enables a more efficient search by eliminating large portions of the data space early in the process.

Parallelized KD Tree Construction:

The construction of the KD tree is parallelized to expedite the process of organizing the dataset. Each thread is responsible for constructing specific portions of the tree, and load-balancing techniques ensure that the workload is evenly distributed. Lock-free algorithms, such as Compare-and-Swap, are employed to manage concurrent access to shared data structures during tree construction.

Parallel kNN Search Algorithm:

During the kNN search, the parallelized KD tree traversal enables multiple threads to explore different branches simultaneously. As threads reach leaf nodes, they independently perform local kNN searches within those nodes. Proper synchronization mechanisms and thread-safe data structures facilitate the aggregation of local results to obtain the final set of k-nearest neighbors.

Benefits of Parallelism:

The compute-intensive nature of kNN search benefits significantly from parallelism. Parallelized KD tree construction allows for faster organization of the dataset, and the parallel kNN search ensures that multiple threads contribute to identifying nearest neighbors concurrently. As the size of the dataset grows, the parallel approach enables efficient exploration of the KD tree and reduces the overall search time. The inherent parallelism in both KD tree construction and kNN search aligns with the computational demands of large-scale datasets, making the algorithm well-suited for modern parallel computing architectures.

CHALLENGE

The challenge lies in efficiently parallelizing the k-nearest neighbors (kNN) search with a parallelized KD tree, balancing high computation-to-communication ratios, and managing divergent execution paths. The workload exhibits complex memory access patterns, with dependencies arising during KD tree traversal and result aggregation. Locality within leaf nodes enhances parallelism, but effective coordination is crucial during the merging of local kNN results. Constraints arise from the irregular nature of the KD tree, dynamic workload, and uneven data distribution, demanding sophisticated load-balancing strategies. Addressing these challenges not only accelerates kNN search but also contributes valuable insights into optimizing irregular and computation-intensive algorithms in parallel, enhancing the efficiency of real-world applications in parallel computing environments.

RESOURCES

For this project, the primary computational resources include multi-core CPU platforms for initial development and testing, taking advantage of parallelism in algorithm design. The project builds upon existing parallel computing frameworks, such as OpenMP and MPI, to enable workload distribution and coordination among threads during KD tree construction and kNN search. The ISPC compiler, known for its efficiency in parallel processing, may be explored for optimizing certain aspects of the algorithm.

Reference materials include research papers on parallel traffic simulations, KD trees, and kNN search algorithms, with specific emphasis on "The k-d tree data structure and a proof for neighborhood computation in expected logarithmic time” by Martin Skrdozki et al. (2019) and "Parallel Batch Dynamic kD trees" by MIT CSAIL. While the project currently leverages general-purpose multi-core CPUs, access to specialized machines with GPU capabilities, such as the late-days cluster's GPU-enabled machines, could potentially enhance performance and facilitate comparative analysis.

GOALS AND DELIVERABLES

Primary Goals (Must Achieve):

1. Implement Parallelized KD Tree Construction: Develop a parallel algorithm for constructing KD trees, ensuring efficient load balancing and lock-free mechanisms to accommodate concurrent access.

2. Parallel kNN Search Algorithm: Design and implement a parallel kNN search algorithm that leverages the constructed KD tree. Achieve effective workload distribution, synchronization, and result merging among threads.

3. Performance Benchmarking: Conduct comprehensive benchmarking to assess the parallelized algorithm's performance. Measure speedup and scalability with varying dataset sizes and thread counts. Aiming for a substantial speedup over the sequential version, with a target of at least 5x improvement.

4. Thorough Documentation: Provide clear and comprehensive documentation detailing the design, implementation, and performance analysis of the parallelized KD tree for kNN search. Include pseudocode, flowcharts, and explanation of key algorithmic decisions.

Additional Goals (Hope to Achieve):

1. Optimization for GPU Acceleration: If progress allows, explore the adaptation of the parallel algorithm for GPU acceleration, aiming to further enhance performance.

2. Real-Time Responsiveness: If ahead of schedule, aspire to achieve real-time responsiveness for the kNN search by optimizing the algorithm and minimizing overhead.

Demo at Poster Session:

Showcase an interactive demo highlighting the parallelized KD tree construction and kNN search. Display the efficiency of the algorithm through speedup graphs, illustrating its performance benefits over a sequential approach. Provide visualizations of KD tree structures and demonstrate the algorithm's responsiveness to varying query scenarios.

Analysis Objectives:

- Explore the impact of varying dataset characteristics (e.g., density, dimensionality) on the algorithm's performance.

- Investigate the scalability of the parallel algorithm concerning the number of threads and dataset sizes.

- Answer key questions related to the efficiency gains achieved through parallelization, identifying potential bottlenecks, and providing insights for future optimizations.

System Capabilities and Performance Expectations:

- The parallelized system should be capable of efficiently handling large datasets for kNN queries in real-time or near-real-time, demonstrating the scalability of the algorithm.

- Aim for a speedup of at least 5x compared to the sequential version, ensuring that the parallelized solution provides significant performance benefits for practical applications in kNN search.

The ultimate goal is to deliver a highly efficient, scalable, and well-documented parallel algorithm for kNN search, showcasing its real-world applicability and performance advantages in parallel computing environments.

PLATFORM CHOICE

Multi-core CPUs provide a cost-effective and widely accessible environment for initial algorithm design and testing. OpenMP simplifies shared-memory parallelism for KD tree construction and intra-node kNN search, while MPI offers scalability to distributed computing if needed. This choice allows for adaptability to more specialized platforms, such as GPU-enabled systems, for potential performance optimization. The platform's flexibility aligns with the project's goal of delivering an efficient and scalable parallel algorithm for kNN search.

SCHEDULE

Task	**Dates**
Project Proposal	Nov 7 - Nov 15
Parallelize KD tree Construction	Nov 19 - Nov 29
Parallel kNN Search	Nov 30 - Dec 5
Milestone Report	Nov 29 - Dec 3
Performance Benchmarking	Dec 6 - Dec 10
Final Report	December 7 - Dec 14