Parallelizing the NSW Search Algorithm

Anudeep Rao Perala
Guide: Dr. Sathya Peri
Indian Institute of Technology Hyderabad

Introduction

- We started the project under the goal for parallelizing the HNSW algorithm which is a method for efficient approximate nearest neighbor search in high-dimensional spaces.
- This method uses Navigable Small Worlds (NSW) for the implementation of hierarchical layers.
- Now we are working on parallelizing the NSW search algorithm.

Navigable Small Worlds

- Navigable Small Worlds (NSW) are networks that have the attributes of small-world structures, which are characterized by short paths among nodes, and navigability.
- Navigable Small Worlds (NSW) networks are formed by sequentially adding elements as vertices, connecting them with their nearest neighbors. This structure, characterized by short paths and efficient navigation, is maintained through bidirectional links.
- NSWs use a variation of the greedy algorithm for efficient searching, where each new element is optimally connected to facilitate quick and efficient traversal.
- While NSWs can face challenges like getting trapped at local minima, their effectiveness measured by search path length and recall rate can be enhanced by adjusting the number of entry points.
- The small world properties are validated using experiments.

Problem Statement

- The primary objective of this project is to implement a parallelized version of the Navigable Small World (NSW) graph building and search algorithms. This task involves redesigning the traditional NSW algorithm to function efficiently in a parallel computing environment.

Parallel Implementation of the NSW Search algorithm

- We have implemented two versions of the parallel algorithm
  - Static allocation method of data to threads (SAM)
  - Dynamic allocation method of data to threads (DAM)
- In these both parallelization algorithms the basic idea is that we give an entry index to the thread for start searching and thread returns its found closest neighbors which is done using the greedy search algorithm.

SAM

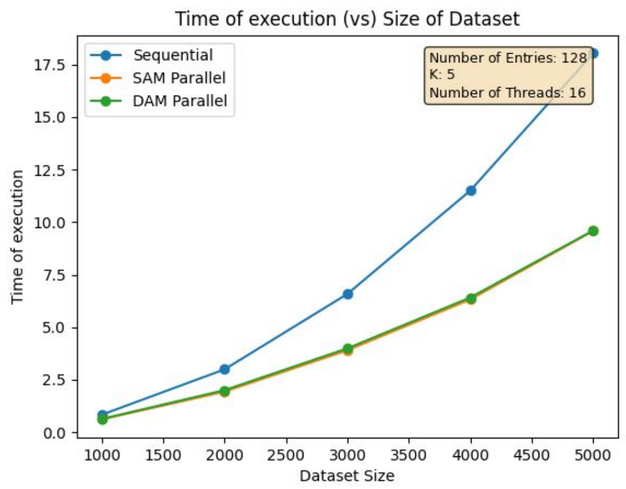
- In static data allocation method, the distribution of data to threads is determined at compile time, before the program is executed. This means that each thread knows in advance which data it will work with throughout the program's execution.
- In this algorithm the threads are assigned entry points by storing the entry points in the array and assigning **(thread_num + i * (total_threads))** entry index to thread number **thread_num**.

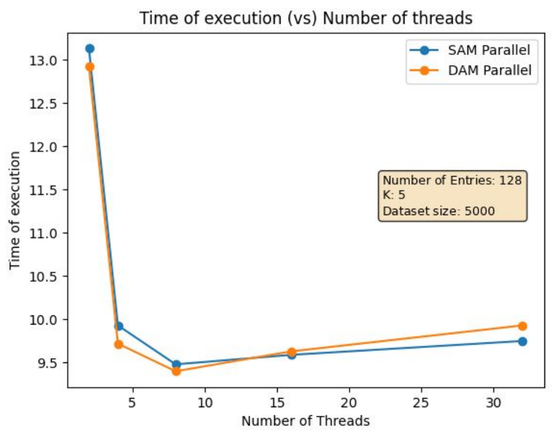
DAM

- Dynamic data allocation involves distributing data to threads at runtime. The allocation can change as the program executes, based on the current workload or other factors.
- Here the assigning is done using a class which assigns entry indexes to thread which first requests the class and then increments a counter.
- Assigning is stopped when the counter reaches the required number of random entries.

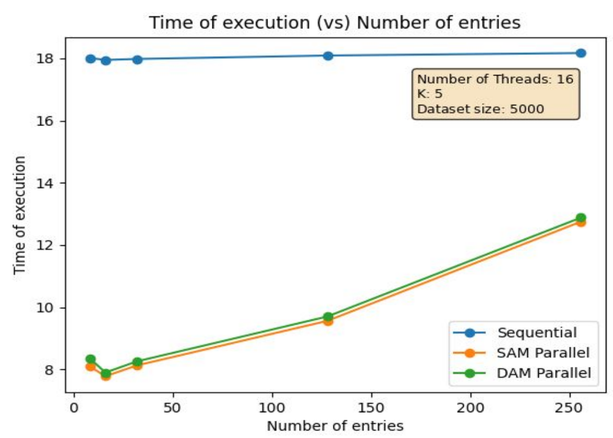
References: Approximate nearest neighbor algorithm based on navigable small world graphs

- The following are the graphs for Time required for building the graph in seconds vs different parameters.





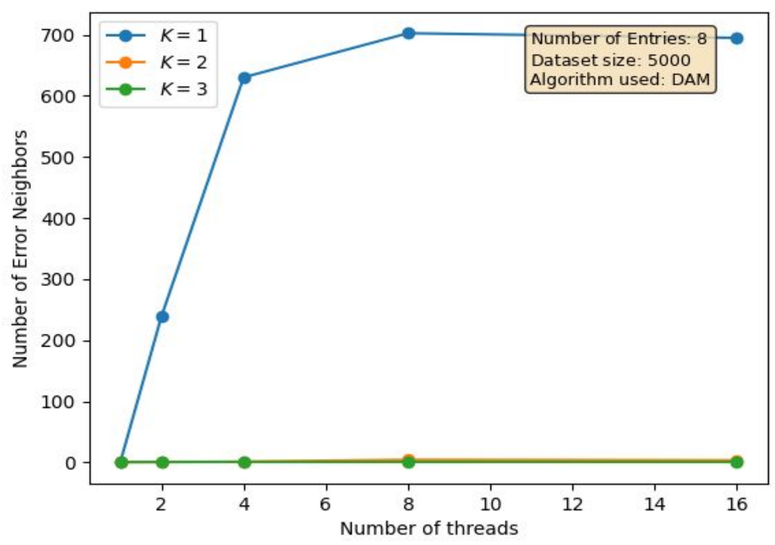
- From the above graphs we can see that SAM and DAM performs nearly with equal performance but as the number of threads increases DAM performance decreases because of race condition between threads to access the shared counter variable.



- From Graph-3 we can observe that as the number of entries increases time of execution is nearly same for sequential algorithm because of the nature of greedy algorithm.

Comparison with Sequential Building

- The following graphs correspond to errors obtained on average in the resultant graph built using DAM NSW compared with the graph built using sequential NSW. The Graph results are same for SAM NSW.



- From here we can see that as K increases the set difference between the neighbours of built graph is different using the parallel algorithm. This is because if the value of K is high then we have less chance that we end up at the local minima.

Future Work

- The efficiency of the parallel algorithm can be significantly enhanced by employing **thread pools**. This approach mitigates the overhead associated with frequent thread creation, management, and synchronization, which occurs each time a search is initiated in the current implementation.
- During the initial iteration of the algorithm, if a local minimum is not encountered, the current design leads to traversing the entire graph from the first entry point of the greedy algorithm. An optimization strategy is under consideration to address this inefficiency, and further research and development are required to finalize this improvement.