CSS 311 – Parallel and Distributed Computing

Title of the work:

Parallel implementation of TF-IDF Index Search Engine

Authors:

Name 1: Anirudh Sandeep Gupta Roll No:2023BCS0209

Name 2: Aditya Dubey Roll No:2023BCD0062

Course Instructor: John Paul Martin

Department of Computer Science, IIIT Kottayam

Date of Submission: 11/10/2025

Abstract

Text parallelization is a key part of natural language processing. It aims to improve the effectiveness of information analysis and retrieval. This project focuses on using the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm for text parallelization. TF-IDF is a common technique for information retrieval and measuring similarity between documents. In this research, we present a new method that uses the TF-IDF algorithm to find and parallelize the most relevant sections of text. This approach speeds up and scales text processing operations. We provide a detailed analysis of our method and evaluate its performance against standard practices. Our results show that TF-IDF-based text parallelization can simplify information extraction tasks. This work adds to ongoing efforts to develop text processing methods, particularly for large-scale document analysis.

1.Introduction

What?

● A system for retrieving texts that turns raw documents into searchable formats using inverted indexing and TF-IDF weighting.

Why?

● Direct search (linear scan) is slow for large text collections.

● Inverted indexes and TF-IDF improve both efficiency (fast lookup) and effectiveness (relevant ranking).

● Parallelization shows how multi-core computing speeds up real-world search engines.

Where?

● Search engines (Google, Bing).

● Digital libraries (Wikipedia, Project Gutenberg).

● Enterprise search systems (research papers, technical notes).

● Any application needing fast and relevant text retrieval.

2.Literature Survey

Existing literature in text summarization mainly focuses on improving the speed and accuracy of summarization models using both serial and parallel implementations. In serial implementations, text data is processed one by one; this makes it easy to design but not suitable for large text collections due to longer processing times. To address this issue, researchers have explored parallel methods that distribute the workload across several processors or threads, reducing execution time while improving scalability. However, parallel implementations also have limitations. The overhead from managing multiple threads can sometimes outweigh the benefits when working with smaller datasets. This indicates that parallelization can enhance performance with larger datasets, but it may not always be the best choice. Therefore, the approach—either serial or parallel—should be selected based on data size and available computing power.

3.Problem Statement and Objectives

3.1 Problem Definition

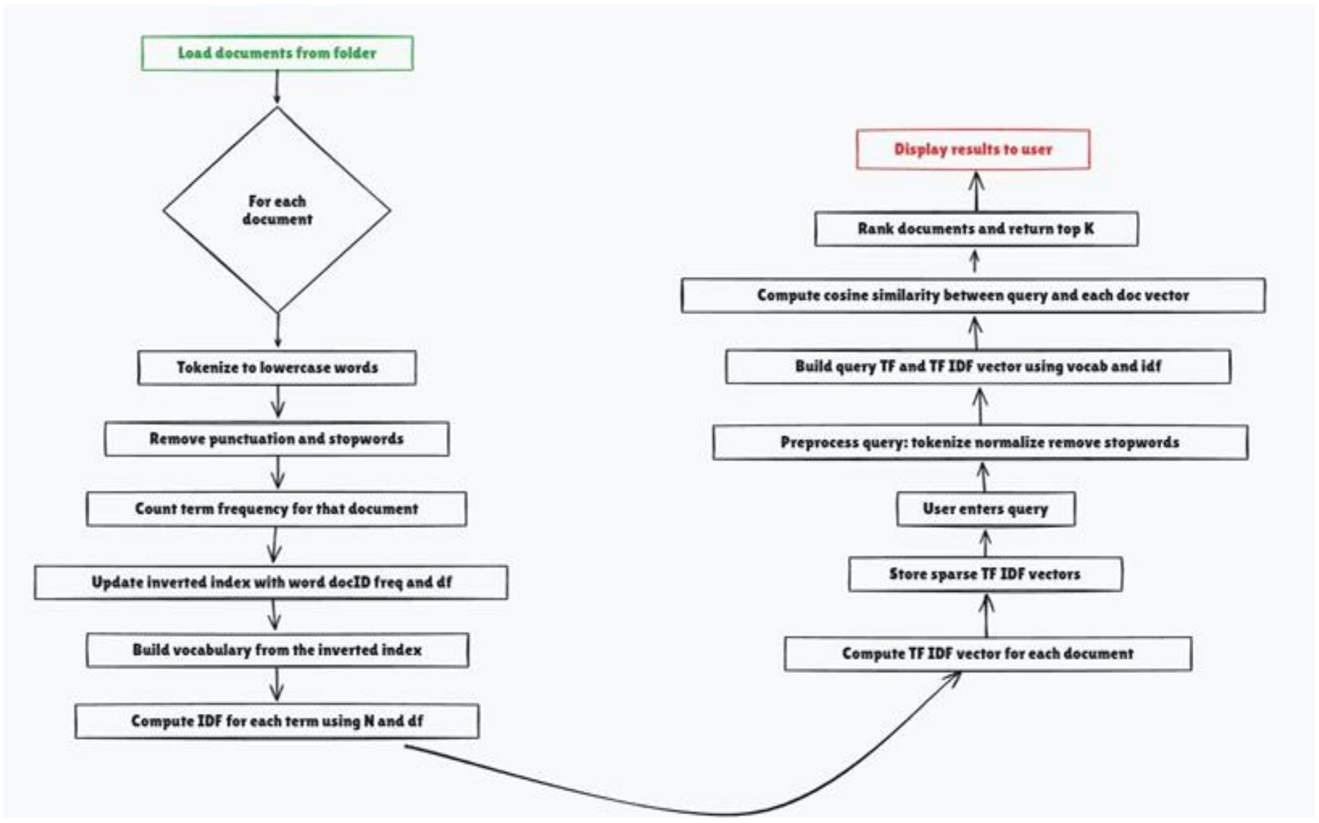
In today's world of overflowing information, it is crucial to find quick and effective ways to summarize texts. By combining the power of parallelization and with the smart use of TF-IDF, we are diving into practical ways to handle the overflow of information in today's fast paced world.

3.2 Problem Objectives

- To implement a small-scale serial TF-IDF search engine
- To identify parallelizable segments of the code.
- To design an OpenMP-based implementation.
- To analyse and compare performance and scalability

4.Methodology and System Architecture

Program flow:



Load documents from folder
For each document
Tokenize to lowercase words
Remove punctuation and stopwords
Count term frequency for that document
Update inverted index with word docID freq and df
Build vocabulary from the inverted index
Compute IDF for each term using N and df
Compute TF IDF vector for each document
Store sparse TF IDF vectors
User enters query
Preprocess query: tokenize normalize remove stopwords
Build query TF and TF IDF vector using vocab and idf
Compute cosine similarity between query and each doc vector
Rank documents and return top K
Display results to user

4.1 Serial Algorithm

```
1. // Step 1: Document Processing
2. Load all documents from the folder
3. Initialize InvertedIndex = {}
4. For each document d in documents:
        a. tokens ← tokenize(d)        // split text, lowercase
        b. tokens ← remove_punctuation(tokens)
        c. tokens ← remove_stopwords(tokens)
        d. term_freq ← count_frequency(tokens)
        e. For each term t in term_freq:
            - Update InvertedIndex[t] with (docID, term_freq[t])
            - Update document frequency (df) for term
5. Vocabulary ← all unique terms in InvertedIndex
6. For each term t in Vocabulary:
        a. idf[t] ← log(N / df[t])        // N = total number of documents
7. For each document d:
        a. Compute tfidf_vector[d] using term_freq[d] * idf
8. // Step 2: Query Processing
9. query ← user input
10.query_tokens ← tokenize(query)
11.query_tokens ← remove_punctuation(query_tokens)
12.query_tokens ← remove_stopwords(query_tokens)
13.query_term_freq ← count_frequency(query_tokens)
14.query_vector ← compute TF-IDF(query_term_freq, idf)
15.// Step 3: Similarity Computation
16.For each document d:
        a. similarity[d] ← cosine_similarity(query_vector, tfidf_vector[d])
17.RankedDocs ← sort documents by similarity in descending order
18.Display top K documents from RankedDocs
End Algorithm
```

4.2 Parallel Algorithm (OpenMP)

```
1.  // Step 1: Collect Files
2.  For each file in folder:
        a. Add file path to list_of_documents
3.  // Step 2: Parallel Per-Document Preprocessing
4.  Parallel for each document d:
        a. Read file[d] → tokenize words
        b. Initialize local_seen_set
        c. For each token in tokens:
            - If token is not a stopword:
                • Increment TF count in doc_term_counts[d]
                • Increment total term count in doc_total_terms[d]
                • Insert token into local_seen_set
        d. For each token in local_seen_set:
            - Increment thread-local DF map
5.  Merge all thread-local DF maps → global_doc_freq
6.  // Step 3: Vocabulary Indexing
7.  For each term in global_doc_freq:
        a. Assign unique index to term
8.  // Step 4: Parallel TF-IDF Vector Build
9.  Parallel for each document d:
        a. For each term in doc_term_counts[d]:
            - tf = count / total_terms[d]
            - idf = log(N / (doc_freq[term] + 1))
            - Store (tf * idf) in doc_vectors[d]
        b. Normalize doc_vectors[d]
10. // Step 5: Interactive Query Loop
11. While user wants to search:
        a. Read query_string → tokenize
        b. Filter stopwords → build query_vector (TF-IDF, normalized)
        c. Parallel similarity computation:
            Parallel for each document d:
                - Compute cosine similarity between query_vector and
    doc_vectors[d]
                - Store score[d]
        d. Collect all scores → sort top-k results
        e. Display top-k documents
End Algorithm
```

4.3 Implementation Details

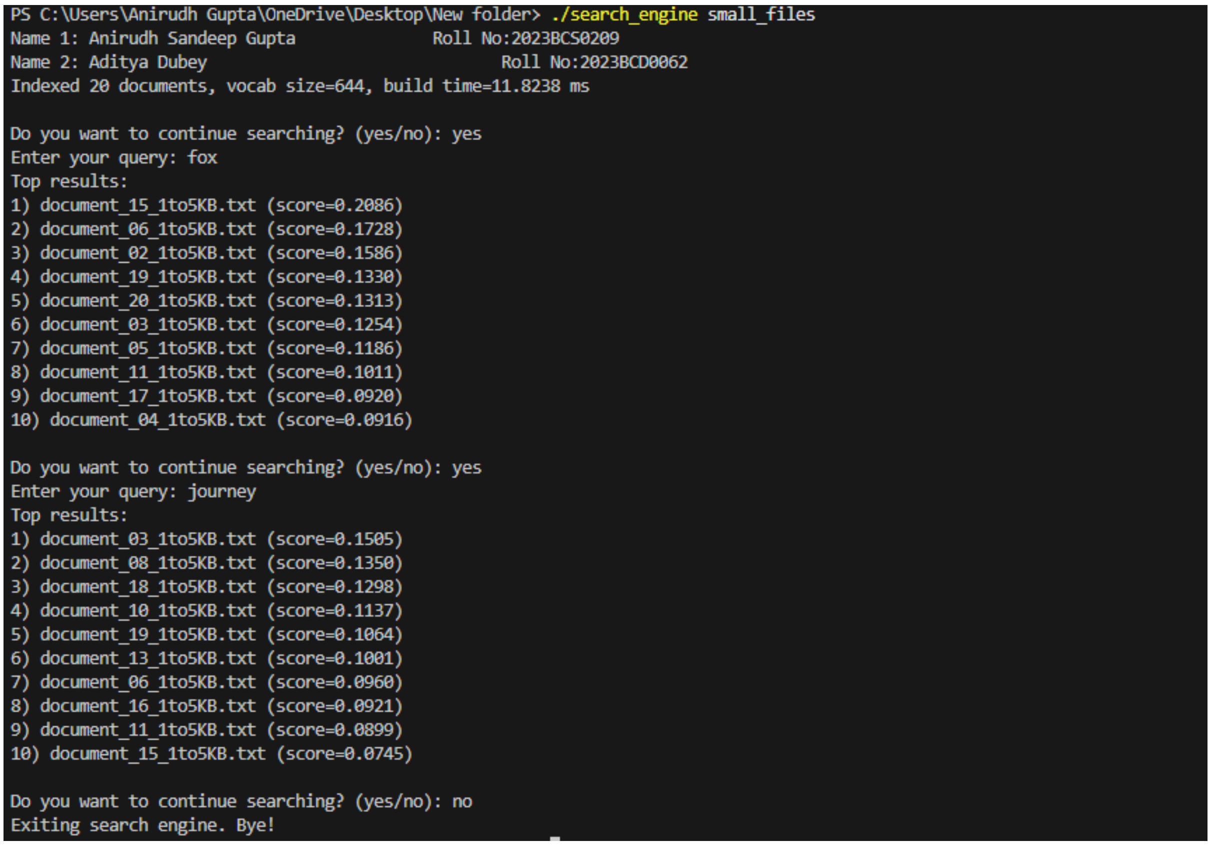
Programming Language: C++
Platform: Windows 11
Compiler: G++ (-fopenmp)
Hardware Configuration:

- Processor: Intel Core i7 11th Generation (8 cores, 16 threads, up to 4.6 GHz)
- RAM: 16 GB DDR4
- Storage: 512 GB SSD
- GPU: NVIDIA GeForce RTX 3050 Ti (4 GB GDDR6)

5. Result and Analysis

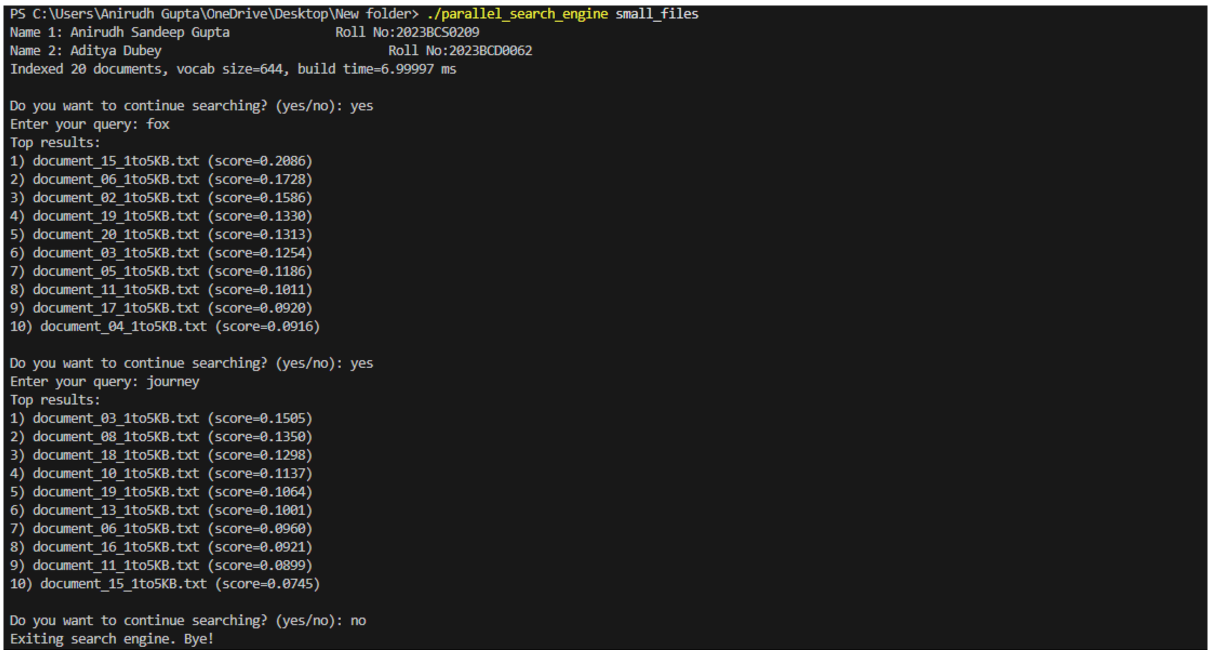
For small-sized files:

Serial-



```
PS C:\Users\Anirudh Gupta\OneDrive\Desktop\New folder> ./search_engine small_files
Name 1: Anirudh Sandeep Gupta              Roll No:2023BCS0209
Name 2: Aditya Dubey                                 Roll No:2023BCD0062
Indexed 20 documents, vocab size=644, build time=11.8238 ms

Do you want to continue searching? (yes/no): yes
Enter your query: fox
Top results:
1) document_15_1to5KB.txt (score=0.2086)
2) document_06_1to5KB.txt (score=0.1728)
3) document_02_1to5KB.txt (score=0.1586)
4) document_19_1to5KB.txt (score=0.1330)
5) document_20_1to5KB.txt (score=0.1313)
6) document_03_1to5KB.txt (score=0.1254)
7) document_05_1to5KB.txt (score=0.1186)
8) document_11_1to5KB.txt (score=0.1011)
9) document_17_1to5KB.txt (score=0.0920)
10) document_04_1to5KB.txt (score=0.0916)

Do you want to continue searching? (yes/no): yes
Enter your query: journey
Top results:
1) document_03_1to5KB.txt (score=0.1505)
2) document_08_1to5KB.txt (score=0.1350)
3) document_18_1to5KB.txt (score=0.1298)
4) document_10_1to5KB.txt (score=0.1137)
5) document_19_1to5KB.txt (score=0.1064)
6) document_13_1to5KB.txt (score=0.1001)
7) document_06_1to5KB.txt (score=0.0960)
8) document_16_1to5KB.txt (score=0.0921)
9) document_11_1to5KB.txt (score=0.0899)
10) document_15_1to5KB.txt (score=0.0745)

Do you want to continue searching? (yes/no): no
Exiting search engine. Bye!
```

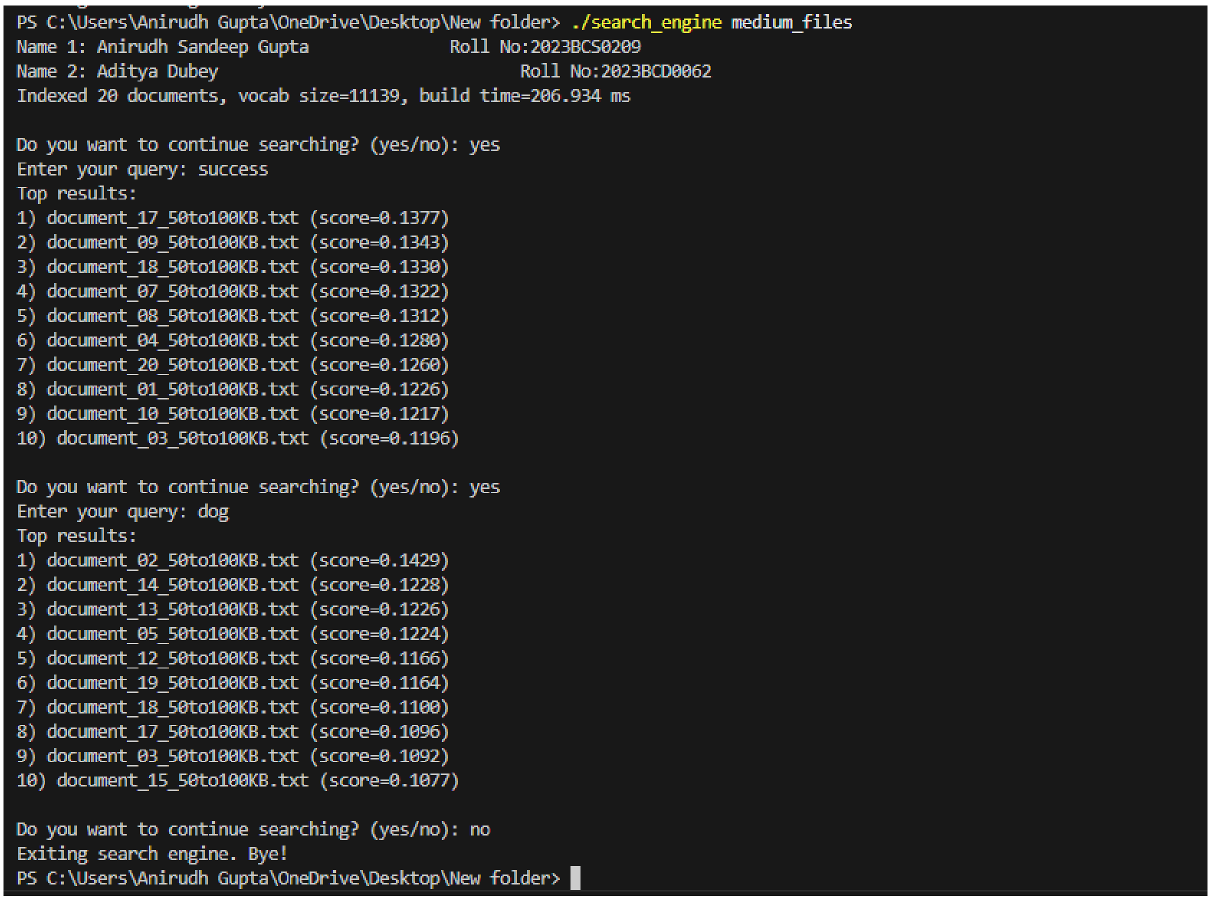
Parallel –



```
PS C:\Users\Anirudh Gupta\OneDrive\Desktop\New folder> ./parallel_search_engine small_files
Name 1: Anirudh Sandeep Gupta              Roll No:2023BCS0209
Name 2: Aditya Dubey                                 Roll No:2023BCD0062
Indexed 20 documents, vocab size=644, build time=6.99997 ms

Do you want to continue searching? (yes/no): yes
Enter your query: fox
Top results:
1) document_15_1to5KB.txt (score=0.2086)
2) document_06_1to5KB.txt (score=0.1728)
3) document_02_1to5KB.txt (score=0.1586)
4) document_19_1to5KB.txt (score=0.1330)
5) document_20_1to5KB.txt (score=0.1313)
6) document_03_1to5KB.txt (score=0.1254)
7) document_05_1to5KB.txt (score=0.1186)
8) document_11_1to5KB.txt (score=0.1011)
9) document_17_1to5KB.txt (score=0.0920)
10) document_04_1to5KB.txt (score=0.0916)

Do you want to continue searching? (yes/no): yes
Enter your query: journey
Top results:
1) document_03_1to5KB.txt (score=0.1505)
2) document_08_1to5KB.txt (score=0.1350)
3) document_18_1to5KB.txt (score=0.1298)
4) document_10_1to5KB.txt (score=0.1137)
5) document_19_1to5KB.txt (score=0.1064)
6) document_13_1to5KB.txt (score=0.1001)
7) document_06_1to5KB.txt (score=0.0960)
8) document_16_1to5KB.txt (score=0.0921)
9) document_11_1to5KB.txt (score=0.0899)
10) document_15_1to5KB.txt (score=0.0745)

Do you want to continue searching? (yes/no): no
Exiting search engine. Bye!
```

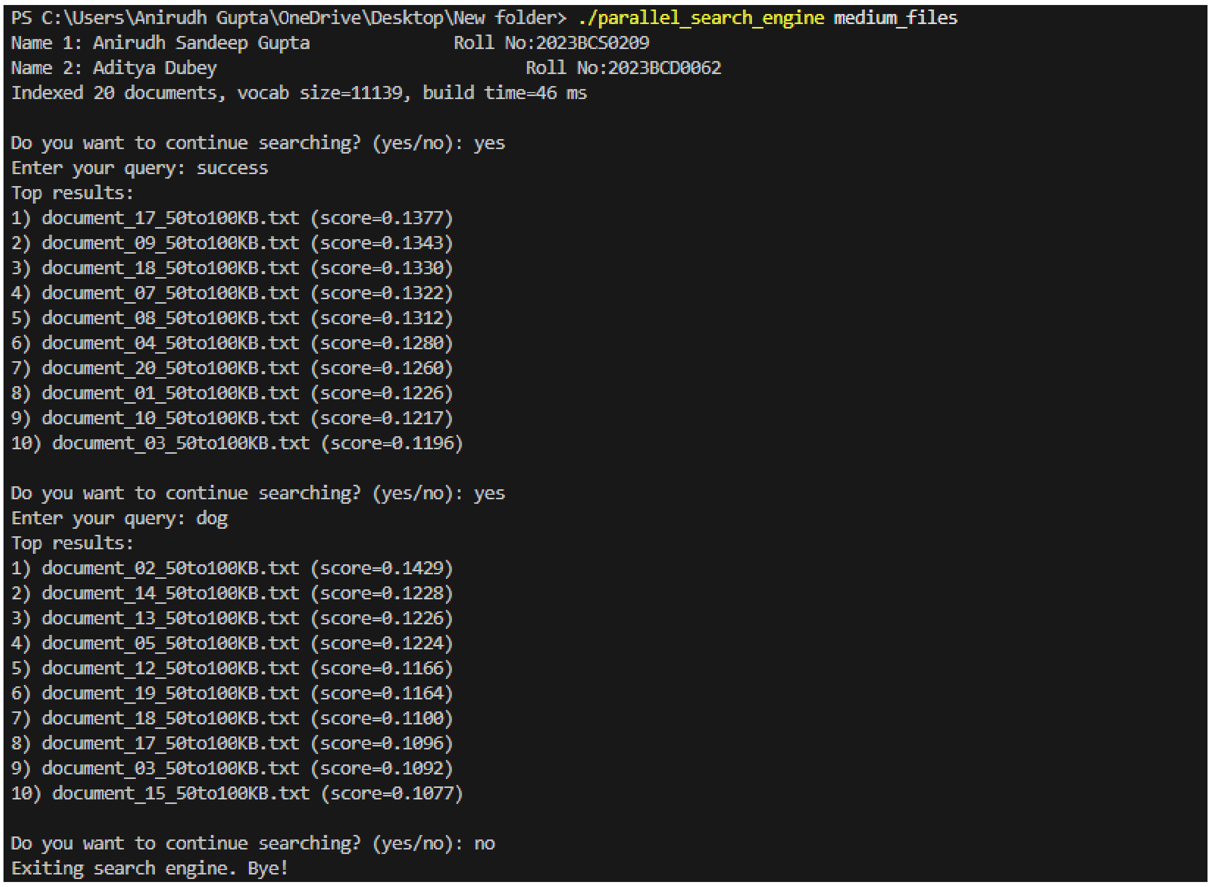
For medium-sized files:

Serial –



```
PS C:\Users\Anirudh Gupta\OneDrive\Desktop\New folder> ./search_engine medium_files
Name 1: Anirudh Sandeep Gupta              Roll No:2023BCS0209
Name 2: Aditya Dubey                               Roll No:2023BCD0062
Indexed 20 documents, vocab size=11139, build time=206.934 ms

Do you want to continue searching? (yes/no): yes
Enter your query: success
Top results:
1) document_17_50to100KB.txt (score=0.1377)
2) document_09_50to100KB.txt (score=0.1343)
3) document_18_50to100KB.txt (score=0.1330)
4) document_07_50to100KB.txt (score=0.1322)
5) document_08_50to100KB.txt (score=0.1312)
6) document_04_50to100KB.txt (score=0.1280)
7) document_20_50to100KB.txt (score=0.1260)
8) document_01_50to100KB.txt (score=0.1226)
9) document_10_50to100KB.txt (score=0.1217)
10) document_03_50to100KB.txt (score=0.1196)

Do you want to continue searching? (yes/no): yes
Enter your query: dog
Top results:
1) document_02_50to100KB.txt (score=0.1429)
2) document_14_50to100KB.txt (score=0.1228)
3) document_13_50to100KB.txt (score=0.1226)
4) document_05_50to100KB.txt (score=0.1224)
5) document_12_50to100KB.txt (score=0.1166)
6) document_19_50to100KB.txt (score=0.1164)
7) document_18_50to100KB.txt (score=0.1100)
8) document_17_50to100KB.txt (score=0.1096)
9) document_03_50to100KB.txt (score=0.1092)
10) document_15_50to100KB.txt (score=0.1077)

Do you want to continue searching? (yes/no): no
Exiting search engine. Bye!
PS C:\Users\Anirudh Gupta\OneDrive\Desktop\New folder>
```

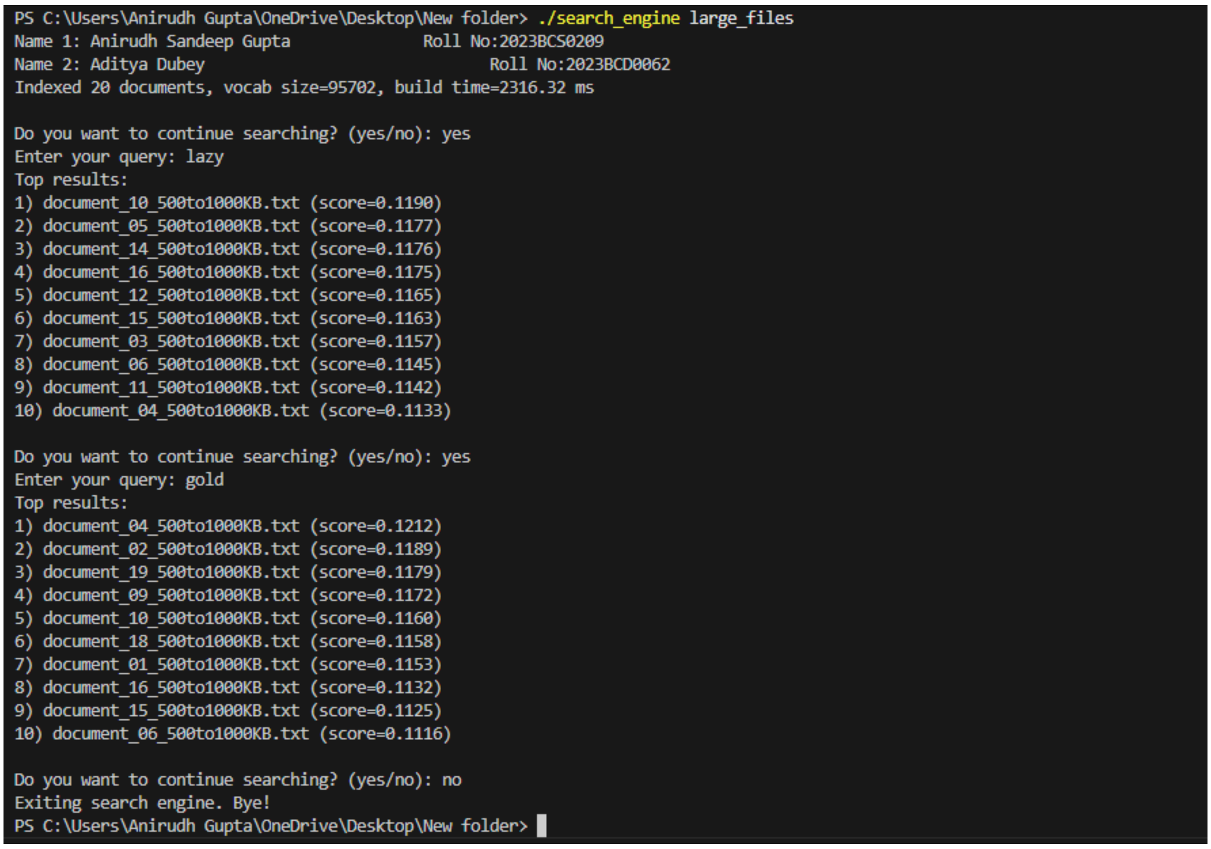
Parallel –



```
PS C:\Users\Anirudh Gupta\OneDrive\Desktop\New folder> ./parallel_search_engine medium_files
Name 1: Anirudh Sandeep Gupta              Roll No:2023BCS0209
Name 2: Aditya Dubey                              Roll No:2023BCD0062
Indexed 20 documents, vocab size=11139, build time=46 ms

Do you want to continue searching? (yes/no): yes
Enter your query: success
Top results:
1) document_17_50to100KB.txt (score=0.1377)
2) document_09_50to100KB.txt (score=0.1343)
3) document_18_50to100KB.txt (score=0.1330)
4) document_07_50to100KB.txt (score=0.1322)
5) document_08_50to100KB.txt (score=0.1312)
6) document_04_50to100KB.txt (score=0.1280)
7) document_20_50to100KB.txt (score=0.1260)
8) document_01_50to100KB.txt (score=0.1226)
9) document_10_50to100KB.txt (score=0.1217)
10) document_03_50to100KB.txt (score=0.1196)

Do you want to continue searching? (yes/no): yes
Enter your query: dog
Top results:
1) document_02_50to100KB.txt (score=0.1429)
2) document_14_50to100KB.txt (score=0.1228)
3) document_13_50to100KB.txt (score=0.1226)
4) document_05_50to100KB.txt (score=0.1224)
5) document_12_50to100KB.txt (score=0.1166)
6) document_19_50to100KB.txt (score=0.1164)
7) document_18_50to100KB.txt (score=0.1100)
8) document_17_50to100KB.txt (score=0.1096)
9) document_03_50to100KB.txt (score=0.1092)
10) document_15_50to100KB.txt (score=0.1077)

Do you want to continue searching? (yes/no): no
Exiting search engine. Bye!
```

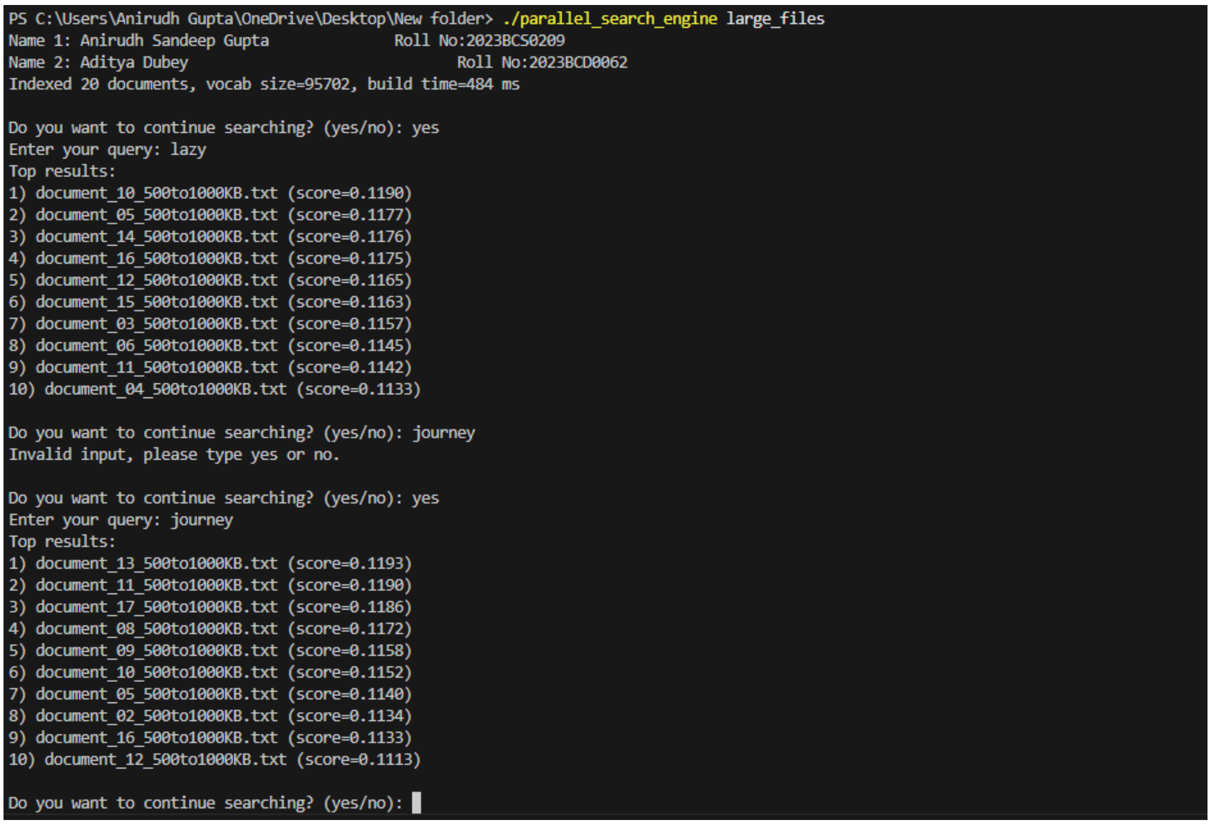
For big-sized files:

Serial –



```
PS C:\Users\Anirudh Gupta\OneDrive\Desktop\New folder> ./search_engine large_files
Name 1: Anirudh Sandeep Gupta             Roll No:2023BCS0209
Name 2: Aditya Dubey                             Roll No:2023BCD0062
Indexed 20 documents, vocab size=95702, build time=2316.32 ms

Do you want to continue searching? (yes/no): yes
Enter your query: lazy
Top results:
1) document_10_500to1000KB.txt (score=0.1190)
2) document_05_500to1000KB.txt (score=0.1177)
3) document_14_500to1000KB.txt (score=0.1176)
4) document_16_500to1000KB.txt (score=0.1175)
5) document_12_500to1000KB.txt (score=0.1165)
6) document_15_500to1000KB.txt (score=0.1163)
7) document_03_500to1000KB.txt (score=0.1157)
8) document_06_500to1000KB.txt (score=0.1145)
9) document_11_500to1000KB.txt (score=0.1142)
10) document_04_500to1000KB.txt (score=0.1133)

Do you want to continue searching? (yes/no): yes
Enter your query: gold
Top results:
1) document_04_500to1000KB.txt (score=0.1212)
2) document_02_500to1000KB.txt (score=0.1189)
3) document_19_500to1000KB.txt (score=0.1179)
4) document_09_500to1000KB.txt (score=0.1172)
5) document_10_500to1000KB.txt (score=0.1160)
6) document_18_500to1000KB.txt (score=0.1158)
7) document_01_500to1000KB.txt (score=0.1153)
8) document_16_500to1000KB.txt (score=0.1132)
9) document_15_500to1000KB.txt (score=0.1125)
10) document_06_500to1000KB.txt (score=0.1116)

Do you want to continue searching? (yes/no): no
Exiting search engine. Bye!
PS C:\Users\Anirudh Gupta\OneDrive\Desktop\New folder>
```

Parallel –



```
PS C:\Users\Anirudh Gupta\OneDrive\Desktop\New folder> ./parallel_search_engine large_files
Name 1: Anirudh Sandeep Gupta             Roll No:2023BCS0209
Name 2: Aditya Dubey                             Roll No:2023BCD0062
Indexed 20 documents, vocab size=95702, build time=484 ms

Do you want to continue searching? (yes/no): yes
Enter your query: lazy
Top results:
1) document_10_500to1000KB.txt (score=0.1190)
2) document_05_500to1000KB.txt (score=0.1177)
3) document_14_500to1000KB.txt (score=0.1176)
4) document_16_500to1000KB.txt (score=0.1175)
5) document_12_500to1000KB.txt (score=0.1165)
6) document_15_500to1000KB.txt (score=0.1163)
7) document_03_500to1000KB.txt (score=0.1157)
8) document_06_500to1000KB.txt (score=0.1145)
9) document_11_500to1000KB.txt (score=0.1142)
10) document_04_500to1000KB.txt (score=0.1133)

Do you want to continue searching? (yes/no): journey
Invalid input, please type yes or no.

Do you want to continue searching? (yes/no): yes
Enter your query: journey
Top results:
1) document_13_500to1000KB.txt (score=0.1193)
2) document_11_500to1000KB.txt (score=0.1190)
3) document_17_500to1000KB.txt (score=0.1186)
4) document_08_500to1000KB.txt (score=0.1172)
5) document_09_500to1000KB.txt (score=0.1158)
6) document_10_500to1000KB.txt (score=0.1152)
7) document_05_500to1000KB.txt (score=0.1140)
8) document_02_500to1000KB.txt (score=0.1134)
9) document_16_500to1000KB.txt (score=0.1133)
10) document_12_500to1000KB.txt (score=0.1113)

Do you want to continue searching? (yes/no):
```

File size (KB) Total files = 20	Time taken for parallel execution (ms)	Time taken for serial execution (ms)
1-5	6.99997	11.8238
50-100	46	206.934
500-1000	484	2316.32

Hence, we can see that the time is reduced significantly when we parallelise our search engine.

Time complexity for serial search engine:

1. Build (indexing)

- Tokenize & count TF/DF: $O(T)$ (I/O + per-token work, amortized).
- Vocabulary indexing & TF–IDF vector creation:

$O(\sum_i U_i) \approx O(V + T)$.

- Practical build cost: $O(I/O + T + V)$ (I/O often dominates).

2. Query (current full-scan implementation)

- Tokenize + build query vector: $O(q)$.
- Cosine similarity via full scan: lookup q_u terms per document $\rightarrow O(N \cdot q_u)$.
- Sorting results (top-K via full sort): $O(N \log N)$.
- Practical query cost: $O(q + N \cdot q_u + N \log N)$ (dominant: $N \cdot q_u$)

Where-

N = number of documents, V = vocabulary size, T = total tokens in corpus, L = avg tokens/doc ($\approx T/N$), q = query tokens, q_u = unique query tokens.

Time Complexity for a parallel search engine:

If we parallelise per-document TF-IDF computation and query scoring:

$$\mathbf{O((N \times V)/p)}$$

Where- N = no. of documents, V = vocabulary size- p = no. of threads

6. Discussions and Observations

It can be observed from the experimental results that the parallel OpenMP-based TF-IDF search engine really decreases execution time significantly compared to the serial version. With the growth of dataset size, the acceleration increases. With small files, it does not help much as there are too many thread overheads and not much workload in each thread. With medium and large datasets, parallelization shows as much as a 4–5× increase in speeds, which shows good scalability as the number and size of documents increase.

Scalability relies mainly on the memory bandwidth and the number of threads of a system. As text preprocessing and computing of TF-IDF vector are embarrassingly parallel computations (independent document processing), scaling goes well until the number of core(s) that are physical. Hotspot just provides a diminishing return after that.

A few limitations were observed:

-Synchronization overhead of threads while performing global frequency merging (DF map) marginally affected the performance for smaller corpora.

-Memory consumption increases with dataset size since TF-IDF vectors for all documents are stored simultaneously.

-Sorting step of query results remains serial and can become a bottleneck for very large datasets.

-I/O time remains paramount, particularly when opening a large number of small text files from disk.

In general, the OpenMP parallelization enhances efficiency and retains correctness and ranking quality.

7.Conclusion and Future Scope

This project (a parallel TF-IDF–based search engine) is successfully implemented and analysed using OpenMP. The system demonstrated clear performance improvements over the serial version, especially for medium to large document collections. The results confirm that search performance is greatly improved by parallelizing document-level processes (tokenization, term-frequency computation, and TF-IDF vector generation).

In the future, there will many possible ways this project can grow:

- A distributed MPI (message passing interface) algorithm will allow for massive amounts of data to be managed on several computers or nodes with computing power that can scale upwards.

-CUDA or GPU acceleration for massive amounts of data will be employed to take advantage of the enormous parallelism of GPUs to perform the vector operations, such as computations of cosine similarity.

-By using hybrid models (MPI + OpenMP or CUDA) which utilize both ideas of distributed and shared memory, it is hoped that resource utilization can be maximized.

-Lastly, there will be other methods of optimization to increase memory efficiency, such as compressed data structures, asynchronous I/O, and efficient parallel sorting.

In conclusion, this project has shown that parallel computing can greatly improve information retrieval tasks, this provides a solid and robust platform for developing a future large-scale or GPU-accelerated search engine systems.

8.References

[1] Quinn, M. J. Parallel Programming in C with MPI and OpenMP, McGraw-Hill, 2004.

[2] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 5.0, 2023.

[3] Tanenbaum, A. S. Distributed Systems: Principles and Paradigms, Pearson, 2019.

[4] D. Chaurasia, P. V. Devi K and M. Bhatta, "Enhancing Text Summarization through Parallelization: A TF-IDF Algorithm Approach," *2024 Second International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI)*, Coimbatore, India, 2024, pp. 1503-1508, doi: 10.1109/ICoICI62503.2024.10696641.