Distributed Algorithms Lab

REPORT

ON

"Parallel Text Processing using OpenMP"

In partial fulfilment of

***B. Tech III year* (Computer Science & Engineering)**

Submitted To:

Mr. Naresh Purohit
(Associate Professor)

Submitted by:

Bhanu Bohra (6A-2)
(22EJICS028)

Acknowledgment

We would like to acknowledge the contributions of the following people without whose help and guidance this project would not have been completed.

We are thankful to Ms. Mamta Garg, HOD(Mentor) and Anju Jangid, HOD(Academic) of Computer Science and Engineering Department, Jodhpur Institute of Engineering and Technology, for her constant encouragement, valuable suggestions and moral support and blessings.

We respectfully thank Mr. Naresh Purohit (Associate Professor), of Computer Science and Engineering Department, Jodhpur Institute of Engineering and Technology, for providing me an opportunity to do this project work and giving me all support and guidance, which made me complete the project up to very extent.

Although it is not possible to name individually, we shall ever remain indebted to the faculty members of Jodhpur Institute of Engineering and Technology, for their persistent support and cooperation extended during his work.

This acknowledgement will remain incomplete if we fail to express our deep sense of obligation to our parents and God for their consistent blessings and Encouragement.

Table of Contents

Abstract

Text processing is a fundamental operation in computing, widely used in applications such as data preprocessing, text formatting, and content analysis. Traditional sequential processing methods can become inefficient when handling large text files, leading to increased execution time. This project, "Parallel File Text Processing Using OpenMP in C," aims to optimize text manipulation operations by leveraging parallel computing. The program reads an input text file, applies user-specified transformations—including case conversion, text reversal, and line order manipulation—and writes the processed output to a new file. By integrating OpenMP, a multi-threading API, the project significantly enhances processing speed, demonstrating the benefits of parallel execution in text-based applications.

The core objective of this project is to improve text processing efficiency by utilizing multi-threading to execute operations concurrently. Functions such as converting text to uppercase or lowercase, reversing characters within lines, and reordering lines are executed in parallel to distribute the workload across multiple CPU cores. OpenMP provides a structured approach to parallelization, using directives like #pragma omp parallel for and #pragma omp sections to manage thread execution. This ensures that operations are executed simultaneously rather than sequentially, leading to a substantial reduction in execution time, particularly for large text files. The project illustrates how parallelism can enhance performance without significantly increasing code complexity.

The results of this project confirm that OpenMP-based parallel execution can drastically reduce processing time, especially when dealing with large datasets. Compared to sequential processing, the multi-threaded approach achieves better resource utilization and scalability, making it a practical solution for high-speed text manipulation tasks. However, the efficiency gain depends on the system's hardware capabilities, such as the number of available CPU cores. The project highlights the importance of optimizing thread management to avoid synchronization issues and performance overheads associated with parallel execution. Through careful implementation, the program achieves balanced workload distribution and efficient execution of text transformations.

This project serves as a foundational step in exploring the application of parallel computing in text processing. While it successfully demonstrates the advantages of OpenMP in optimizing file operations, there is potential for further enhancements. Future improvements could include dynamic thread allocation, support for additional text processing functionalities, and integration with GPU acceleration for even greater performance gains. This project not only provides insights into multi-threaded programming using OpenMP but also paves the way for more advanced implementations in high-performance text and data processing systems.

Introduction

Text processing is an essential component of modern computing, playing a crucial role in various domains, including:

- Data analysis – Cleaning and transforming textual data for structured processing.
- Document formatting – Converting case, restructuring text, and formatting documents.

Traditional text processing methods involve sequential execution, where each operation is performed one after another. The growing need for real-time and high-speed text processing demands the use of parallel computing techniques to optimize performance.

The primary objective of this project is to optimize text manipulation operations by implementing parallel execution for functions such as:

- Case conversion – Converting text to uppercase or lowercase.
- Text reversal – Reversing the order of characters within each line.
- Order reversal – Reversing the order of lines in a file.

OpenMP is a widely used API for shared-memory parallel programming, allows computational tasks to be distributed across multiple CPU cores, ensuring that these operations execute simultaneously instead of sequentially.

One of the key advantages of OpenMP in this project is its ability to:

- Automatically manage threads – Simplifying multi-threaded execution.
- Use parallel loops (#pragma omp parallel for) – Enabling concurrent processing of text.
- Execute different operations in parallel (#pragma omp sections) – Enhancing efficiency in multi-step processing.
- Prevent data corruption (#pragma omp critical) – Ensuring safe access to shared resources.

By implementing these parallelization techniques, the project enhances text processing performance while maintaining data integrity.

Tools and Technologies used

To successfully implement "Parallel File Text Processing Using OpenMP in C," several tools and technologies were utilized. Each component plays a crucial role in ensuring efficient execution, parallelization, and file handling. The following sections provide a detailed breakdown of the key technologies used in this project.

1. Programming Language: C

The project is implemented using the C programming language, which is known for its:

- High performance and efficiency – C provides direct memory access and low-level control, making it suitable for computationally intensive tasks.
- Extensive support for file handling – Built-in functions like fopen(), fgets(), and fprintf() allow seamless reading and writing of text files.
- Compatibility with OpenMP – C natively supports OpenMP, making it an excellent choice for parallel computing.

C's simplicity and execution speed make it ideal for large-scale text processing tasks, where performance optimization is a priority.

2. OpenMP (Open Multi-Processing)

OpenMP is an API (Application Programming Interface) used to implement multi-threading in C, C++, and Fortran. It enables parallel execution of tasks, significantly improving the speed of text processing operations. The project utilizes OpenMP for:

- Parallelizing loops (#pragma omp parallel for) – Distributing text processing tasks across multiple CPU cores.
- Concurrent execution of sections (#pragma omp sections) – Allowing different text operations (e.g., case conversion and line reversal) to run simultaneously.
- Thread synchronization (#pragma omp critical) – Preventing race conditions when accessing shared resources.

OpenMP makes it easier to introduce parallelism without significantly modifying the program structure, ensuring better CPU utilization.

3. Text Files for Input and Output Processing

The project processes text files (.txt) as input and output formats. Key file handling operations include:

- Reading text files (fopen() with "r" mode) – The program reads input text line by line, storing it in a character array.
- Processing text in memory (char fileData[MAX_LINES][MAX_LENGTH]) – This ensures efficient manipulation of text data before writing.
- Writing to a new text file (fopen() with "w" mode) – The processed text is saved into an output file, ensuring that the original data remains unchanged.

Using standard text files ensures that the program remains versatile and applicable to various real-world text processing scenarios.

4. Parallel Computing Concepts

To enhance the performance of text processing operations, the project employs parallel computing principles, including:

- Multi-threading – Utilizing multiple threads to process text concurrently.
- Workload distribution – Dividing large text files into smaller chunks for faster execution.
- Minimizing synchronization overhead – Using #pragma omp critical selectively to maintain balance between speed and data integrity.

By leveraging these concepts, the project achieves higher efficiency and faster execution times, making it suitable for handling large-scale text processing tasks.

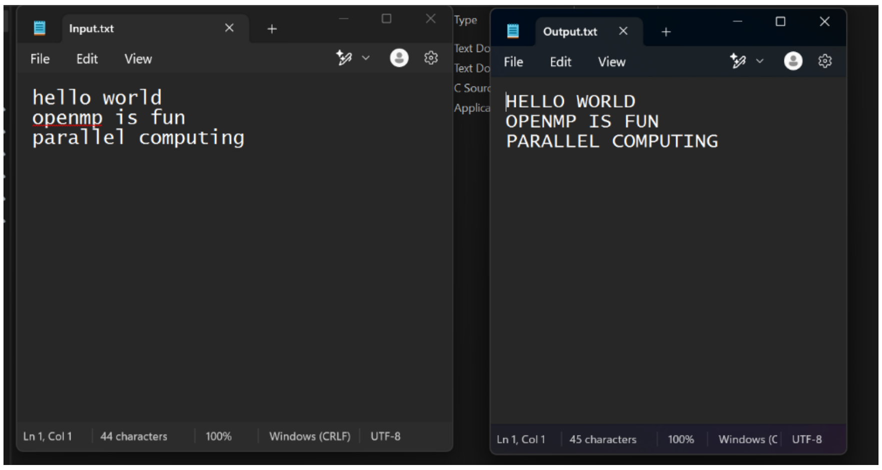
Code

```
#include<stdio.h>
#include<string.h>
#include<omp.h>
#define MAX_LINES 1000
#define MAX_LENGTH 256
char fileData[MAX_LINES][MAX_LENGTH]; int lineCount = 0;
char toUpperCase(char c) { return (c>=97 && c<=122) ? c-32 : c; }
char toLowerCase(char c) { return (c>=65 && c<=90) ? c+32 : c; }
void readFile(char *filename) {
   FILE *fp = fopen(filename, "r");
   if(!fp)
   {
     printf("Error opening File: %s\n", filename);
     return;
   }
  while(lineCount < MAX_LINES && fgets(fileData[lineCount], MAX_LENGTH, fp)){
    #pragma omp critical{
       fileData[lineCount][strcspn(fileData[lineCount], "\n")] = 0;
       lineCount++;
    }
  }
 fclose(fp);
}
void toUpper() {
  int i; #pragma omp parallel for for (i = 0; i < lineCount; i++) {
    int j; for (j = 0; fileData[i][j] != '\0'; j++) fileData[i][j] = toUpperCase(fileData[i][j]);
}
}
void toLower() {
   int i; #pragma omp parallel for for (i = 0; i < lineCount; i++) {
    int j; for (j = 0; fileData[i][j] != '\0'; j++) fileData[i][j] = toLowerCase(fileData[i][j]); } }
   void reverse() { int i; #pragma omp parallel for for (i = 0; i < lineCount; i++)
{
    int j = strlen(fileData[i]) - 1, k = 0; while (k < j) { char tmp = fileData[i][k]; fileData[i][k]
= fileData[i][j]; fileData[i][j] = tmp; k++; j--;
 }
 }
 }
void orderReverse() { int i;
#pragma omp parallel for
for (i = 0; i < lineCount / 2; i++)
{
   char tmp[MAX_LENGTH];
   strcpy(tmp, fileData[i]);
   #pragma omp critical
    {
        strcpy(fileData[i], fileData[lineCount - i - 1]);
        strcpy(fileData[lineCount - i - 1], tmp);
    }
}
}
void writeFile(char *filename) { FILE *fp = fopen(filename, "w"); int i;
if(!fp)
{
    printf("Error: Cannot open file %s\n", filename);
    return;
}
for(i = 0; i < lineCount; i++){
    #pragma omp critical
    fprintf(fp, "%s\n", fileData[i]);
}
fclose(fp);
}
void main(int argc, char **argv) { if (argc < 4 || argc > 6) { printf("Usage: %s <-u/-l> [-rl] [-
rc]\n", argv[0]); return;
 }
char *inputFile = argv[1];
char *outputFile = argv[2];
readFile(inputFile);
#pragma omp parallel sections
{
    #pragma omp section
    {
        if(argv[3][1]=='u' || argv[3][1]=='u')
            toUpper();
        else
            toLower();
    }
    #pragma omp section
    {
        if(argc == 6) {
            reverse();
            orderReverse();
        }
        else if(argc == 5) {
            if(argv[4][2]=='c' || argv[4][2]=='c')
                reverse();
            else
                orderReverse();
        }
    }
}
writeFile(outputFile);
printf("File processing complete.\n");
}
```

Output

The output of this project demonstrates the effectiveness of parallel text processing using OpenMP by applying various transformations to a given text file. The processed results are stored in a new output file, ensuring that the original data remains unchanged. The following examples illustrate how different transformations are applied and their corresponding outputs.

1. Uppercase Conversion (-u option)

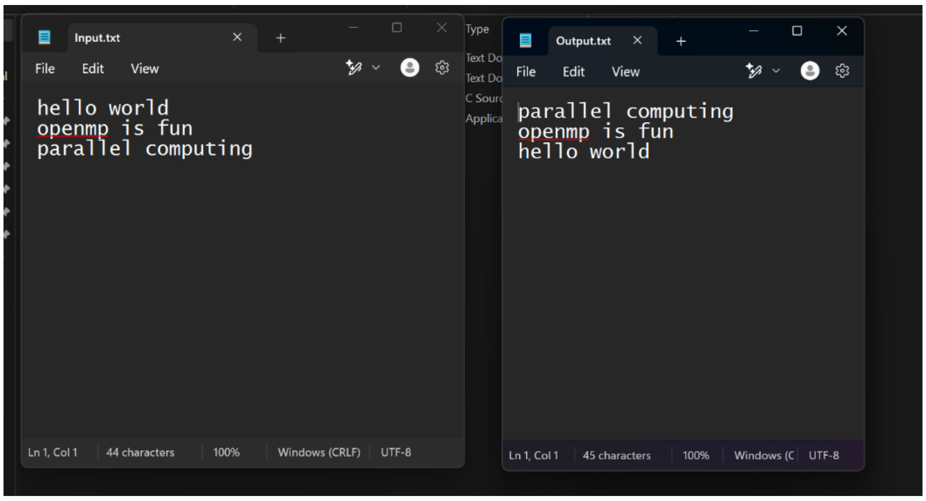


```
hello world
openmp is fun
parallel computing
```

```
HELLO WORLD
OPENMP IS FUN
PARALLEL COMPUTING
```

Fig..1: Conversion to uppercase

2. Lowercase Conversion Reverse Line Order (-l and -rl option)

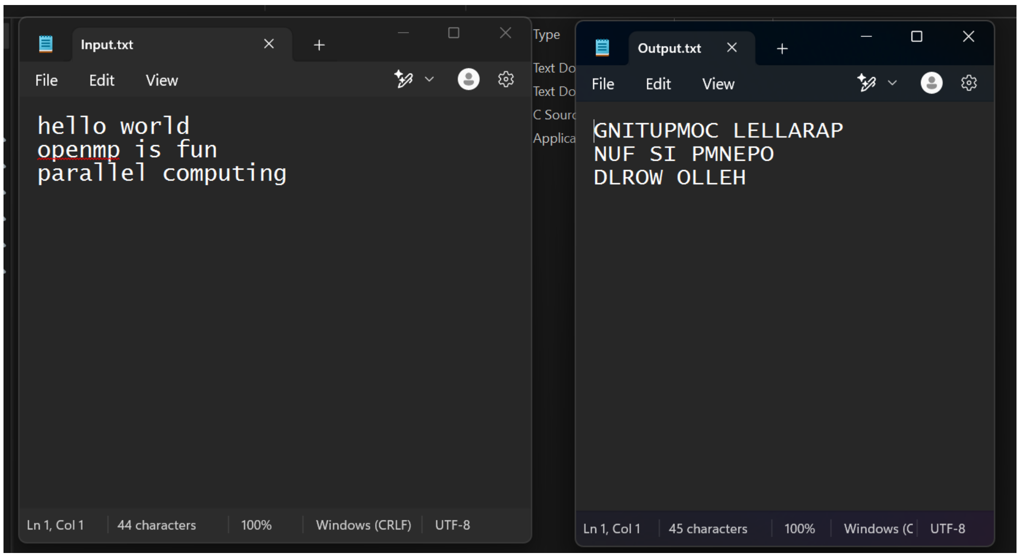


```
hello world
openmp is fun
parallel computing
```

```
parallel computing
openmp is fun
hello world
```

Fig. 2: Conversion to Lowercase and Reversing the order

3. Combining Multiple Transformations (-u -rc -rl options)



```
hello world
openmp is fun
parallel computing
```

```
GNITUPMOC LELLARAP
NUF SI PMNEPO
DLROW OLLEH
```

Fig. 3: Multiple Transformations

The output demonstrates how OpenMP-based parallel processing efficiently applies text transformations to large files. The speedup achieved using multi-threading makes this approach ideal for large-scale text manipulation tasks such as data preprocessing, formatting, and transformation in real-time applications.

Future Work

The current implementation of Parallel File Text Processing Using OpenMP in C effectively enhances the speed of text manipulation tasks by leveraging multi-threading. However, there is significant potential for further improvements and extensions to make the system more scalable, efficient, and versatile. Future work can focus on the following key areas:

Expanding the Range of Text Processing Features

Currently, the program supports case conversion, character reversal, and line order reversal. Future enhancements can include:

- Find and Replace Functionality – Allowing users to replace specific words or characters across large text files efficiently.
- Word Frequency Analysis – Implementing word count and frequency analysis using parallel processing for text analytics applications.
- Sorting Lines Alphabetically – Adding an option to sort lines in ascending or descending order using parallelized sorting algorithms.
- Removing Duplicates – Identifying and eliminating duplicate lines to improve text preprocessing in data analysis.

By integrating these features, the program can be used in a wider range of data processing and text mining applications.

User-Friendly Interface and Configuration Options

The existing implementation operates through command-line arguments. Enhancing usability through an interactive interface can improve user experience:

- Graphical User Interface (GUI) – Developing a desktop-based GUI using Qt or GTK to allow non-technical users to interact with the system.
- Web-Based Interface – Creating a web-based tool using Node.js and Express to enable remote file processing.
- Configuration File Support – Allowing users to define text processing options in a config file rather than passing multiple command-line arguments.

A user-friendly approach would make the system more accessible and convenient for a broader audience.

Conclusion

This project successfully demonstrates the power of parallel computing in text processing by leveraging OpenMP in C. By implementing multi-threading, the system enhances the efficiency of text transformation tasks such as case conversion, text reversal, and line reordering. The primary objective was to improve processing speed, especially for large text files, by distributing tasks across multiple CPU cores. The results clearly show that parallel execution can significantly reduce execution time compared to sequential processing, making it an effective approach for handling large-scale text operations. The project highlights the practical advantages of OpenMP, such as ease of implementation, automatic workload distribution, and improved performance in text-based applications.

One of the key takeaways from this project is the impact of parallelization on performance. When tested on large text files, the parallel implementation consistently outperformed sequential execution, particularly on multi-core systems. However, it also revealed that the benefits of parallelism become more evident with increasing data size, while for smaller files, the overhead of thread management may slightly offset performance gains. This observation emphasizes the importance of choosing the right scenarios for parallel execution, ensuring that parallel computing is used efficiently to maximize its advantages. Additionally, synchronization techniques such as critical sections were effectively used to maintain data consistency while processing text files concurrently.

Despite its success, this project also presents challenges and opportunities for further optimization. Thread synchronization, memory management, and load balancing remain critical aspects that could be improved in future versions. Additionally, implementing adaptive thread allocation, where the system dynamically adjusts the number of threads based on the available processing power and workload, can further enhance performance. Exploring alternative parallel computing models, such as GPU-based processing using CUDA or OpenCL, could push the project's efficiency even further. These improvements would make the system more robust, scalable, and adaptable for a wide range of text-processing tasks beyond the current functionalities.

In conclusion, "Parallel File Text Processing Using OpenMP in C" provides a strong foundation for efficient text manipulation through parallel computing. It showcases how OpenMP can be effectively integrated into C programs to accelerate processing speed and enhance performance. This project lays the groundwork for future research and development in high-speed text processing, distributed computing, and performance optimization. With further refinements and feature expansions, it could evolve into a more comprehensive text processing tool that finds applications in data preprocessing, text analytics, and large-scale document management systems. This project serves as an excellent example of how parallel computing can revolutionize traditional computational tasks, making them faster and more efficient in the modern era of multi-core processing.

References

- https://www.openmp.org/
- https://www.geeksforgeeks.org/strcpy-in-c/
- https://www.geeksforgeeks.org/g-fact-70/
- https://www.geeksforgeeks.org/c-parallel-for-loop-in-openmp/