OS Assignment - 5
Report

Submitted by Group ID Number - **62**

1. **Akshit Gupta 2022058**
2. **Manit Kaushik 2022277**

Contributions -

Manit Kaushik	Akshit Gupta
• Implemented void void parallel_for(int low1, int high1, int low2, int high2, std::function<void(int, int)> &&lambda, int numThreads); for matrix.cpp • Making the design document needed	• Implemented void parallel_for(int low, int high, std::function<void(int)> &&lambda, int numThreads); for vector.cpp • Implementing error checks throughout the code wherever necessary

Simple-MultiThreader Implementation -

The **simple-multithreader.h** header file provides a simple framework for parallelizing tasks using threads in C++. It includes functionalities for passing lambda functions as parameters to parallelize vector addition and matrix multiplication. The primary components include:

1. **Lambda Function Demonstration**:
   The header file demonstrates how to pass lambda functions as parameters using r-value references (&&). It showcases capturing variables by value and reference, emphasizing that global variables are captured by reference by default.
2. **Vector Summation**:
   The vector_sum_func function performs parallel vector addition. It divides the vector into segments and assigns each segment to a separate thread. The

function calculates the execution time and handles thread creation errors.

3. **Parallel Vector Addition**:
   The parallel_for function facilitates parallel vector addition by creating multiple threads. It accepts a lambda function representing the vector addition operation and splits the vector into segments for parallel processing. The function calculates the execution time and checks for thread creation errors.
4. **Matrix Multiplication**:
   The matrix_sum_func function performs parallel matrix multiplication. Similar to vector addition, it divides the matrix into segments and assigns each segment to a separate thread. It calculates the execution time and handles thread creation errors.
5. **Parallel Matrix Multiplication**:
   The parallel_for function is extended to handle parallel matrix multiplication. It accepts a lambda function representing the matrix multiplication operation and divides the matrix into segments for parallel processing. The function calculates the execution time and checks for thread creation errors.
6. **Argument Storage**:
   Two-dimensional arrays (arguments_arr_vector and arguments_arr_matrix) store arguments for each thread. These arrays store information such as the low and high indices for vector operations and the dimensions of matrix operations.
7. **Thread Argument Structures:**

   Two structures (ThreadArgs_vector and ThreadArgs_matrix) encapsulate thread-related arguments. They include the thread number and the lambda function to be executed.

8. **Execution Time Measurement**:
   The clock function is used to measure the execution time of parallelized functions. The time taken for execution is printed for both vector addition and matrix multiplication.
9. **Error Handling**:
   The code includes error checks for thread creation to ensure that threads are successfully created. If an error occurs, the program prints an error message and exits with failure status.
10. **Main Function Demonstration**:
    The main function serves as a demonstration and testing ground for the provided functionalities. It showcases the use of lambda functions for welcoming messages, executes user-defined tasks, and prints a farewell message.