ASSIGNMENT-5

This code implements a multithreaded **parallel for loop** abstraction for both 1D and 2D operations. It divides a range of indices (or index pairs) into chunks, assigns them to threads, and applies a user-provided lambda function to each index or pair. The `parallel_for` functions handle thread creation, workload distribution, and thread synchronization using `pthread_create` and `pthread_join`. It ensures even and uneven workloads are properly managed, with additional logic for any leftover indices. Finally, the program measures execution time, providing performance insights for the multithreaded operation

CODE STRUCTURE

1. Header Inclusions

```
#include <iostream>
#include <list>
#include <functional>
#include <stdlib.h>
#include <cstring>
#include <sys/time.h>
```

- `#include <iostream>`: Includes the standard input/output stream library for basic I/O operations like `cout` and `endl`.
- `#include <list>`: Includes the `std::list` container, although it's not explicitly used in this code.
- `#include <functional>`: Provides the `std::function` class template, allowing functions to be passed as objects (used for lambda expressions in this program).
- `#include <stdlib.h>`: Includes standard library functions like `exit()`.
- `#include <cstring>`: Provides functions for C-style string manipulation.
- `#include <sys/time.h>`: Enables time measurement by providing `gettimeofday()` for capturing timestamps.

2. Function Declarations

```
void *to_be_executed_by_thread_matrix(void* ptr);
void *to_be_executed_by_thread_vector(void *ptr);
void parallel_for(int start, int end,function<void(int)> &&lambda, int no_of_threads);
void parallel_for(int start_1, int end_1, int start_2, int end_2,function<void(int, int)> &&lambda, int no_of_threads);
```

- **Purpose**: Declares the main utility functions for the program to ensure proper order during the compilation process.
- **Functions**:
    - `to_be_executed_by_thread_matrix`: Thread function for 2D (matrix) computations.
    - `to_be_executed_by_thread_vector`: Thread function for 1D (vector) computations.
    - `parallel_for`: Two overloaded versions for 1D and 2D parallel computations, respectively.

3. Struct Definitions

```
typedef struct {
  function<void(int,int)> lambda;
  int start_1;
  int end_1;
  pthread_t tid;
  int start_2;
  int end_2;
} threads_matrix;

typedef struct{
  function<void(int)> lambda;
  pthread_t tid;
  int start;
  int end;
} threads_vector;
```

- **Purpose**: Define custom data structures for thread arguments.
- **Details**:
    - **`threads_matrix`**:
        - Holds data for 2D computation, including:
            - `lambda`: Lambda function to be applied on matrix indices.
            - `start_1` & `end_1`: Indices for the outer loop range.
            - `start_2` & `end_2`: Indices for the inner loop range.
            - `tid`: Thread identifier.
    - **`threads_vector`**:
        - Similar to `threads_matrix`, but for 1D computation:
            - `start` & `end`: Index range for the vector.

4. Function: `parallel_for` (1D Version)

```
void parallel_for(int start, int end,function<void(int)> &&lambda, int no_of_threads) {
  ...
}
```

- **Purpose**: Distributes vector computation across multiple threads.
- **Key Operations**:
  1. **Input Validation**:
     - Checks if `end <= start` or `no_of_threads <= 0`. If invalid, the program terminates.
  2. **Time Capture**:
     - Captures the start time using `gettimeofday()` for performance measurement.
  3. **Thread Setup**:
     - Defines an array `threads_vector args[]` to hold arguments for each thread.
     - Calculates `chunk_size` to evenly distribute indices among threads.
     - Handles the edge case where the range cannot be evenly divided (`starting_incase_not_completed`).
  4. **Thread Creation**:
     - Loops through the number of threads, assigning index ranges (`start`, `end`) and passing the lambda function.
     - Creates threads using `pthread_create()`, pointing to `to_be_executed_by_thread_vector`.
  5. **Remaining Indices**:
     - If indices remain, an additional thread (`rem_th`) handles them.
  6. **Thread Joining**:
     - Joins all threads using `pthread_join()` to ensure computation completes.
  7. **Execution Time**:
     - Captures the end time and calculates the execution duration.

5. Function: `parallel_for` (2D Version)

```
void parallel_for(int start_1, int end_1, int start_2, int end_2,function<void(int, int)> &&lambda, int no_of_threads) {
  ...
}
```

- **Purpose**: Distributes matrix computation across multiple threads.
- **Key Operations**:
    - Similar to the 1D version but with added complexity to handle two dimensions.
    - Divides the outer loop (`start_1`, `end_1`) and inner loop (`start_2`, `end_2`) among threads.
    - Handles edge cases for uneven division of the outer loop.

6. Thread Worker Functions

```
void *to_be_executed_by_thread_matrix(void* ptr) {
  ...
}

void *to_be_executed_by_thread_vector(void *ptr) {
  ...
}
```

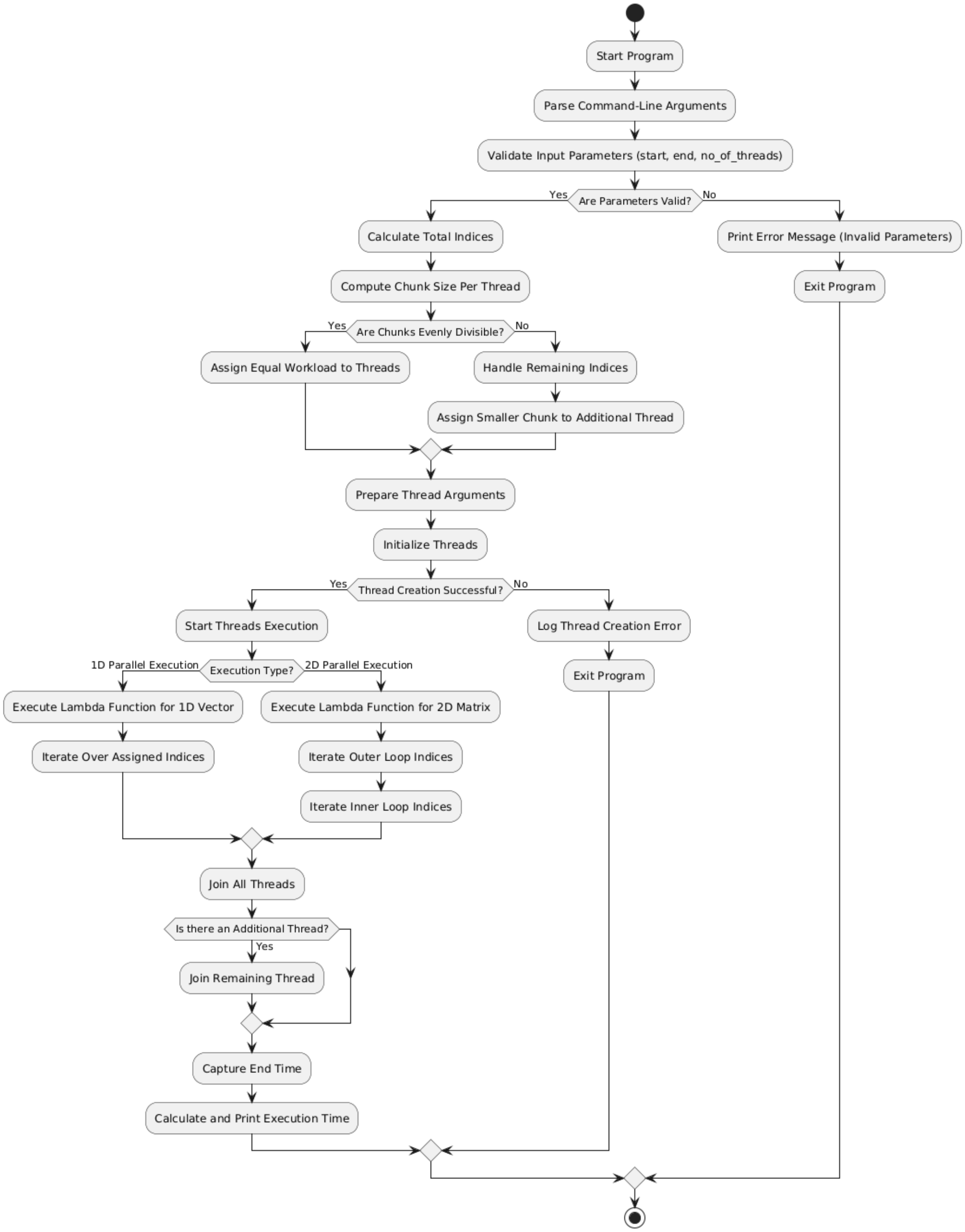
- **Purpose**: Define the computation each thread performs.
- **Details**:
    - `to_be_executed_by_thread_matrix`:
        - Iterates through the specified range of matrix indices and applies the lambda function on each pair `(i, j)`.
    - `to_be_executed_by_thread_vector`:
        - Iterates through the specified range of vector indices and applies the lambda function on each index `i`.

7. Main Function

```
int user_main(int argc, char **argv);
int main(int argc, char **argv) {
    int x = user_main(argc, argv);
    return x;
}
#define main user_main
```

- **Purpose**: Wraps the user-defined `main()` function (`user_main`) and ensures it runs with the correct signature.

CODE FLOW


Start Program
Parse Command-Line Arguments
Validate Input Parameters (start, end, no_of_threads)
Are Parameters Valid?
Yes
No
Calculate Total Indices
Print Error Message (Invalid Parameters)
Compute Chunk Size Per Thread
Exit Program
Are Chunks Evenly Divisible?
Yes
No
Assign Equal Workload to Threads
Handle Remaining Indices
Assign Smaller Chunk to Additional Thread
Prepare Thread Arguments
Initialize Threads
Thread Creation Successful?
Yes
No
Start Threads Execution
Log Thread Creation Error
Execution Type?
1D Parallel Execution
2D Parallel Execution
Exit Program
Execute Lambda Function for 1D Vector
Execute Lambda Function for 2D Matrix
Iterate Over Assigned Indices
Iterate Outer Loop Indices
Iterate Inner Loop Indices
Join All Threads
Is there an Additional Thread?
Yes
Join Remaining Thread
Capture End Time
Calculate and Print Execution Time

CONTRIBUTIONS

1. Priyanshu Sharma / 2023408

- Code logic
- Commenting
- GitHub contribution

2. Vansh Tyagi / 2023582

- Code logic
- Documentation
- GitHub contribution