Struct Definitions:

● thread_args: Structure containing parameters for the one-dimensional parallel_for.

● thread_args2: Structure containing parameters for the two-dimensional parallel_for.

● Function Declarations:

●

● void *thread_func(void* ptr): Function to be executed by each thread in the one-dimensional parallel_for.

● void *thread_func2(void* ptr): Function to be executed by each thread in the two-dimensional parallel_for.

● void parallel_for(int low, int high, std::function<void(int)> &&lambda, int numThreads): One-dimensional parallel_for function.

● void parallel_for(int low1, int high1, int low2, int high2, std::function<void(int, int)> &&lambda, int numThreads): Two-dimensional parallel_for function.

● void demonstration(std::function<void()> && lambda): Function to demonstrate how to pass a lambda as a parameter.

● int user_main(int argc, char **argv): Placeholder for the user's main function.

● Main Function (main):

● Demonstrates the use of lambda functions.

● Calls the user's main function (user_main).

● Displays a farewell message.

● Timing Measurement:

● Uses std::chrono to measure the execution time of parallel_for functions.

● Records the start and end times and calculates the duration in microseconds.

● Execution in the user_main Function:

● Calls the parallel_for functions with appropriate lambda functions and loop parameters.

● Demonstrates the use of the demonstration function with lambda expressions.

Multithreading:

● Creates multiple threads using pthreads to execute the specified functions concurrently.

● Uses thread arguments (thread_args and thread_args2) to pass parameters to the threads.

Error Handling:

● Prints an error message if there is an issue creating a thread.

Cleanup:

● Joins threads to ensure that the main program waits for all threads to finish.

Output:

● Prints the execution time for both one-dimensional and two-dimensional parallel_for functions.

Design Considerations

1. Error Handling

Errors during thread creation and joining are reported to standard error.

The program may exit upon encountering an error in thread creation or joining.

2. Timing

Execution time for both 1D and 2D parallel_for is measured using std::chrono.

3. Lambda Functions

The framework allows users to pass lambda functions for parallel execution.

4. Thread Count

Users can specify the number of threads to be used for parallel execution.

Conclusion

The parallel processing framework provides a simple and flexible mechanism for parallelizing computations. Users can leverage this framework by providing their own functions and customizing the range of values for parallel execution.