Simple Multithreading

This C++ code provides a basic implementation of a parallel_for construct using pthreads, allowing for parallel execution of loops. The code includes examples of parallel_for with a single loop and a nested double loop, along with a demonstration of passing a lambda function as a parameter.

Introduction:

Multithreading is a powerful technique to enhance the performance of programs by executing multiple threads concurrently. This code showcases a basic implementation of parallelized loops using the pthreads library in C++. It includes examples of parallel_for with both single and nested double loops. Additionally, the code demonstrates the use of lambda functions, a feature introduced in C++11, as parameters to achieve flexibility in defining parallelizable tasks.

Usage:

To compile and run the provided C++ code, follow the steps below:

1. Navigate to the Directory:

-> Open a terminal.

- >Change your current working directory to the one containing the `simple_multithreader` code.

```bash
cd path/to/simple_multithreader
```

2. Compile the Code:

- >Use the `make` command to compile the code.

```bash
make
```

3. Run the Executable:

- >Execute the compiled program using `./filename` this will run the file on 2 threads and array size of 1024.

  - >Else you can specify the number of threads and size like `./filename (NUM_THREADS) (SIZE)`

```bash
  `./filename (NUM_THREADS) (SIZE)`
  ```

Replace ``./filename (NUM_THREADS) (SIZE)` with the actual name of the compiled executable: and num threads and size of your choice.

4. View Output:

  -> Observe the output in the terminal, which will include information about the execution time of parallel_for calls and the demonstration of lambda functions.

5. Clean Up (Optional):

  - If needed, you can use `make clean` to remove the compiled executable and object files.

```bash
  make clean
  ```

Note→

- >Make sure you install the necessary build tools on your system, such as `g++` and `make`. If not, you may need to install them before running the instructions.

Overview of Functions:

1. `demonstration` Function

  -> Purpose: Demonstrates how to pass a lambda function as a parameter.

  - >Parameters:

    - >`std::function<void()> && lambda`: A lambda function to be executed.

- > Usage: The function takes a lambda function as a parameter and executes it.

```cpp
void demonstration(std::function<void()> && lambda);
```

2. `parallel_for` Function (Single Loop)

->Purpose: Implements a parallelized version of a single loop using pthreads.

- >Parameters:
  - int low: Lower bound of the loop.
  - int high: Upper bound of the loop.
  - std::function<void(int)>&& lambda : Lambda function to be executed in parallel.
  - int numThreads : Number of threads to be used.
- > Usage: Splits the loop range into chunks and executes the lambda function in parallel using multiple threads.

```cpp
void parallel_for(int low, int high, std::function<void(int)>&& lambda, int numThreads);
```

3. `parallel_for` Function (Double Loop)

- >Purpose: Implements a parallelized version of a nested double loop using pthreads.
- >Parameters:
  - int low1 : Lower bound of the outer loop.
  - int high1 : Upper bound of the outer loop.
  - int low2 : Lower bound of the inner loop.
  - int high2 : Upper bound of the inner loop.
  - std::function<void(int, int)>&& lambda : Lambda function to be executed in parallel.
  - int numThreads : Number of threads to be used.
- > Usage: Similar to the single loop version, but with an additional set of loop bounds for the inner loop.

```cpp
void parallel_for(int low1, int high1, int low2, int high2, std::function<void(int, int)>&& lambda, int numThreads);
```

4. `for_loop` Function

->Purpose: Executes a simple for loop within the given range and lambda function.

- >Parameters:

  - int low : Lower bound of the loop.

  - int high : Upper bound of the loop.

  - std::function<void(int)>&& lambda : Lambda function to be executed.

- Usage: Called by the `thread_func_for` function to perform the loop within a specific thread.

```cpp
void for_loop(int low, int high, std::function<void(int)>&& lambda);
```

5. `double_for_loop` Function

-> Purpose: Executes a nested double loop within the given ranges and lambda function.

- > Parameters:

  - int low1 : Lower bound of the outer loop.

  - int high1 : Upper bound of the outer loop.

  - int low2 : Lower bound of the inner loop.

  - int high2 : Upper bound of the inner loop.

  - std::function<void(int, int)>&& lambda : Lambda function to be executed.

- > Usage: Called by the `thread_func_double_for` function to perform the nested loop within a specific thread.

```cpp
void double_for_loop(int low1, int high1, int low2, int high2, std::function<void(int, int)>&& lambda);
```

6. `thread_func_for` Function

-> Purpose: Thread function for executing the `for_loop` in parallel.

- > Parameters:

  - void* ptr : Pointer to thread arguments (`thread_args_for` structure).

-> Usage: Created by the `parallel_for` function to run a portion of the loop in a separate thread.

```cpp
void* thread_func_for(void* ptr);
```

7. `thread_func_double_for` Function

-> Purpose: Thread function for executing the `double_for_loop` in parallel.

-> Parameters:

  - void* ptr : Pointer to thread arguments (`thread_args_double_for` structure).

-> Usage: Created by the `parallel_for` function (double loop version) to run a portion of the nested loop in a separate thread.

```cpp
void* thread_func_double_for(void* ptr);
```

8. `main` Function

-> Purpose: Entry point of the program.

- > Usage: Demonstrates the usage of parallel_for with examples and showcases lambda function usage.

```cpp
int main(int argc, char **argv);
```

These functions collectively provide a framework for parallelizing loops and demonstrate the use of lambda functions in C++. The `main` function serves as the entry point for the program, showcasing the functionality of the implemented constructs.

Contribution:

->Angadjeet Singh (2022071): Implemented the logic building of the code and error handling.

-> Apaar IIITD (2022089): Implemented the basic structure of the code and worked on error handling.

GitHub Link --→ https://github.com/apaar0001/OS_Assignment_5