Open MP "Hello World" with output ordered by rank.

PDC Assignment 3 Part 1 2023

Gia Bao Hoang – a1814824

I. Objective:

The objective of this project was to modify Program 5.1 from our textbook into an OpenMP (OMP) program that delivers ordered output. We were tasked with developing a communication mechanism suitable for the shared memory model, which forms the basis of OMP. The usage of directives was restricted to:

```
#pragma omp parallel
#pragma omp single
#pragma omp for
```

II. Implementation Strategy:

To begin the parallelization of the program, a parallel region was created using the "#pragma omp parallel" directive. This directive forms a team of threads that execute the code block in parallel. The count of threads is taken from an input argument and stored in 'thread_count'. In the Hello() function, we can determine the number of threads by calling omp_get_num_threads().

2.1 Single Output:

For printing the statement "Number of threads = …" just once, there were two possible routes. The first was to print the statement outside the parallel region. While this approach is correct, it wouldn't have allowed us to use all the required directives, notably "#pragma omp single".

We chose the second approach, which employs the "#pragma omp single" directive. This directive ensures that the enclosed block of code is executed by just one thread, no matter the total number of threads in the team. The other threads wait at an implicit barrier until the single thread finishes executing the block.

2.2 Ordered Output:

In the original Program 5.1, each thread constructed and printed its own greeting, which led to a race condition. To resolve this, we adopted a master-workers partitioning strategy. With OMP, we treated the main program before it forked as the master, and the forked threads as workers.

In our modified version, each worker constructs its own greeting and stores it in a global buffer array by calling the Hello() function. The position and order in the array are determined by the worker thread's rank, which is retrieved from the function omp_get_thread_num(). Here, the "#pragma omp for" directive is used to distribute the loop iterations evenly among the threads in the team. In the context of this program, the number of iterations matches the number of threads, ensuring that each thread only calls the Hello() function once. Thus, each thread constructs its own greeting and stores it in the buffer array.

Finally, once the parallel region ends and all threads have rejoined, the master thread goes through the buffer and prints out the greetings in order. This approach ensures an ordered output and avoids potential race conditions related to console output. Through this method, we were able to achieve effective inter-thread communication and ordered output, while adhering to the shared memory model of OpenMP.