Lab Report 1

CSE455 - High Performance Computing

REDA MOHSEN REDA
18P5141
CESS

Problem Statement

Given an array with size N=100, search in the array for a specific number and print **ALL** indices in which this number appears, and the id of the thread in which this number was found.

Introduction About Solution

The code searches for a specific number in an array of size 100. It uses OpenMP to parallelize the search across multiple threads. The code prints out all the indices in which the number appears and the ID of the thread in which it was found.

The code initializes an array of size 100 with random numbers between 0 and 19 using the rand() function. It then declares two variables, "tid" and "nthreads," where tid represents the thread ID, and nthreads specifies the number of threads to use in parallel execution. It then parallelizes the for loop using OpenMP by defining a private "tid" variable and setting the number of threads using the "num_threads" directive.

Inside the parallel region, the code searches for the number 15 in each portion of the array assigned to each thread using a nested for loop. If it finds the number, it prints out the index of the number and the ID of the thread in which it was found.

Solution

Code

```
#include <iostream>
#include <omp.h>

#define N 100
#define MOD 20

int main() {
    int arr[N];
    for (int i = 0; i < N; i++) {
        arr[i] = rand() % MOD;
    }
    int num = 15;
    int nthreads = 4;
    //function to return the ID of the current thread
    #pragma omp parallel num_threads(nthreads)
    {
        int tid = omp_get_thread_num();

        //divide the array evenly among the threads
        int start = tid * N / nthreads;
        int end = (tid + 1) * N / nthreads;
        //each thread search for a separate part of the array
        for (int i = start; i < end; i++) {

            //the number is found
            if (arr[i] == num) {

                //the index and thread ID are printed.
                std::cout << "Thread " << tid << " found " << num << " at index " << i <<
std::endl;
            }
        }
    }

    return 0;
}
```

Conclusion

In conclusion, this code demonstrates how to use OpenMP to parallelize a search algorithm to improve its performance. By dividing the array into smaller portions and searching for the number in parallel, the code can utilize the resources of multiple threads to speed up the search process.