Indian Institute of Information Technology Surat

Lab Report on
High Performance Computing (CS 602) Practical

Submitted by

[RAHUL KUMAR SINGH] (UI21CS44)

Course Faculty

Dr. Sachin D. Patil

Department of Computer Science and Engineering
Indian Institute of Information Technology Surat
Gujarat-394190, India

Jan-2024

Lab No: 2

Aim: To run the sample program given for OpenMP and design the solution for assignment 1 for the sum of the dot product.

Description: Important Directives of OpenMP:

- #pragma omp parallel: Begins a parallel region, where a block of code will be executed by multiple threads.
- #pragma omp for: Distributes the iterations of a loop among the available threads in a parallel region.
- #pragma omp critical: Defines a critical section, ensuring that only one thread at a time can execute the enclosed code.
- #pragma omp atomic: Specifies that a specific operation should be executed atomically, preventing race conditions.
- #pragma omp barrier: Synchronizes all threads at the barrier, ensuring that no thread proceeds beyond the barrier until all threads have reached it.
- #pragma omp threadprivate: Declares a variable to be private to each thread in a parallel region.
- #pragma omp reduction: Performs a reduction operation on a specified variable (result) across all threads.
- #pragma omp parallel num_threads(): Sets the number of threads to be used in a parallel region.

Sample Program:

```c
#include <omp.h>
#include <stdio.h>

int main() {
    #pragma omp parallel
    {
        int ID = omp_get_thread_num();
        printf(" hello(%d) ", ID);
        printf(" world(%d) \n", ID);
    }

    return 0;
}
```

```
vboxuser@Ubuntu22: ~/Assignment/HCP/P2
vboxuser@Ubuntu22:~/Assignment/HCP/P2$ gcc -fopenmp hello.c -o hello
vboxuser@Ubuntu22:~/Assignment/HCP/P2$ ./hello
 hello(0)  world(0)
 hello(1)  world(1)
vboxuser@Ubuntu22:~/Assignment/HCP/P2$
```

Source Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define N 10
#define T 5

int main() {
    int n = N;
    double a[N], b[N];
    double result = 0.0;
    for (int i = 0; i < n; ++i) {
        a[i] = (double)rand() / RAND_MAX;
        b[i] = (double)rand() / RAND_MAX;
    }
    int num_threads = T;
    #pragma omp parallel for num_threads(num_threads) reduction(+:result)
    for (int i = 0; i < n; ++i) {
        result += a[i] * b[i];
        int thread_id = omp_get_thread_num();
        printf("%.2f dot %.2f = %.2f from thread %d\n",a[i],b[i],result,thread_id);
    }
    printf("Dot product result: %.2f\n", result);
    return 0;
}
```

Output:

When T (Number of threads) < N (Array Length):

```
vboxuser@Ubuntu22:~/Downloads/openmpi-5.0.1$ gcc -fopenmp P2_dot.c -o P2_dot
vboxuser@Ubuntu22:~/Downloads/openmpi-5.0.1$ ./P2_dot
0.84 dot 0.39 = 0.33 from thread 0
0.78 dot 0.80 = 0.96 from thread 0
0.36 dot 0.51 = 0.19 from thread 3
0.95 dot 0.92 = 1.06 from thread 3
0.64 dot 0.72 = 0.46 from thread 4
0.91 dot 0.20 = 0.18 from thread 1
0.34 dot 0.77 = 0.44 from thread 1
0.28 dot 0.55 = 0.15 from thread 2
0.48 dot 0.63 = 0.45 from thread 2
0.14 dot 0.61 = 0.54 from thread 4
Dot product result: 3.45
```

When T (Number of threads) = N (Array Length):

```
vboxuser@Ubuntu22:~/Downloads/openmpi-5.0.1$ gcc -fopenmp P2_dot.c -o P2_dot
vboxuser@Ubuntu22:~/Downloads/openmpi-5.0.1$ ./P2_dot
0.91 dot 0.20 = 0.18 from thread 2
0.36 dot 0.51 = 0.19 from thread 6
0.95 dot 0.92 = 0.87 from thread 7
0.78 dot 0.80 = 0.63 from thread 1
0.84 dot 0.39 = 0.33 from thread 0
0.64 dot 0.72 = 0.46 from thread 8
0.14 dot 0.61 = 0.09 from thread 9
0.28 dot 0.55 = 0.15 from thread 4
0.48 dot 0.63 = 0.30 from thread 5
0.34 dot 0.77 = 0.26 from thread 3
Dot product result: 3.45
```

When T (Number of threads) > N (Array Length):

```
vboxuser@Ubuntu22:~/Downloads/openmpi-5.0.1$ gcc -fopenmp P2_dot.c -o P2_dot
vboxuser@Ubuntu22:~/Downloads/openmpi-5.0.1$ ./P2_dot
0.78 dot 0.80 = 0.63 from thread 1
0.91 dot 0.20 = 0.18 from thread 2
0.28 dot 0.55 = 0.15 from thread 4
0.36 dot 0.51 = 0.19 from thread 6
0.34 dot 0.77 = 0.26 from thread 3
0.48 dot 0.63 = 0.30 from thread 5
0.95 dot 0.92 = 0.87 from thread 7
0.64 dot 0.72 = 0.46 from thread 8
0.14 dot 0.61 = 0.09 from thread 9
0.84 dot 0.39 = 0.33 from thread 0
Dot product result: 3.45
```

Conclusion:

- Understanding of OpenMP directives for the creation and implementation of threads to enable parallelism.
- Leveraging threads for both cross product and dot product computation enables parallel processing, optimizing performance for large N.
- Incorporating random inputs adds realism to the computational model, simulating scenarios with varied data distributions.