

High Performance Computing Lab

Class: Final Year (Computer Science and Engineering)

Year: 2022-23

PRN: 2019BTECS00089 – Piyush Pramod Mhaske

Batch: B3

Practical No. 2

Github link: <https://github.com/Piyush4620/2019BTECS00089HPCLab>

Title of practical:

To Implement a parallel code for vector scalar addition

Problem Statement 1:

To Study and implementation of first private and shared variables

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

int main()
{
    int a[10] = {6, 2, 8, 4, 1, 10, 7, 9, 3, 5};
    int b[10] = {0};
    int i, c = 7;
    #pragma omp parallel for shared(a, b) firstprivate(c) num_threads(5)
    for (i = 0; i < 10; i++)
    {
        b[i] = a[i] + c;
    }
    printf("Vector Scalar Addition\n");
    for (int i = 0; i < 10; i++)
    {
        printf("b[%d] = %d\n", i, b[i]);
    }
    return 0;
}
```

```

PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical2> gcc -o Sc
ectorAddition.c
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical2> .\ScalarV
Vector Scalar Addition
b[0] = 13
b[1] = 9
b[2] = 15
b[3] = 11
b[4] = 8
b[5] = 17
b[6] = 14
b[7] = 16
b[8] = 10
b[9] = 12
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical2>

```

Information 1:

The clause `private (variable list)` indicates that the set of variables specified is local to each thread – i.e., each thread has its own copy of each variable in the list. The clause `firstprivate (variable list)` is similar to the `private` clause, except the values of variables on entering the threads are initialized to corresponding values before the parallel directive. The clause `shared (variable list)` indicates that all variables in the list are shared across all the threads, i.e., there is only one copy.

Problem Statement 2:

To Implement a parallel code for vector-vector addition

```

#include <omp.h>
#include <stdio.h>
#include <time.h>

void main()
{
    double time_spent = 0.0;

```

```

clock_t begin = clock();

printf("Adding Two Arrays\n");
int a1[] = {1, 2, 3, 4, 5,6,7,8};
int a2[] = {11, 12, 13, 14, 15,16,17,18};
int a3[8];
omp_set_num_threads(5);
int i;
#pragma omp parallel for shared(a1,a2,a3) private(i)
for (i = 0; i < 8; i++)
{
    a3[i] = a1[i] + a2[i];
    #pragma omp critical
    printf("Thread Number:%d = %d\n",omp_get_thread_num(), a3[i]);
}

clock_t end = clock();

time_spent += (double)(end - begin) / CLOCKS_PER_SEC;

printf("Runtime is %f seconds", time_spent);
}

```

output:

```

PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical2> .\addingTwoArrays.c
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical2> ^C
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical2> .\addingTwoArrays.exe
Adding Two Arrays
Thread Number:1 = 16
Thread Number:1 = 18
Thread Number:2 = 20
Thread Number:2 = 22
Thread Number:4 = 26
Thread Number:3 = 24
Thread Number:0 = 12
Thread Number:0 = 14
Runtime is 0.003000 seconds
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical2>

```

Information :

The schedule clause of the for directive deals with the assignment of iterations to threads. The general form of the schedule directive is schedule (scheduling_class[, parameter]). Open MP supports four scheduling classes: static, dynamic, guided, and runtime. The general form of the static scheduling class is schedule (static[, chunk-size]). This technique splits the iteration space into equal chunks of size chunk-size and assigns them to threads in a round-robin fashion. Open MP has a dynamic scheduling class. The general form of this class is schedule (dynamic [, chunk-size]). The iteration

space is partitioned into chunks given by chunk-size. However, these are assigned to threads as they become idle.