Date: 05/09/2022

High Performance Computing Lab

Assignment - 2

PRN: 2019BTECS00029
Name: Harshal Kodgire

SPMD: Single Program Multiple Data
It is a parallel programming style. In SPMD style tasks are split up and run simultaneously on multiple processors with different data. This is done to achieve results faster.

Worksharing
Threads are assigned an independent subset of the total workload For example, different chunks of an iteration are distributed among the threads.
eg.
T1 -> iteration from 1 to 5
T2 -> iteration from 6 to 10 and so on…

OpenMP provides various constructs for worksharing.

OpenMP loop worksharing construct

> OpenMP's loop worksharing construct splits loop iterations among all active threads
> *#pragma omp for*

Types of variables

1. Shared Variables :
There exist one instance of this variable which is shared among all threads

2. Private Variables :
Each thread in a team of threads has its own local copy of the private variable

Two ways to assign variables as private and shared are
Implicit and Explicit
Implicit : All the variables declared outside of the pragma are by default shared and all the variables declared inside pragma are private

Explicit:

Shared Clause

eg. *#pragma omp parallel for shared(n, a)* => n and a are declared as shared vavriables

Private Clause

eg. *#pragma omp parallel for shared(n, a) private(c)* => here c is private variable

Default Clause

eg. *#pragma omp parallel for default(shared)* => now all variables are shared

#pragma omp parallel for default(private) => now all variables are private

firstprivate

firstprivate make the variable private but that variable is initialised with the value that it has before the parallel region

lastprivate

lastprivate make the variable private but it retain the last value of that private variable outside of the private region

Schedule
a specification of how iterations of associated loops are divided into contiguous non-empty subsets.
syntax: *#pragma omp parallel for schedule([modifier [modifier]:]kind[,chunk_size])*

Program for vector to vector addition :

Sequential :

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

int main() {
    int N = 1000;
    int A[1000];
    for (int i = 0; i < N; i++)
        A[i] = 10;

    int B[1000];
    for (int i = 0; i < N; i++)
        B[i] = 20;

    int C[1000] = {0};
    double stime = omp_get_wtime();
    for (int i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
        printf("Index: %d Thread: %d\n", i,
omp_get_thread_num());
    }

    for (int i = 0; i < N; i++) {
        printf("%d ", C[i]);
    }

    double etime = omp_get_wtime();
```

```
    double time = etime - stime;
    printf("Time taken is %f\n", time);
    return 0;
}
```

```
Index: 39 Thread: 0
Index: 40 Thread: 0
Index: 41 Thread: 0
Index: 42 Thread: 0
Index: 43 Thread: 0
Index: 44 Thread: 0
Index: 45 Thread: 0
Index: 46 Thread: 0
Index: 47 Thread: 0
Index: 48 Thread: 0
Index: 49 Thread: 0
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 3
0 30 30 30 30 30 30

 Time taken is 0.010000
```

Parallel：

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

int main() {
    int N = 1000;
    int A[1000];
    for (int i = 0; i < N; i++)
        A[i] = 10;

    int B[1000];
    for (int i = 0; i < N; i++)
        B[i] = 20;

    int C[1000] = {0};
```

```
    double stime = omp_get_wtime();
#pragma omp parallel for reduction(+ \
                                : C)
    for (int i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
        printf("Index: %d Thread: %d\n", i,
omp_get_thread_num());
    }

    for (int i = 0; i < N; i++) {
        printf("%d ", C[i]);
    }

    double etime = omp_get_wtime();
    double time = etime - stime;
    printf("\nTime taken is %f\n", time);
    printf("\n");
    return 0;
}
```

```
Index: 12 Thread: 1
Index: 13 Thread: 1
Index: 14 Thread: 2
Index: 15 Thread: 2
Index: 16 Thread: 2
Index: 17 Thread: 2
Index: 4 Thread: 0
Index: 5 Thread: 0
Index: 6 Thread: 0
Index: 38 Thread: 6
Index: 39 Thread: 6
Index: 40 Thread: 6
Index: 41 Thread: 6
Index: 42 Thread: 6
Index: 43 Thread: 6
Index: 18 Thread: 2
Index: 19 Thread: 2

30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 3
0 30 30 30 30 30 30
```

Vector Scaler multiplication :

Sequential :

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

int main() {
    int N = 500;
    int A[1000];
    for (int i = 0; i < N; i++)
        A[i] = 20;
    int S = 2;

    double itime;
    itime = omp_get_wtime();
    for (int i = 0; i < N; i++) {
        A[i] *= S;
        printf("Index: %d Thread: %d\n", i, omp_get_thread_num());
    }

    for (int i = 0; i < N; i++) {
        printf("%d ", A[i]);
    }

    double ftime = omp_get_wtime();
    double exec_time = ftime - itime;
    printf("\nTime taken is %f\n", exec_time);
    printf("\n");
}
```

```
Index: 41 Thread: 0
Index: 42 Thread: 0
Index: 43 Thread: 0
Index: 44 Thread: 0
Index: 45 Thread: 0
Index: 46 Thread: 0
Index: 47 Thread: 0
Index: 48 Thread: 0
Index: 49 Thread: 0
60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 6
0 60 60 60 60 60 60 60
Time taken is 0.010000


D:\WCE ENGINEERING\BTECH SEM1\HPC Lab>
```

Parallel :

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

int main() {
   int N = 500;
   int A[1000];
   for (int i = 0; i < N; i++)
      A[i] = 20;
   int S = 2;

   double itime;
   itime = omp_get_wtime();
#pragma omp parallel for
   for (int i = 0; i < N; i++) {
      A[i] *= S;
      printf("Index: %d Thread: %d\n", i, omp_get_thread_num());
   }

   for (int i = 0; i < N; i++) {
      printf("%d ", A[i]);
   }
```

```
    double ftime = omp_get_wtime();
    double exec_time = ftime - itime;
    printf("\nTime taken is %f\n", exec_time);
    printf("\n");
}
```

```
Index: 15 Thread: 2
Index: 16 Thread: 2
Index: 17 Thread: 2
Index: 18 Thread: 2
Index: 19 Thread: 2
Index: 49 Thread: 7
60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
0 60 60 60 60 60 60
Time taken is 0.014000
```