HPC LAB : 02
OpenMp Directives, Spmd, worksharing

PRN : 2019BTECS00055
Name : Aditya Rajendra Manapure

Problem Statement 01 . Spmd vs worksharing in OpenMp

WorkSharing : Using the OpenMP pragmas, most loops with no loop-carried dependencies can be threaded with one simple statement. This concept of OpenMP to parallelize loops, is called **worksharing**. A worksharing region has no barrier on entry; however, an implied barrier exists at the end of the worksharing region, unless a clause nowait is specified. If a **nowait** clause is present, an implementation may omit the barrier at the end of the worksharing region. In this case, threads that finish early may proceed straight to the instructions that follow the worksharing region without waiting for the other members of the team to finish the worksharing region, and without performing a flush operation.

SPMD : Computers in which each processing element is capable of executing a different program independent of the other processing elements are called **multiple instruction stream, multiple data** stream **MIMD** computers. A simple variant of this model, called the **single program multiple data SPMD** model, relies on multiple instances of the same program executing on different data.

Problem Statement 01 : OpenMP Clauses

For general attributes:

Clause	Description
if	Specifies whether a loop should be executed in parallel or in serial.
num_threads	Sets the number of threads in a thread team.
ordered	Required on a parallel for statement if an ordered directive is to be used in the loop.
schedule	Applies to the for directive.
nowait	Overrides the barrier implicit in a directive.

For data-sharing attributes:

Clause	Description
private	Specifies that each thread should have its own instance of a variable.
firstprivate	Specifies that each thread should have its own instance of a variable, and that the variable should be initialized with the value of the variable, because it exists before the parallel construct.
lastprivate	Specifies that the enclosing context's version of the variable is set equal to the private version of whichever thread executes the final iteration (for-loop construct) or last section (#pragma sections).
shared	Specifies that one or more variables should be shared among all threads.
default	Specifies the behavior of unscoped variables in a parallel region.
reduction	Specifies that one or more variables that are private to each thread are the subject of a reduction operation at the end of the parallel region.

Problem Statement 1: To Implement a parallel code for vector scalar addition

Screenshot 1:

Assignment-2 > Que1_Addition_vector_scalar.cpp > main()

```cpp
//Implement a parallel code for vector scalar addition
//2019BTECS00055 Aditya Manapure

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

#define THREAD_COUNT 8
#define ARRAY_SIZE 100
#define COUNT_PER_THREAD ARRAY_SIZE/THREAD_COUNT


int main(){

    int A[ARRAY_SIZE],  C[ARRAY_SIZE];
    int scalar = 5;

    for(int i=0; i<ARRAY_SIZE; i++){
        A[i] = i;
    }

    omp_set_num_threads(THREAD_COUNT);
    int i=0;
    double start = omp_get_wtime();
    #pragma omp parallel
    {

        #pragma omp single
            printf("Number of threds Allocated %d\n", omp_get_num_threads());

        #pragma omp single
            printf("Processing Data ... ");

        #pragma omp for
        for(i=0; i<ARRAY_SIZE; i++){
            printf("\n Thread id %d is proccessing on index %d\n", omp_get_thread_num(), i);
            C[i] = A[i] + scalar;
```

```cpp

        #pragma omp barrier

        #pragma omp single nowait
            printf("Displaying Data ... ");

        #pragma omp for
        for(i=0; i<ARRAY_SIZE; i++){
            printf("\n Thread id %d is printing on index %d : %d + %d = %d", omp_get_thread_num(), i, A[i], scalar, C[i]);
        }

    }
    double end = omp_get_wtime();
    double total_time = end - start;
    printf("\n\nTime taken : %lf\n", total_time);

    return 0;
}
```

Output :

```
Thread id 2 is proccessing on index 34

Thread id 2 is proccessing on index 35

Thread id 2 is proccessing on index 36

Thread id 2 is proccessing on index 37

Thread id 2 is proccessing on index 38
isplaying Data ...
Thread id 0 is printing on index 0 : 0 + 5 = 5
Thread id 0 is printing on index 1 : 1 + 5 = 6
Thread id 0 is printing on index 2 : 2 + 5 = 7
Thread id 0 is printing on index 3 : 3 + 5 = 8
Thread id 0 is printing on index 4 : 4 + 5 = 9
Thread id 0 is printing on index 5 : 5 + 5 = 10
Thread id 0 is printing on index 6 : 6 + 5 = 11
Thread id 0 is printing on index 7 : 7 + 5 = 12
Thread id 0 is printing on index 8 : 8 + 5 = 13
Thread id 0 is printing on index 9 : 9 + 5 = 14
Thread id 0 is printing on index 10 : 10 + 5 = 15
Thread id 0 is printing on index 11 : 11 + 5 = 16
Thread id 0 is printing on index 12 : 12 + 5 = 17
Thread id 7 is printing on index 88 : 88 + 5 = 93
Thread id 7 is printing on index 89 : 89 + 5 = 94
```

```
Thread id 5 is printing on index 67 : 67 + 5 = 72
Thread id 5 is printing on index 68 : 68 + 5 = 73
Thread id 5 is printing on index 69 : 69 + 5 = 74
Thread id 5 is printing on index 70 : 70 + 5 = 75
Thread id 5 is printing on index 71 : 71 + 5 = 76
Thread id 5 is printing on index 72 : 72 + 5 = 77
Thread id 5 is printing on index 73 : 73 + 5 = 78
Thread id 5 is printing on index 74 : 74 + 5 = 79
Thread id 5 is printing on index 75 : 75 + 5 = 80

Time taken : 0.094000
```

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.

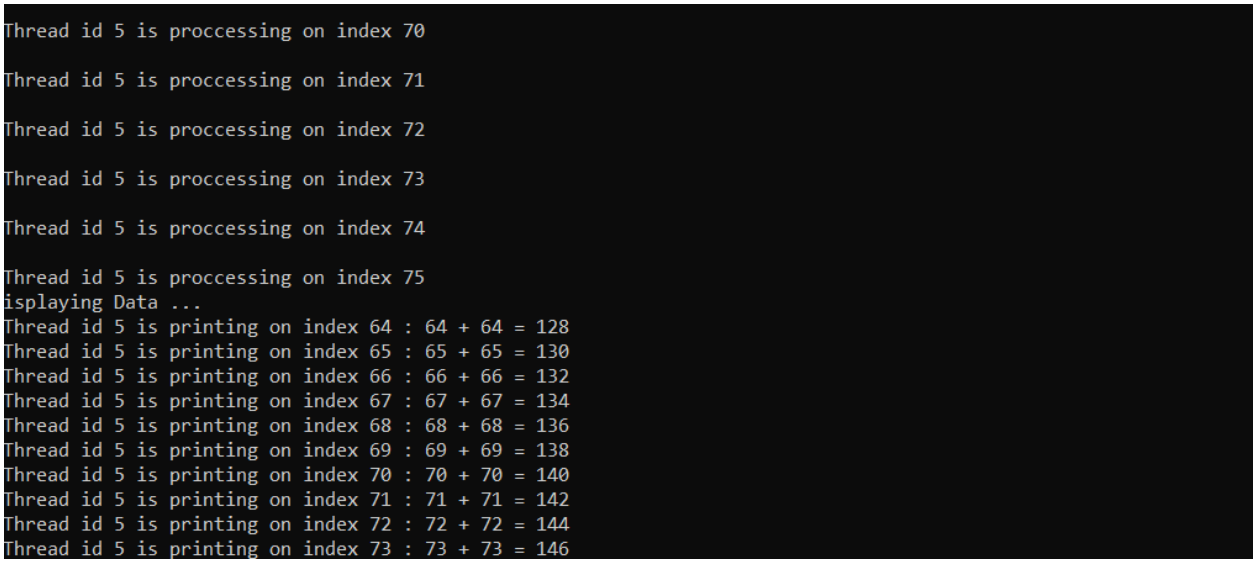
Problem Statement 2: To Implement a parallel code for vector-vector addition

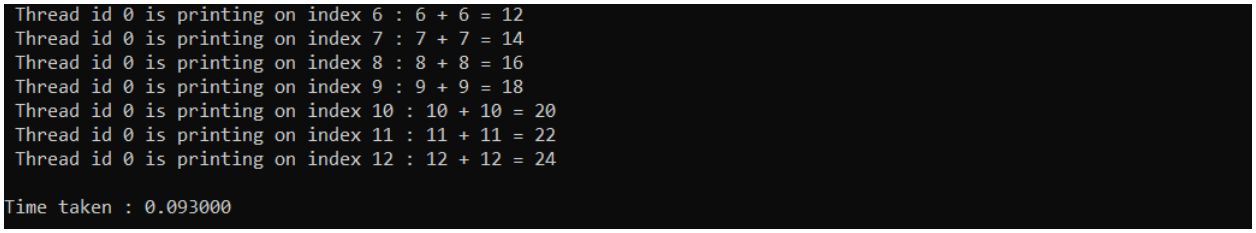
Screenshot 1:

Assignment-2 > Que2_Addition_vector_vector.cpp > main()

```cpp
//Implement a parallel code for vector-vector addition
//Author : 2019BTECS00055 Aditya Manapure

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

#define THREAD_COUNT 8
#define ARRAY_SIZE 100
#define COUNT_PER_THREAD ARRAY_SIZE/THREAD_COUNT


int main(){

    int A[ARRAY_SIZE], B[ARRAY_SIZE], C[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; i++){
        A[i] = i;
        B[i] = i;
    }

    omp_set_num_threads(THREAD_COUNT);
    int i=0;
    double start = omp_get_wtime();
    #pragma omp parallel
    {

        #pragma omp single
            printf("Number of threds Allocated %d\n", omp_get_num_threads());

        #pragma omp single
            printf("Processing Data ... ");

        #pragma omp for
        for(i=0; i<ARRAY_SIZE; i++){
            printf("\n Thread id %d is proccessing on index %d\n", omp_get_thread_num(), i);
            C[i] = A[i] + B[i];
        }


        #pragma omp barrier

        #pragma omp single nowait
            printf("Displaying Data ... ");

        #pragma omp for
        for(i=0; i<ARRAY_SIZE; i++){
            printf("\n Thread id %d is printing on index %d : %d + %d = %d", omp_get_thread_num(), i,  A[i], B[i], C[i]);
        }

    }
    double end = omp_get_wtime();
    double total_time = end - start;
    printf("\n\nTime taken : %lf\n", total_time);

    return 0;
}
```

Screenshot 2:





Information 2:

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.