Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Final Year: High Performance Computing Lab 2022-23 Sem I
Class: Final Year (Computer Science and Engineering)
Year: 2022-23 Semester: 1
Course: High Performance Computing Lab
Assignment No : 2

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

Q1.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

Q.2 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

Q.3 Two ways to assign variables as private ans 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 variables

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])

Q.4 Five kinds of schedules for OpenMP loop:

• static

• dynamic

• guided

• auto

• runtime

Static Schedule ->

In static scheduling openMP assingn interations to threads in a cyclic order

eg: #pragma omp parallel for schedule(static,1)

=> Number “1” indicates that we assign one iteration to each thread before switching to the next thread — we use chunks of size 1.

Dynamic Schedule ->

In dynamic scheduling, each thread will take one iteration, process it, and then see what is the next iteration that is currently not being processed by anyone. This way it will never happen that one thread finishes while other threads have still lots of work to do:

eg: #pragma omp parallel for schedule(dynamic,1)

nowait->

When we use a parallel region, OpenMP will automatically wait for all threads to finish before execution continues. There is also a synchronization point after each omp for loop

However, if we do not need synchronization after the loop, we can disable it with nowait

eg: #pragma omp for nowait

reduction->

The OpenMP reduction clause lets you specify one or more thread-private variables that are subject to a reduction operation at the end of the parallel region.

eg: if we want to calculate the sum of the first 100 natural numbers and want to save it into variable 'sum', we can save individual thread sums and then add all sum to get resultant sum.

Or we can use reduction clause and it will automatically handle each threads sum and then the final sum, syntax is

#pragma omp for reduction(+:sum)

Q5. Write parallel Program in open mp for vector addition

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
#define ARRAY_SIZE 100000
#define NUM_THREADS 8
int main ()
{
   double start , finish;
   int * a;
   int * b;
   int * c;
   int n = ARRAY_SIZE;
   int n_per_thread;
   int total_threads = NUM_THREADS;
   int i;
   a = (int *) malloc(sizeof(int)*n);
   b = (int *) malloc(sizeof(int)*n);
   c = (int *) malloc(sizeof(int)*n);
   for(i=0; i<n; i++) {
       a[i] = i;
   }
   for(i=0; i<n; i++) {
       b[i] = i;
   }


   omp_set_num_threads(total_threads);


   n_per_thread = n/total_threads;



   start = omp_get_wtime();
```

```
    #pragma omp parallel for
    for(i=0; i<n; i++) {
        c[i] = a[i]+b[i];
    }


    finish = omp_get_wtime();



    printf("Execution Time : %f\n",finish - start);

    free(a);  free(b); free(c);

    return 0;
}
```

```
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_02$ ./a.out
Execution Time : 0.026229
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_02$
```

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.

Q6. Write parallel Program in open mp for Scalar Multiplication

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

```c
#include <omp.h>

#define ARRAY_SIZE 100000
#define NUM_THREADS 8

int main ()
{

   double start , finish;



   int * a;
   int * b;

   int n = ARRAY_SIZE;
   int n_per_thread;
   int total_threads = NUM_THREADS;
   int i;


   a = (int *) malloc(sizeof(int)*n);
   b = (int *) malloc(sizeof(int)*n);


   for(i=0; i<n; i++) {
       a[i] = i;
   }

   for(i=0; i<n; i++) {
       b[i] = i;
   }


   omp_set_num_threads(total_threads);


   n_per_thread = n/total_threads;
```

```
   start = omp_get_wtime();

   int sum = 0;

   #pragma omp parallel for shared(a, b , sum) private(i) schedule(static,
n_per_thread)
   for(i=0; i<n; i++) {
       sum += a[i]*b[i];
   }



   finish = omp_get_wtime();

   printf("%f",finish - start);

   free(a);  free(b);

   return 0;
}
```

```
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_02$ ./a.out

Execution TIme : 0.030178
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_02$
```

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.