

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

Class: Final Year (Computer Science and Engineering)

Year: 2022-23

Semester: 1

Course: High Performance Computing Lab

Practical No. 2

Exam Seat No: 2019BTECS00064

Name – Kunal Santosh Kadam

Title of practical: Study on Parallel Programming

Concepts

Problem Statement 1:

Write a short note on following points:

Information :

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 or shared are:

1. Implicit:

All the variables declared outside of the pragma are by default shared and all the variables declared inside pragma are private

2. 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])`

Five kinds of schedules for OpenMP loop:

- static
- dynamic
- guided
- auto
- runtime

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

Static Schedule

In static scheduling openMP assigns iterations 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 sum of 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

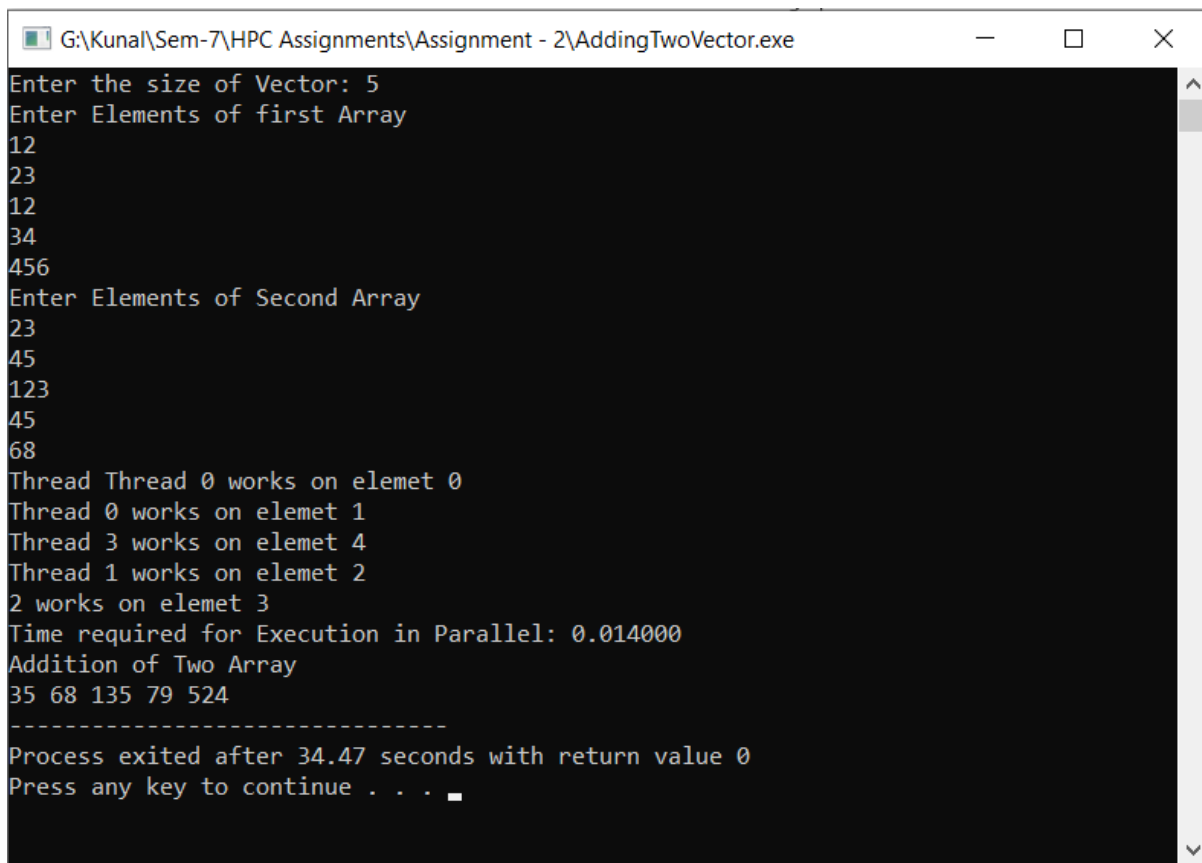
Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

#pragma omp for reduction(+:sum)

Problem Statement 2:

Write a program for Vector-Vector Addition using the OpenMP

Screenshot #:



```
G:\Kunal\Sem-7\HPC Assignments\Assignment - 2\AddingTwoVector.exe
Enter the size of Vector: 5
Enter Elements of first Array
12
23
12
34
456
Enter Elements of Second Array
23
45
123
45
68
Thread Thread 0 works on element 0
Thread 0 works on element 1
Thread 3 works on element 4
Thread 1 works on element 2
2 works on element 3
Time required for Execution in Parallel: 0.014000
Addition of Two Array
35 68 135 79 524
-----
Process exited after 34.47 seconds with return value 0
Press any key to continue . . .
```

Information #:

```
#include<bits/stdc++.h>
```

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

```
using namespace std;
```

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

```
int main()
{
    int n,i;
    cout<<"Enter the size of Vector: ";
    cin>>n;
    vector<int> a(n),b(n),c(n);

    cout<<"Enter Elements of first Array\n";
    for(i=0;i<n;i++)
        cin>>a[i];

    cout<<"Enter Elements of Second Array\n";
    for(i=0;i<n;i++)
        cin>>b[i];

    double getInTime = omp_get_wtime();

    #pragma omp parallel for shared(a, b, c) private(i) schedule(static)

    for(i=0;i<n;i++)
    {
        c[i] = a[i] + b[i];
        printf("Thread %d works on element %d \n",
omp_get_thread_num(), i);
    }

    double getOutTime = omp_get_wtime();

    double exptTime= getOutTime - getInTime;

    printf("Time required for Execution in Parallel: %f\n", exptTime);

    cout<<"Addition of Two Array\n";
```

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

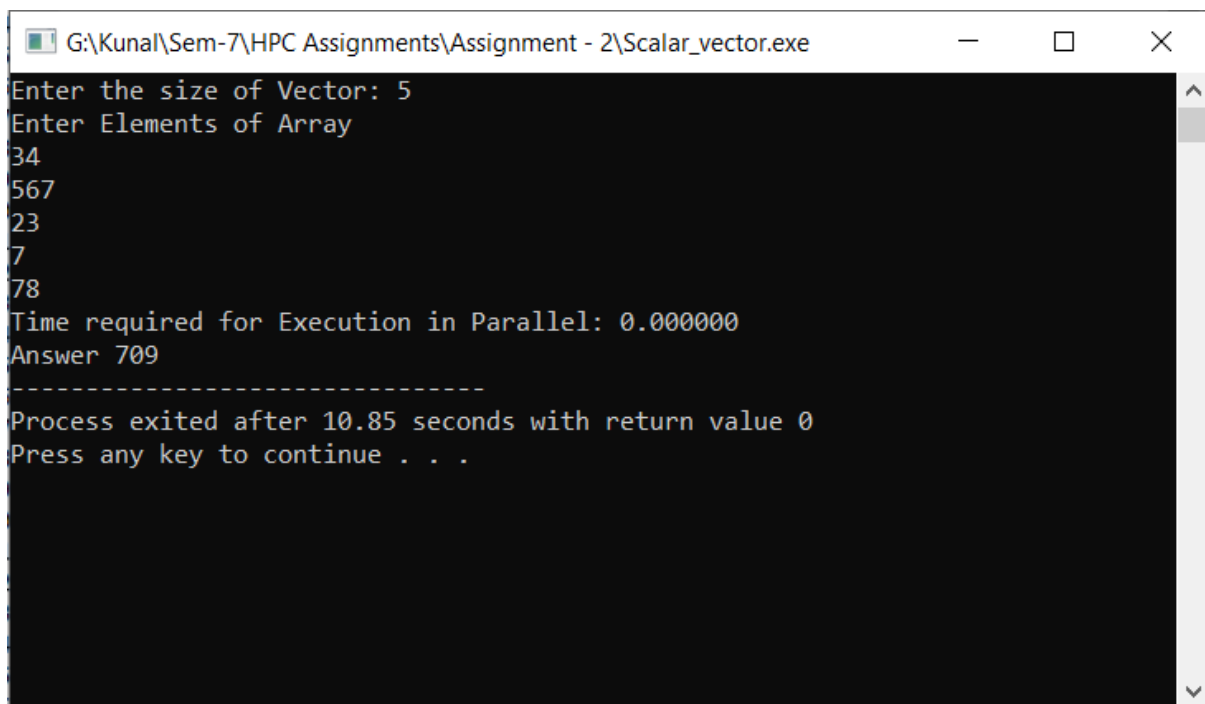
```
for(i=0;i<n;i++)  
{  
    printf("%d ", c[i]);  
}  
  
return 0;  
}
```

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

Problem Statement 3:

Write a program for Vector Scalar Addition using OpenMP

Screenshot #:



```
G:\Kunal\Sem-7\HPC Assignments\Assignment - 2\Scalar_vector.exe
Enter the size of Vector: 5
Enter Elements of Array
34
567
23
7
78
Time required for Execution in Parallel: 0.000000
Answer 709
-----
Process exited after 10.85 seconds with return value 0
Press any key to continue . . .
```

Information #:

```
#include<bits/stdc++.h>
#include<omp.h>

using namespace std;

int main()
{
    int n,i;
```


Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

```
cout<<"Enter the size of Vector: ";
cin>>n;
vector<int> a(n);

cout<<"Enter Elements of Array\n";
for(i=0;i<n;i++)
    cin>>a[i];
int answer = 0;

double getInTime = omp_get_wtime();

#pragma omp parallel for
for (i = 0; i < 5; i++)
{
    answer += a[i];
}

double getOutTime = omp_get_wtime();

double exptTime= getOutTime - getInTime;

printf("Time required for Execution in Parallel: %f\n", exptTime);

printf("Answer %d ", answer);

return 0;
}
```

Github Link:

<https://github.com/Kunalkadam179/HPC-Assignment/tree/main/Assignment%20-%20202>

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