PRIYAL AGRAWAL **ROLL NO -27**
BATCH H2 **1032201406**

Parallel Programming

Assignment no: 2

Problem Statement: Write a program for large vector addition. Convert the serial vector addition program to parallel vector addition. Measure the time taken for serial vs parallel vector addition for various size of the computation. Plot a graph of the execution times (Ts and Tp) identify the threshold for which you start getting speedup.

Aim: To write a C program to add two large vectors and run the program on a multi-core parallel system.

Objective: To understand the conversion of serial code to parallel work.

Theory :

For Vector Addition

Step 1: Divide given vector in N equal part where N is the number of thread available in device.
Step 2: Assign each pair of vectors to a unique thread and perform element-wise addition.
Step 3: simultaneously store the sum into a third vector.
Step 4: That third vector will be considered as result.

#pragma omp parallel for private(x)

Causes the work done in a for loop inside a parallel region to be divided among threads. The private clause allow each thread to have (i) as local variables i.e, have different address space.

Total Cost = Time complexity × Number of processors used

= 4O(n)

Speedup (Max Size) = Ts/Tp

= 0.355817 / 0.14358

= 2.478179

Efficiency = Speedup / p

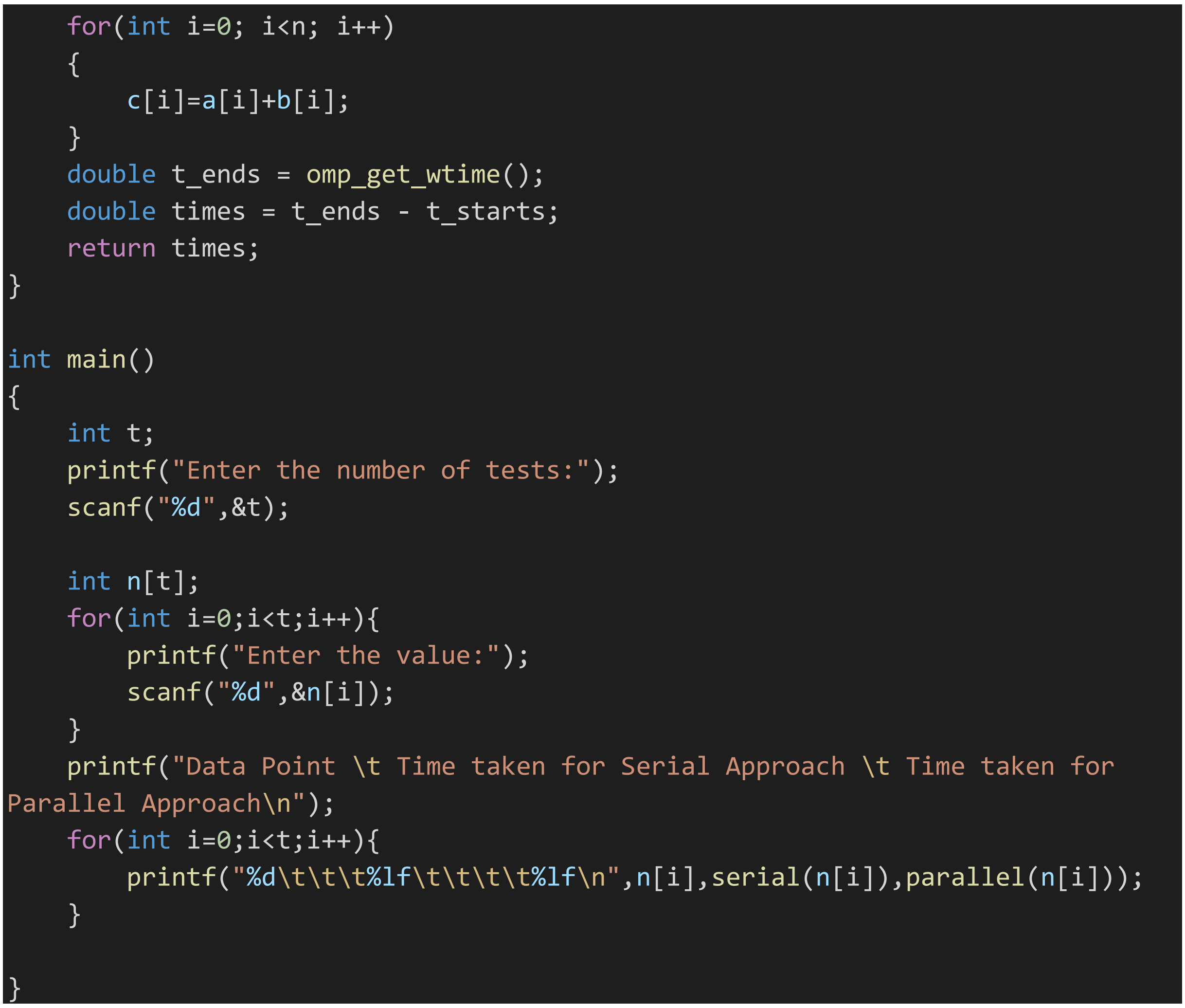
= 2.478179 / 4

= 0.619544853

Where p: number of processors = 4

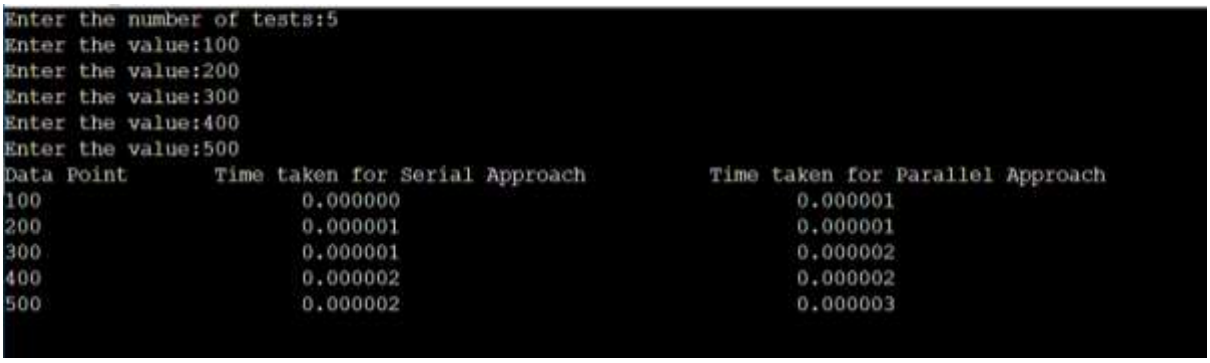
CODE:

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

double parallel(int n){

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

    for(int i=0; i<n;i++)
    {
        a[i] = rand()%n;
        b[i] = rand()%n;
    }

    double t_startp = omp_get_wtime();
    #pragma omp parallelfor private(i)
    for(int i=0; i<n; i++)
    {
        c[i]=a[i]+b[i];
    }
    double t_endp = omp_get_wtime();
    double timep = t_endp - t_startp;
    return timep;
}

double serial(int n){

    int *a,*b,*c;

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

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

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

    double t_starts = omp_get_wtime();
```



```
    for(int i=0; i<n; i++)
    {
        c[i]=a[i]+b[i];
    }
    double t_ends = omp_get_wtime();
    double times = t_ends - t_starts;
    return times;
}

int main()
{
    int t;
    printf("Enter the number of tests:");
    scanf("%d",&t);

    int n[t];
    for(int i=0;i<t;i++){
        printf("Enter the value:");
        scanf("%d",&n[i]);
    }
    printf("Data Point \t Time taken for Serial Approach \t Time taken for 
Parallel Approach\n");
    for(int i=0;i<t;i++){
        printf("%d\t\t\t%lf\t\t\t\t%lf\n",n[i],serial(n[i]),parallel(n[i]));
    }

}
```

OUTPUT:

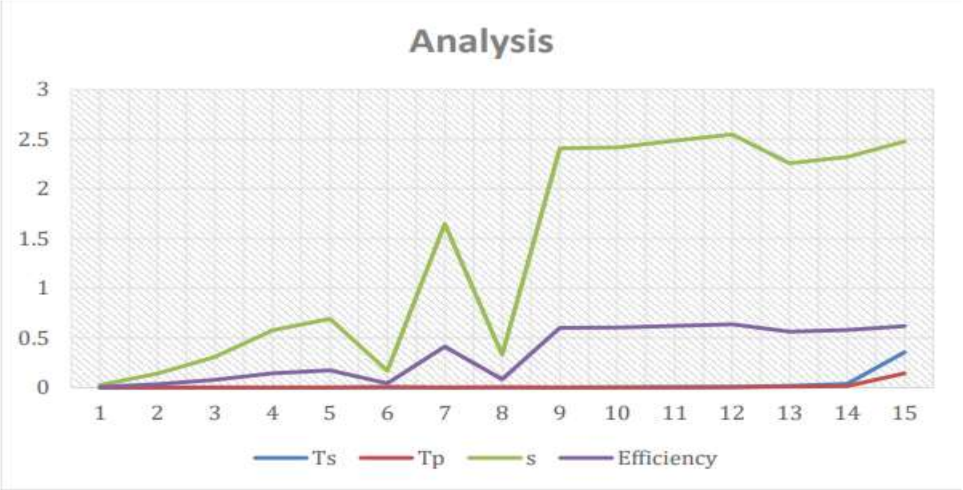


```
Enter the number of tests:5
Enter the value:100
Enter the value:200
Enter the value:300
Enter the value:400
Enter the value:500
Data Point       Time taken for Serial Approach          Time taken for Parallel Approach
100                     0.000000                                 0.000001
200                     0.000001                                 0.000001
300                     0.000001                                 0.000002
400                     0.000002                                 0.000002
500                     0.000002                                 0.000003
```

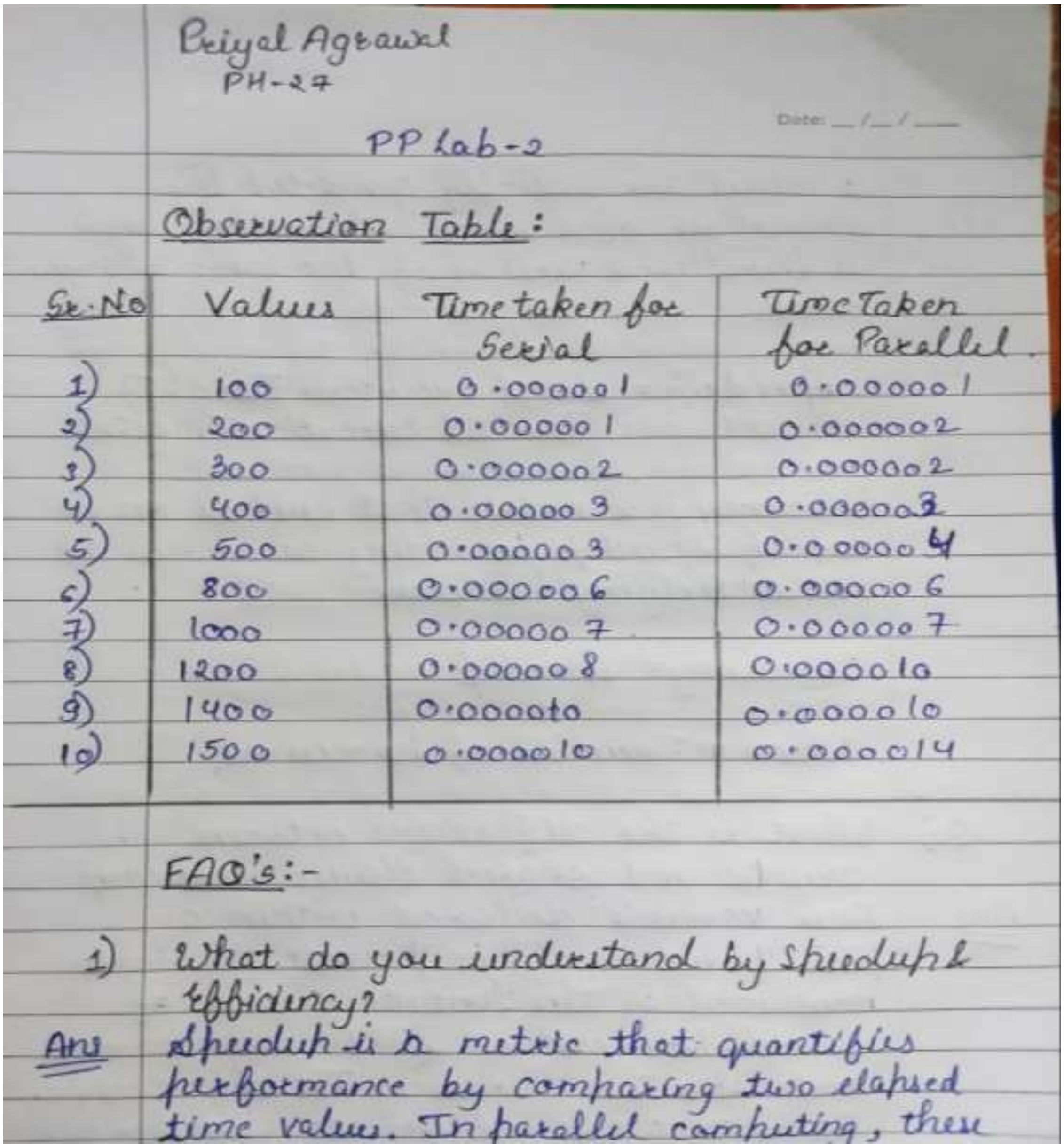
Output: Sorted Array of data points/values.

Platform: Online C compiler GDB

Conclusion: Successfully studied serial to parallel conversion of vector addition code and analyze speedup and efficiency.



FAQS:-



Priyal Agrawal
PH-27

Date: __/__/____

PP Lab-2

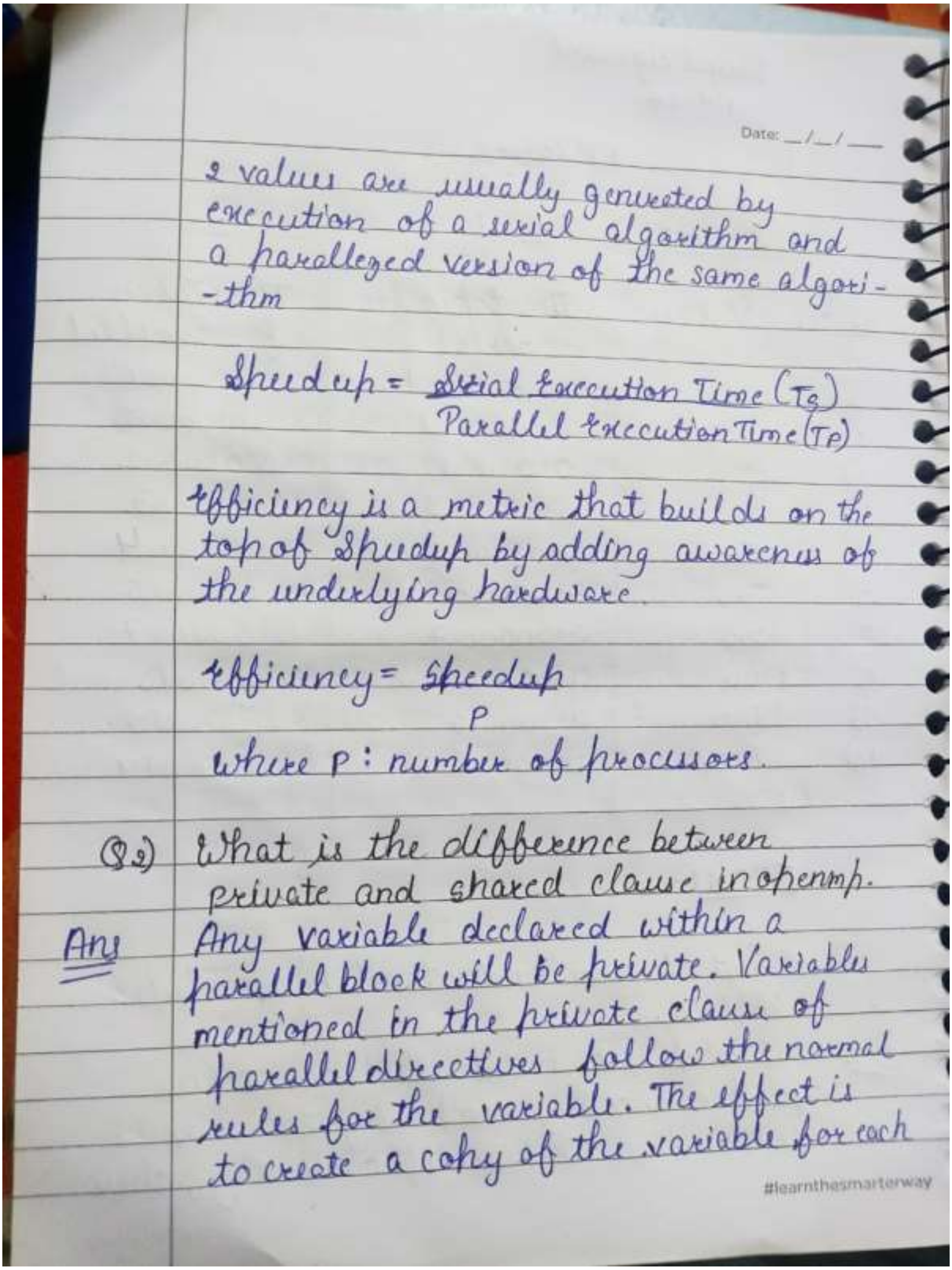
Observation Table:

Sr. No	Values	Time taken for Serial	Time Taken for Parallel.
1)	100	0.000001	0.000001
2)	200	0.000001	0.000002
3)	300	0.000002	0.000002
4)	400	0.000003	0.000003
5)	500	0.000003	0.000004
6)	800	0.000006	0.000006
7)	1000	0.000007	0.000007
8)	1200	0.000008	0.000010
9)	1400	0.000010	0.000010
10)	1500	0.000010	0.000014

FAQ's:-

1) What do you understand by Speedup & Efficiency?

Ans Speedup is a metric that quantifies performance by comparing two elapsed time values. In parallel computing, these



2 values are usually generated by execution of a serial algorithm and a paralleged version of the same algorithm

$$\text{Speedup} = \frac{\text{Serial Execution Time } (T_S)}{\text{Parallel Execution Time } (T_P)}$$

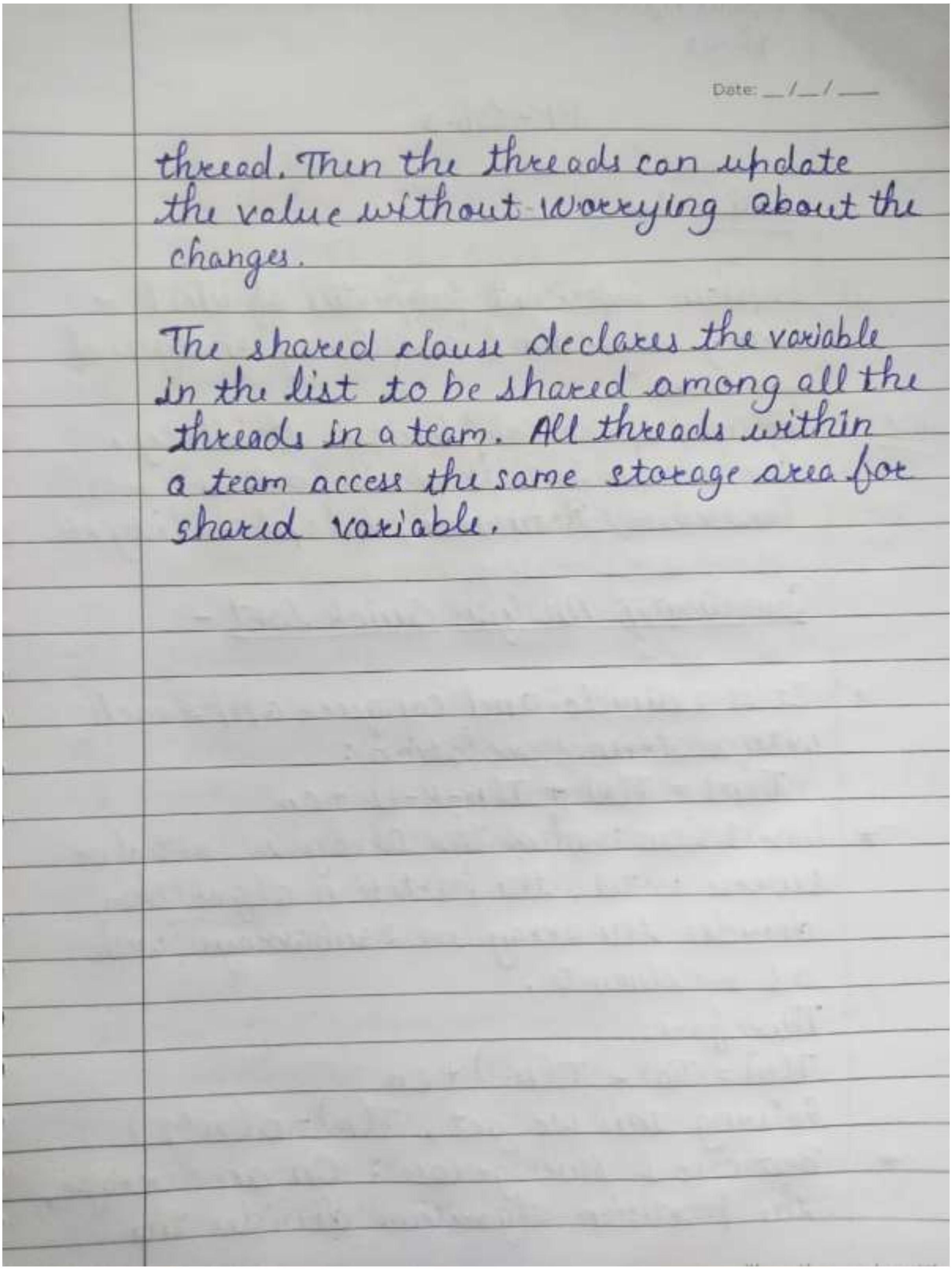
Efficiency is a metric that builds on the top of Speedup by adding awareness of the underlying hardware.

$$\text{Efficiency} = \frac{\text{Speedup}}{P}$$

where P : number of processors.

(Q2) What is the difference between private and shared clause in openmp.

Ans Any variable declared within a parallel block will be private. Variables mentioned in the private clause of parallel directives follow the normal rules for the variable. The effect is to create a copy of the variable for each



Date: __/__/____

thread. Then the threads can update the value without worrying about the changes.

The shared clause declares the variable in the list to be shared among all the threads in a team. All threads within a team access the same storage area for shared variable.