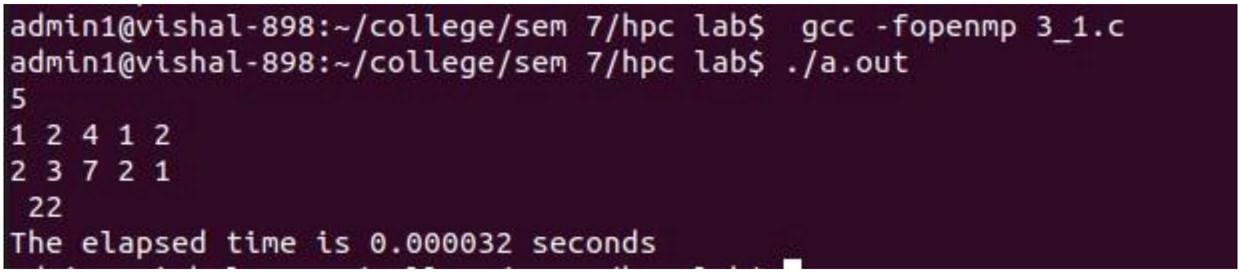
**SEM - VII - 2022-23**
**High-Performance Computing Lab**
**Assignment 3**
**Name: Borse Vishal Subhash**
**PRN: 2019BTECS00066**

Q1: Analyse and implement a Parallel code for the below program using OpenMP.

```
// C Program to find the minimum scalar product of two vectors (dot product)
#include<stdio.h>
int sort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++) for (j = 0;
        j < n - i - 1; j++) if (arr[j] >
        arr[j + 1])
            { int temp = arr[j]; arr[j]
                = arr[j + 1]; arr[j
                + 1] = temp;
            } } int
sort_des(int arr[], int n)
{
    int i, j;


    for (i = 0; i < n; ++i)
    { for (j = i + 1; j < n; ++j)
        { if (arr[i] < arr[j])
            { int a = arr[i]; arr[i]
                = arr[j];
                arr[j] = a;
            }
        }
    }
}


int main()
{
//fill the code; int n; scanf("
    % d", &n); int
    arr1[n], arr2[n]; int i;
    for (i = 0; i < n ; i++)
    { scanf(" % d", &arr1[i]);
    }
    for (i = 0; i < n ; i++)
    { scanf(" % d", &arr2[i]);
```

```
        } sort(arr1, n);
        sort_des(arr2, n);
        int sum = 0; for (i =
        0; i < n ; i++)
        { sum = sum + (arr1[i] * arr2[i]);
        }
        printf("% d", sum);

        return 0;
}
```



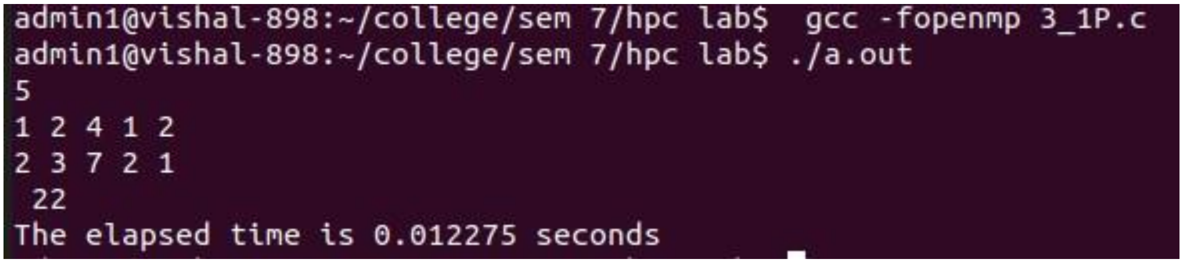
```
admin1@vishal-898:~/college/sem 7/hpc lab$  gcc -fopenmp 3_1.c
admin1@vishal-898:~/college/sem 7/hpc lab$ ./a.out
5
1 2 4 1 2
2 3 7 2 1
 22
The elapsed time is 0.000032 seconds
```

Parallel

```
// C Program to find the minimum scalar product of two vectors (dot product)
#include<stdio.h>
#include <time.h>
int n;
int sort(int arr[])
{
        int i, j; for (i = 0; i < n;
        i++) { int turn = i % 2;
                #pragma omp parallel for
                for (j = turn; j < n - 1; j += 2)
                        if (arr[j] > arr[j + 1])
                        { int temp = arr[j]; arr[j]
                                = arr[j + 1]; arr[j
                                + 1] = temp;
                        }
```

```
    } } int
sort_des(int arr[])
{
    int i, j;
    for (i = 0; i < n; ++i)
    { int turn = i % 2;
        #pragma omp parallel for
        for (j = turn; j < n - 1; j += 2)
        {


            if (arr[j] < arr[j + 1])
            { int temp = arr[j]; arr[j]
                = arr[j + 1]; arr[j
                + 1] = temp;
            }
        }
    }
}

int main()
{
//fill the code;

    scanf(" %d", &n);
    int arr1[n], arr2[n];
    int i; for (i = 0; i < n
    ; i++)
    { scanf("%d", &arr1[i]);
    } for (i = 0; i < n ;
    i++)
    { scanf(" %d", &arr2[i]);
    }

    double time_spent = 0.0;

    clock_t begin = clock();

    sort(arr1);
    sort_des(arr2); int
```

```
        sum = 0; for (i = 0; i
        < n ; i++)
        { sum = sum + (arr1[i] * arr2[i]);
        }
        printf("% d", sum);

        clock_t end = clock(); time_spent += (double)(end - begin) /
        CLOCKS_PER_SEC; printf("\nThe elapsed time is %f
        seconds\n", time_spent);

        return 0;
}
```



```
admin1@vishal-898:~/college/sem 7/hpc lab$  gcc -fopenmp 3_1P.c
admin1@vishal-898:~/college/sem 7/hpc lab$ ./a.out
5
1 2 4 1 2
2 3 7 2 1
 22
The elapsed time is 0.012275 seconds
```

Q2: Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculating the execution time or use GPROF)
i. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads. ii. Explain whether or not the scaling behavior is as expected.

Serial :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define N 100
void add(int** a, int** b, int** c) { for (int i
    = 0; i < N; i++) { for (int j = 0; j <
    N; j++) { c[i][j] = a[i][j] + b[i][j];
        }
    }
}
void getMatrix(int** a, int num) { for
    (int i = 0; i < N; i++) { for (int j =
    0; j < N; j++) { a[i][j] = num;
        }
    }
} void display(int** a) { for (int i = 0; i <
N; i++) { for (int j = 0; j < N; j++) {
printf("%d ", a[i][j]);
        }
        printf("\n");
    }
} int main() { int** a; int** b; int** c; a =
malloc(sizeof(int*) * N); b =
malloc(sizeof(int*) * N); c =
malloc(sizeof(int*) * N); for (int i = 0; i <
N; i++) { a[i] = malloc(sizeof(int) * N); b[i]
= malloc(sizeof(int) * N); c[i] =
malloc(sizeof(int) * N);
    } getMatrix(a,
    1);
    getMatrix(b,
    1);


    double time_spent = 0.0;
    clock_t begin = clock();


    add(a, b, c);
```

```
          clock_t end = clock(); time_spent += (double)(end - begin) /
          CLOCKS_PER_SEC; printf("\nThe elapsed time is %f
          seconds\n", time_spent);


}
```

| N | 250 | 500 | 750 | 1000 | 2000 |
|---|---|---|---|---|---|
| Time | 0.001 | 0.0015 | 0.00251 | 0.0031 | 0.0150 |

**Parallel :**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define N 100
void add(int** a, int** b, int** c) {
          #pragma omp parallel for
          for (int i = 0; i < N; i++) { for (int j
                    = 0; j < N; j++) { c[i][j] =
                    a[i][j] + b[i][j];
                    }
          }
} void getMatrix(int** a, int num) { for
(int i = 0; i < N; i++) { for (int j = 0; j <
N; j++) { a[i][j] = num;
                    }
          }
} void display(int** a) { for (int i = 0; i <
N; i++) { for (int j = 0; j < N; j++) {
printf("%d ", a[i][j]);
                }
                printf("\n");
        }
}
int main() { int** a; int** b; int** c; a =
        malloc(sizeof(int*) * N); b =
        malloc(sizeof(int*) * N); c =
        malloc(sizeof(int*) * N); for (int i =
        0; i < N; i++) { a[i] =
        malloc(sizeof(int) * N); b[i] =
        malloc(sizeof(int) * N); c[i] =
        malloc(sizeof(int) * N);
        } getMatrix(a,
        1);
        getMatrix(b,
        1);


        //omp_set_num_threads(2);

        double time_spent = 0.0;
        clock_t begin = clock();


        add(a, b, c);


        clock_t end = clock(); time_spent += (double)(end - begin) /
        CLOCKS_PER_SEC; printf("\nThe elapsed time is %f
        seconds\n", time_spent);


}
```

| N | 250 | 500 | 750 | 1000 | 2000 |
|---|---|---|---|---|---|
| Time | 0.002 | 0.001 | 0.00254 | 0.0029 | 0.0034 |

Q3. For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:

i. Use the STATIC schedule and set the loop iteration chunk size to varioussizes when changing the size of your matrix. Analyze the speedup.

ii. Use the DYNAMIC schedule and set the loop iteration chunk size to

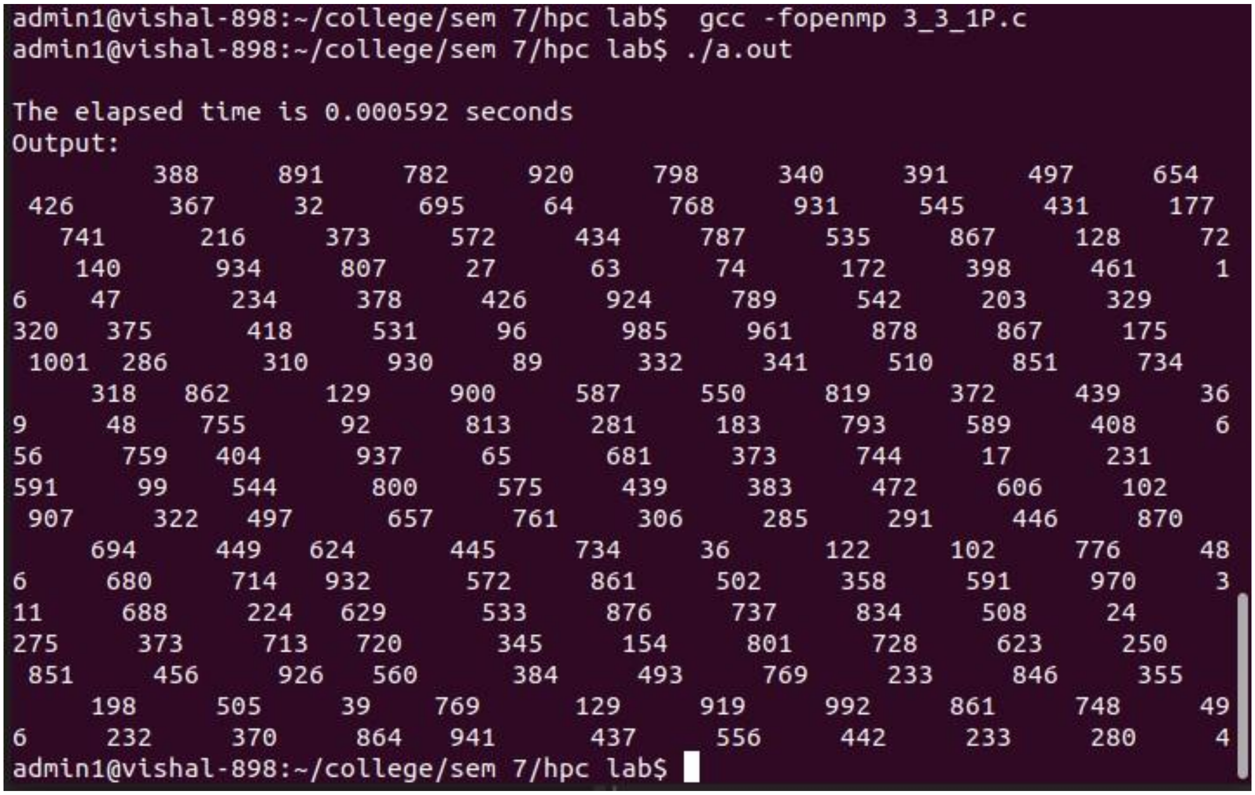
varioussizes when changing the size of your matrix. Analyze the speedup.

iii. Demonstrate the use of nowait clause. i)

i) Static

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

int main() { int
    n = 200; int
    arr[n];

    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;
    }

    int ans[n];
    int val = 5;

    double time_spent = 0.0;
    clock_t begin = clock();



    #pragma omp parallel for schedule(static,20) shared(arr, ans,val)
    for (int i = 0; i < n; i++)
    { ans[i] = arr[i] + val;
    }
    clock_t end = clock(); time_spent += (double)(end - begin) /
    CLOCKS_PER_SEC; printf("\nThe elapsed time is %f
    seconds\n", time_spent);


    printf("Output: \n"); for
    (int i = 0; i < n; i++) {
        printf("\t %d", ans[i]);
    }

    return 0;
}
```

| Chunk Size | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| Time | .0019 | .0010 | 0.0009 | 0.0009 |



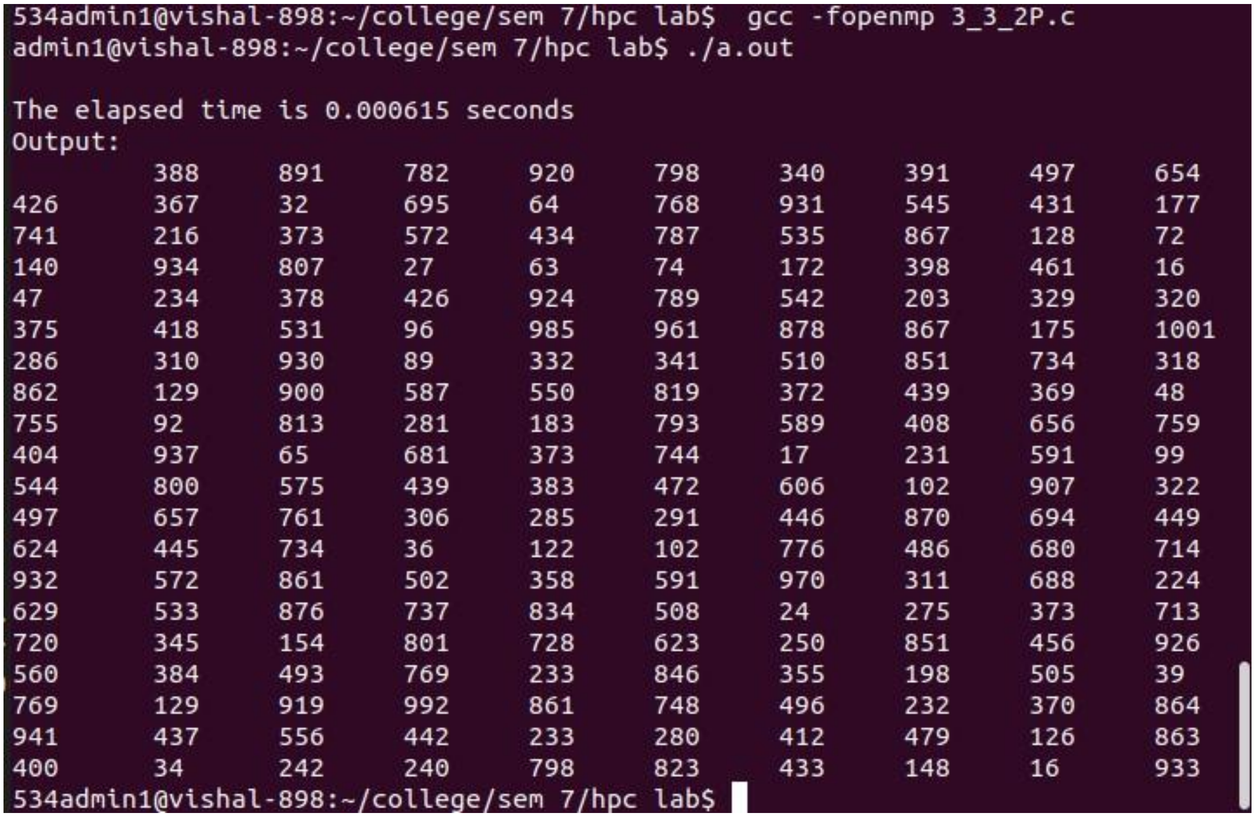
```
admin1@vishal-898:~/college/sem 7/hpc lab$  gcc -fopenmp 3_3_1P.c
admin1@vishal-898:~/college/sem 7/hpc lab$ ./a.out

The elapsed time is 0.000592 seconds
Output:
         388     891     782     920     798     340     391     497     654
 426      367     32      695     64      768     931     545     431     177
   741      216     373     572     434     787     535     867     128     72
    140      934     807     27      63      74      172     398     461     1
6    47       234     378     426     924     789     542     203     329
320   375      418     531     96      985     961     878     867     175
 1001  286      310     930     89      332     341     510     851     734
     318   862      129     900     587     550     819     372     439     36
9     48    755      92      813     281     183     793     589     408     6
56     759   404      937     65      681     373     744     17      231
591     99    544      800     575     439     383     472     606     102
 907     322   497      657     761     306     285     291     446     870
     694     449   624      445     734     36      122     102     776     48
6     680     714   932      572     861     502     358     591     970     3
11     688     224   629      533     876     737     834     508     24
275     373     713   720      345     154     801     728     623     250
 851     456     926   560      384     493     769     233     846     355
     198     505     39    769      129     919     992     861     748     49
6     232     370     864   941      437     556     442     233     280     4
admin1@vishal-898:~/college/sem 7/hpc lab$ 
```

ii) Dynamic

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

```
int main() { int
    n = 200; int
    arr[n];

    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;
    }

    int ans[n];
    int val = 5;

    double time_spent = 0.0;
    clock_t begin = clock();



    #pragma omp parallel for schedule(dynamic,1) shared(arr, ans,val)
    for (int i = 0; i < n; i++)
    { ans[i] = arr[i] + val;
    }

    clock_t end = clock(); time_spent += (double)(end - begin) /
    CLOCKS_PER_SEC; printf("\nThe elapsed time is %f
    seconds\n", time_spent);


    printf("Output: \n"); for
    (int i = 0; i < n; i++) {
    printf("\t %d", ans[i]);
    }

    return 0;
}
```

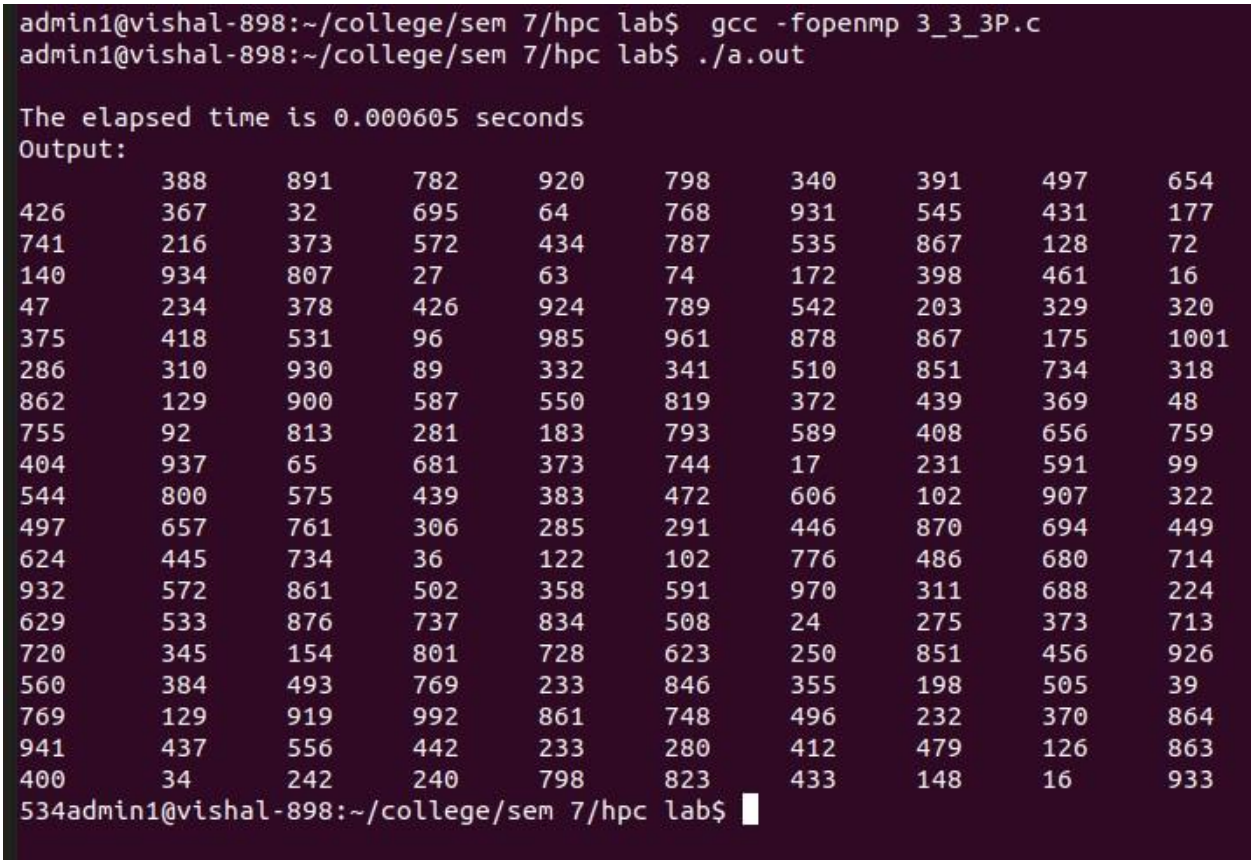
| Chunk Size | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| Time | 0.00187 | 0.0017 | 0.0015 | 0.0012 |



```
534admin1@vishal-898:~/college/sem 7/hpc lab$  gcc -fopenmp 3_3_2P.c
admin1@vishal-898:~/college/sem 7/hpc lab$ ./a.out

The elapsed time is 0.000615 seconds
Output:
         388     891     782     920     798     340     391     497     654
426      367     32      695     64      768     931     545     431     177
741      216     373     572     434     787     535     867     128     72
140      934     807     27      63      74      172     398     461     16
47       234     378     426     924     789     542     203     329     320
375      418     531     96      985     961     878     867     175     1001
286      310     930     89      332     341     510     851     734     318
862      129     900     587     550     819     372     439     369     48
755      92      813     281     183     793     589     408     656     759
404      937     65      681     373     744     17      231     591     99
544      800     575     439     383     472     606     102     907     322
497      657     761     306     285     291     446     870     694     449
624      445     734     36      122     102     776     486     680     714
932      572     861     502     358     591     970     311     688     224
629      533     876     737     834     508     24      275     373     713
720      345     154     801     728     623     250     851     456     926
560      384     493     769     233     846     355     198     505     39
769      129     919     992     861     748     496     232     370     864
941      437     556     442     233     280     412     479     126     863
400      34      242     240     798     823     433     148     16      933
534admin1@vishal-898:~/college/sem 7/hpc lab$ 
```

iii) Nowait

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

int main() { int n = 200; int

   arr[n]; for (int i = 0; i <
   n; i++) { arr[i] = rand()

   % 1000;

   }

   int ans[n];
   int val = 5;

   double time_spent = 0.0;
   clock_t begin = clock();



   #pragma omp parallel
   {
       #pragma omp for nowait
       for (int i = 0; i < n; i++)
       { ans[i] = arr[i] + val;
       }
   }

   clock_t end = clock(); time_spent += (double)(end - begin) /
   CLOCKS_PER_SEC; printf("\nThe elapsed time is %f
   seconds\n", time_spent);


   printf("Output: \n"); for
   (int i = 0; i < n; i++) {
       printf("\t %d", ans[i]);
   }

   return 0;
}
```



```
admin1@vishal-898:~/college/sem 7/hpc lab$  gcc -fopenmp 3_3_3P.c
admin1@vishal-898:~/college/sem 7/hpc lab$ ./a.out

The elapsed time is 0.000605 seconds
Output:
         388     891     782     920     798     340     391     497     654
426      367     32      695     64      768     931     545     431     177
741      216     373     572     434     787     535     867     128     72
140      934     807     27      63      74      172     398     461     16
47       234     378     426     924     789     542     203     329     320
375      418     531     96      985     961     878     867     175     1001
286      310     930     89      332     341     510     851     734     318
862      129     900     587     550     819     372     439     369     48
755      92      813     281     183     793     589     408     656     759
404      937     65      681     373     744     17      231     591     99
544      800     575     439     383     472     606     102     907     322
497      657     761     306     285     291     446     870     694     449
624      445     734     36      122     102     776     486     680     714
932      572     861     502     358     591     970     311     688     224
629      533     876     737     834     508     24      275     373     713
720      345     154     801     728     623     250     851     456     926
560      384     493     769     233     846     355     198     505     39
769      129     919     992     861     748     496     232     370     864
941      437     556     442     233     280     412     479     126     863
400      34      242     240     798     823     433     148     16      933
534admin1@vishal-898:~/college/sem 7/hpc lab$ █
```