Class: Final Year (Computer Science and Engineering)

Year: 2021-22 Semester: 1

Course: High Performance Computing Lab

Practical No. 3

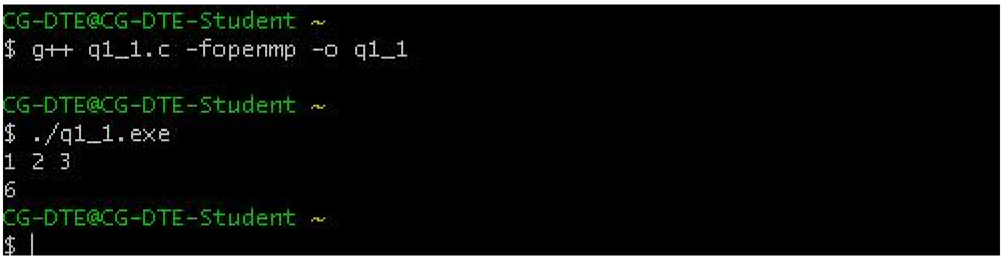
Exam Seat No:2019BTECS00205

Name - Shweta Nandkumar Arbune

Problem Statement 1: Analyse and implement a Parallel code for below program using OpenMP

```
#include<stdio.h>
int sort(int arr[], int n)
{
int i, j;
#pragma omp parallel private(j)
{
#pragma omp for schedule (dynamic)
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;
#pragma omp parallel private(j)
{
#pragma omp for schedule (dynamic)
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;
#pragma omp parallel for reduction (+:sum)
for(i = 0; i < n ; i++)
{
sum = sum + (arr1[i] * arr2[i]);
}

printf("%d",sum);
return 0;
}
```

ScreenShoot 1:



```
CG-DTE@CG-DTE-Student ~
$ g++ q1_1.c -fopenmp -o q1_1

CG-DTE@CG-DTE-Student ~
$ ./q1_1.exe
1 2 3
6
CG-DTE@CG-DTE-Student ~
$ |
```

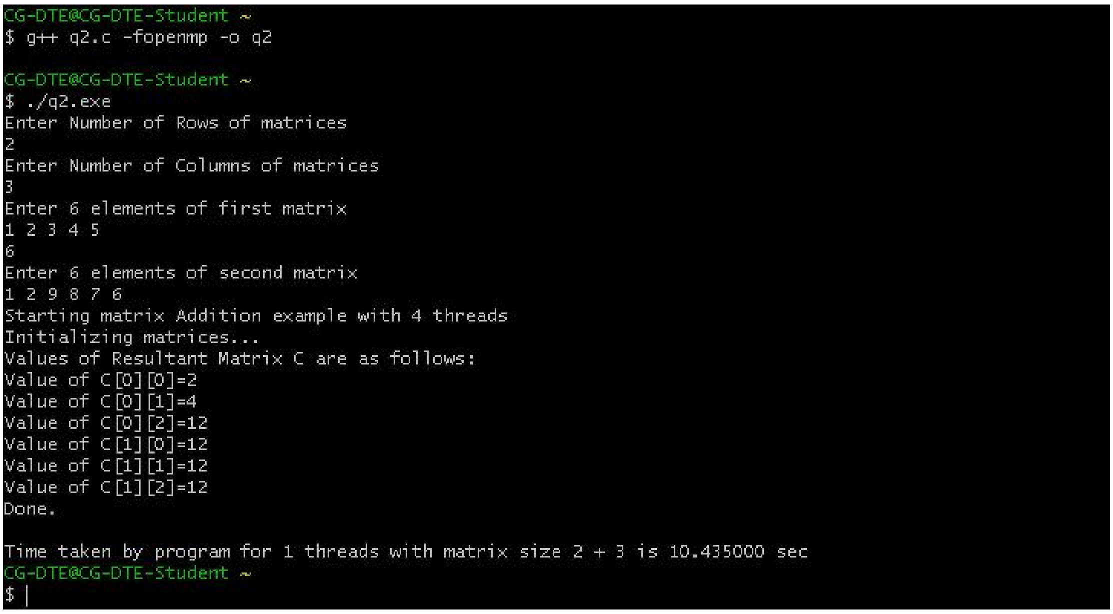
Problem Statement 2: Write OpenMPcode for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)i.For each matrix size, change the number of threads from 2,4,8.,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

.

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#include<time.h>
#define tot_threads 8
int main(int argc, char *argv[])
{
clock_t t;
t = clock();
int tid;
int i,j;
int rows,cols;

printf("Enter Number of Rows of matrices\n");
scanf("%d",&rows);
printf("Enter Number of Columns of matrices\n");
scanf("%d",&cols);

int a[rows][cols];
int b[rows][cols];
int c[rows][cols];

int *d,*e,*f;
int nthreads;

printf("Enter %d elements of first matrix\n",rows*cols);
for(i=0;i<rows;i++)
  for(j=0;j<cols;j++)
    {
      scanf("%d",&a[i][j]);
    }

printf("Enter %d elements of second matrix\n",rows*cols);
for(i=0;i<rows;i++)
  for(j=0;j<cols;j++)
    {
      scanf("%d",&b[i][j]);
    }

d=(int *)malloc(sizeof(int)*rows*cols);
e=(int *)malloc(sizeof(int)*rows*cols);
f=(int *)malloc(sizeof(int)*rows*cols);

d=(int *)a;
e=(int *)b;
f=(int *)c;
#pragma omp parallel shared(d,e,f,nthreads) private(tid,i,j) num_threads(8)
```

```
        {

                tid = omp_get_thread_num();
                if (tid == 0)
                {
                        nthreads = omp_get_num_threads();
                        printf("Starting matrix Addition example with %d
threads\n",nthreads);
                        printf("Initializing matrices...\n");
                }
}


#pragma omp parallel num_threads(rows*cols)
  {
   tid=omp_get_thread_num();
   f[tid]=d[tid]+e[tid];
  }

printf("Values of Resultant Matrix C are as follows:\n");

for(i=0;i<rows;i++)
  for(j=0;j<cols;j++)
    {
      printf("Value of C[%d][%d]=%d\n",i,j,c[i][j]);
    }
printf ("Done.\n");
        t = clock() - t;
        double time_taken = ((double)t)/CLOCKS_PER_SEC;

   printf("\nTime taken by program for %d threads with matrix size %d + %d is %f
sec",tot_threads,rows,cols,time_taken);

return 0;

}
```

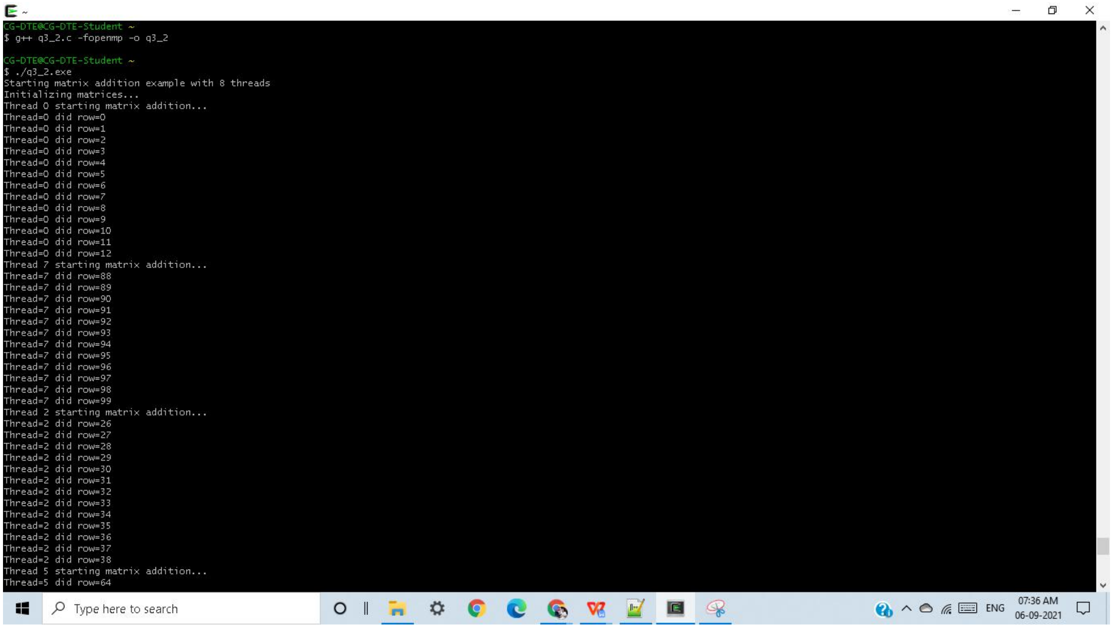
Screenshoot 1:



```
CG-DTE@CG-DTE-Student ~
$ g++ q2.c -fopenmp -o q2

CG-DTE@CG-DTE-Student ~
$ ./q2.exe
Enter Number of Rows of matrices
2
Enter Number of Columns of matrices
3
Enter 6 elements of first matrix
1 2 3 4 5
6
Enter 6 elements of second matrix
1 2 9 8 7 6
Starting matrix Addition example with 4 threads
Initializing matrices...
Values of Resultant Matrix C are as follows:
Value of C[0][0]=2
Value of C[0][1]=4
Value of C[0][2]=12
Value of C[1][0]=12
Value of C[1][1]=12
Value of C[1][2]=12
Done.

Time taken by program for 1 threads with matrix size 2 + 3 is 10.435000 sec
CG-DTE@CG-DTE-Student ~
$ |
```

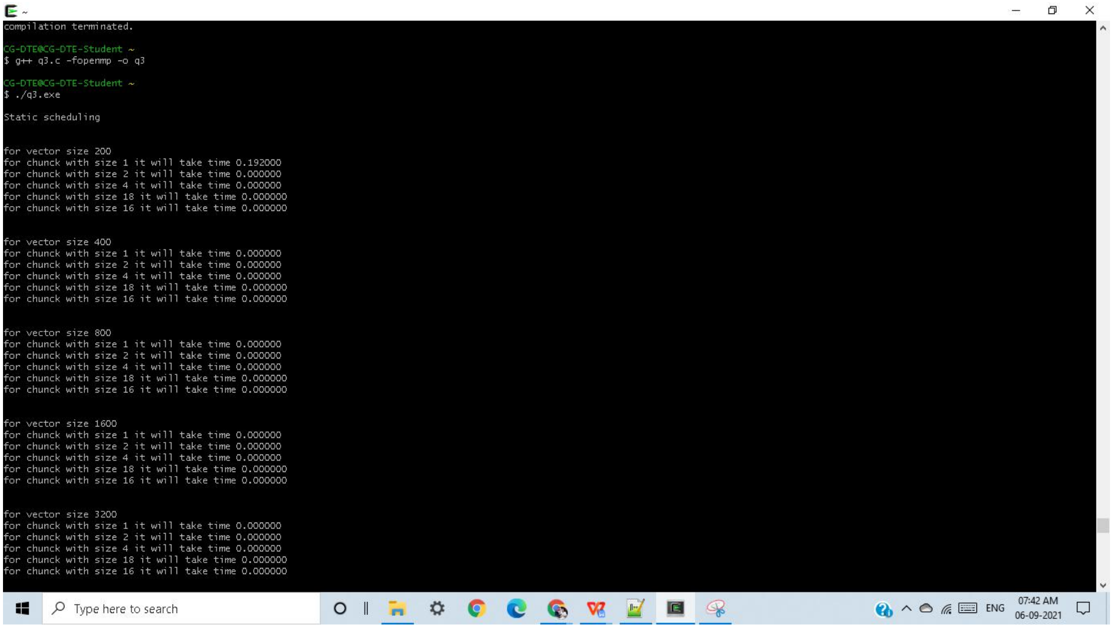
```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

#define N 100

int main (int argc, char *argv[])
{
int  tid, nthreads, i, j;
double a[N][N], b[N][N], c[N][N];
omp_set_num_threads(8);
 double time = omp_get_wtime();
#pragma omp parallel shared(a,b,c,nthreads) private(tid,i,j)
  {

  tid = omp_get_thread_num();
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Starting matrix addition example with %d threads\n",nthreads);
    printf("Initializing matrices...\n");
    }
  /* Initialize matrices */
  #pragma omp for
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      a[i][j]= i+j;

  #pragma omp for
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
```

```
      b[i][j]= i+j;



  printf("Thread %d starting matrix addition...\n",tid);

  #pragma omp for
  for (i=0; i<N; i++)
    {
    printf("Thread=%d did row=%d\n",tid,i);
    for(j=0; j<N; j++)
        c[i][j] = a[i][j] + b[i][j];
    }
  }
printf ("Done In %f Seconds",omp_get_wtime() - time);
printf("\n Using %d Threads",omp_get_max_threads());

return(0);
}
```

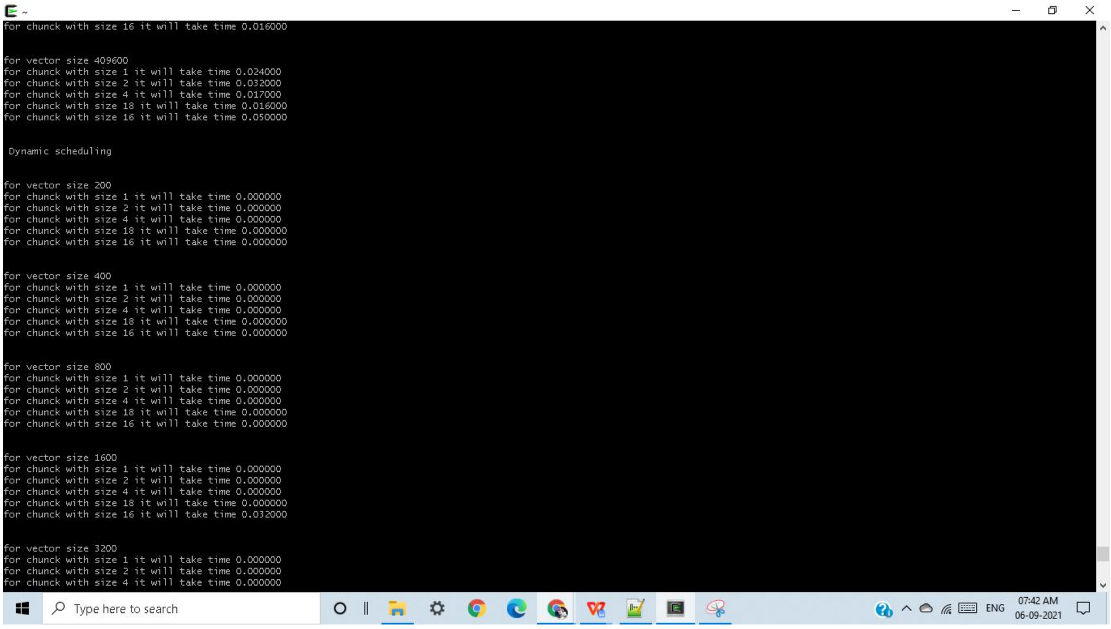


```
CG-DTE@CG-DTE-Student ~
$ g++ q3_2.c -fopenmp -o q3_2

CG-DTE@CG-DTE-Student ~
$ ./q3_2.exe
Starting matrix addition example with 8 threads
Initializing matrices...
Thread 0 starting matrix addition...
Thread=0 did row=0
Thread=0 did row=1
Thread=0 did row=2
Thread=0 did row=3
Thread=0 did row=4
Thread=0 did row=5
Thread=0 did row=6
Thread=0 did row=7
Thread=0 did row=8
Thread=0 did row=9
Thread=0 did row=10
Thread=0 did row=11
Thread=0 did row=12
Thread 7 starting matrix addition...
Thread=7 did row=88
Thread=7 did row=89
Thread=7 did row=90
Thread=7 did row=91
Thread=7 did row=92
Thread=7 did row=93
Thread=7 did row=94
Thread=7 did row=95
Thread=7 did row=96
Thread=7 did row=97
Thread=7 did row=98
Thread=7 did row=99
Thread 2 starting matrix addition...
Thread=2 did row=26
Thread=2 did row=27
Thread=2 did row=28
Thread=2 did row=29
Thread=2 did row=30
Thread=2 did row=31
Thread=2 did row=32
Thread=2 did row=33
Thread=2 did row=34
Thread=2 did row=35
Thread=2 did row=36
Thread=2 did row=37
Thread=2 did row=38
Thread 5 starting matrix addition...
Thread=5 did row=64
```

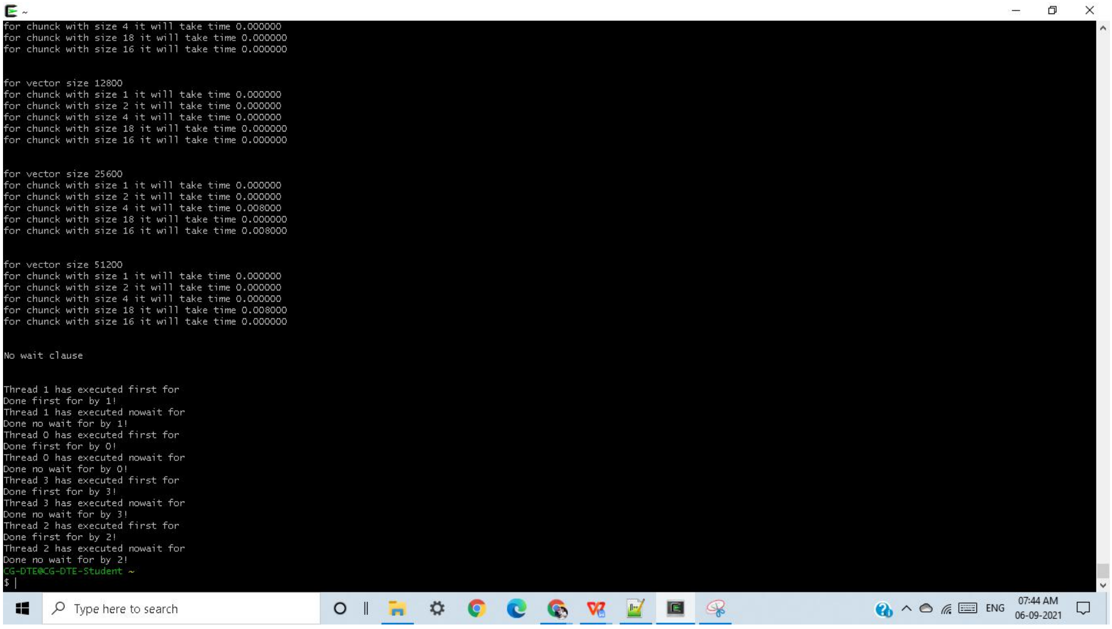
Q3. For 1D Vector (size=200) and scalar addition, Write a OpenMPcode with the following:i.Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.ii.Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.iii.Demonstrate the use of nowait clause.



```
compilation terminated.

CG-DTE@CG-DTE-Student ~
$ g++ q3.c -fopenmp -o q3

CG-DTE@CG-DTE-Student ~
$ ./q3.exe

Static scheduling


for vector size 200
for chunck with size 1 it will take time 0.192000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 400
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 800
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 1600
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 3200
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000
```



```
for chunck with size 16 it will take time 0.016000


for vector size 409600
for chunck with size 1 it will take time 0.024000
for chunck with size 2 it will take time 0.032000
for chunck with size 4 it will take time 0.017000
for chunck with size 18 it will take time 0.016000
for chunck with size 16 it will take time 0.050000


 Dynamic scheduling


for vector size 200
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 400
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 800
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 1600
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.032000


for vector size 3200
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
```



```
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 12800
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.000000


for vector size 25600
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.008000
for chunck with size 18 it will take time 0.000000
for chunck with size 16 it will take time 0.008000


for vector size 51200
for chunck with size 1 it will take time 0.000000
for chunck with size 2 it will take time 0.000000
for chunck with size 4 it will take time 0.000000
for chunck with size 18 it will take time 0.008000
for chunck with size 16 it will take time 0.000000


No wait clause


Thread 1 has executed first for
Done first for by 1!
Thread 1 has executed nowait for
Done no wait for by 1!
Thread 0 has executed first for
Done first for by 0!
Thread 0 has executed nowait for
Done no wait for by 0!
Thread 3 has executed first for
Done first for by 3!
Thread 3 has executed nowait for
Done no wait for by 3!
Thread 2 has executed first for
Done first for by 2!
Thread 2 has executed nowait for
Done no wait for by 2!
CG-DTE@CG-DTE-Student ~
$ |
```

The nowait Clause :

If there are multiple independent loops within a parallel region, you can use the nowait clause to avoid the implied barrier at the end of the loop construct, as follows:

Example nowait.

```
#include <math.h>
void nowait_example(int n, int m, float *a, float *b, float *y, float *z)
{
int i;
 #pragma omp parallel
 {
 #pragma omp for nowait
 for (i=1; i<n; i++)
 b[i] = (a[i] + a[i-1]) / 2.0;
  #pragma omp for nowait
 for (i=0; i<m; i++)
 y[i] = sqrt(z[i]);
  }
 }
```

Github Link: https://github.com/shwetaarbune/HPC-LAB