**Class:** Final Year (Computer Science and Engineering)

**Year:** 2022-23 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 3**

**Exam Seat No: 2019BTECS00064**

**Name – Kunal Santosh Kadam**

**Title of practical: Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses**

**Problem Statement 1:**

Analyse and implement a Parallel code for 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]);

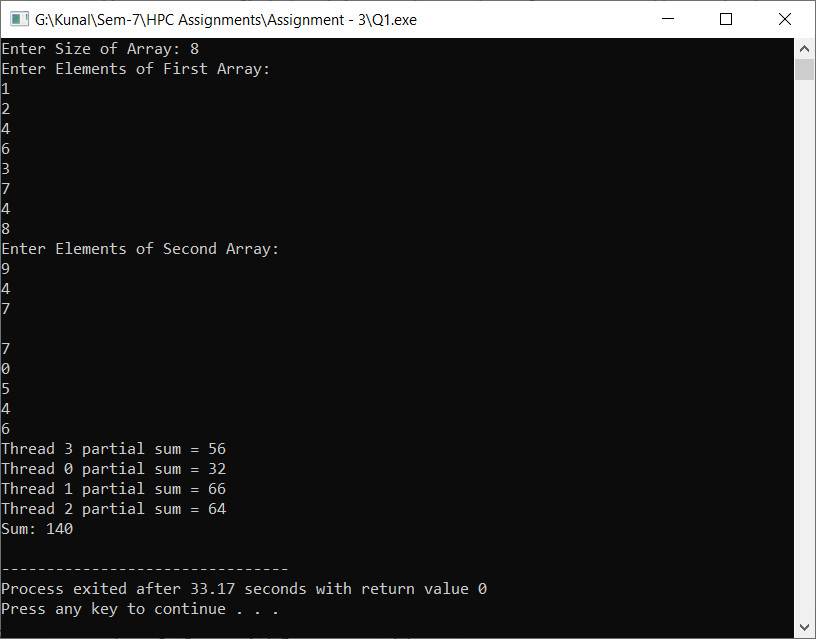
}

printf(“%d”,sum);

return 0;

}

**Screenshot #:**



**Information #:**

// C Program to find the minimum scalar product of two vectors (dot product)

#include<bits/stdc++.h>

#include <omp.h>

using namespace std;

int sort(int arr[], int n)

{

int i, j;

#pragma omp parallel shared(arr) 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 shared(arr) 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 i,tid,n,psum;

int threads = 4;

cout<<"Enter Size of Array: ";

cin>>n;

int arr1[n], arr2[n];

cout<<"Enter Elements of First Array:\n";

for(i = 0; i < n ; i++)

{

cin>>arr1[i];

}

cout<<"Enter Elements of Second Array:\n";

for(i = 0; i < n ; i++)

{

cin>>arr2[i];

}

sort(arr1, n);

sort\_des(arr2, n);

int sum = 0;

#pragma omp parallel private(i,tid,psum) num\_threads(threads)

{

psum=0;

tid = omp\_get\_thread\_num();

#pragma omp for reduction(+:sum)

for(int i=0; i<n; i++)

{

sum += arr1[i] \* arr2[i];

psum+=sum;

}

printf("Thread %d partial sum = %d\n",tid,psum);

}

cout<<"Sum: "<<sum<<endl;

return 0;

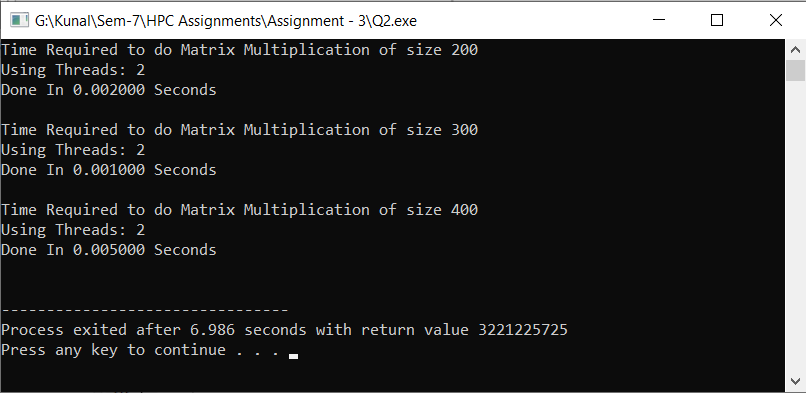
}

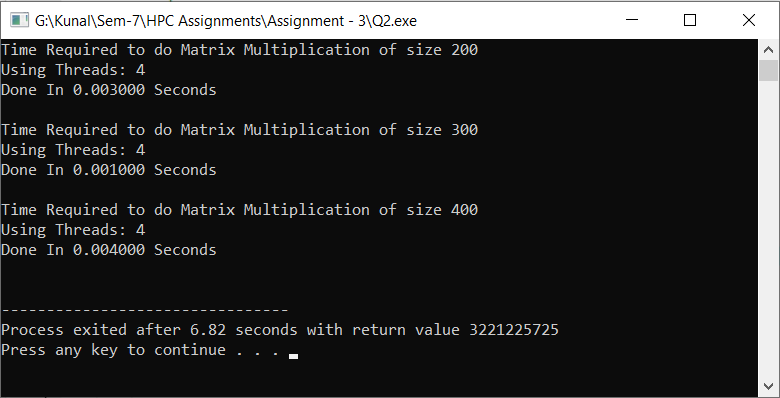
**Problem Statement 2:**

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 calculate the execution time or use GPROF)

1. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.
2. Explain whether or not the scaling behaviour is as expected.

**Screenshot #:**





**Information #:**

#include <bits/stdc++.h>

#include <omp.h>

using namespace std;

int main()

{

int tid, nthreads , i, j;

int n=100;

while(1){

if(n==500)

break;

else

n+=100;

nthreads=4;

int a[n][n], b[n][n], c[n][n];

int index = 0;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

a[i][j] = b[i][j] = (i+j);

}

}

printf("Time Required to do Matrix Multiplication of size %d\nUsing Threads: %d",n,nthreads);

double time = omp\_get\_wtime();

#pragma omp parallel shared(a, b, c, nthreads) private(tid, i, j) num\_threads(nthreads)

{

# 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];

}

}

}

printf("\nDone In %f Seconds\n\n", omp\_get\_wtime() - time);

}

return 0;

}

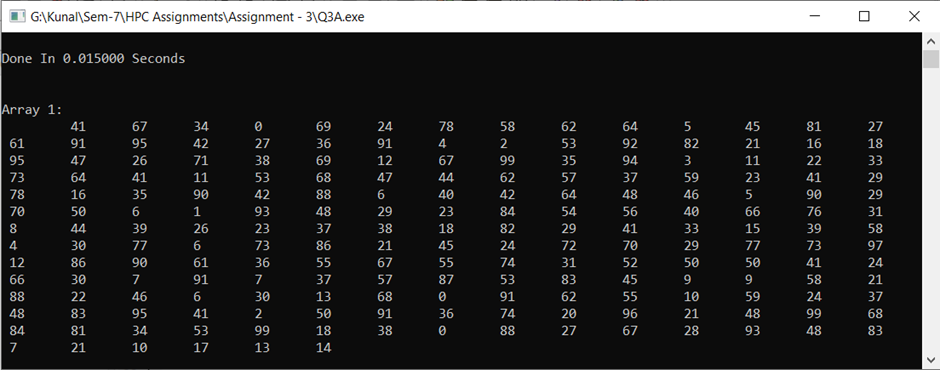
**Problem Statement 3:**

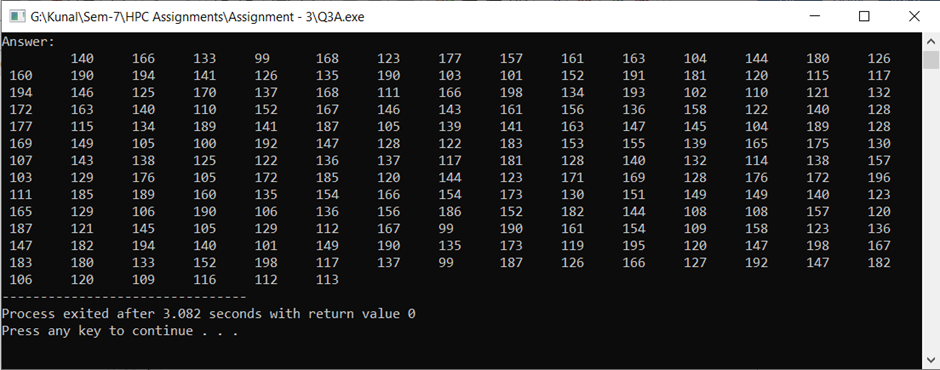
For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:

1. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.
2. Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.
3. Demonstrate the use of nowait clause

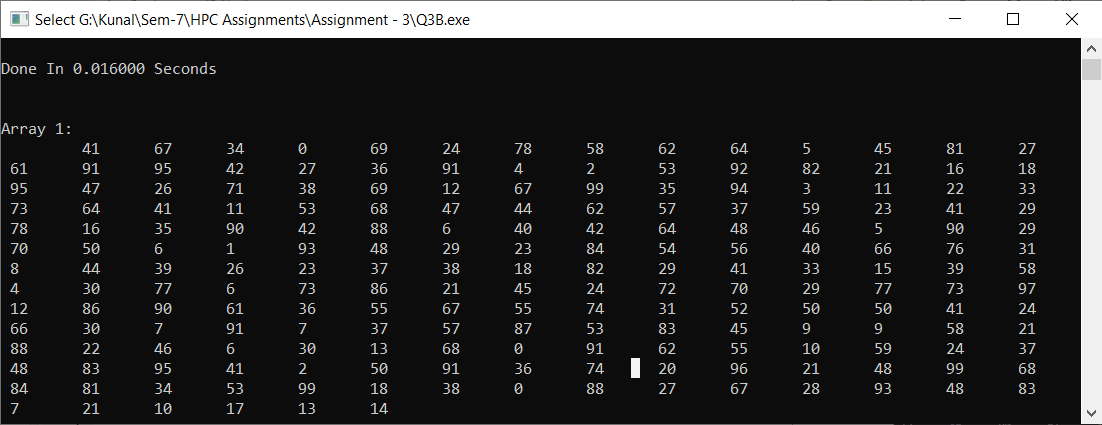
**Screenshot #:**

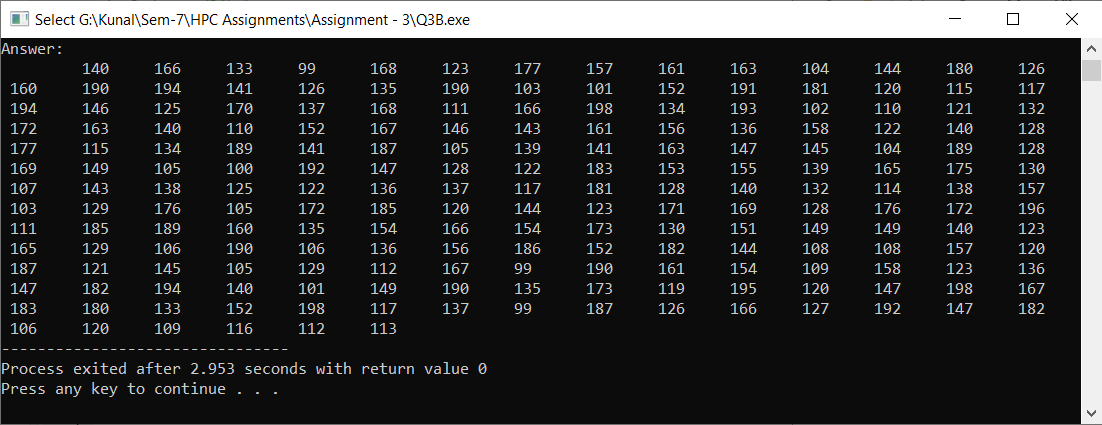
**Use of Static Schedule**



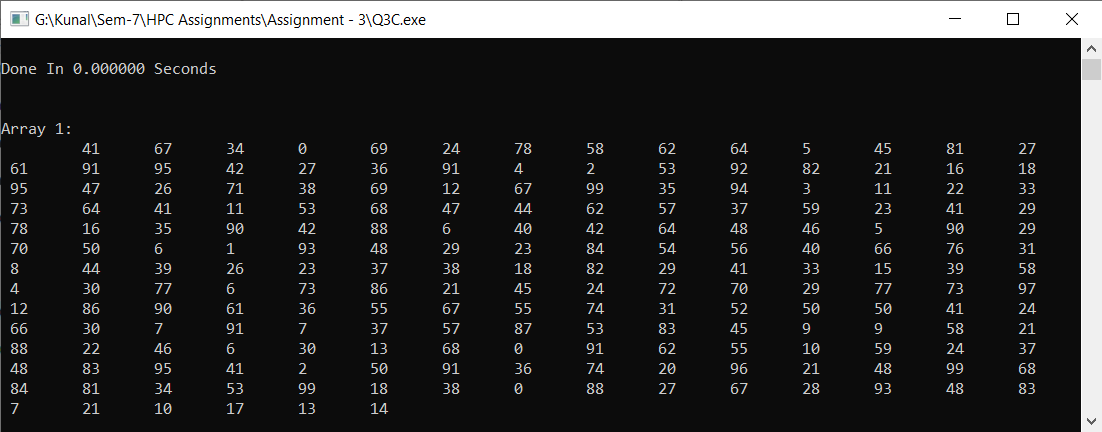


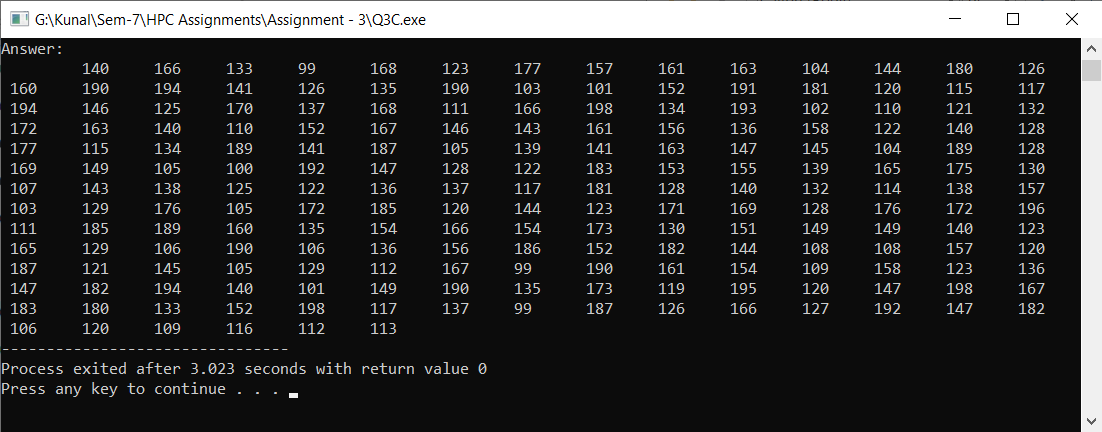
**Use of Dynamic Schedule**





**Use of Nowait Clause**





**Information #:**

**Use of Static Schedule**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(){

int n = 200, i ,j=99;

int arr1[n], answer[n];

for(i = 0; i < n; i++){

arr1[i] = rand()%100;

}

double time = omp\_get\_wtime();

#pragma omp parallel for schedule(static,20) shared(arr1, answer,j) private(i)

for(i = 0; i < n; i++)

{

answer[i] = arr1[i] + j;

}

printf("\nDone In %f Seconds\n\n", omp\_get\_wtime() - time);

printf("\nArray 1: \n");

for(i = 0; i < n; i++){

printf("\t %d", arr1[i]);

}

printf("\nAnswer: \n");

for(i = 0; i < n; i++){

printf("\t %d", answer[i]);

}

return 0;

}

**Use of Dynamic Schedule**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(){

int n = 200, i ,j=99;

int arr1[n], answer[n];

for(i = 0; i < n; i++)

{

arr1[i] = rand()%100;

}

double time = omp\_get\_wtime();

#pragma omp parallel for schedule(dynamic,20) shared(arr1, answer,j) private(i)

for(i = 0; i < n; i++)

{

answer[i] = arr1[i] + j;

}

printf("\nDone In %f Seconds\n\n", omp\_get\_wtime() - time);

printf("\nArray 1: \n");

for(i = 0; i < n; i++)

{

printf("\t %d", arr1[i]);

}

printf("\nAnswer: \n");

for(i = 0; i < n; i++)

{

printf("\t %d", answer[i]);

}

return 0;

}

**Use of Nowait Clause**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

int n = 200, i ,j=99;

int arr1[n], answer[n];

for(i = 0; i < n; i++)

{

arr1[i] = rand()%100;

}

double time = omp\_get\_wtime();

#pragma omp parallel

{

#pragma omp for nowait

for(i = 0; i < n; i++)

{

answer[i] = arr1[i] + j;

}

}

printf("\nDone In %f Seconds\n\n", omp\_get\_wtime() - time);

printf("\nArray 1: \n");

for(i = 0; i < n; i++){

printf("\t %d", arr1[i]);

}

printf("\nAnswer: \n");

for(i = 0; i < n; i++){

printf("\t %d", answer[i]);

}

return 0;

}

**Github Link:**

<https://github.com/Kunalkadam179/HPC-Assignment/tree/main/Assignment%20-%203>