**Program 2**

**Write an OpenMP program that computes a simple matrix-matrix multiplication using dynamic memory allocation.**

1. **Illustrate the correctness of the program.**
2. **Justify the inference when outer “for” loop is parallelized with and without using the explicit data scope variables.**

**a) Illustrate the correctness of the program.**

#include <stdio.h>

int main()

{

int m, n, p, q, c, d, k, sum = 0;

int first[10][10], second[10][10], multiply[10][10];

printf("Enter the number of rows and columns of first matrix\n");

scanf("%d%d", &m, &n);

printf("Enter the elements of first matrix\n");

for ( c = 0 ; c < m ; c++ )

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

scanf("%d", &first[c][d]);

printf("Enter the number of rows and columns of second matrix\n");

scanf("%d%d", &p, &q);

if ( n != p )

printf("Matrices with entered orders can't be multiplied with each other.\n");

else

{

printf("Enter the elements of second matrix\n");

for ( c = 0 ; c < p ; c++ )

for ( d = 0 ; d < q ; d++ )

scanf("%d", &second[c][d]);

for ( c = 0 ; c < m ; c++ )

{

for ( d = 0 ; d < q ; d++ )

{

for ( k = 0 ; k < p ; k++ )

{

sum = sum + first[c][k]\*second[k][d];

}

multiply[c][d] = sum;

sum = 0;

}

}

printf("Product of entered matrices:-\n");

for ( c = 0 ; c < m ; c++ )

{

for ( d = 0 ; d < q ; d++ )

printf("%d\t", multiply[c][d]);

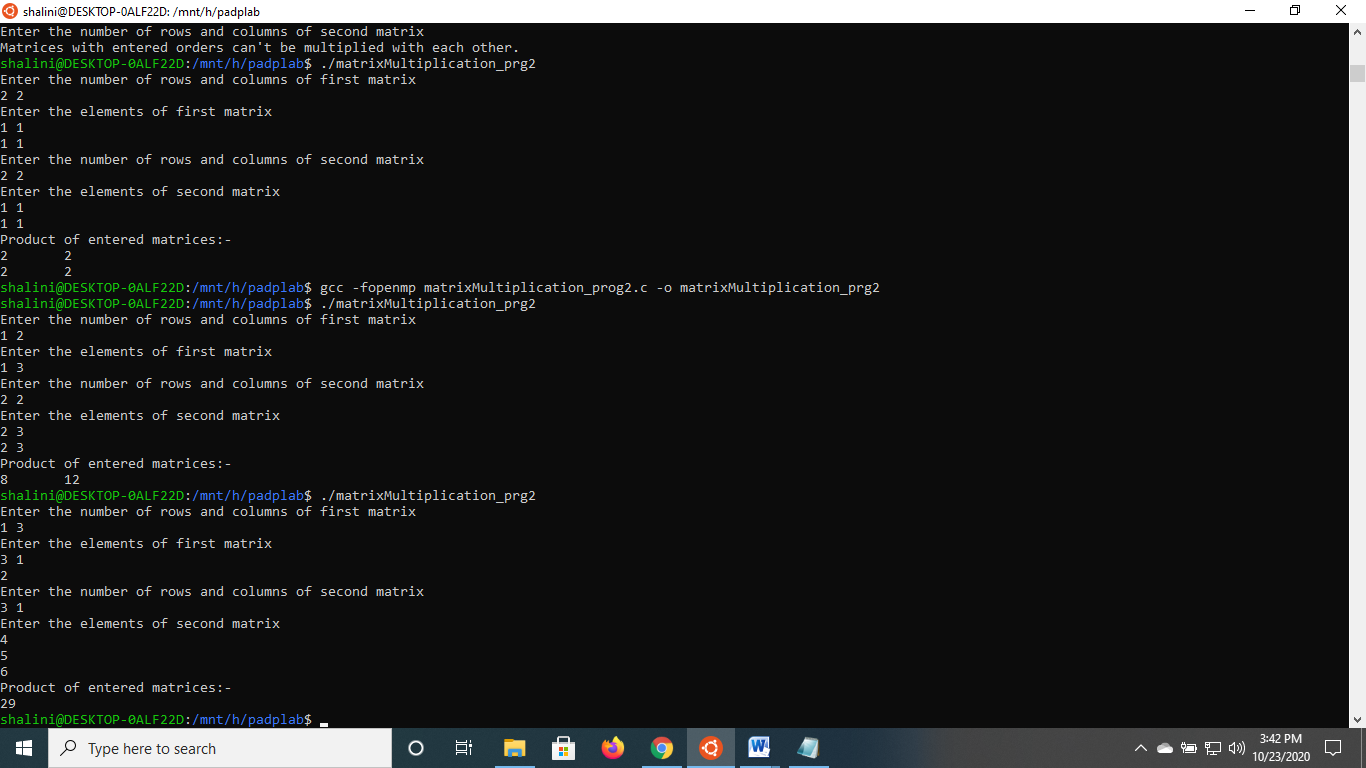
printf("\n");

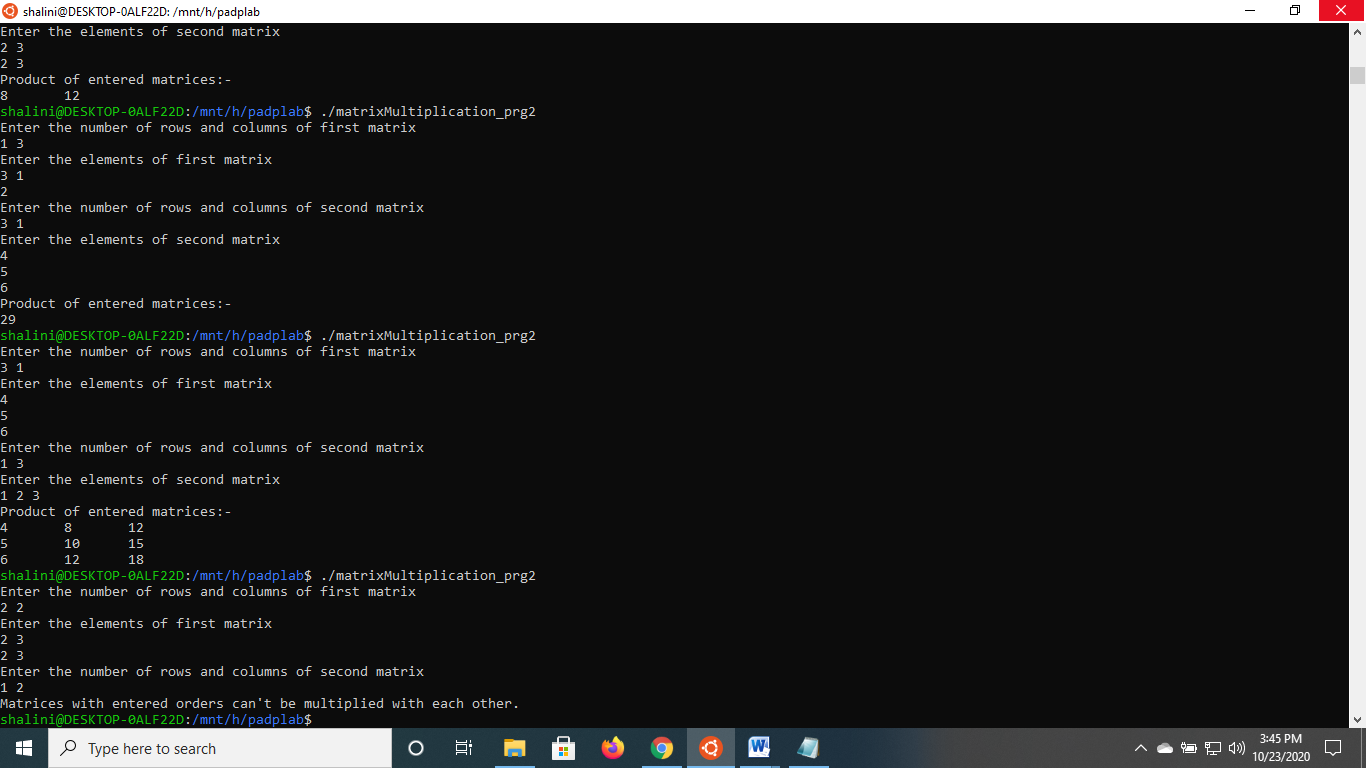
}

}

return 0;

}





1. **Justify the inference when outer “for” loop is parallelized with and without using the explicit data scope variables.**

#include<omp.h>

#include<stdio.h>

#include<stdlib.h>

int main(){

printf("sizes: 500,1000,1500,2000 \n");

printf("threads : 1,2,4,8,16\n");

int r, c, i, j, count=0, sum =0, k;

int threads[5] = {1,2,4,8,16};

long long int sizes[4]={500,1000,1500,2000};

//dynamically allocate arrays

printf("\t\t1\t\t2\t\t4\t\t8\t\t16\n");

for(int sizeloop = 0; sizeloop<4; sizeloop++)

{

printf("\n%lld", sizes[sizeloop]);

for(int threadloop =0; threadloop<5;threadloop++)

{

r = c = sizes[sizeloop];

int \*\*arr1 = (int \*\*)malloc(r \* sizeof(int \*));

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

arr1[i] = (int \*)malloc(c \* sizeof(int));

int \*\*arr2 = (int \*\*)malloc(r \* sizeof(int \*));

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

arr2[i] = (int \*)malloc(c \* sizeof(int));

int \*\*arr3 = (int \*\*)malloc(r \* sizeof(int \*));

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

arr3[i] = (int \*)malloc(c \* sizeof(int));

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

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

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

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

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

arr2[i][j] = count++;

double x = omp\_get\_wtime();

omp\_set\_num\_threads(threadloop);

#pragma omp parallel for private(j, k)

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

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

for(k = 0;k < r; k++)

arr3[i][j] += arr1[i][k] \* arr2[k][j];

double y = omp\_get\_wtime();

printf("\t%lf", y-x);

}//threadloop

}//sizeloop

return 0;

}

