PARALLEL AND DISTRIBUTED COMPUTING

(CSE4001)

CONSOLIDATED LAB REPORT

Name: O G Ragavi

Reg No: 20BCE1988

Slot: D1

Course code: CSE4001

CSE4001-Parallel and distributed computing

Name: O G RAGAVI

Reg no: 20BCE1988

Lab Ex: 1-3

Date: 21.08.2022

1.Sample program

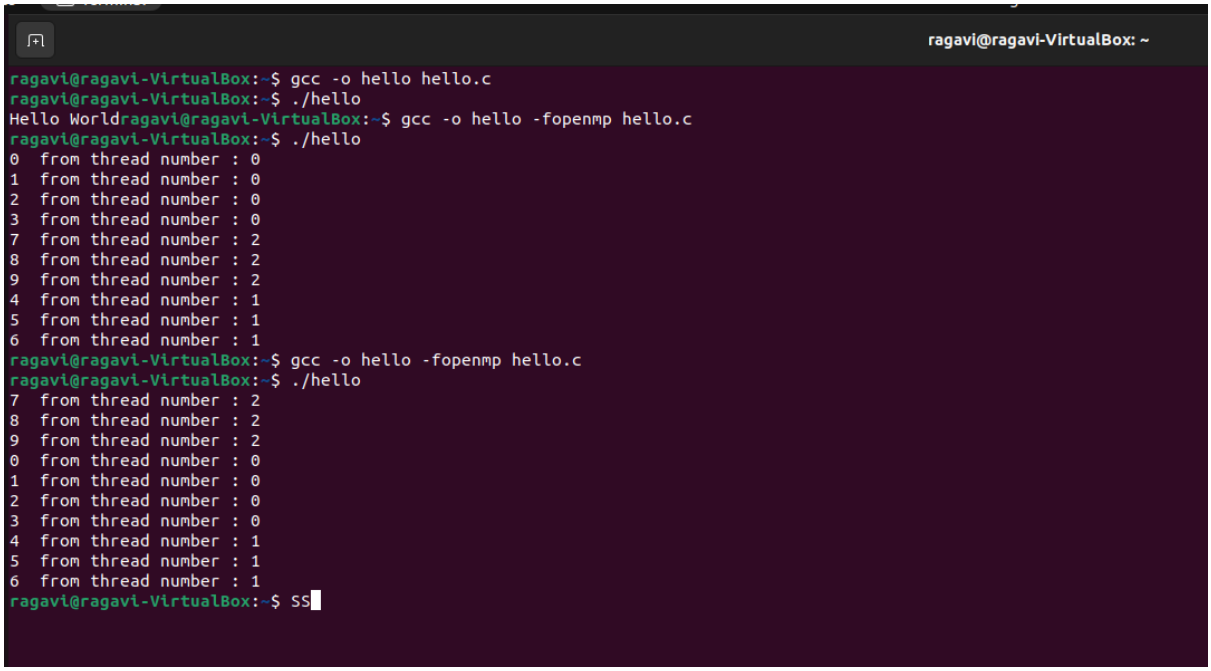
Code:

Open

hello.c
~/

```c
#include<stdio.h>
#include<omp.h>
int main()
{
   #pragma omp parallel for
   for(int i=0;i<10;i++)
   {
      printf("%d  from thread number : %d\n",i,omp_get_thread_num());
   }
}
```

Output:



2.Program to find the sum of first n numbers.

Algorithm:

- **Start the clock before each and every iteration and stop it before it ends.**
- **N is initialised and for every iteration no of threads is set.**
- **Using reduction , we run a for loop and increment the sum by i**
- **We end the clock and note the time difference(total time taken)**

Code:

Open

prog1.c
~/

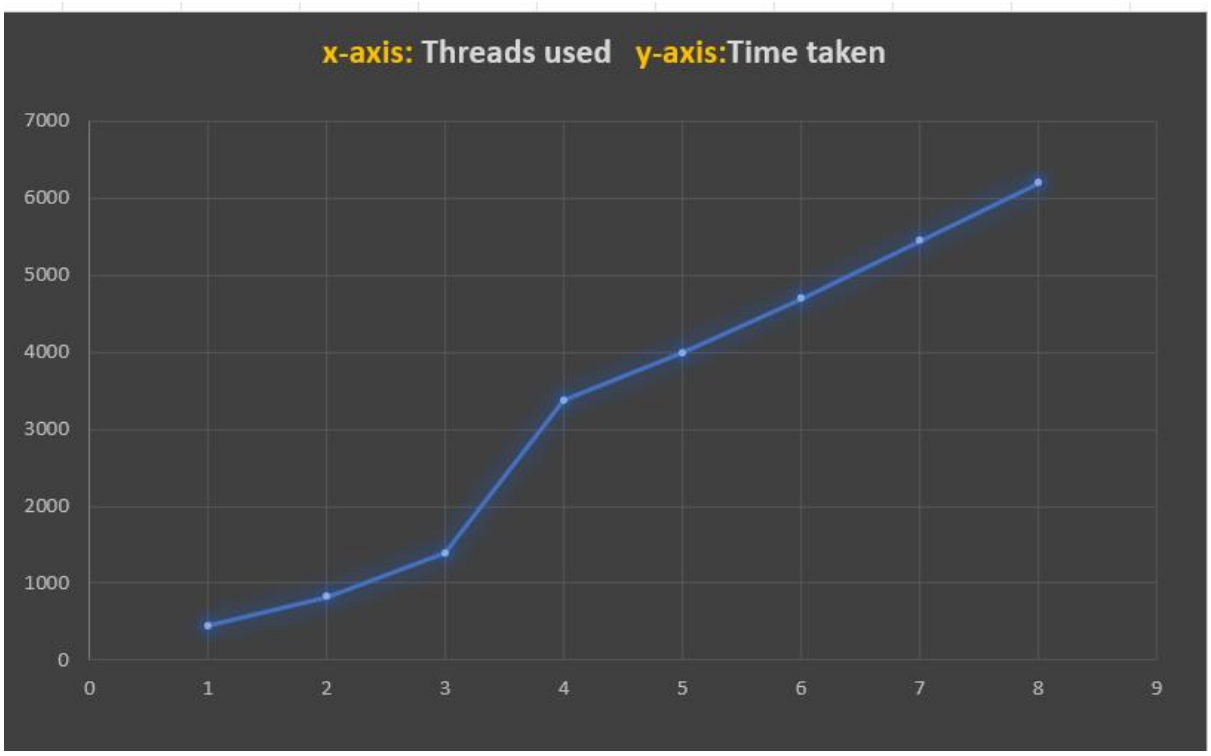
```c
//program to find the sum of first n numbers where n>10000 with time constraint.
#include<stdio.h>
#include<omp.h>
#include<time.h>
int main()
{
   clock_t s,e;

   int i;
   int n=100000,sum=0;
   for(int j=1;j<=8;j++)
   {
   s=clock();
   omp_set_num_threads(j);// In which thread the corresponding function is working
   #pragma omp parallel for reduction(+: sum)
   for(i=0;i<n;i++)
   {
      sum=sum+i;
   }

   printf("Sum is : %d",sum);
   e=clock();
   printf("\nTime taken : %ld for thread  : %d\n",e-s,j);
   sum=0;
   }
}
```

Output:

ragavi@ragavi-VirtualBox: ~

```
ragavi@ragavi-VirtualBox:~$ gcc -o prog1 -fopenmp prog1.c
ragavi@ragavi-VirtualBox:~$ ./prog1
Sum is : 704982704
Time taken : 373 Sum is : 704982704
Time taken : 716 Sum is : 704982704
Time taken : 978 Sum is : 704982704
Time taken : 2272 Sum is : 704982704
Time taken : 2795 Sum is : 704982704
Time taken : 3402 Sum is : 704982704
Time taken : 3955 Sum is : 704982704
Time taken : 4961 ragavi@ragavi-VirtualBox:~$ gcc -o prog1 -fopenmp prog1.c
ragavi@ragavi-VirtualBox:~$ ./prog1
Sum is : 704982704
Time taken : 442 for thread  : 1
Sum is : 704982704
Time taken : 821 for thread  : 2
Sum is : 704982704
Time taken : 1392 for thread  : 3
Sum is : 704982704
Time taken : 3377 for thread  : 4
Sum is : 704982704
Time taken : 3987 for thread  : 5
Sum is : 704982704
Time taken : 4696 for thread  : 6
Sum is : 704982704
Time taken : 5440 for thread  : 7
Sum is : 704982704
Time taken : 6196 for thread  : 8
ragavi@ragavi-VirtualBox:~$
```

Graph Plot:



2.Program to find the sum of n numbers: n>10000

Algorithm:

- **Start the clock before each and every iteration and stop it before it ends.**
- **N is initialised and for every iteration no of threads is set.**
- **Using reduction , we run a for loop and increment the sum by any random number generated.(0-100)**
- **We end the clock and note the time difference(total time taken)**

Code:

Open

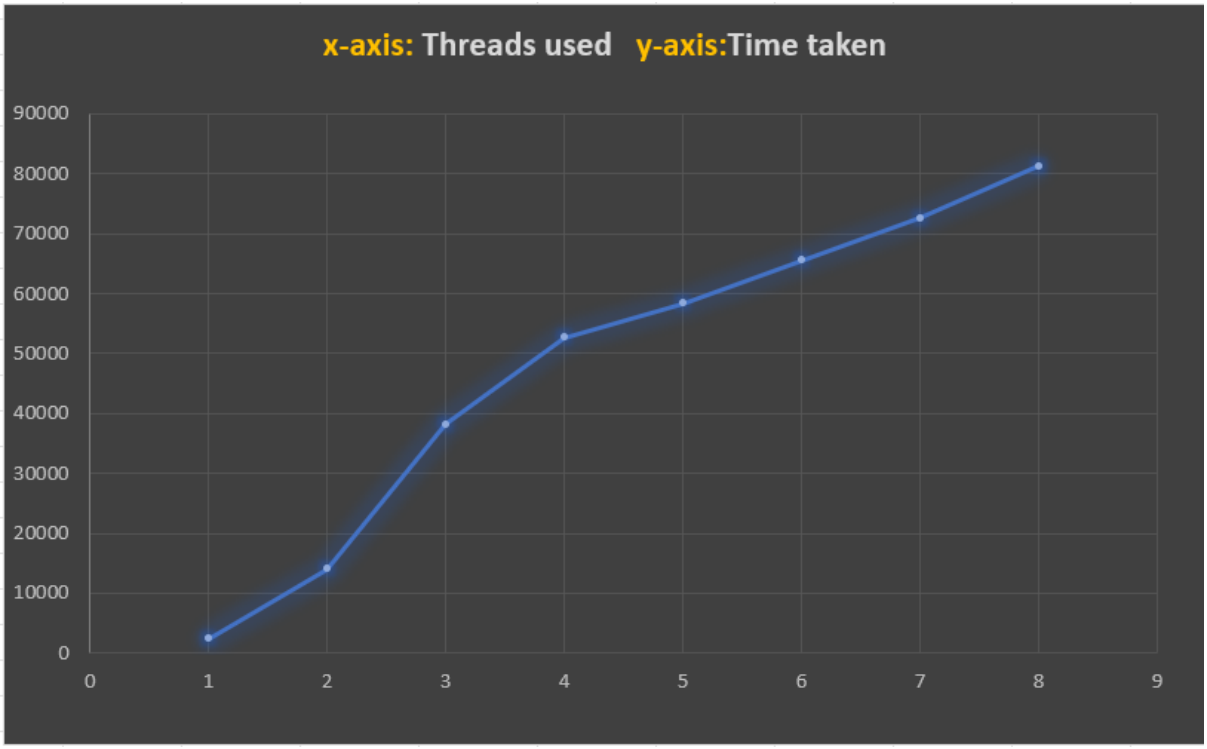
prog2.c
~/

```c
//program to find the sum of n numbers where n>10000 with time constraint.
#include<stdio.h>
#include<omp.h>
#include<time.h>
#include<stdlib.h>
int main()
{
   clock_t s,e;

   int i;
   int n=100001,sum=0;
   for(int j=1;j<=8;j++)
   {
   s=clock();
   omp_set_num_threads(j);// In which thread the corresponding function is working
   #pragma omp parallel for reduction(+: sum)
   for(i=0;i<n;i++)
   {

      sum=sum+(rand()%100);
   }

   printf("Sum is : %d",sum);
   e=clock();
   printf("\nTime taken : %ld for thread  : %d\n",e-s,j);
   sum=0;
   }
}
```

Output:

```
ragavi@ragavi-VirtualBox:~$ ./prog2
Sum is : 4952488
Time taken : 2436 for thread  : 1
Sum is : 4949527
Time taken : 14468 for thread  : 2
Sum is : 4942504
Time taken : 43845 for thread  : 3
Sum is : 4948385
Time taken : 62964 for thread  : 4
Sum is : 4953382
Time taken : 66958 for thread  : 5
Sum is : 4953961
Time taken : 90371 for thread  : 6
Sum is : 4960295
Time taken : 94243 for thread  : 7
Sum is : 4940623
Time taken : 97459 for thread  : 8
```

Graph Plot:



3.Program to find the addition of 2 integer arrays :

Algorithm:

- **Initialise 2 arrays a and b.**
- **Use a for loop to get the sum at each index.**
- **Omp_get_num_threads – building function that takes void input and returns no threads.**
- **Stop the clock.**

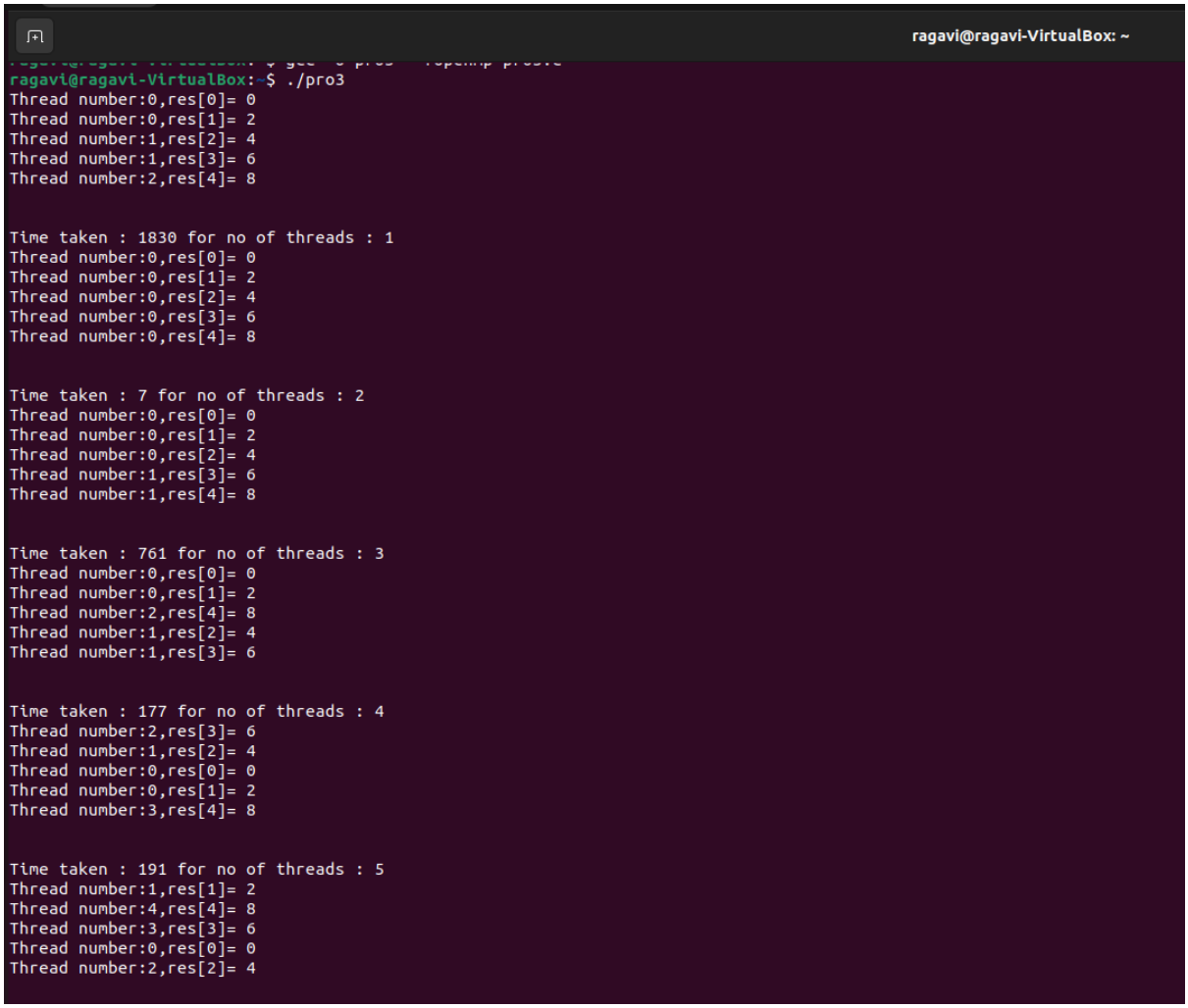
Code:

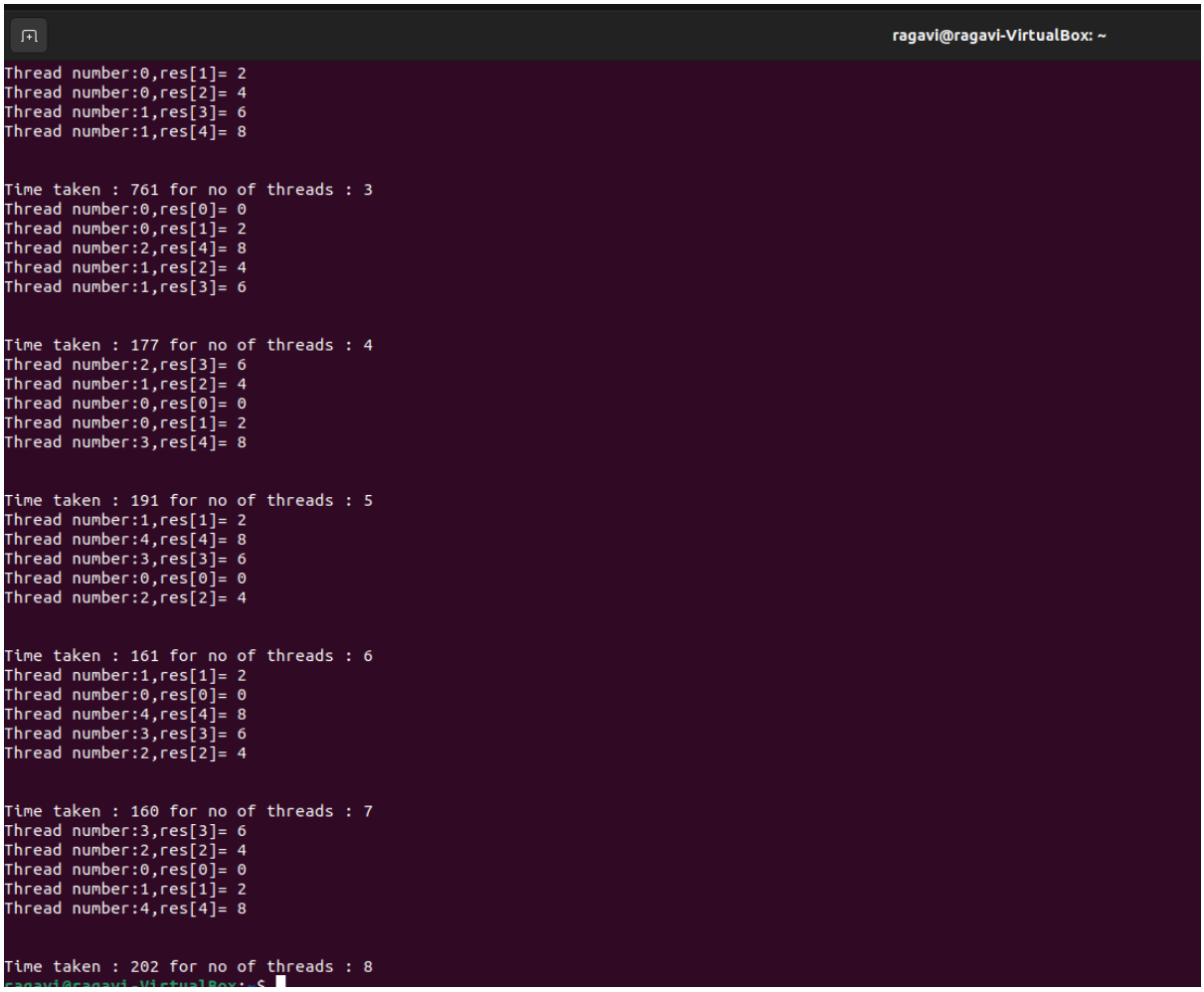
Open

pro3.c
~/

```c
//program to find the addition of 2 integer arrays  with time constraint.
#include<stdio.h>
#include<omp.h>
#include<time.h>
#include<stdlib.h>
int main()
{
   clock_t s,e;

   int i;
   int tid;
   int a[5]={0,1,2,3,4};
   int b[5]={0,1,2,3,4};
   int res[5];
   for(int j=1;j<=8;j++)
   {
      s=clock();
      #pragma omp parallel for schedule(static) num_threads(j-1)
      for(i=0;i<5;i++)
      {
        res[i]=a[i]+b[i];
        printf("Thread number:%d,res[%d]= %d\n", omp_get_thread_num(),i,res[i]);
      }



   printf("\n");
   e=clock();
   printf("\nTime taken : %ld for no of threads : %d\n",e-s,j);

   }
}
```

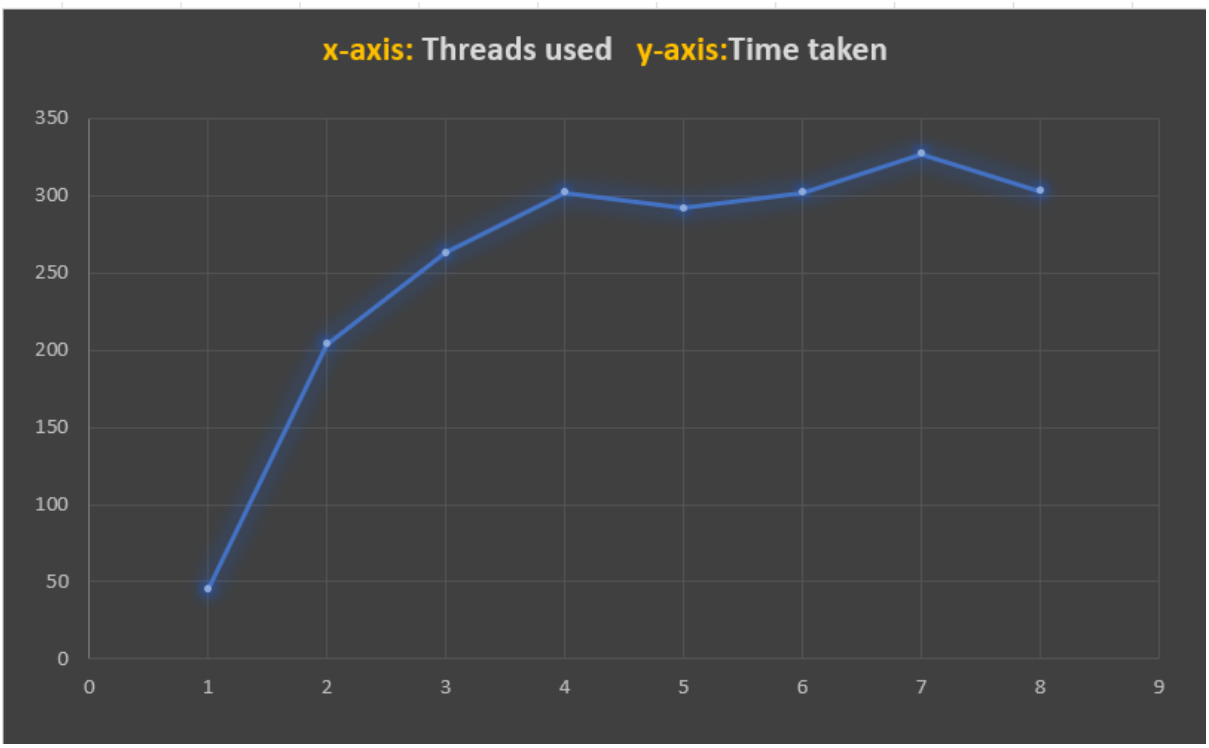
Output:


ragavi@ragavi-VirtualBox: ~
ragavi@ragavi-VirtualBox:~$ ./pro3
Thread number:0,res[0]= 0
Thread number:0,res[1]= 2
Thread number:1,res[2]= 4
Thread number:1,res[3]= 6
Thread number:2,res[4]= 8
Time taken : 1830 for no of threads : 1
Thread number:0,res[0]= 0
Thread number:0,res[1]= 2
Thread number:0,res[2]= 4
Thread number:0,res[3]= 6
Thread number:0,res[4]= 8
Time taken : 7 for no of threads : 2
Thread number:0,res[0]= 0
Thread number:0,res[1]= 2
Thread number:0,res[2]= 4
Thread number:1,res[3]= 6
Thread number:1,res[4]= 8
Time taken : 761 for no of threads : 3
Thread number:0,res[0]= 0
Thread number:0,res[1]= 2
Thread number:2,res[4]= 8
Thread number:1,res[2]= 4
Thread number:1,res[3]= 6
Time taken : 177 for no of threads : 4
Thread number:2,res[3]= 6
Thread number:1,res[2]= 4
Thread number:0,res[0]= 0
Thread number:0,res[1]= 2
Thread number:3,res[4]= 8
Time taken : 191 for no of threads : 5
Thread number:1,res[1]= 2
Thread number:4,res[4]= 8
Thread number:3,res[3]= 6
Thread number:0,res[0]= 0
Thread number:2,res[2]= 4



Graph Plot:

x axis	y axis
Threads used	Time(s)
1	45
2	204
3	263
4	302
5	292
6	302
7	327
8	303



4.Program to find the maximum element from an array of 1000 numbers.

Code:

Open

prog4.c
~/

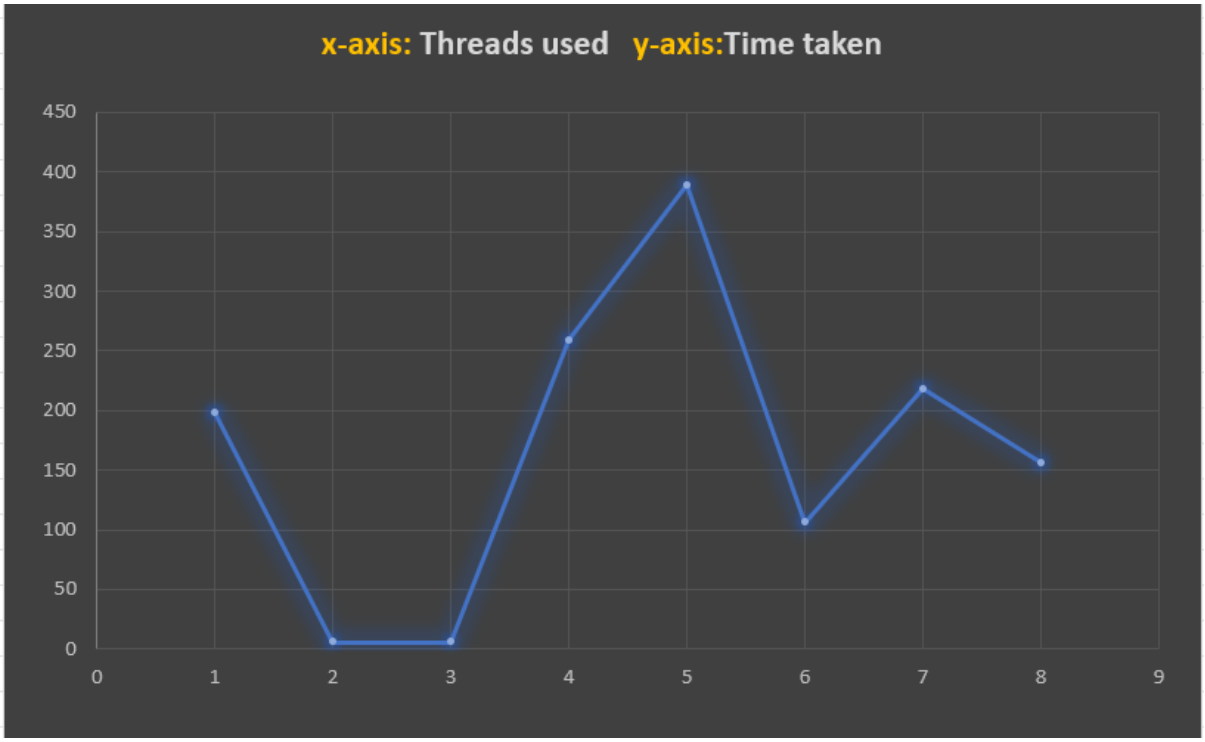
```c
//program to find the maximum element from an array of 1000 numbers with time constraint.
#include<stdio.h>
#include<omp.h>
#include<time.h>
#include<stdlib.h>
int main()
{
   clock_t s,e;

   int i,max=0;
   int tid,a[1000];
   for(int k=0;k<1000;k++)
   {
      a[k]=rand()%50;
   }
   for(int j=1;j<=8;j++)
   {
      s=clock();
      #pragma omp parallel for schedule(static) num_threads(j-1)
      for(i=0;i<1000;i++)
      {
        if(max<a[i])
        max=a[i];

      }
      printf("Thread number:%d, max= %d\n", omp_get_thread_num(),max);
      max=0;


   printf("\n");
   e=clock();
   printf("\nTime taken : %ld for no of threads : %d\n",e-s,j);

   }
}
```

Output:

```
ragavi@ragavi-VirtualBox:~$ gcc -o prog4  -fopenmp prog4.c
ragavi@ragavi-VirtualBox:~$ ./prog4
Thread number:0, max= 49


Time taken : 198 for no of threads : 1
Thread number:0, max= 49


Time taken : 6 for no of threads : 2
Thread number:0, max= 49


Time taken : 6 for no of threads : 3
Thread number:0, max= 49


Time taken : 259 for no of threads : 4
Thread number:0, max= 49


Time taken : 389 for no of threads : 5
Thread number:0, max= 49


Time taken : 106 for no of threads : 6
Thread number:0, max= 49


Time taken : 218 for no of threads : 7
Thread number:0, max= 49


Time taken : 156 for no of threads : 8
ragavi@ragavi-VirtualBox:~$
```

Graph Plot:

x axis	y axis
Threads used	Time(s)
1	198
2	6
3	6
4	259
5	389
6	106
7	218
8	156



5.Program to find the minimum element from the array of n numbers.

Code:

Open prog5.c

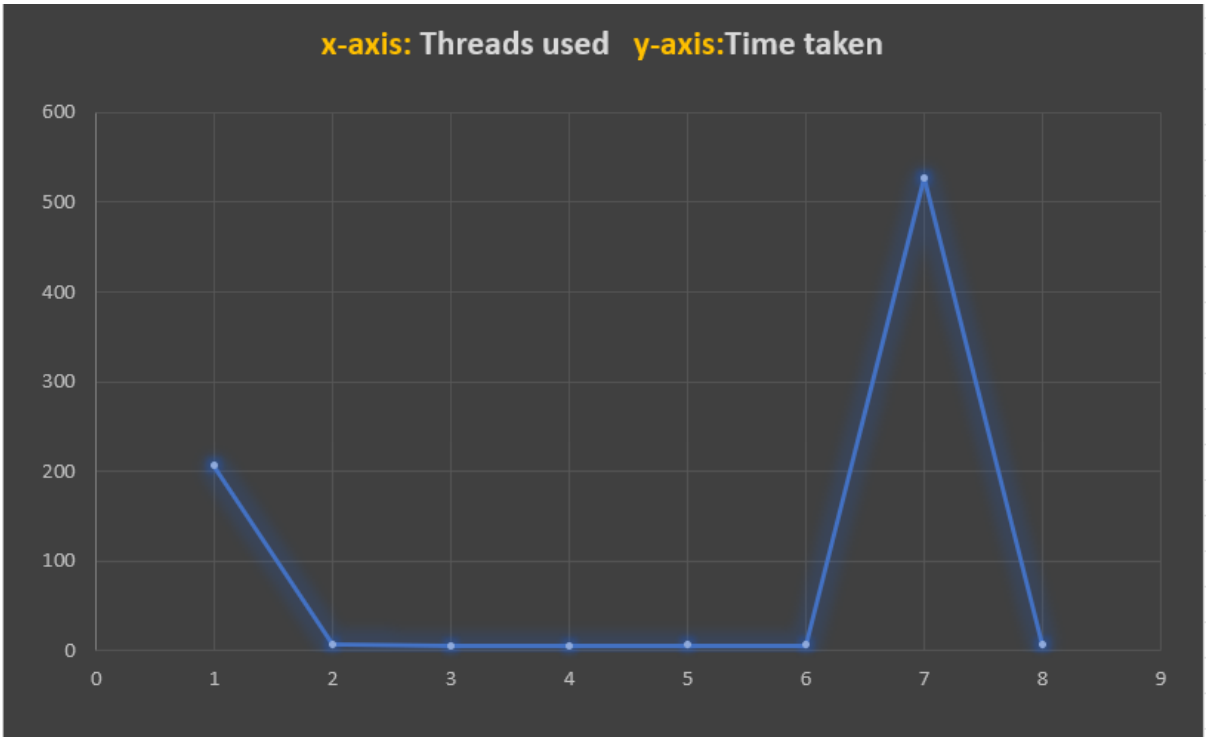
```c
//program to find the minimum element from an array of 10000 numbers with time constraint.
#include<stdio.h>
#include<omp.h>
#include<time.h>
#include<stdlib.h>
int main()
{
   clock_t s,e;

   int i,min1=1000000;
   int tid,a[10000];
   for(int k=0;k<10000;k++)
   {
      a[k]=rand()%100;
   }
   for(int j=1;j<=8;j++)
   {
      s=clock();
      #pragma omp parallel reduction(min:min1)
      #pragma omp for
      for(i=0;i<1000;i++)
      {
        if(min1>a[i])
        min1=a[i];

      }
      printf("Thread number:%d, min= %d\n", omp_get_thread_num(),min1);
      min1=100000;


   printf("\n");
   e=clock();
   printf("\nTime taken : %ld for no of threads : %d\n",e-s,j);

   }
}
```

Output:

```
ragavi@ragavi-VirtualBox:~$ gcc -o prog5  -fopenmp prog5.c
ragavi@ragavi-VirtualBox:~$ ./prog5
Thread number:0, min= 0


Time taken : 206 for no of threads : 1
Thread number:0, min= 0


Time taken : 7 for no of threads : 2
Thread number:0, min= 0


Time taken : 5 for no of threads : 3
Thread number:0, min= 0


Time taken : 5 for no of threads : 4
Thread number:0, min= 0


Time taken : 6 for no of threads : 5
Thread number:0, min= 0


Time taken : 6 for no of threads : 6
Thread number:0, min= 0


Time taken : 527 for no of threads : 7
Thread number:0, min= 0


Time taken : 6 for no of threads : 8
```

Graph Plot:

x axis	y axis
Threads used	Time(s)
1	206
2	7
3	5
4	5
5	6
6	6
7	527
8	6



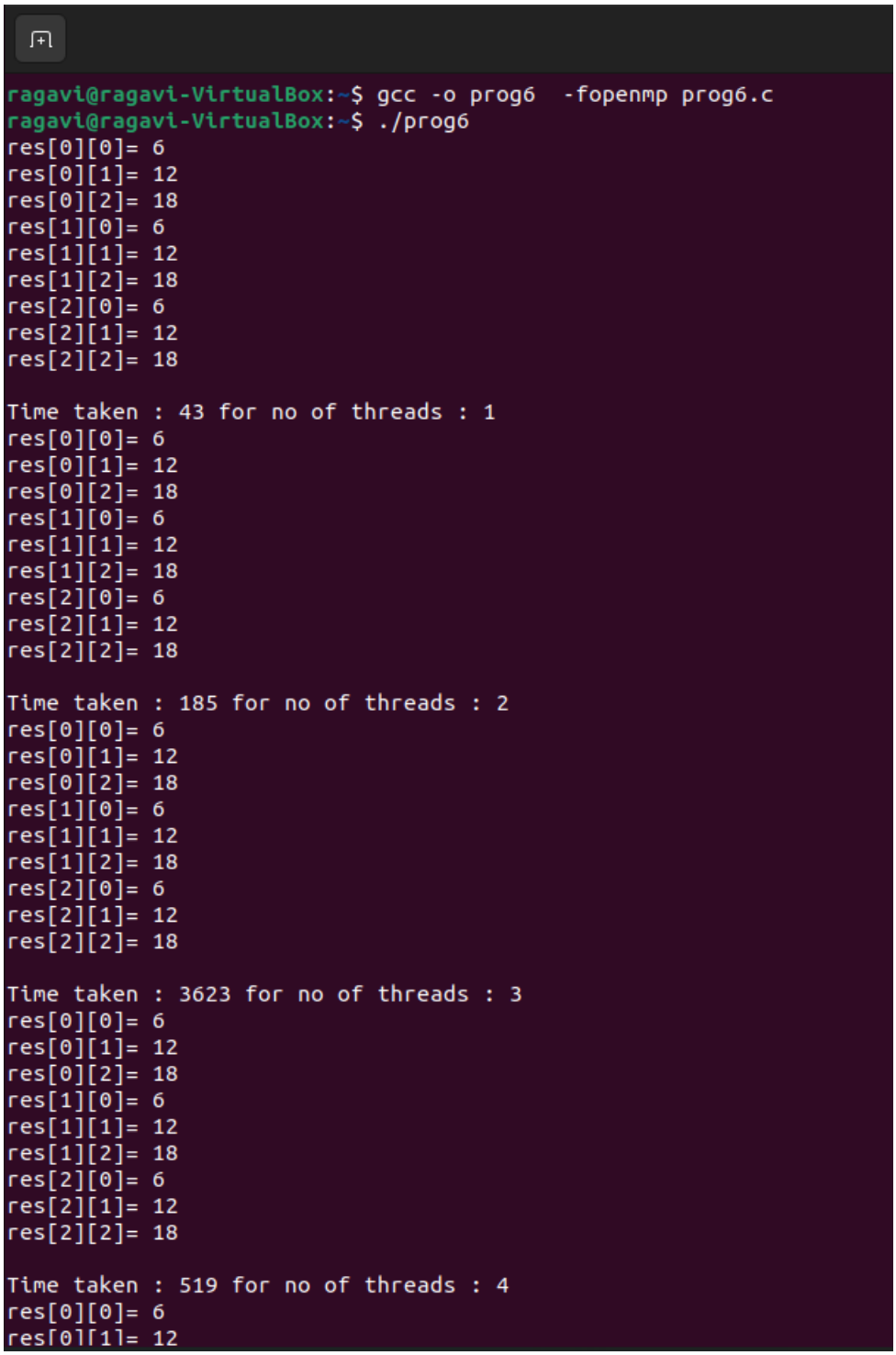
6.Program to find resultant of matrix multiplication.

Code:

Open prog6.c

```
1 //program to perform matrix multiplication  with time constraint.
2 #include<stdio.h>
3 #include<omp.h>
4 #include<time.h>
5 #include<stdlib.h>
6 int main()
7 {
8    clock_t s,e;
9
10   int i;
11   int tid;
12   int a[3][3]={{1,2,3},{1,2,3},{1,2,3}};
13   int b[3][3]={{1,2,3},{1,2,3},{1,2,3}};
14   int res[3][3];
15   for(int j=1;j<=8;j++)
16   {
17      s=clock();
18      omp_set_num_threads(j);
19      #pragma omp parallel for collapse(2) schedule(static)
20      for(int i1=0;i1<3;i1++)
21      {
22         for(int j1=0;j1<3;j1++)
23         {
24            res[i1][j1]=0;
25           for(int k=0;k<3;k++)
26           {
27             res[i1][j1]+= a[i1][k]*b[k][j1];
28           }
29         }
30       }
31
32        for(i=0;i<3;i++)
33        {
34          for(int k=0;k<3;k++)
35          {
36
37             printf("res[%d][%d]= %d\n",i,k,res[i][k]);
38           }
39
40         }
41
42
43
44   printf("\n");
45   e=clock();
46   printf("Time taken : %ld for no of threads : %d\n",e-s,j);
```

Output:



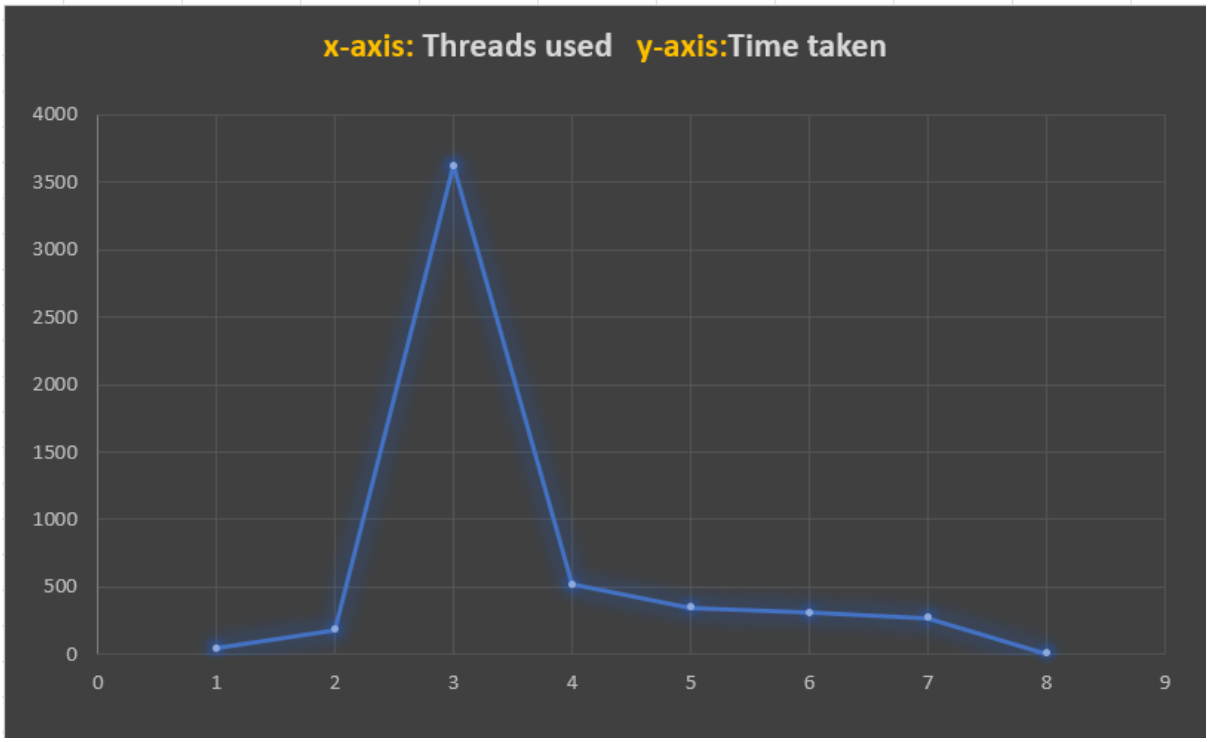
```
ragavi@ragavi-VirtualBox:~$ gcc -o prog6  -fopenmp prog6.c
ragavi@ragavi-VirtualBox:~$ ./prog6
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 43 for no of threads : 1
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 185 for no of threads : 2
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 3623 for no of threads : 3
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 519 for no of threads : 4
res[0][0]= 6
res[0][1]= 12
```

```
Time taken : 3623 for no of threads : 3
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 519 for no of threads : 4
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 348 for no of threads : 5
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 308 for no of threads : 6
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 273 for no of threads : 7
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
```

```
res[2][2]= 18

Time taken : 519 for no of threads : 4
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 348 for no of threads : 5
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 308 for no of threads : 6
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 273 for no of threads : 7
res[0][0]= 6
res[0][1]= 12
res[0][2]= 18
res[1][0]= 6
res[1][1]= 12
res[1][2]= 18
res[2][0]= 6
res[2][1]= 12
res[2][2]= 18

Time taken : 328 for no of threads : 8
ragavi@ragavi-VirtualBox:~$
```

Graph Plot:

x axis	y axis
Threads used	Time(s)
1	43
2	185
3	3623
4	519
5	348
6	308
7	273
8	328



7.a)No-wait:

```
#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]);
}
```

b)Barrier:

```
#pragma omp parallel default(shared)
{
    #pragma omp for
        for (i=0; i<n; i++)
        {
            #pragma omp parallel shared(i, n)
            {
```

```
                #pragma omp for
                    for (j=0; j<n; j++)
                        work(i, j);
            }
        }
}
```

CSE4001-PARALLEL AND DISTRIBUTED COMPUTING

LAB-4

Name: O G Ragavi

Reg No: 20BCE1988

Lab: 4

—----------

1.Aim : Sorting an array of 1000 numbers

Algorithm:

- Include the preprocessor directives
- Use the clock function of time.h library to start the timer before entering into the parallel section
- Use a loop that runs from 1 to size of the array which is 1000 .• Use the phase #pragma omp parallel for, so that the iterations of the inner loop gets distributed among the given number of threads to reach maximum efficiency .

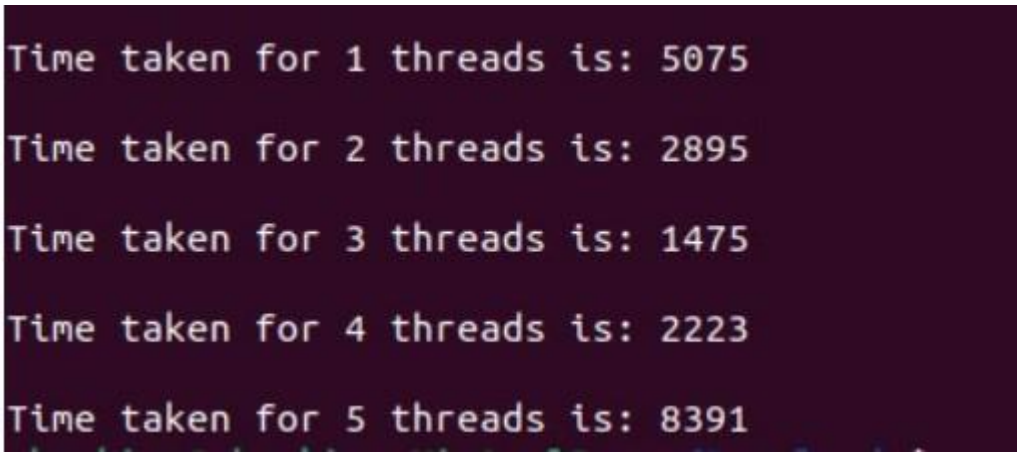
Code:

```
#include<stdio.h>

#include<omp.h>

#include<time.

h>

#include<stdlib

.h> int main () {

int
A[1000];
int   N   =
```

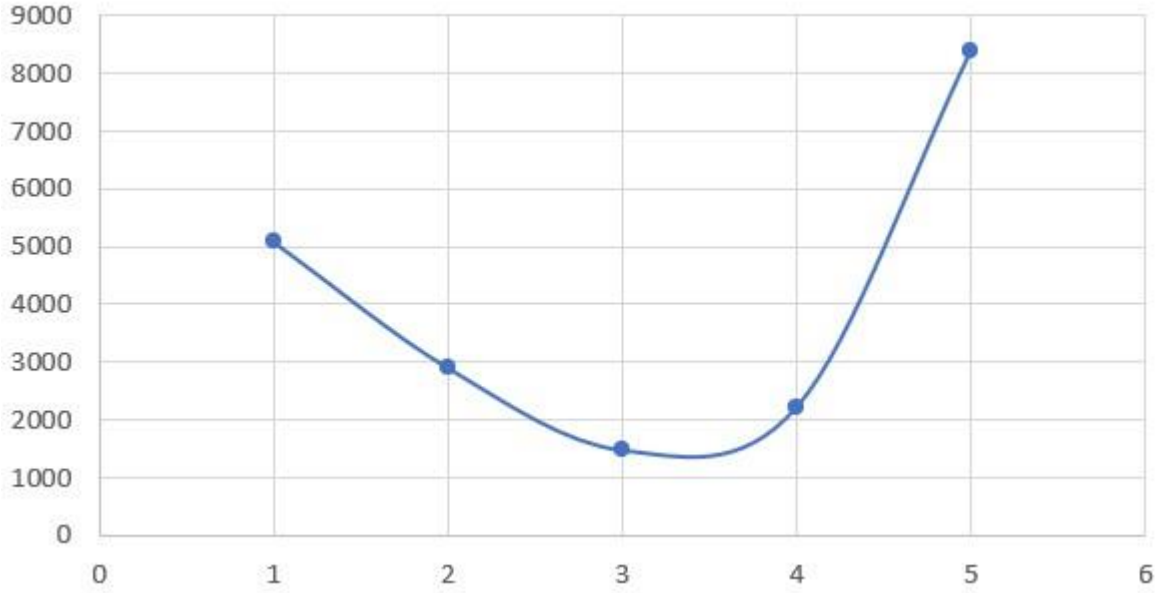
```
1000; int
i=0;
for(int i=0;i<N;i++)
{
A[i]=rand()%50;
}
clock_t
s,e;

for(int k=0;k<5;k++)
{
  s=clock();
  omp_set_num_threads(k+1);
  #pragma omp parallel for default(none), shared(A, N)
        for(int i=0;i<N;i++)
        {


          for(int j = 0; j < N-1; j++)
          {
if(A[j] > A[j
+ 1])
{ int temp =
   A[j]; A[j] =
   A[j + 1];
    A[j + 1] = temp;
}
```

```
    }
        }


/*
 for(i=0;i<N;i
 ++) {
  //printf("%d ",A[i]);
}*/ e = clock(); printf("\nTime taken for %d threads
        is: %ld\n",k+1,(e-s));
         }

}
```

Output:



Graph:



2.Aim : Searching an element in an array of elements:

Algorithm:

- Include the required pre-processor directives
- Declare an array of the required size and initialize it with random numbers
- Use the time function to start the clock before the parallel section
- User #pragma omp parallel for, so that the iterations gets distributed among the given number of threads
- Inside the loop use a binary search to search the key from the corresponding array, print the location of the element found.

Code:

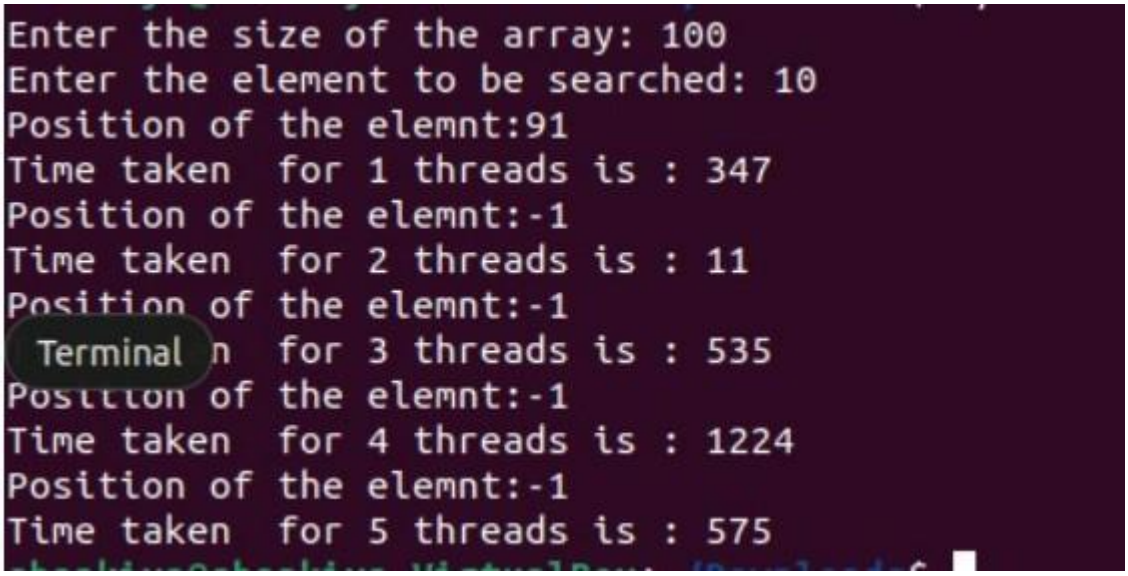
```
#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#include<time.h>
```

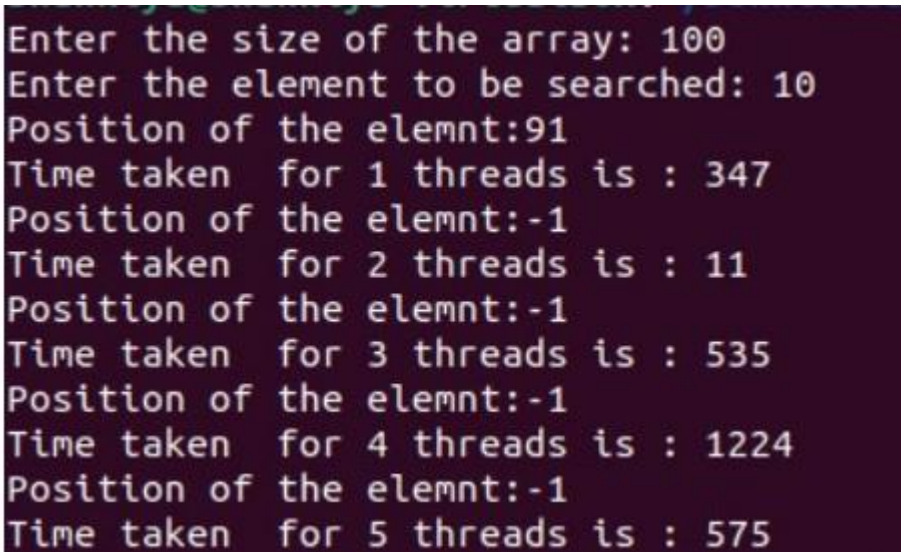
```
int bin(int a[], int l, int r, int key, int n)
{ int index = -1; int
   size = (r - l + 1) /
   n; if (size == 0 ||
   n == 1)
   {
      #pragma omp parallel
      for for (int i = l; i <= r;
      i++)
      { if (a[i] ==
         key)
            index = i;
      }
      return index;
   } int
   left = l;
   int
   right =
   r;
   omp_set_num_threads(n);
   omp_set_nested(1);
   #pragma omp parallel
   {
      int id = omp_get_thread_num();
      int lt = l + id *
      size; int rt = lt
      + size - 1; if (id
      == n - 1)
```

```
        rt = r;

      if (a[lt] <= key && a[rt] >= key)
      { left =
        lt;
        right
        = rt;
      } } if (left == l
   && right == r)
      return -1;
   return bin(a, left, right, key, n);
}

int main()
{ int n; int nthread; clock_t s, e;
   printf("Enter the size of the
   array: "); scanf("%d", &n); int
   a[n];

   for (int i = 0; i < n; i++)
      a[i] = rand()%50; printf("Enter the
      element to be searched: ");
   int key;
   scanf("%d",
   &key); for(int
   i=0;i<5;i++){


   s = clock(); printf("Position of the elemnt:%d\n",
   bin(a, 0, n- 1, key, i+1)); e = clock(); printf("Time
   taken  for %d threads is : %ld\n",i+1,(e-s));
```

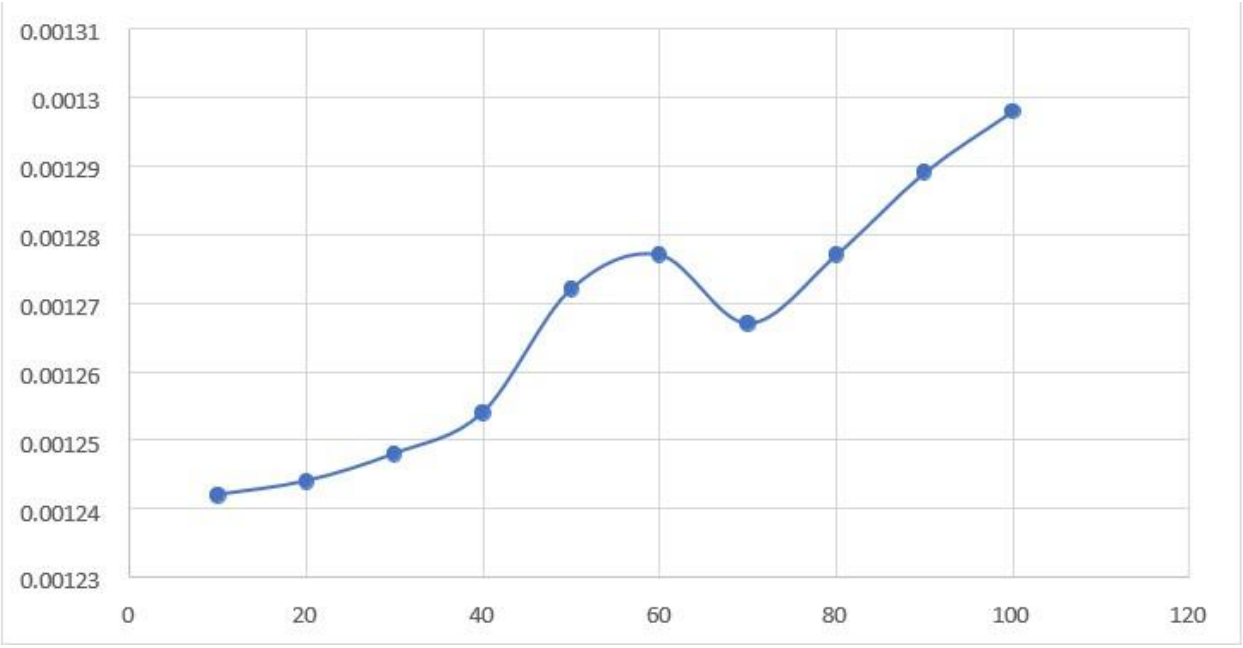
```
    }

}
```

Output:





Graph:

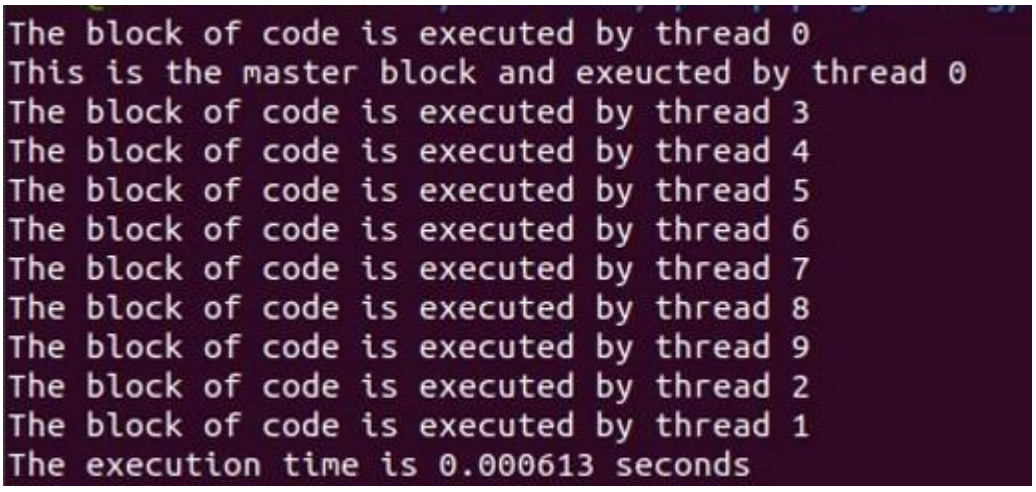


3.Aim: To write an OpenMP program to illustrate master, single, firstprivate, lastprivate and ordered . **a. Master Code:**

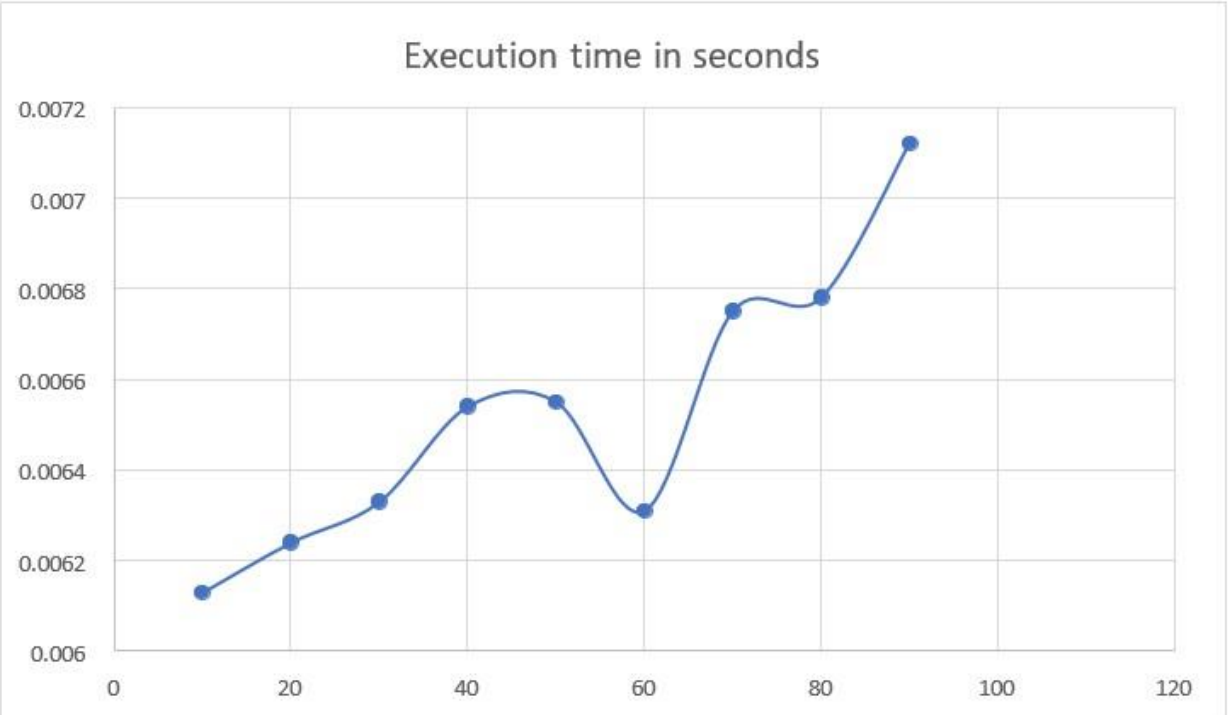
```
#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#include<time.h>

void main()

{

   time_t
   start,end; start
   = clock();

   #pragma omp parallel

   {

     printf("The block of code is executed by thread
%d\n",omp_get_thread_num());

      #pragma omp master
```

```
    {
        printf("This is the master block and exeucted by thread
%d\n",omp_get_thread_num());
    }
  }
  end = clock();
                                double diff = (end-
                start)*1.0/CLOCKS_PER_SEC;
  printf("The execution time is %lf seconds\n",diff);
}
```

Output:



Graph:



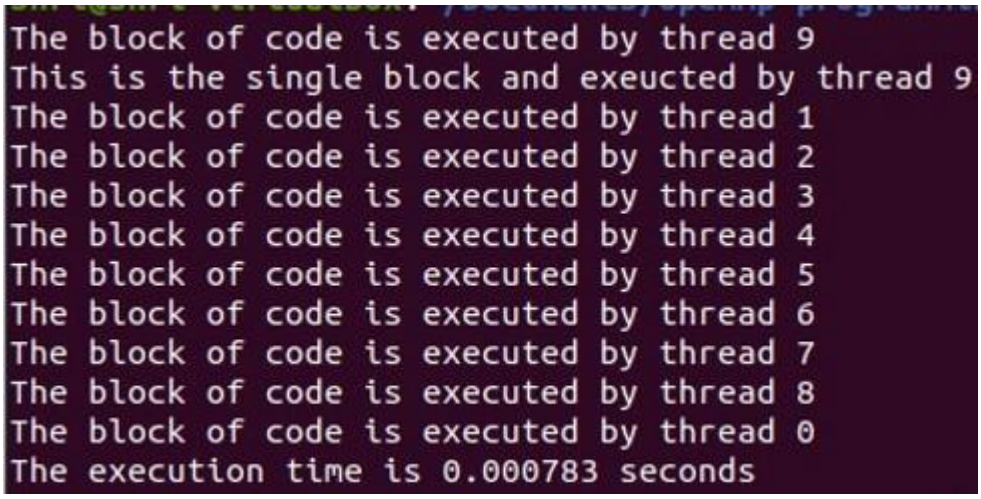
b.Single:

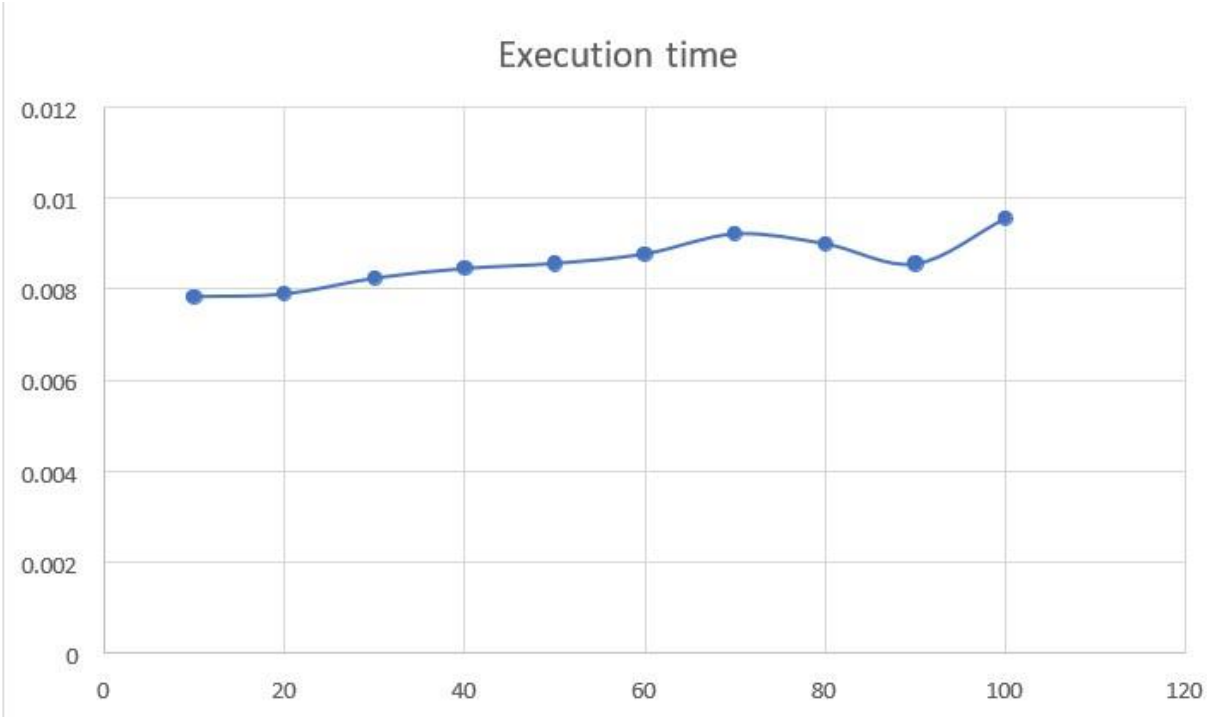
Code:

```
#include<st
dio.h>

#include<omp.h>

#include<stdlib.h>

#include<time.h
 > void main()

{    time_t
   start,end; start =
   clock();

   #pragma omp parallel

   {

      printf("The block of code is executed by thread

%d\n",omp_get_thread_num());       #pragma omp single

      {
```

```
        printf("This is the single block and exeucted by thread
%d\n",omp_get_thread_num());
      }
   }
   end = clock();
   double diff = (end-start)*1.0/CLOCKS_PER_SEC;     printf("The
   execution time
is %lf
seconds\n",diff);
}
```

Output:





c.Firstprivat

e: Code:

```
#include<stdio.h>

#include<omp.h>

#include<stdlib.
 h>
 #include<time.h
 > void main()

{

    time_t
    start,end;
    int i = 127;
    start =
    clock();

    #pragma omp parallel firstprivate(i)

    {
```

```
                    printf("The value of i in the thread %d is
                        %d\n",omp_get_thread_num(),i);
    }
    end = clock(); double diff = (end-
    start)*1.0/CLOCKS_PER_SEC;
printf("The execution time is %lf seconds\n",diff);
}
```

Output:

```
The value of i in the thread 9 is 127
The value of i in the thread 0 is 127
The value of i in the thread 1 is 127
The value of i in the thread 2 is 127
The value of i in the thread 3 is 127
The value of i in the thread 4 is 127
The value of i in the thread 5 is 127
The value of i in the thread 6 is 127
The value of i in the thread 7 is 127
The value of i in the thread 8 is 127
The execution time is 0.000731 seconds
```

Graph:

d.Lastprivat

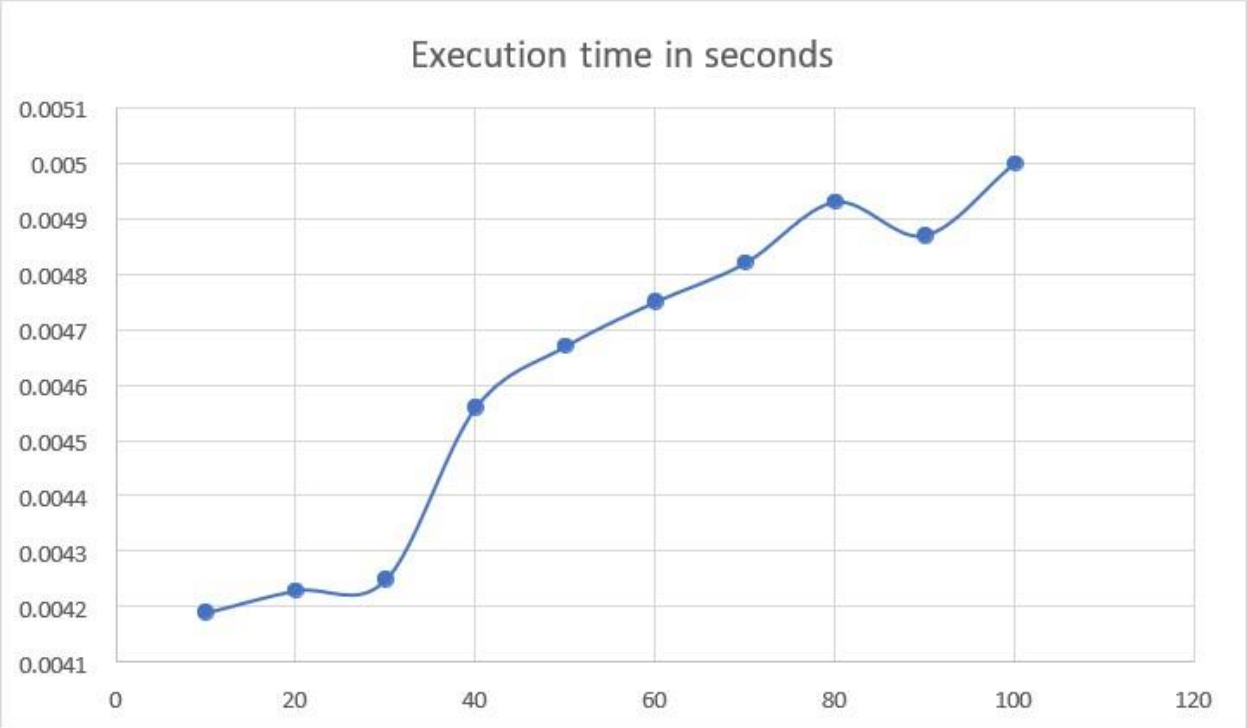
e: Code:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.
h>
#include<time.h
> void main()
{
        time_t
start,end; start
= clock(); int j =
            0;
  #pragma omp parallel for
    lastprivate(j) for(int i=0;i<10;i++)
    {
                        printf("The value of j in the thread %d is
                            %d\n",omp_get_thread_num(),j);
     j++;
    } printf("The value of j outside the parallel
  region is %d\n",j);
 end = clock();
  double diff = (end-start)*1.0/CLOCKS_PER_SEC;
printf("The execution time is %lf seconds\n",diff);
}
```

Output:

```
The value of j in the thread 3 is 0
The value of j in the thread 3 is 1
The value of j in the thread 1 is 0
The value of j in the thread 1 is 1
The value of j in the thread 1 is 2
The value of j in the thread 2 is 0
The value of j in the thread 2 is 1
The value of j in the thread 0 is 0
The value of j in the thread 0 is 1
The value of j in the thread 0 is 2
The value of j outside the parallel region is 2
The execution time is 0.000419 seconds
```

Graph:



D.ordered:

Code:

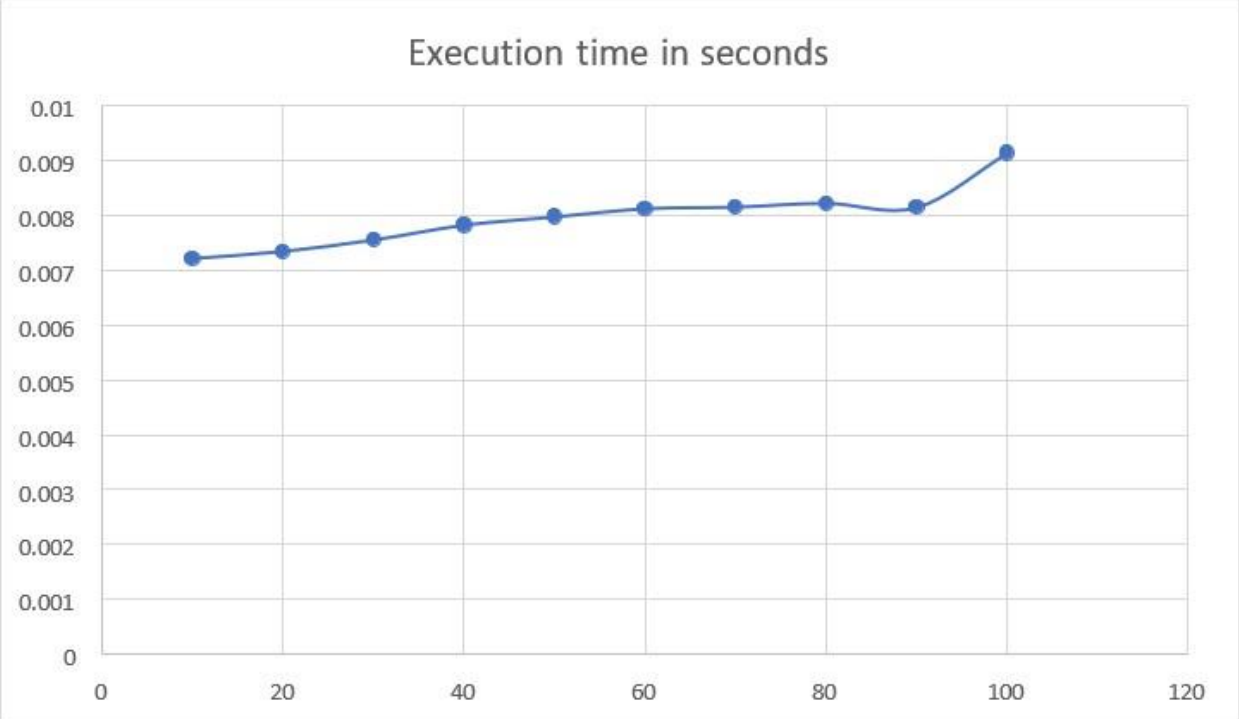
```
#include<stdio.h>

#include<omp.h>

#include<stdlib.
 h>
 #include<time.h
 > void main()
```

```
{
   time_t
   start,end;
   start =
   clock(); int j
   = 0;
   #pragma omp parallel
   { int x
      = 0;
      #pragma omp for
      ordered for(int
      i=0;i<10;i++)
      { x =
         x+i;
         #pragma omp ordered printf("in i=%d x=%d
         thread=%d\n",i,x,omp_get_thread_num());
      }
   }
   end = clock(); double diff = (end-
   start)*1.0/CLOCKS_PER_SEC;
  printf("The execution time is %lf seconds\n",diff);
}
```

Output:

```
in i=0 x=0 thread=0
in i=1 x=1 thread=1
in i=2 x=2 thread=2
in i=3 x=3 thread=3
in i=4 x=4 thread=4
in i=5 x=5 thread=5
in i=6 x=6 thread=6
in i=7 x=7 thread=7
in i=8 x=8 thread=8
in i=9 x=9 thread=9
The execution time is 0.000721 seconds
```

Graph:



CSE4001-PARALLEL AND DISTRIBUTED COMPUTING

Name: O G Ragavi

Reg No: 20BCE1988

Date: 9-9-2022

Lab Ex: 5

—----------

1.Finding the prime numbers in the given range:(for 2^1 to 2^17)

Algorithm:

- We use a simple sieve algorithm to find the prime numbers in a given range.
- We run a loop for 1-5 threads ; each time we set the number of threads from 1-5 and run the function to find the number of prime numbers within the range.
- Start the clock before setting the thread to the corresponding number and close the clock after printing the count value.

Code:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#include<limits.
h>
#include<time.h
> int main()
{
```

```
int
prime[1000000];
long long i, j, n;
clock_t start,end;
double
cpu_time_used;
  printf("\nEnter the value of n");
scanf("%lld",&n);
for(int
k=0;k<5;k++) {
start=clock();
omp_set_num_thr
eads(k+1);


for(i=1;i<=n;i++)
{
   prime[i]=1;
}
prime[1]=0;
for(i=2;i*i<=n;i
++) {

      #pragma omp parallel for
      for(j=i*i;j<=n;j=j+i)
       {
          if(prime[j]
          ==1)
            prime[j]=0;
```

```
        }
}
int count = 0;
for(i=2;i<=n;i++)
{ if(prime[i] ==
   1)
   {
      count ++;
   }
} printf("%d\n",
count);
end=clock();
cpu_time_used=(
(double)(end-
start))/CLOCKS_P
ER_SEC;
printf("Execution
Time is:%lf for
%d
threads\n",cpu_t
ime_used,k+1);
```

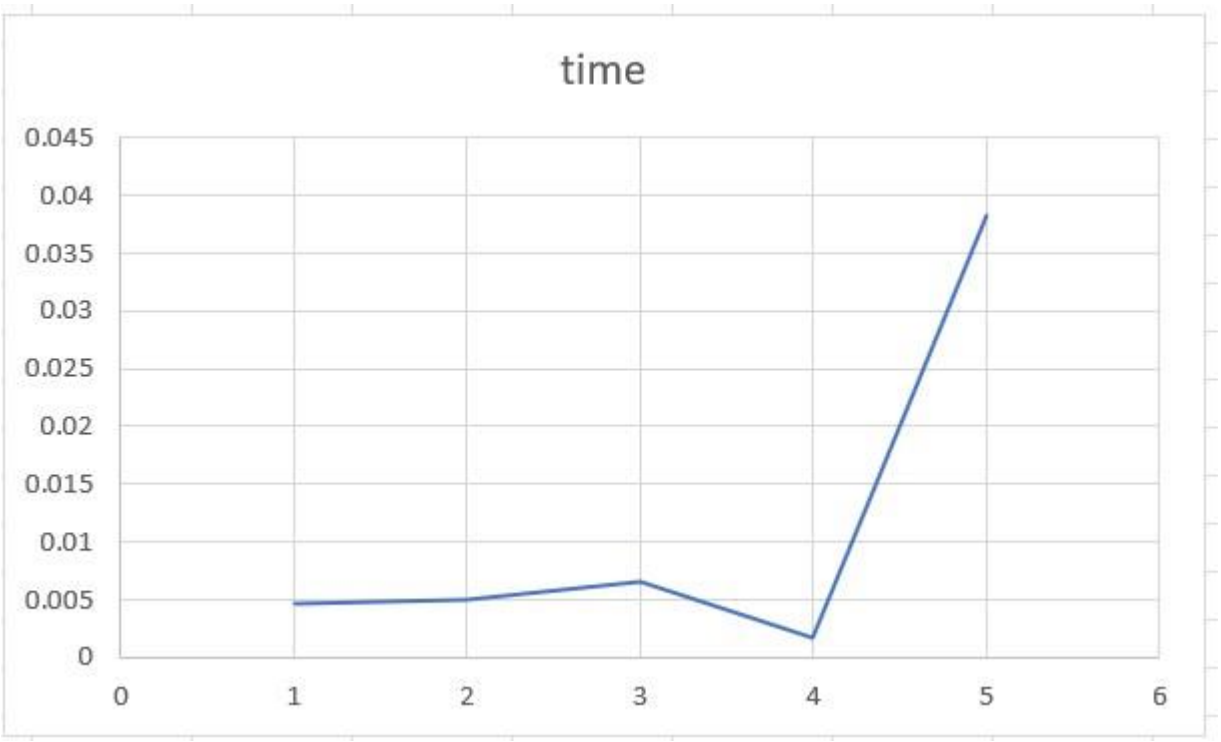
```
    }
}
```

Output:

```
Enter the value of n 131072
12251
Execution Time is:0.004724 for 1 threads
12251
Execution Time is:0.005071 for 2 threads
12251
Execution Time is:0.006595 for 3 threads
12251
Execution Time is:0.001666 for 4 threads
12251
Execution Time is:0.038366 for 5 threads
```

Graph:

n=131072	
No of threads	time
1	0.004724
2	0.005071
3	0.006595
4	0.001666
5	0.038366


time
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
1
2
3
4
5
6

CSE4001-PARALLEL AND DISTRIBUTED COMPUTING

lab-6

Name: O G Ragavi

Reg No: 20BCE1988

Lab: 6

—----------

1.OMP program to multiply 2 matrices:

Code:

```
// C program to multiply two
square matrices.
#include <stdio.h>
#include<math.h>
#include<omp.h>
#include<time.h>
#define N 4

// This function multiplies mat1[][]
and mat2[][],
// and stores the result in res[][]

void multiply(int mat1[][N], int
mat2[][N], int res[][N])
{ int
  i, j,
  k;
  #pragma omp parallel for
  collapse(2) schedule(static) for (i
  = 0; i < N; i++) { for (j = 0; j < N;
  j++) { res[i][j] = 0; for (k = 0; k <
```

```
   N; k++) res[i][j] += mat1[i][k] *
   mat2[k][j]; }
   }
}

int main()
{ clock_t
     s,e;
     s=cl
     ock()
     ;
   int mat1[N][N] = { { 1, 1, 1, 1 },
            { 2, 2, 2, 2 },
            { 3, 3, 3, 3 },
            { 4, 4, 4, 4 } };

  int mat2[N][N] = { { 1, 1, 1, 1 },
            { 2, 2, 2, 2 },
       { 3, 3, 3, 3 },
       { 4, 4, 4, 4 } };

  int res[N][N]; // To
   store result int i, j;
   multiply(mat1, mat2, res);

  printf("Result
   matrix is \n");
   for (i = 0; i <
   N; i++) { for (j
   = 0; j < N; j++)
 printf("%d ", res[i][j]);
     printf("\n");
   }
     e=clock();
     printf("Time
```

```
    taken %ld ",(e-
    s));
  return 0;
}
```

matrix_mul.c

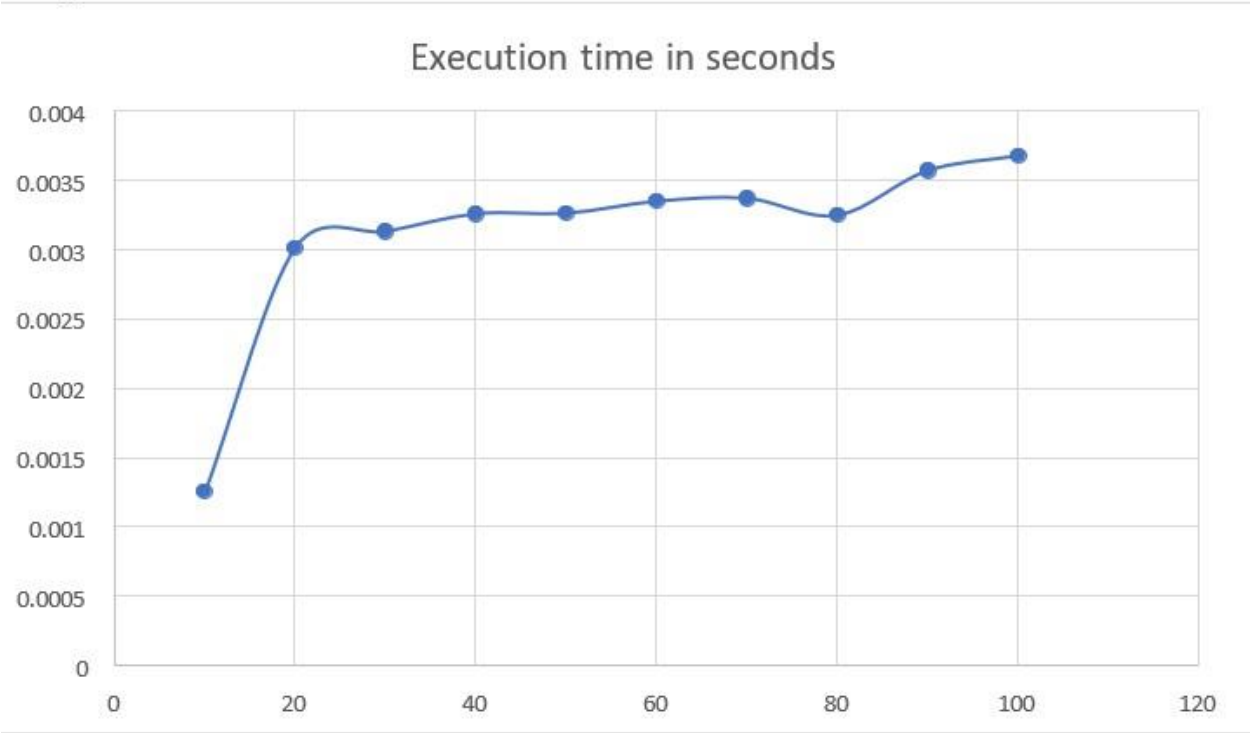
```c
// C program to multiply two square matrices.
#include <stdio.h>
#include<math.h>
#include<omp.h>
#include<time.h>
#define N 4
//20BCE1988- O G RAGAV1
// This function multiplies mat1[][] and mat2[][],
// and stores the result in res[][]

void multiply(int mat1[][N], int mat2[][N], int res[][N])
{
        int i, j, k;
        #pragma omp parallel for collapse(2) schedule(static)
        for (i = 0; i < N; i++) {
                for (j = 0; j < N; j++) {
                        res[i][j] = 0;
                        for (k = 0; k < N; k++)
                                res[i][j] += mat1[i][k] * mat2[k][j];
                }
        }
}

int main()
{
        clock_t s,e;
        s=clock();
        int mat1[N][N] = { { 1, 1, 1, 1 },
                                        { 2, 2, 2, 2 },
                                        { 3, 3, 3, 3 },
                                        { 4, 4, 4, 4 } };

        int mat2[N][N] = { { 1, 1, 1, 1 },
                                        { 2, 2, 2, 2 },
                                        { 3, 3, 3, 3 },
                                        { 4, 4, 4, 4 } };

        int res[N][N]; // To store result
        int i, j;
```

C Tab Width: 8 Ln 7, Col 24 INS

Output:

```
ex2@AB1210SCOPE70: ~
File  Edit  View  Search  Terminal  Help
ex2@AB1210SCOPE70:~$ gcc -o mul matrix_mul.c
ex2@AB1210SCOPE70:~$ ./mul
Result matrix is
10 10 10 10
20 20 20 20
30 30 30 30
40 40 40 40
ex2@AB1210SCOPE70:~$ gcc -o mul matrix_mul.c
ex2@AB1210SCOPE70:~$ ./mul
Result matrix is
10 10 10 10
20 20 20 20
30 30 30 30
40 40 40 40
ex2@AB1210SCOPE70:~$ gcc -o mul -fopenmp matrix_mul.c
ex2@AB1210SCOPE70:~$ ./mul
Result matrix is
9 10 10 10
20 20 20 20
30 30 30 30
40 60 40 40
ex2@AB1210SCOPE70:~$ gcc -o mul -fopenmp matrix_mul.c
ex2@AB1210SCOPE70:~$ ./mul
Result matrix is
10 10 10 10
20 20 20 20
30 30 30 30
20 40 40 40
Time taken 2726 ex2@AB1210SCOPE70:~$ gcc -o tsp tsp.c
gcc: error: tsp.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
ex2@AB1210SCOPE70:~$ gcc -o tsp tsp.c
ex2@AB1210SCOPE70:~$ ./tsp.c
bash: ./tsp.c: Permission denied
ex2@AB1210SCOPE70:~$ ./tsp
Enter Total Number of Cities:   4

Enter Cost Matrix
```

Graph:



2.OMP Program-Traveling salesman problem:

Code:

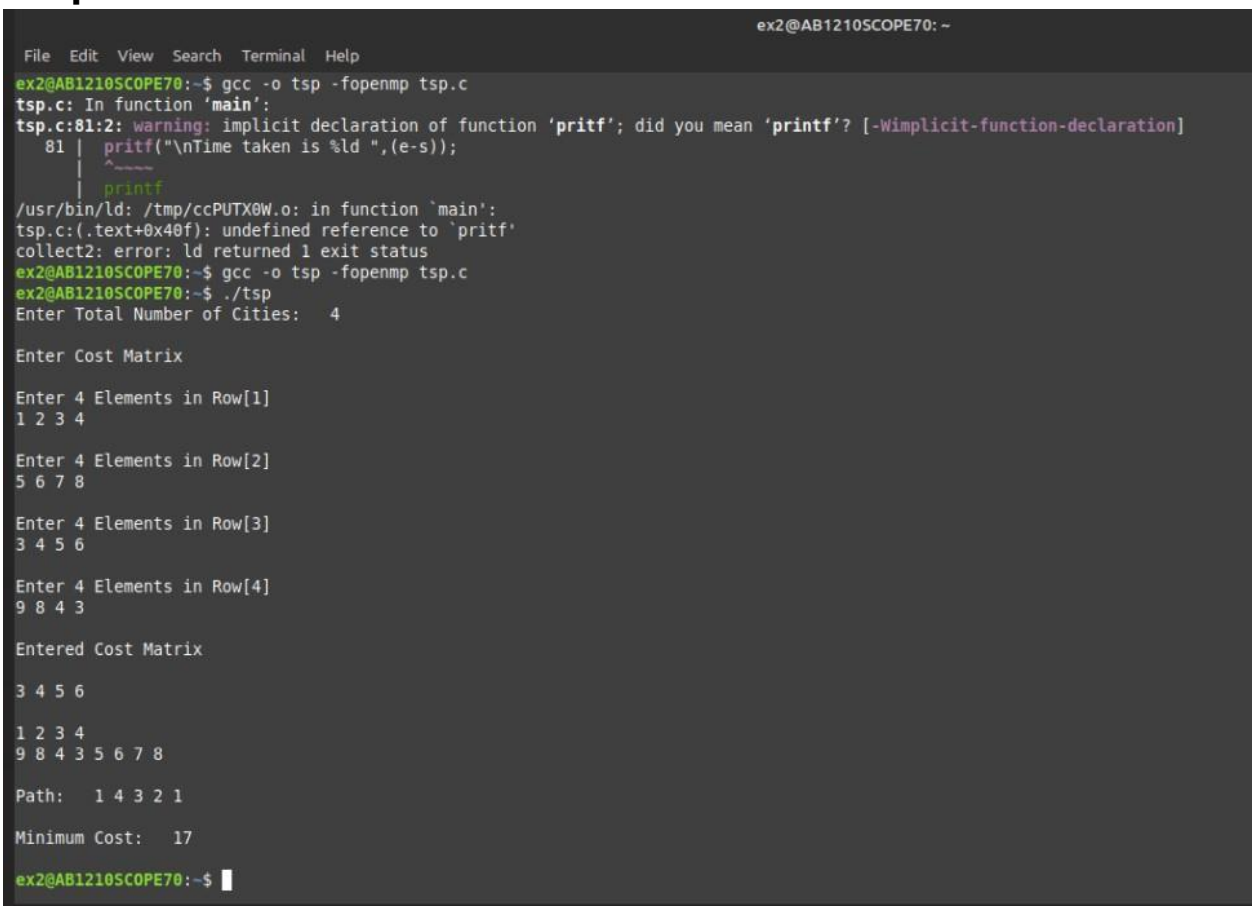
```
#include <stdio.h>
#include<omp.h>
#include<time.h>
#include<stdlib.h>
//20BCE1988- O G RAGAVI
int matrix[25][25], visited_cities[10], limit, cost
= 0;

int tsp(int c)
{
int count, nearest_city =
999; int minimum = 999,
temp; for(count = 0; count
< limit; count++)
{ if((matrix[c][count] != 0) &&
(visited_cities[count] == 0))
{
```

```
if(matrix[c][count] < minimum)
{
minimum = matrix[count][0] +
matrix[c][count];
}
temp = matrix[c][count];
nearest_city = count;
}
}
if(minimum != 999)
{
cost = cost + temp;
}
return nearest_city;
}

void minimum_cost(int city)
{ int
nearest_city;
visited_cities[cit
y] = 1;
printf("%d ", city
+ 1);
nearest_city =
tsp(city);
if(nearest_city
== 999)
{
nearest_city = 0; printf("%d",
nearest_city + 1); cost =
cost +
matrix[city][nearest_city];
return;
```

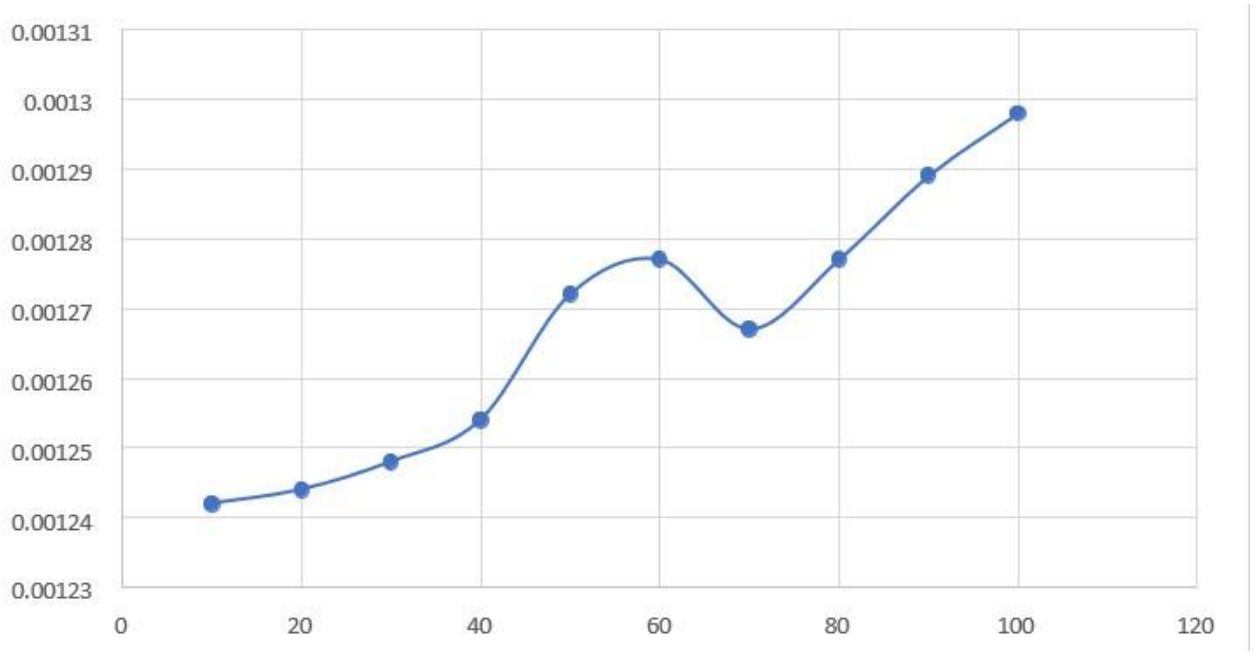
```
}
minimum_cost(nearest_city);
}

int main()
{
int
i,
j;
clock_t s,e; s=clock();
printf("Enter Total Number of
Cities:\t"); scanf("%d",
&limit); printf("\nEnter Cost
Matrix\n");
for(i = 0; i < limit; i++)
{
printf("\nEnter %d Elements in Row[%d]\n",
limit, i + 1); for(j = 0; j < limit; j++)
{ scanf("%d",
&matrix[i][j]);
}
visited_cities
[i] = 0;
}

printf("\nEntered Cost Matrix\n");
  #pragma omp parallel for
for(i = 0; i < limit; i++)
{ printf("\n"); for(j
   = 0; j < limit;
   j++)
   { printf("%d ",
     matrix[i][j]);
```

```
   } } printf("\n\nPath:\t");
minimum_cost(0);
printf("\n\nMinimum Cost:
\t"); printf("%d\n", cost);
e=clock(); printf("\nTime
taken is %ld ",(e-s));
return 0; }
```

Output:



Graph:


0.00131
0.0013
0.00129
0.00128
0.00127
0.00126
0.00125
0.00124
0.00123
0
20
40
60
80
100
120

CSE-4001-Parallel and Distributed computing

Name: O G RAGAVI

Reg No: 20BCE1988

Title: Scholarly articles on OpenMP

Lab : 7

Scholarly article 1: Research on OpenMP Model of the Parallel
Programming Technology for Homogeneous Multicore DSP

Abstract:

As application complexity continues to grow, using multicore processors has been proved to be an effective methodology to meet the ever-increasing processing demand across the industry association. The Master/Slave model, the Data Flow model and the OpenMP model are the three dominant models for parallel programming. In this paper, the first two models are briefly discussed while the OpenMP model is focused. Some factors (e.g. the number of threads, the scheduling strategy, the load balance, etc.) that affect the execution performance of OpenMP programs were also studied in this paper.

Synopsis:

The use of parallel programming in the IT industry has grown significantly over the past few years.
Using the multi core processors to implement high performance computing has become the consensus of the industry. OpenMP is widely used in general purpose processors for its scalability and flexibility. Different hardware platforms correspond to different parallel programming models. The method using the OpenMP to realize the rapid and efficient image edge detection is presented in this paper. This paper presents a method of taking advantage of the OpenMP model to realize the image edge detection within the platform of TMS320C6678 Digital Signal Processors. The modification of the program only involves

the number of cores which proves the Open MP model has a good scalability.

Introduction:

- Due to the limitation of the CMOS technology, it is difficult to continuously improve the frequency of the processor chip after it reaches 4GHz. Moore's Law leading the semiconductor market nearly 40 years may fail in the next 10 to 20 years. Using the multi core processors to implement high performance computing has become the consensus of the industry. Multicore Digital Signal Processors (DSP) are widely used in wireless telecommunication, industrial control, audio and video processing, etc. As embedded multicore hardware enables more functions to be implemented on the same device, efficient parallel programming methods are required to achieve desired performance without increasing software complexity.
- OpenMP is now widely used in general purpose processors for its scalability and flexibility, while the application in multicore DSP is still in the initial stage. The state-of-the-art approach to program multi-core DSPs is based on proprietary vendor Software Development Kits(SDKs), which only provides low-level, non-portable primitives
- Different hardware platforms correspond to different parallel programming models. This paper will take this platform as an instance, analyze the number of threads, the scheduling strategy on the performance of the OpenMP model. The method using the OpenMP to realize the rapid and efficient image edge detection is also presented in this paper.

Highlights:

- This paper presents a method of taking advantage of the OpenMP model to realize the image edge detection within the platform of TMS320C6678 Digital Signal Processors (DSP)
- We study the two main factors which will influence the performance of

  OpenMP model

- The best performance can be obtained at the point where the number of threads is equal to the number of cores which are available within the platform
- In terms of image edge detection, we adopt the runtime scheduling strategy which gets a better performance compared to others
- The modification of the program only involves the number of cores which proves the OpenMP model has a good scalability.

Study subjects and analysis:

The experimental results show that the OpenMP model has a better advantage on scalability and flexibility compared to the Master/Slave model and the Data Flow model.The Data Flow model is used for distributed control and execution **Results:**

TABLE I. EXECUTION TIME VERSUS NUMBER OF THREADS

	Number of threads		
	4	8	12
Execution Time/cycle	17112903	10353808	16902841

The table shows the text result that execution time varies with the number within the DSP platform. As can be seen, the execution time will bottom at a point where the number of threads is equal to the number of cores which are available within the platform. If the number of threads is less than the number of cores, then there will be some cores in idle state. Inversely, too many threads will result in the increasing overhead of threads creating and switching. Both conditions above are not conducive to performance improvement.

- When the SIZE is 80, then there are 80x80 multiply-add operations, the execution time is 19.2ms when the schedule(static) is specified. Since the main objective is to compare the pros and cons between various scheduling mechanisms, we want to calculate the relative value among them.

Conclusion:

OpenMP is one of the most widely used parallel programming techniques in the modern multicore era. But applying this model to homogeneous multicore DSP is still a great challenge. In this paper, we studied the two main factors which will influence the performance of the OpenMP model.

The best performance can be obtained at the point where the number of threads is equal to the number of cores which are available within the platform. The scheduling strategy is determined by the specific application. In terms of image edge detection, we adopt the runtime scheduling strategy which gets a better performance compared to others. For the sake of validating the speedup, we use 1 to 8 cores to realize the edge detection and record the respective cost. The modification of the program only involves the number of cores which proves the OpenMP model has a good scalability. **The conclusion that the speedup increases linearly with the number of cores satisfies Gustafson's law. What's more, under the case of 8 cores running simultaneously, the speedup reaches 7.233.** Compared with the Master/Slave model, the OpenMP model has approximately improved one-third on performance.

Scholarly article2: The Design of OpenMP Tasks

Abstract:

OpenMP has been very successful in exploiting structured parallelism in applications. With increasing application complexity, there is a growing need for addressing irregular parallelism in the presence of complicated control structures. This is evident in various efforts by the industry and research communities to provide a solution to this challenging problem. One of the primary goals of OpenMP was to define a standard dialect to express and to exploit unstructured parallelism efficiently.This paper presents the design goals and key features of the tasking model, including a rich set of examples and an in-depth discussion of the rationale behind various design choices. It compares a prototype implementation of the tasking model with existing models, and evaluates it on a wide range of applications. The comparison shows that the OpenMP tasking model provides expressiveness, flexibility, and huge potential for performance and scalability

Introduction:

In the last few decades, OpenMP has emerged as the de facto standard for shared-memory parallel programming. OpenMP provides a simple and flexible interface for developing portable and scalable parallel applications. OpenMP grew in the 1990s out of the need to standardize the different vendor specific directives related to parallelism. It was structured around parallel loops and was meant to handle dense numerical applications.

The set of features in the OpenMP 2.5 specification is ill equipped to exploit the concurrency available in modern applications.Users now need a simple way to identify independent units of work and not concern themselves with scheduling these work units. This model is typically called “tasking”.With the goal of defining a simple tasking dialect for expressing irregular and unstructured parallelism, a subcommittee of the OpenMP 3.0 language committee was formed.

Nested Parallelism Vs Tasking:

New OpenMP tasks allow a programmer to express parallelism that in OpenMP 2.5 would be expressed using nested parallelism. The versions using nested OpenMP, while simple to write, usually do not perform well because of a variety of problems (load imbalance, synchronization overheads etc.).Handling recursive code structures is another simple case that motivated tasking in OpenMP was recursive work generation.Nested parallelism can be used to allow recursive work generation but at the expense of the overhead in creating a rigid tree structure of thread teams and their associated (unnecessary) implicit barriers.

Prototype implementation:

In order to test the proposal in terms of expressiveness and performance, they have developed their own implementation of the proposed tasking model . They developed the prototype on top of a research OpenMP compiler (source-to-source restructuring tool) and runtime infrastructure. The runtime infrastructure is an implementation of a user level thread package based on the nano-threads programming model introduced first by Polychronopoulos.

The nano-thread layer is implemented on top of POSIX Threads (also known as pthreads).They decided to use pthreads to ensure that they will be portable across a wide range of systems. This layered implementation can have a slight impact on efficiency.
Once the task is first executed by a thread, and if the task has task scheduling points, we can expect two different behaviors. First, the task is bound to that thread (so, it can only be executed by that thread), and second, the task is not attached to any thread and can be executed by any other thread of the team. The library offers the possibility to move a task from the team queues to the local queues. This ability covers the requirements of the united clause of the task construct, which allows a task suspended by one thread to be resumed by a different one. The synchronization construct is provided through task counters that keep track of the number of tasks that are created in the current scope (i.e., the current task). Each task data structure has a successor field that points to the counter the task must decrement.

Conclusion:

They have presented the work of the OpenMP 3.0 tasking subcommittee which was a proposal **to integrate task parallelism into the OpenMP specification**. This proposal allows programmers to **parallelize program structures like while loops and recursive functions more easily and efficiently**. We have shown that, in fact, these structures are easy to parallelize with the new proposal.

The process of defining the proposal has not been without difficult decisions, as they tried to achieve conflicting goals: simplicity of use, simplicity of specification, and consistency with the rest of OpenMP.Their discussions identified trade-offs between the goals, and our decisions reflected our best judgments of the relative merits of each. The comparisons of these results show that expressiveness is not incompatible with performance and the OpenMP tasks implementation can achieve very promising speedups when compared to other established models. Overall, **OpenMP tasks provide a balanced, flexible, and very expressive dialect for expressing unstructured parallelism** in OpenMP programs.

CSE4001-Parallel and distributed computing Name: OG RAGAVI

Reg No: 20BCE1988

Lab: 8

Steps:

PARALLEL AND DISTRIBUTED COMPUTING

Initial set up

Lab 8

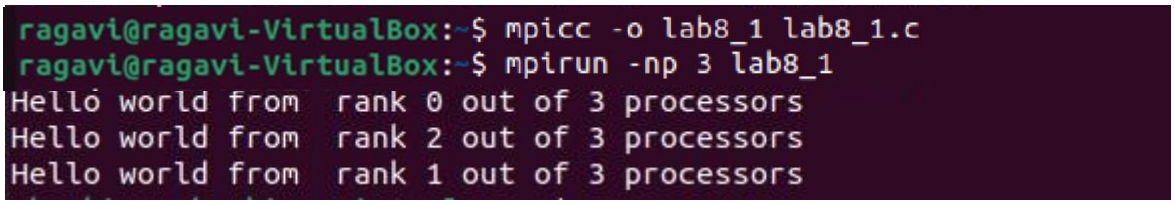
```
sudo apt-get update

sudo apt-get install libopenmpi-dev
sudo apt-get install openmpi-bin

sudo apt-get install mpich
```

1.Hello World Program

```
#includ
e
<mpi.h>
#includ
e
<stdio.h
>
int main(int
argc, char**
argv) {     //
Initialize the
MPI
environment
MPI_Init(NULL,
NULL);

    // Get the
number of
processes     int
world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

// Print off a hello world message
    printf("Hello world from  rank %d out of %d processors\n",
world_rank, world_size);
```

```
    // Finalize the MPI environment.
    MPI_Finalize();
}
```

output:



2. Sum of Array Program

CODE:

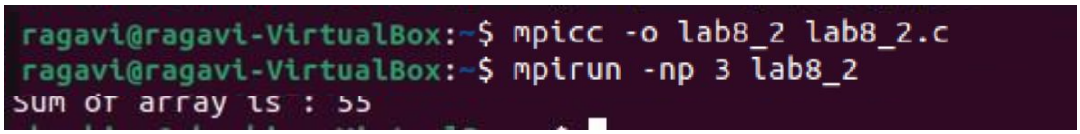
```
#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

// size
of
array
#defin
e n 10

int a[] = { 1, 2, 3, 4, 5, 6, 7,
8, 9, 10 }; // Temporary
array for slave process int
a2[1000];

int main(int argc, char* argv[])

{

int pid, np,elements_per_process,n_elements_recieved;

// np -> no. of processes

// pid -> process id

MPI_Status status;
```

```
// Creation of parallel processes
MPI_Init(&argc, &argv);

// find out process ID,
// and how many processes were started
MPI_Comm_size(MPI_COMM_WORLD,&np);
MPI_Comm_rank(MPI_COMM_WORLD,&pid);

//
m
as
te
r
pr
oc
es
s
if
(p
id
=
=
0)
{
int index, i;
elements_per_process =n/np;

// check if more than 1
processes are run if (np > 1) {
// distributes the portion of array
// to child processes to calculate
// their partial sums
for (i = 1; i < np - 1;
i++) { index = i *
elements_per_proce
ss;
MPI_Send(&elements_per_process, 1, MPI_INT, i, 0,MPI_COMM_WORLD);
MPI_Send(&a[index], elements_per_process,MPI_INT, i, 0,
MPI_COMM_WORLD);
```

```
}

// last process adds remaining
elements index = i *
elements_per_process;

int elements_left = n - index;

MPI_Send(&elements_left, 1, MPI_INT,i, 0, MPI_COMM_WORLD);

MPI_Send(&a[index], elements_left,MPI_INT, i, 0, MPI_COMM_WORLD);

}

// master process add its own
sub array int sum = 0;

for (i = 0; i <
elements_per_process;i++) sum +=
a[i];

// collects partial sums from other
processes int tmp;

for (i = 1; i < np; i++) {

MPI_Recv(&tmp, 1, MPI_INT, MPI_ANY_SOURCE,

0,MPI_COMM_WORLD,
&status); int sender =
status.MPI_SOURCE; sum
+= tmp;

}

// prints the final sum of array
printf("Sum of array is : %d\n",
sum);

}

//
slave
proce
sses
else {

MPI_Recv(&n_elements_recieved, 1, MPI_INT, 0,0,MPI_COMM_WORLD,
```

```
&status);

// stores the received array segment
// in local array a2
MPI_Recv(&a2, n_elements_recieved, MPI_INT, 0,0,MPI_COMM_WORLD,
&status);

// calculates its
partial sum int
partial_sum = 0;
for (int i = 0; i < n_elements_recieved;i++)
partial_sum += a2[i];

// sends the partial sum to the root process
MPI_Send(&partial_sum, 1, MPI_INT,0, 0, MPI_COMM_WORLD);
}
// cleans up all MPI state before exit of process
MPI_Finalize();

return 0;
}
```

output:



CSE4001-Parallel and distributed computing

Name: OG RAGAVI

Reg No: 20BCE1988

Lab Ex: 9

Date: 27.09.2022

Important commands:

- **MPI_Init (NULL, NULL)** –Initializes the MPI program.
- **MPI_Comm_size(MPI_COMM_WORLD,&no of processors)** – tells how many processors are there in communication.
- **int rank** – indicates the processor id according to its priority.
- **int MPI_Send(void *data_to_send, int send_count, MPI_Datatype send_type, int destination_ID, int tag, MPI_Comm comm)**- used to send the data to the destination processor.Tag value depends on the number of different processes allocated to the system.If we have only one function, we'll use tag=0.
- **int MPI_Recv(void *received_data, int receive_count, MPI_Datatype receive_type,int sender_ID, int tag, MPI_Comm comm, MPI_Status *status) –**used to receive data from a particular processor.Status maintains the success result.

Q1) Program to sort an array of 100 numbers.

Algorithm:

- Initialize the MPI program to run for multiple processors.
- Using rand() function , take input of 100 random numbers.
- Divide the program to np processors= no of elements in the array/np.

- Initially the master program gets into action and sorts the first 25 elements(in case of 100 numbers , as 100/4 processors= 25).
- Then it passes the index one by one (incrementing by 25) to the processor with pid 1, 2 and so on till np.
- Each slave computes the sorted array of their particular subpart. For eg)Pid=1 sorts the array from index 26 to 50 and so on.
- These partially sorted arrays are sent to the master processor , which are finally merged and sorted once again to display the output>

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
//sort for 100 numbers
void merge(int *, int *, int, int, int);
void mergeSort(int *, int *, int, int);

int main(int argc, char** argv) {

    /********** Create and populate the array **********/
    int n = atoi(argv[1]);
    int *original_array = malloc(n * sizeof(int));

    int c;
    srand(time(NULL));
```

```
printf("The initial unsorted array is: ");
for(c = 0; c < n; c++) {

        original_array[c] = rand() % n;
        printf("%d ", original_array[c]);

        }

printf("\n");
printf("\n");

/********** Initialize MPI **********/
int world_rank;//priority given to the processor
int world_size;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

/********** Divide the array in equal-sized chunks **********/
int size = n/world_size;

/********** Send each subarray to each process **********/
int *sub_array = malloc(size * sizeof(int));
MPI_Scatter(original_array, size, MPI_INT, sub_array, size, MPI_INT, 0,
MPI_COMM_WORLD);
```

```
        /********** Perform the mergesort on each process **********/
        int *tmp_array = malloc(size * sizeof(int));
        mergeSort(sub_array, tmp_array, 0, (size - 1));

        /********** Gather the sorted subarrays into one **********/
        int *sorted = NULL;
        if(world_rank == 0) {

                sorted = malloc(n * sizeof(int));

                }

        MPI_Gather(sub_array, size, MPI_INT, sorted, size, MPI_INT, 0,
MPI_COMM_WORLD);

        /********** Make the final mergeSort call **********/
        if(world_rank == 0) {

                int *other_array = malloc(n * sizeof(int));
                mergeSort(sorted, other_array, 0, (n - 1));

                /********** Display the sorted array **********/
                printf("The final sorted array is: ");
                for(c = 0; c < n; c++) {

                        printf("%d ", sorted[c]);
```

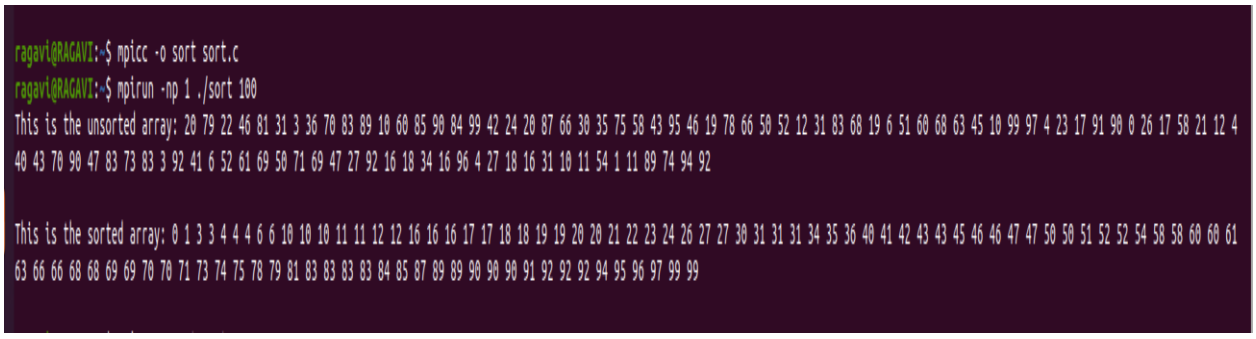
```
                }

                printf("\n");
                printf("\n");

                /********** Clean up root **********/
                free(sorted);
                free(other_array);

                }

        /********** Clean up rest **********/
        free(original_array);
        free(sub_array);
        free(tmp_array);

        /********** Finalize MPI **********/
        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Finalize();

        }

/********** Merge Function **********/
void merge(int *a, int *b, int l, int m, int r) {

        int h, i, j, k;
```

```
h = l;
i = l;
j = m + 1;

while((h <= m) && (j <= r)) {

        if(a[h] <= a[j]) {

                b[i] = a[h];
                h++;

                }

        else {

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

                }

        i++;

        }

if(m < h) {
```

```
        for(k = j; k <= r; k++) {

                b[i] = a[k];
                i++;

                }

        }

else {

        for(k = h; k <= m; k++) {

                b[i] = a[k];
                i++;

                }

        }

for(k = l; k <= r; k++) {

        a[k] = b[k];

        }
```

```
        }

/********** Recursive Merge Function **********/
void mergeSort(int *a, int *b, int l, int r) {

        int m;

        if(l < r) {

                m = (l + r)/2;

                mergeSort(a, b, l, m);
                mergeSort(a, b, (m + 1), r);
                merge(a, b, l, m, r);

                }

        }
```

Output:



Q2) Program to perform matrix multiplication:

Code:

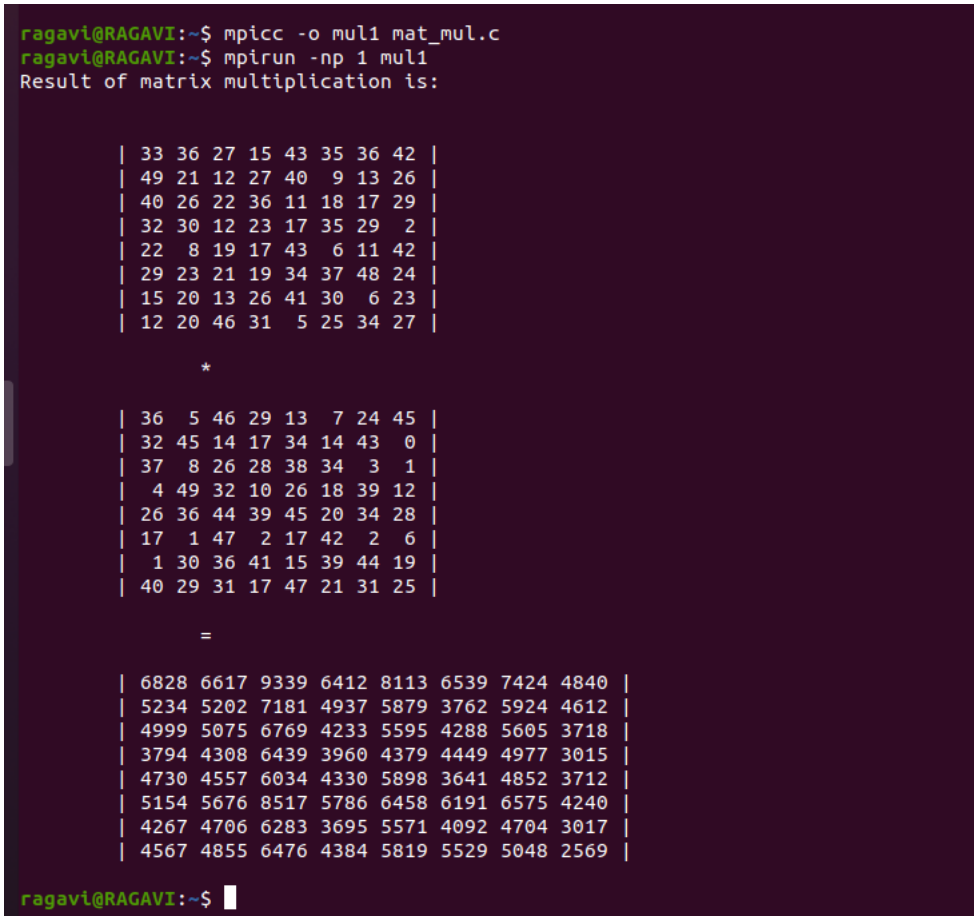
```
#include <mpi.h>
#include <stdio.h>
#include<stdlib.h>
#include<math.h>
#define SIZE 8                          /* Size of matrices */

int A[SIZE][SIZE], B[SIZE][SIZE], C[SIZE][SIZE];

void fill_matrix(int m[SIZE][SIZE])
{
  static int n=0;
  int i, j;
  for (i=0; i<SIZE; i++)
    for (j=0; j<SIZE; j++)
      //m[i][j] = n++;
      m[i][j]=rand()%50;
}

void print_matrix(int m[SIZE][SIZE])
{
  int i, j = 0;
  for (i=0; i<SIZE; i++) {
    printf("\n\t| ");
    for (j=0; j<SIZE; j++)
      printf("%2d ", m[i][j]);
    printf("|");
```

```
  }
}


int main(int argc, char *argv[])
{
  int myrank, P, from, to, i, j, k;
  int tag = 666;                /* any value will do */
  MPI_Status status;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);       /* who am i */
  MPI_Comm_size(MPI_COMM_WORLD, &P); /* number of processors */

  /* Just to use the simple variants of MPI_Gather and MPI_Scatter we */
  /* impose that SIZE is divisible by P. By using the vector versions, */
  /* (MPI_Gatherv and MPI_Scatterv) it is easy to drop this restriction. */

  if (SIZE%P!=0) {
    if (myrank==0) printf("Matrix size not divisible by number of processors\n");
    MPI_Finalize();
    exit(-1);
  }

  from = myrank * SIZE/P;
  to = (myrank+1) * SIZE/P;
```

```
 /* Process 0 fills the input matrices and broadcasts them to the rest */
 /* (actually, only the relevant stripe of A is sent to each process) */

 if (myrank==0) {
  fill_matrix(A);
  fill_matrix(B);
 }

 MPI_Bcast (B, SIZE*SIZE, MPI_INT, 0, MPI_COMM_WORLD);
 MPI_Scatter (A, SIZE*SIZE/P, MPI_INT, A[from], SIZE*SIZE/P, MPI_INT, 0,
MPI_COMM_WORLD);

 printf("Result of matrix multiplication is:");
 for (i=from; i<to; i++)
  for (j=0; j<SIZE; j++) {
   C[i][j]=0;
   for (k=0; k<SIZE; k++)
       C[i][j] += A[i][k]*B[k][j];
  }

 MPI_Gather (C[from], SIZE*SIZE/P, MPI_INT, C, SIZE*SIZE/P, MPI_INT, 0,
MPI_COMM_WORLD);

 if (myrank==0) {
  printf("\n\n");
  print_matrix(A);
```

```
    printf("\n\n\t      * \n");
    print_matrix(B);
    printf("\n\n\t      = \n");
    print_matrix(C);
    printf("\n\n");
  }

  MPI_Finalize();
  return 0;
}
```

Output:



CSE4001-Parallel and distributed computing

Name: O G RAGAVI

Reg No: 20BCE1988

Lab Ex: 10

Title: Dijshktra's algorithm implementation using open MP and mpi

---

Open MP implementation:

Code:

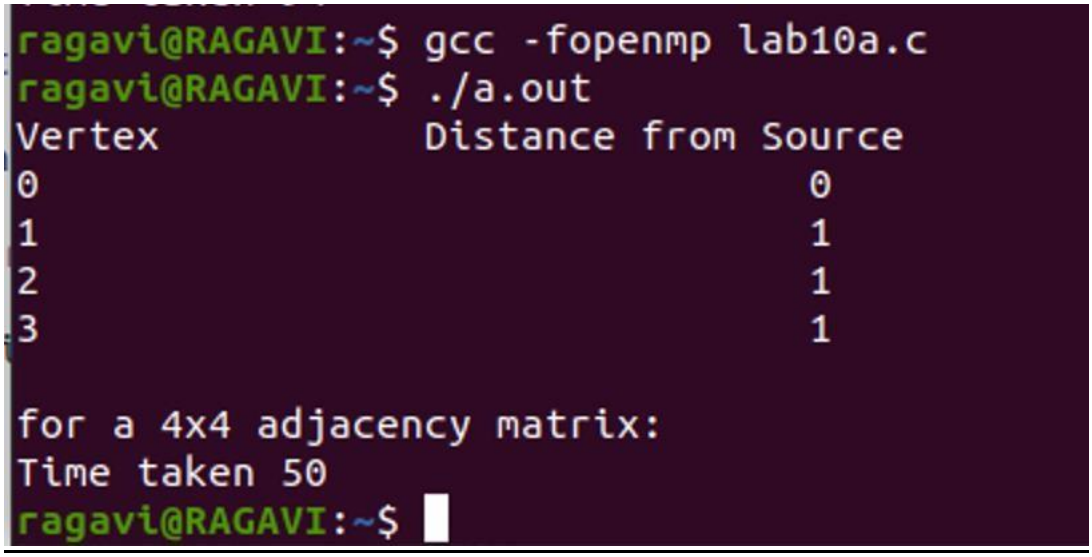
```
#include<omp.h>
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#include<time.h>
// Number of vertices in the graph
#define V 9

int minDistance(int dist[], bool sptSet[])
{
        // Initialize min value     int min =
INT_MAX, min_index;

        for (int v = 0; v < V; v++) if (sptSet[v] == false && dist[v] <=
                min)            #pragma omp critical                    min
                = dist[v], min_index = v;

            return min_index;
}

// A utility function to print the constructed distance
// array
void printSolution(int dist[])
{
```

```
        printf("Vertex \t\t Distance from Source\n");      for (int i = 0;
i < V; i++)                    printf("%d \t\t\t\t %d\n", i, dist[i]);
}


void dijkstra(int graph[V][V], int src)
{
            int dist[V];

        bool sptSet[V];
        for (int i = 0; i < V; i++)
                        dist[i] = INT_MAX, sptSet[i] = false;


            dist[src] = 0;

                #pragma omp parallel for

                    for (int count = 0; count < V - 1; count++) {

                            int u = minDistance(dist, sptSet);

                    // Mark the picked vertex as processed              sptSet[u] = true;

                            for (int v = 0; v < V; v++)

                                    if (!sptSet[v] && graph[u][v]
                                            && dist[u] != INT_MAX
                                               && dist[u] + graph[u][v] < dist[v])
```

```
                                        dist[v] = dist[u] + graph[u][v];
            }

                printSolution(dist);
}

// driver's code int
main()
{

                int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                                                { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                                                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                                                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                                                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                                                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                                                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                                                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

            // Function call
            clock_t s,e;
       s=clock();
            dijkstra(graph, 0);          e=clock();
             printf("\nfor a 9x9 adjacency matrix:");        printf("\nTime taken %ld
\n",(e-s));

               return 0;
}
```

Output:

```
ragavi@RAGAVI:~$ ./a.out
Vertex           Distance from Source
0                              0
1                              4
2                              12
3                              19
4                              21
5                              11
6                              9
7                              8
8                              14

for a 9x9 adjacency matrix:
Time taken 527
```



2.MPI Implementation:

Code:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include "mpi.h"
#define N 5
#define SOURCE 1
#define MAXINT 9999999
void SingleSource(int n, int source, int *wgt, int *lengths, MPI_Comm comm)
{
 int temp[N];
```

```
 int i, j;
 int nlocal; /* The number of vertices stored locally */  int *marker; /* Used to mark
the vertices belonging to Vo */
 int firstvtx; /* The index number of the first vertex that is stored locally
*/
 int lastvtx; /* The index number of the last vertex that is stored locally
*/  int u,
udist; int
lminpair[2],
gminpair[2];
int npes,
myrank;
 MPI_Status status;
 MPI_Comm_size(comm, &npes);  MPI_Comm_rank(comm, &myrank);
 nlocal = n / npes;  firstvtx = myrank *
nlocal;  lastvtx = firstvtx + nlocal - 1;
 /* Set the initial distances from source to all the other vertices */  for (j = 0; j < nlocal; j++)
 {
 lengths[j] = wgt[source * nlocal + j];
 }
 /* This array is used to indicate if the shortest part to a vertex has been found  or not. */
 /* if marker [v] is one, then the shortest path to v has been found, otherwise it  has not. */
 marker = (int *)malloc(nlocal * sizeof(int));  for (j = 0; j < nlocal;
j++)
 {
 marker[j] = 1;
 }
 /* The process that stores the source vertex, marks it as being seen */  if (source >= firstvtx &&
source <= lastvtx)
 {
 marker[source - firstvtx] = 0;
 }
 /* The main loop of Dijkstra's algorithm */  for (i = 1; i < n; i++)
```

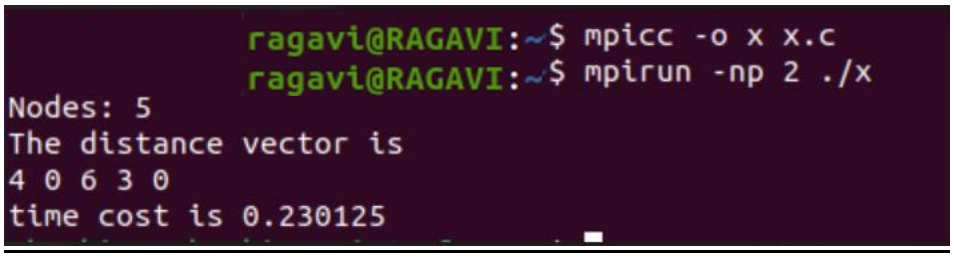
```
{
/* Step 1: Find the local vertex that is at the smallest distance from source */
lminpair[0] = MAXINT; /* set it to an architecture dependent large number
*/
lminpair[1] = -1;  for (j = 0; j <
nlocal; j++)
{
if (marker[j] && lengths[j] < lminpair[0])
{
lminpair[0] = lengths[j];  lminpair[1] = firstvtx +
j;
}
}
/* Step 2: Compute the global minimum vertex, and insert it into Vc */
MPI_Allreduce(lminpair, gminpair, 1, MPI_2INT, MPI_MINLOC, comm);  udist = gminpair[0];  u =
gminpair[1];
/* The process that stores the minimum vertex, marks it as being seen
*/
if (u == lminpair[1])
{
marker[u - firstvtx] = 0;
}
/* Step 3: Update the distances given that u got inserted */  for (j = 0; j < nlocal; j++)
{
if (marker[j] && ((udist + wgt[u * nlocal + j]) < lengths[j]))
{
lengths[j] = udist + wgt[u * nlocal + j];
}
}
}
free(marker);
}
```

```
int main(int argc, char *argv[])
{
 int npes, myrank, nlocal;  int weight[N][N]; /*adjacency matrix*/
int distance[N]; /*distance vector*/  int *localWeight; /*local weight
array*/  int *localDistance; /*local distance vector*/  int sendbuf[N
* N]; /*local weight to distribute*/  int i, j, k;
 char fn[255];
 double time_start, time_end;  struct
timeval tv;  struct timezone tz;
gettimeofday(&tv, &tz);
 time_start = (double)tv.tv_sec + (double)tv.tv_usec / 1000000.00;
 /* Initialize MPI and get system information */
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &npes);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 nlocal = N / npes; /* Compute the number of elements to be stored locally. */
 /*allocate local weight and local disatance arrays for each prosess*/  localWeight = (int
*)malloc(nlocal * N * sizeof(int));
localDistance = (int *)malloc(nlocal * sizeof(int));
/* Open input file, read adjacency matrix and prepare for sendbuf */
 // printf("\nThe adjacency matrix: \n");  for (i = 0; i < N;
i++)
 {
 for (j = 0; j < N; j++)
 {
 weight[i][j] = rand() % 11;
 // if (weight[i][j] == 9999999) printf("%4s", "INT");
 // else printf("%4d", weight[i][j]);
 }
 // printf("\n");
 }
```

```
 /*prepare send data */  for (k = 0; k
< npes; ++k)
 {
 for (i = 0; i < N; ++i)
 {
 for (j = 0; j < nlocal; ++j)
 {
 sendbuf[k * N * nlocal + i * nlocal + j] = weight[i][k * nlocal + j];
 }
}
}
 /*distribute data*/
 MPI_Scatter(sendbuf, nlocal * N, MPI_INT, localWeight, nlocal * N,
MPI_INT, SOURCE,
 MPI_COMM_WORLD);
 /*Implement the single source dijkstra's algorithm*/
 SingleSource(N, SOURCE, localWeight, localDistance,
MPI_COMM_WORLD);
 /*collect local distance vector at the source process*/
 MPI_Gather(localDistance, nlocal, MPI_INT, distance, nlocal, MPI_INT,
SOURCE,
 MPI_COMM_WORLD);  if
(myrank == SOURCE)
 {
 printf("Nodes: %d\n", N);  printf("The distance
vector is \n");  for (i = 0; i < N; ++i)
 {
 printf("%d ", distance[i]);
 }
 printf("\n");
gettimeofday(&tv, &tz);
time_end = (double)tv.tv_sec + (double)tv.tv_usec / 1000000.00;
```

```
 printf("time cost is %1f\n", time_end - time_start);
 }
 free(localWeight);
free(localDistance);
MPI_Finalize();  return 0;
}
```

Output:



CSE4001-Parallel and distributed computing

Name: OG RAGAVI

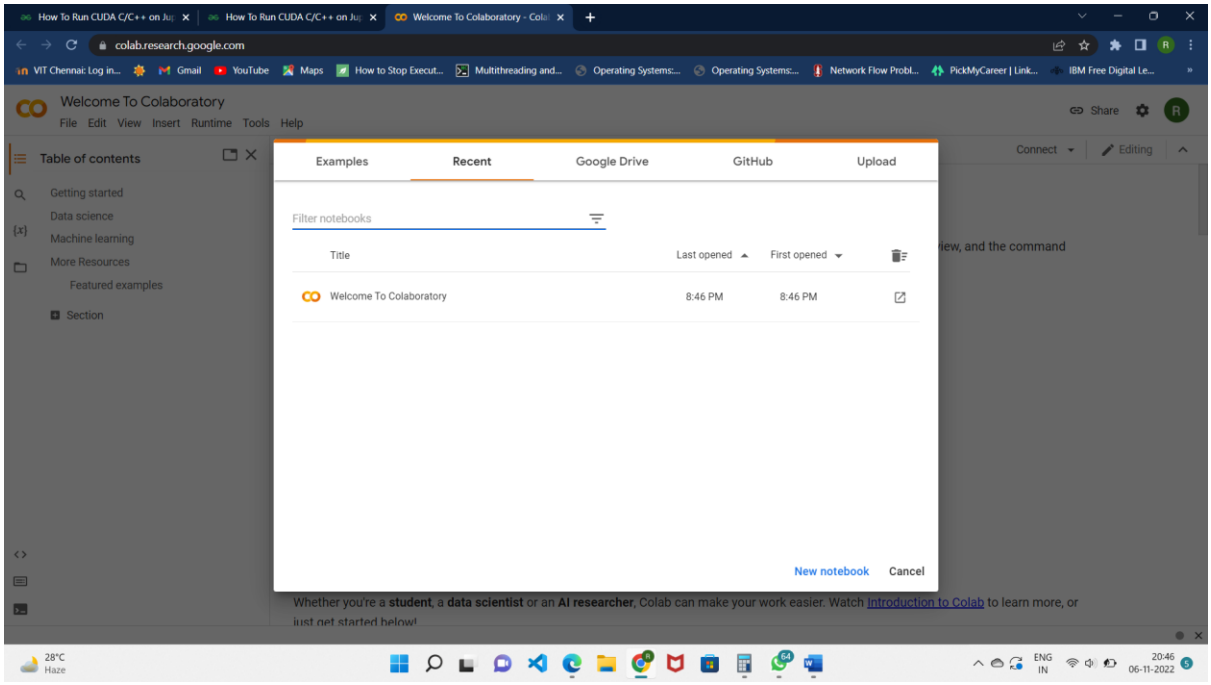
Reg No: 20BCE1988

Lab Ex: 11

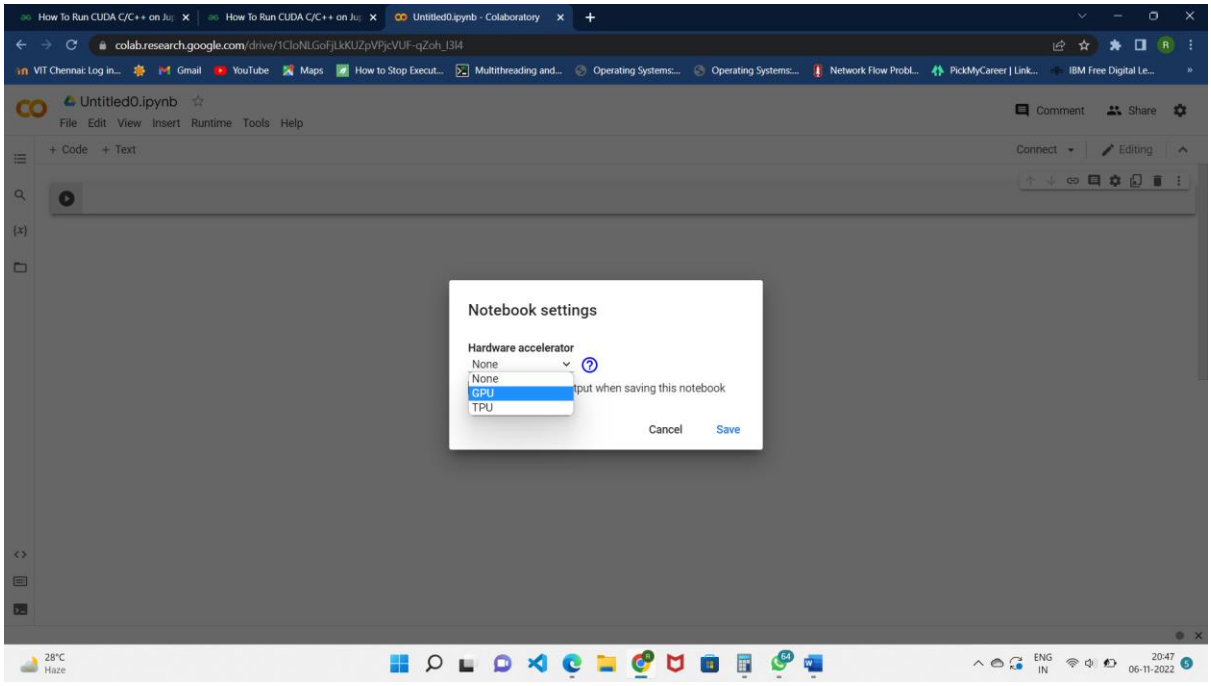
Title: CUDA C/C++

Installation of CUDA C/C++ Google Colab:

1.Open a new notebook in google colab



2.Switch runtime to GPU:



3.To uninstall previous CUDA version:

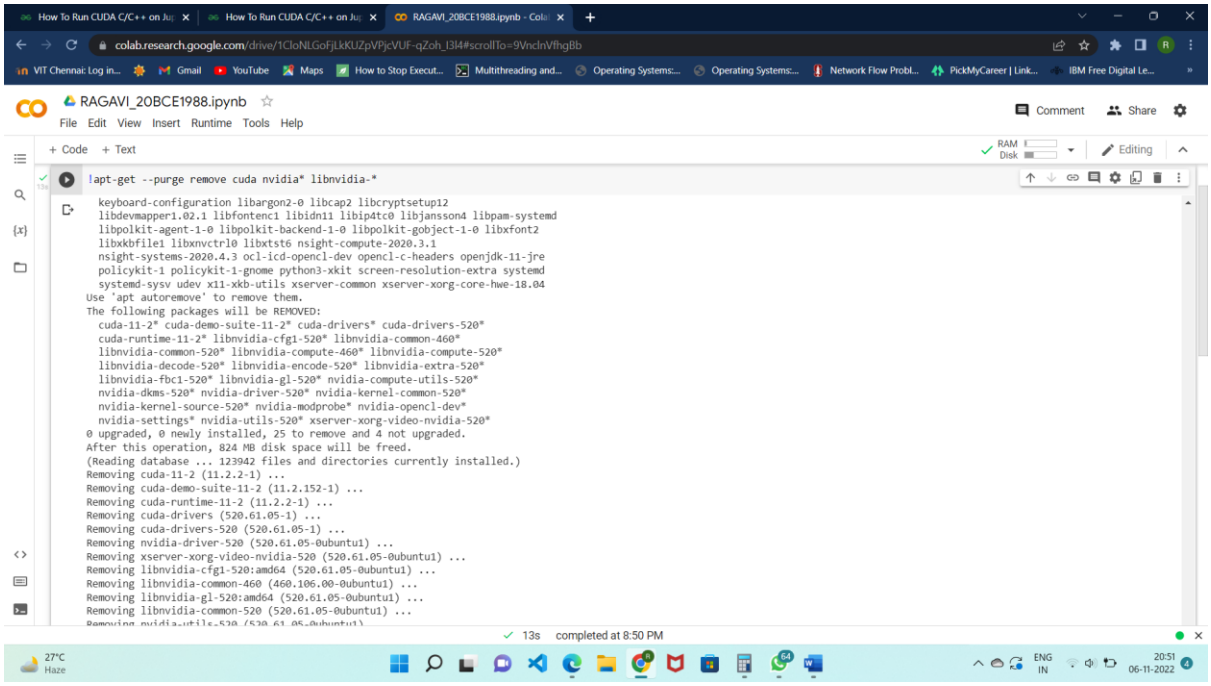
!apt-get --purge remove cuda nvidia* libnvidia-*

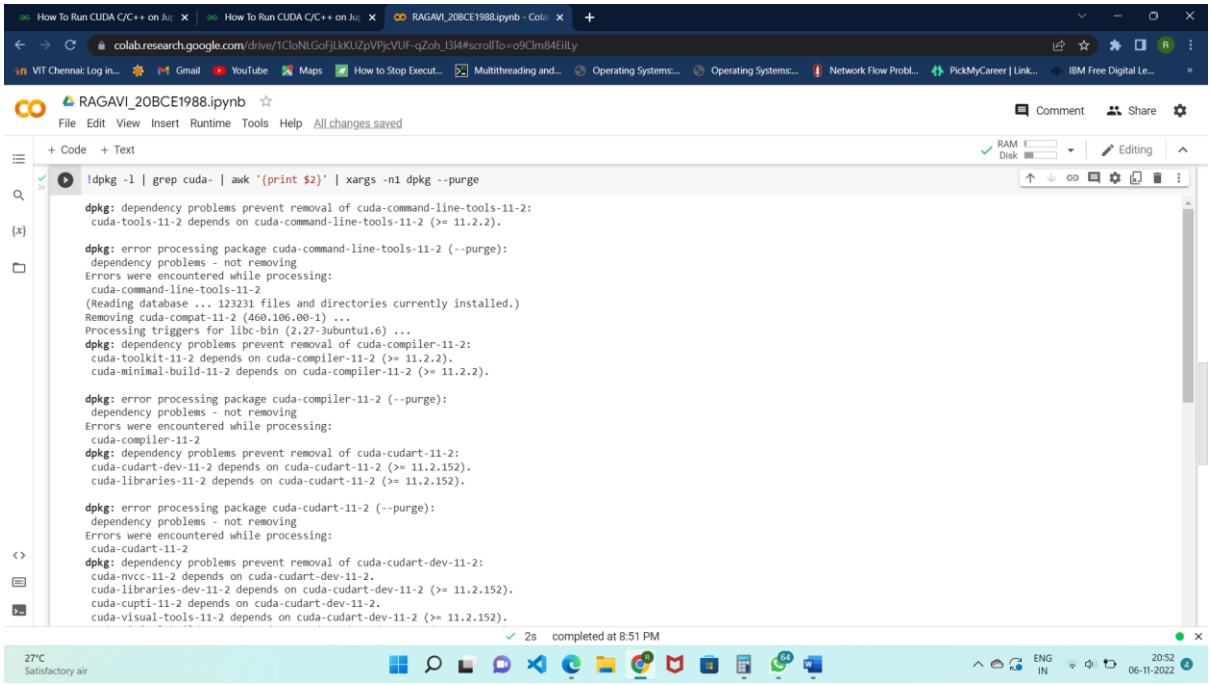
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge

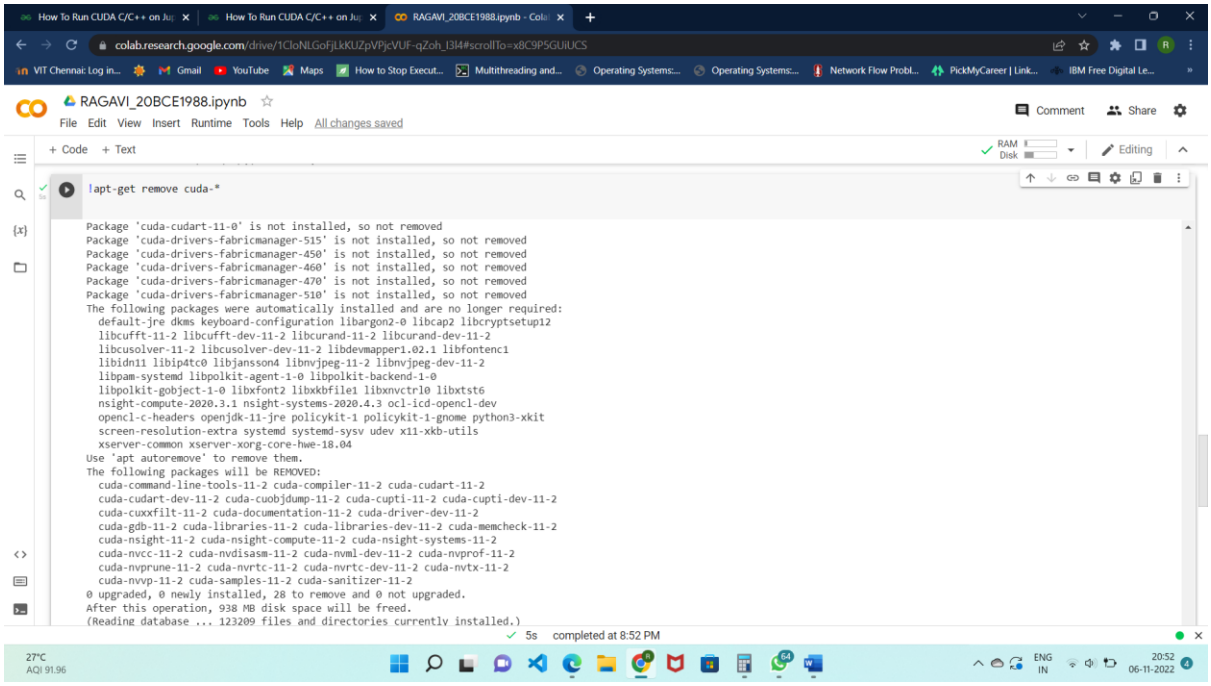
!apt-get remove cuda-*

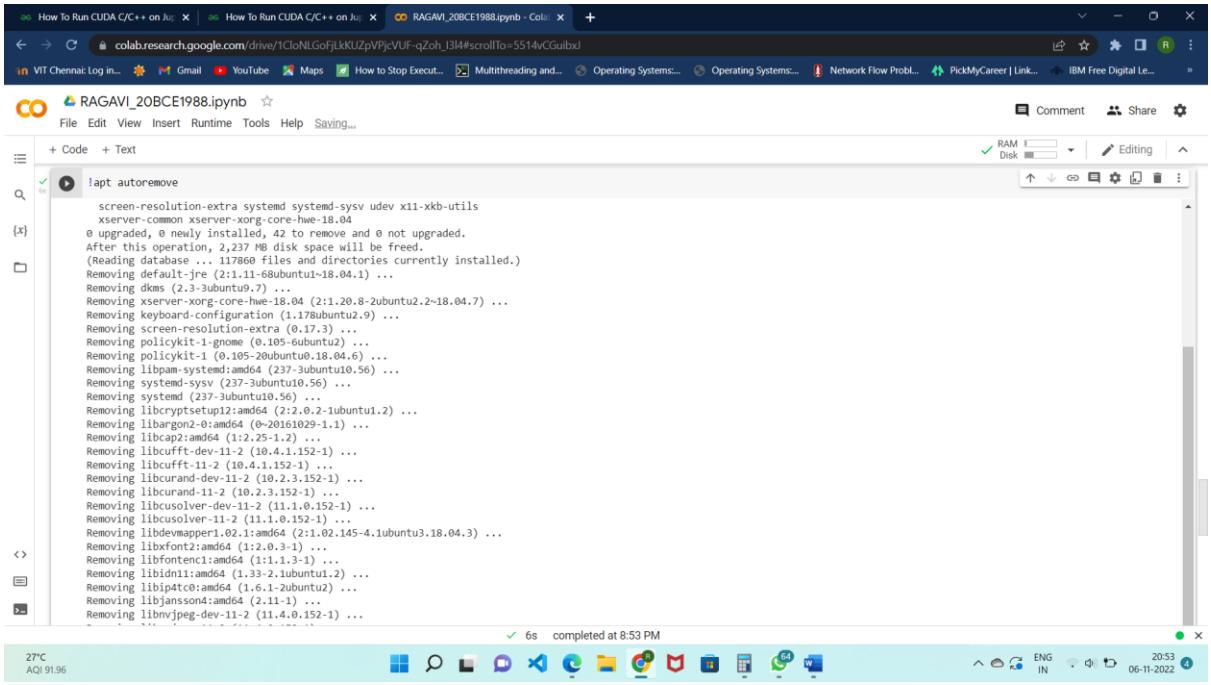
!apt autoremove

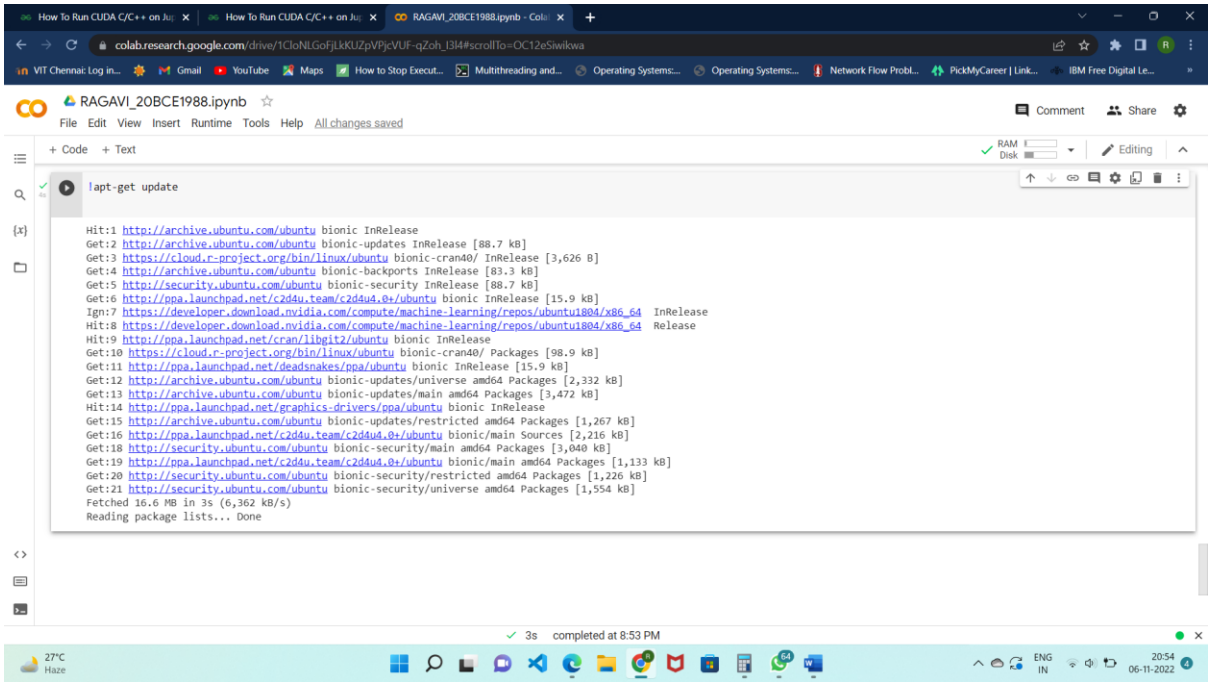
!apt-get update


How To Run CUDA C/C++ on Ju
How To Run CUDA C/C++ on Ju
RAGAVI_20BCE1988.ipynb - Colab
colab.research.google.com/drive/1CloNLGoFjLkKUZpVPjcVUf-qZoh_l3l4#scrollTo=9VnclnVfhgBb
RAGAVI_20BCE1988.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
RAM
Disk
Editing
!apt-get --purge remove cuda nvidia* libnvidia-*
keyboard-configuration libargon2-0 libcap2 libcryptsetup12
libdevmapper1.02.1 libfontenc1 libidn11 libip4tc0 libjansson4 libpam-systemd
libpolkit-agent-1-0 libpolkit-backend-1-0 libpolkit-gobject-1-0 libxfont2
libxkbfile1 libxnvctrl0 libxtst6 nsight-compute-2020.3.1
nsight-systems-2020.4.3 ocl-icd-opencl-dev opencl-c-headers openjdk-11-jre
policykit-1 policykit-1-gnome python3-xkit screen-resolution-extra systemd
systemd-sysv udev x11-xkb-utils xserver-common xserver-xorg-core-hwe-18.04
Use 'apt autoremove' to remove them.
The following packages will be REMOVED:
cuda-11-2* cuda-demo-suite-11-2* cuda-drivers* cuda-drivers-520*
cuda-runtime-11-2* libnvidia-cfg1-520* libnvidia-common-460*
libnvidia-common-520* libnvidia-compute-460* libnvidia-compute-520*
libnvidia-decode-520* libnvidia-encode-520* libnvidia-extra-520*
libnvidia-fbc1-520* libnvidia-gl-520* nvidia-compute-utils-520*
nvidia-dkms-520* nvidia-driver-520* nvidia-kernel-common-520*
nvidia-kernel-source-520* nvidia-modprobe* nvidia-opencl-dev*
nvidia-settings* nvidia-utils-520* xserver-xorg-video-nvidia-520*
0 upgraded, 0 newly installed, 25 to remove and 4 not upgraded.
After this operation, 824 MB disk space will be freed.
(Reading database ... 123942 files and directories currently installed.)
Removing cuda-11-2 (11.2.2-1) ...
Removing cuda-demo-suite-11-2 (11.2.152-1) ...
Removing cuda-runtime-11-2 (11.2.2-1) ...
Removing cuda-drivers (520.61.05-1) ...
Removing cuda-drivers-520 (520.61.05-1) ...
Removing nvidia-driver-520 (520.61.05-0ubuntu1) ...
Removing xserver-xorg-video-nvidia-520 (520.61.05-0ubuntu1) ...
Removing libnvidia-cfg1-520:amd64 (520.61.05-0ubuntu1) ...
Removing libnvidia-common-460 (460.106.00-0ubuntu1) ...
Removing libnvidia-gl-520:amd64 (520.61.05-0ubuntu1) ...
Removing libnvidia-common-520 (520.61.05-0ubuntu1) ...
13s completed at 8:50 PM
27°C
Haze
ENG
IN
20:51
06-11-2022


How To Run CUDA C/C++ on Ju
How To Run CUDA C/C++ on Ju
RAGAVI_20BCE1988.ipynb - Colab
colab.research.google.com/drive/1CloNLGoFjLkKUZpVPjcVUf-qZoh_l3l4#scrollTo=o9Clm84EilLy
RAGAVI_20BCE1988.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
dpkg: dependency problems prevent removal of cuda-command-line-tools-11-2:
cuda-tools-11-2 depends on cuda-command-line-tools-11-2 (>= 11.2.2).
dpkg: error processing package cuda-command-line-tools-11-2 (--purge):
dependency problems - not removing
Errors were encountered while processing:
cuda-command-line-tools-11-2
(Reading database ... 123231 files and directories currently installed.)
Removing cuda-compat-11-2 (460.106.00-1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.6) ...
dpkg: dependency problems prevent removal of cuda-compiler-11-2:
cuda-toolkit-11-2 depends on cuda-compiler-11-2 (>= 11.2.2).
cuda-minimal-build-11-2 depends on cuda-compiler-11-2 (>= 11.2.2).
dpkg: error processing package cuda-compiler-11-2 (--purge):
dependency problems - not removing
Errors were encountered while processing:
cuda-compiler-11-2
dpkg: dependency problems prevent removal of cuda-cudart-11-2:
cuda-cudart-dev-11-2 depends on cuda-cudart-11-2 (>= 11.2.152).
cuda-libraries-11-2 depends on cuda-cudart-11-2 (>= 11.2.152).
dpkg: error processing package cuda-cudart-11-2 (--purge):
dependency problems - not removing
Errors were encountered while processing:
cuda-cudart-11-2
dpkg: dependency problems prevent removal of cuda-cudart-dev-11-2:
cuda-nvcc-11-2 depends on cuda-cudart-dev-11-2.
cuda-libraries-dev-11-2 depends on cuda-cudart-dev-11-2 (>= 11.2.152).
cuda-cupti-11-2 depends on cuda-cudart-dev-11-2.
cuda-visual-tools-11-2 depends on cuda-cudart-dev-11-2 (>= 11.2.152).
2s completed at 8:51 PM
27°C
Satisfactory air
ENG
IN
20:52
06-11-2022


How To Run CUDA C/C++ on Ju
How To Run CUDA C/C++ on Ju
RAGAVI_20BCE1988.ipynb - Colab
colab.research.google.com/drive/1ClONLGoFjLkKUZpVPjcVUF-qZoh_I3l4#scrollTo=x8C9P5GUiUCS
VIT Chennai: Log in...
Gmail
YouTube
Maps
How to Stop Execut...
Multithreading and...
Operating Systems:...
Operating Systems:...
Network Flow Probl...
PickMyCareer | Link...
IBM Free Digital Le...
RAGAVI_20BCE1988.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing
!apt-get remove cuda-*
Package 'cuda-cudart-11-0' is not installed, so not removed
Package 'cuda-drivers-fabricmanager-515' is not installed, so not removed
Package 'cuda-drivers-fabricmanager-450' is not installed, so not removed
Package 'cuda-drivers-fabricmanager-460' is not installed, so not removed
Package 'cuda-drivers-fabricmanager-470' is not installed, so not removed
Package 'cuda-drivers-fabricmanager-510' is not installed, so not removed
The following packages were automatically installed and are no longer required:
default-jre dkms keyboard-configuration libargon2-0 libcap2 libcryptsetup12
libcufft-11-2 libcufft-dev-11-2 libcurand-11-2 libcurand-dev-11-2
libcusolver-11-2 libcusolver-dev-11-2 libdevmapper1.02.1 libfontenc1
libidn11 libip4tc0 libjansson4 libnvjpeg-11-2 libnvjpeg-dev-11-2
libpam-systemd libpolkit-agent-1-0 libpolkit-backend-1-0
libpolkit-gobject-1-0 libxfont2 libxkbfile1 libxnvctrl0 libxtst6
nsight-compute-2020.3.1 nsight-systems-2020.4.3 ocl-icd-opencl-dev
opencl-c-headers openjdk-11-jre policykit-1 policykit-1-gnome python3-xkit
screen-resolution-extra systemd systemd-sysv udev x11-xkb-utils
xserver-common xserver-xorg-core-hwe-18.04
Use 'apt autoremove' to remove them.
The following packages will be REMOVED:
cuda-command-line-tools-11-2 cuda-compiler-11-2 cuda-cudart-11-2
cuda-cudart-dev-11-2 cuda-cuobjdump-11-2 cuda-cupti-11-2 cuda-cupti-dev-11-2
cuda-cuxxfilt-11-2 cuda-documentation-11-2 cuda-driver-dev-11-2
cuda-gdb-11-2 cuda-libraries-11-2 cuda-libraries-dev-11-2 cuda-memcheck-11-2
cuda-nsight-11-2 cuda-nsight-compute-11-2 cuda-nsight-systems-11-2
cuda-nvcc-11-2 cuda-nvdisasm-11-2 cuda-nvml-dev-11-2 cuda-nvprof-11-2
cuda-nvprune-11-2 cuda-nvrtc-11-2 cuda-nvrtc-dev-11-2 cuda-nvtx-11-2
cuda-nvvp-11-2 cuda-samples-11-2 cuda-sanitizer-11-2
0 upgraded, 0 newly installed, 28 to remove and 0 not upgraded.
After this operation, 938 MB disk space will be freed.
(Reading database ... 123209 files and directories currently installed.)
5s completed at 8:52 PM
27°C
AQI 91.96
ENG
IN
20:52
06-11-2022


How To Run CUDA C/C++ on Ju
How To Run CUDA C/C++ on Ju
RAGAVI_20BCE1988.ipynb - Colab
colab.research.google.com/drive/1ClONLGoFjLkKUZpVPjcVUF-qZoh_I3l4#scrollTo=5514vCGuibxJ
VIT Chennai: Log in...
Gmail
YouTube
Maps
How to Stop Execut...
Multithreading and...
Operating Systems:...
Operating Systems:...
Network Flow Probl...
PickMyCareer | Link...
IBM Free Digital Le...
RAGAVI_20BCE1988.ipynb
File Edit View Insert Runtime Tools Help Saving...
Comment
Share
+ Code + Text
RAM
Disk
Editing
!apt autoremove
screen-resolution-extra systemd systemd-sysv udev x11-xkb-utils
xserver-common xserver-xorg-core-hwe-18.04
0 upgraded, 0 newly installed, 42 to remove and 0 not upgraded.
After this operation, 2,237 MB disk space will be freed.
(Reading database ... 117860 files and directories currently installed.)
Removing default-jre (2:1.11-68ubuntu1~18.04.1) ...
Removing dkms (2.3-3ubuntu9.7) ...
Removing xserver-xorg-core-hwe-18.04 (2:1.20.8-2ubuntu2.2~18.04.7) ...
Removing keyboard-configuration (1.178ubuntu2.9) ...
Removing screen-resolution-extra (0.17.3) ...
Removing policykit-1-gnome (0.105-6ubuntu2) ...
Removing policykit-1 (0.105-20ubuntu0.18.04.6) ...
Removing libpam-systemd:amd64 (237-3ubuntu10.56) ...
Removing systemd-sysv (237-3ubuntu10.56) ...
Removing systemd (237-3ubuntu10.56) ...
Removing libcryptsetup12:amd64 (2:2.0.2-1ubuntu1.2) ...
Removing libargon2-0:amd64 (0~20161029-1.1) ...
Removing libcap2:amd64 (1:2.25-1.2) ...
Removing libcufft-dev-11-2 (10.4.1.152-1) ...
Removing libcufft-11-2 (10.4.1.152-1) ...
Removing libcurand-dev-11-2 (10.2.3.152-1) ...
Removing libcurand-11-2 (10.2.3.152-1) ...
Removing libcusolver-dev-11-2 (11.1.0.152-1) ...
Removing libcusolver-11-2 (11.1.0.152-1) ...
Removing libdevmapper1.02.1:amd64 (2:1.02.145-4.1ubuntu3.18.04.3) ...
Removing libxfont2:amd64 (1:2.0.3-1) ...
Removing libfontenc1:amd64 (1:1.1.3-1) ...
Removing libidn11:amd64 (1.33-2.1ubuntu1.2) ...
Removing libip4tc0:amd64 (1.6.1-2ubuntu2) ...
Removing libjansson4:amd64 (2.11-1) ...
Removing libnvjpeg-dev-11-2 (11.4.0.152-1) ...
6s completed at 8:53 PM
27°C
AQI 91.96
ENG
IN
20:53
06-11-2022



To install CUDA version 9:

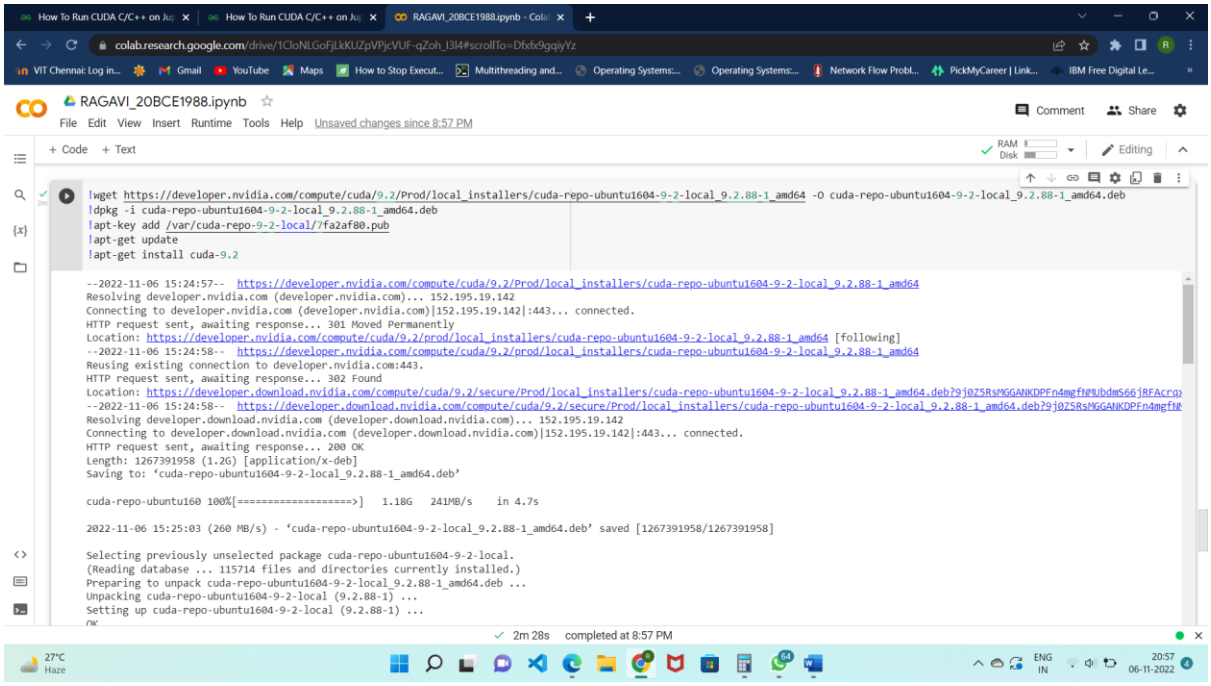
!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -O cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb

!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb

!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub

!apt-get update

!apt-get install cuda-9.2

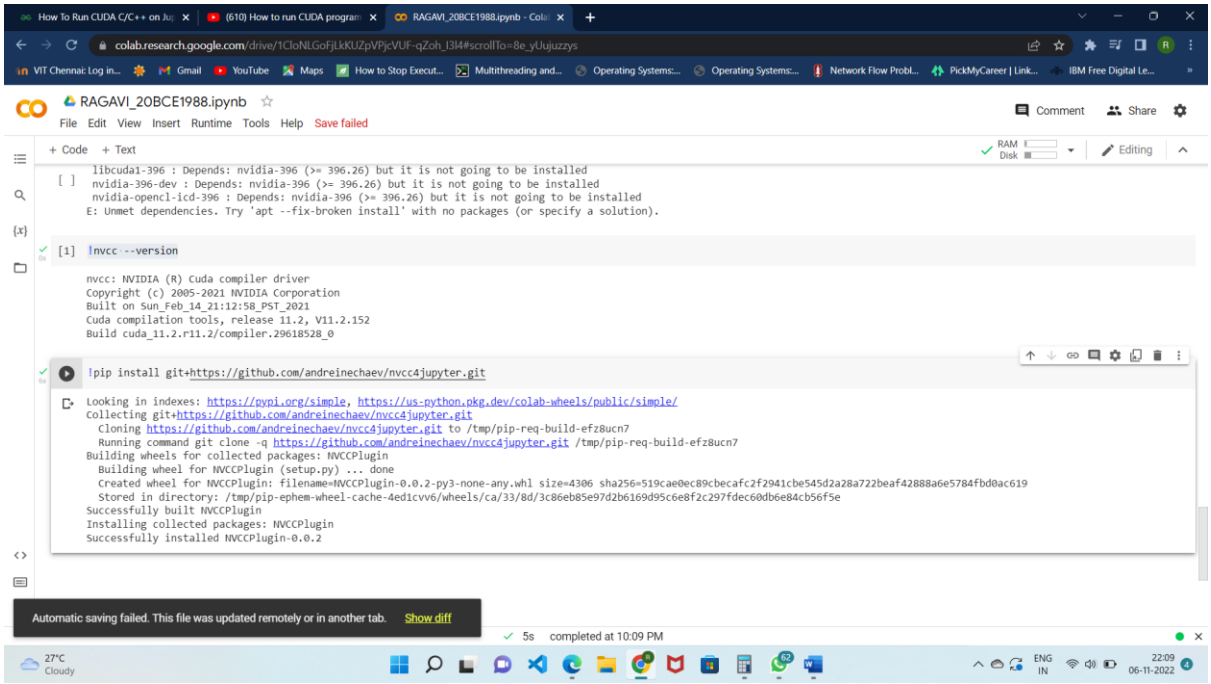


To check CUDA installation:

!nvcc –version

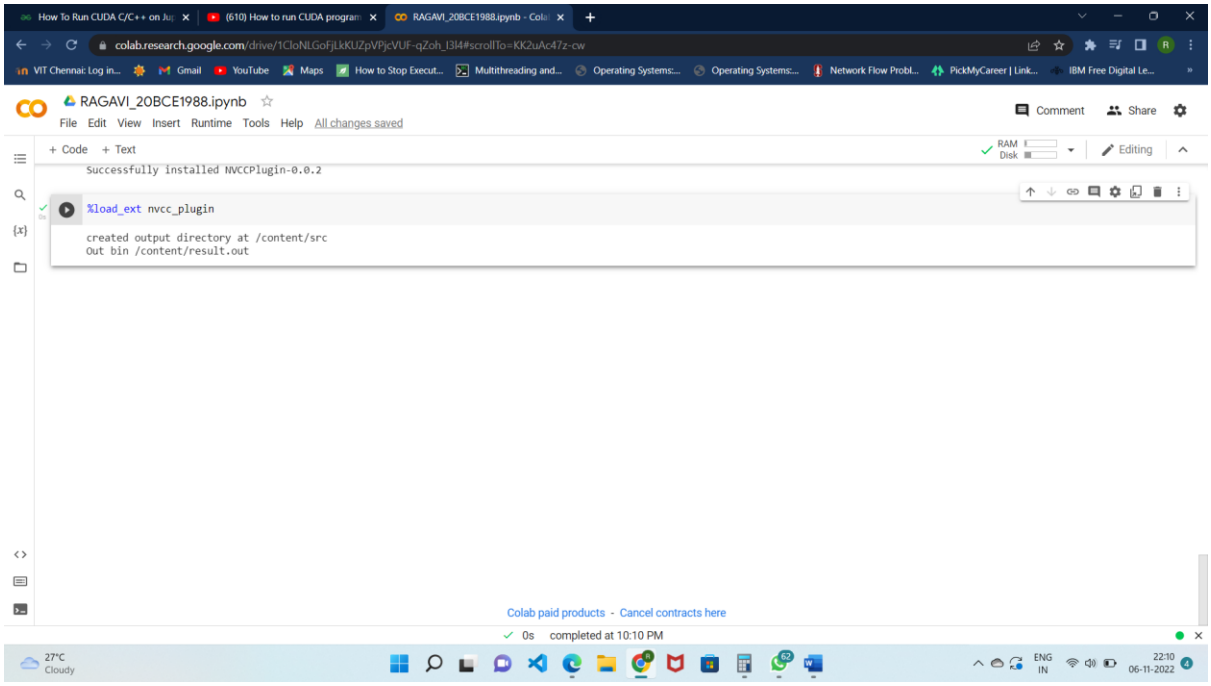
Run the given command to install a small extension to run nvcc from the Notebook cells.

!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git

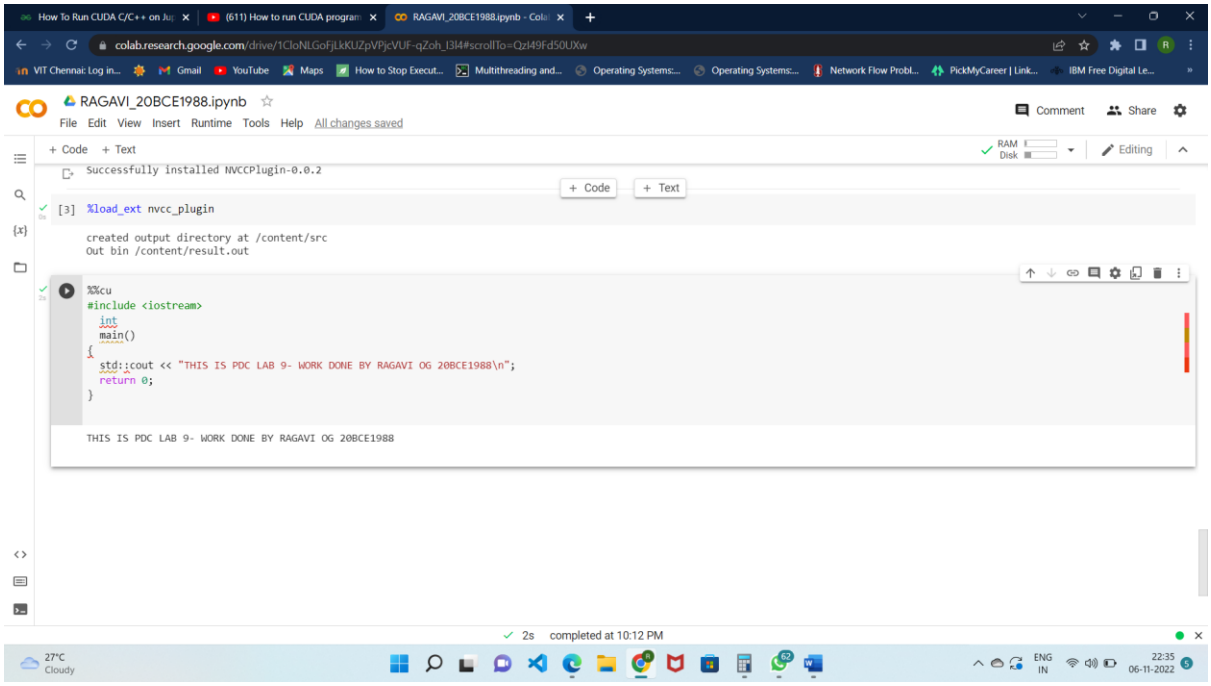


Load the extension using the code given below:

%load_ext nvcc_plugin



Sample output:



<u>Problem: Addition of 2 integer arrays:</u>

CODE:

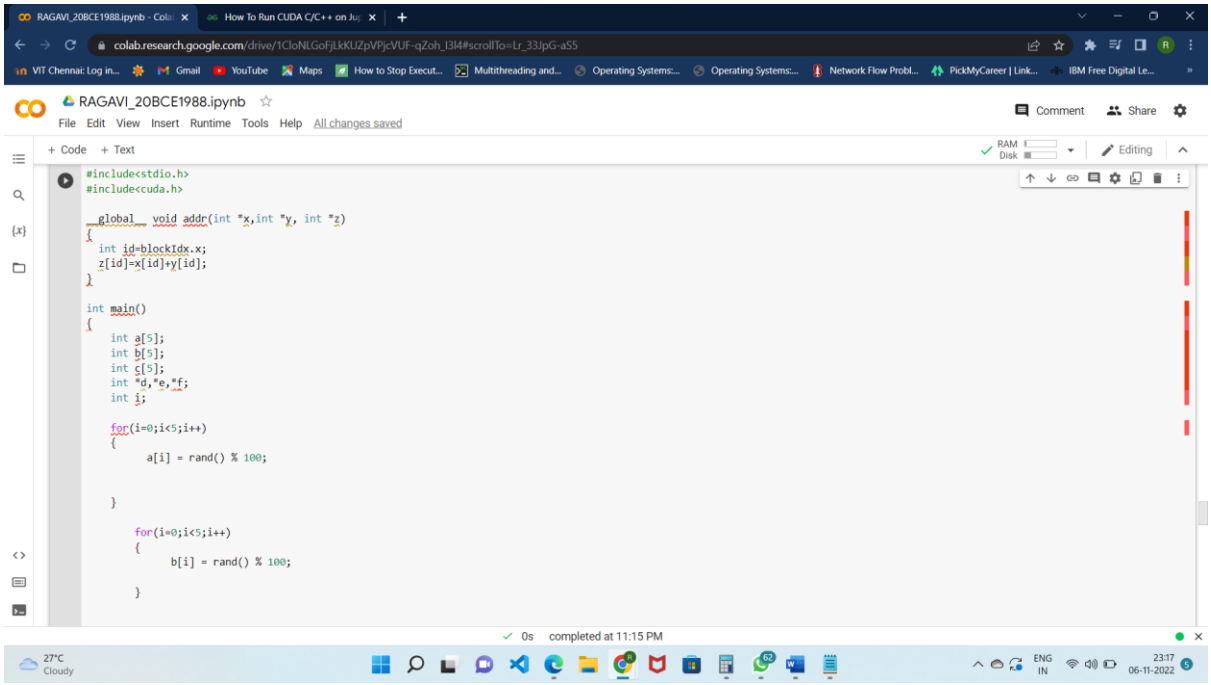
```
%%cu
#include<stdio.h>
#include<cuda.h>

__global__ void addr(int *x,int *y, int *z)
{
  int id=blockIdx.x;
  z[id]=x[id]+y[id];
}

int main()
{
    int a[5];
    int b[5];
    int c[5];
    int *d,*e,*f;
    int i;

    for(i=0;i<5;i++)
    {
          a[i] = rand() % 100;


    }

        for(i=0;i<5;i++)
        {
              b[i] = rand() % 100;

        }


    cudaMalloc((void **)&d,5*sizeof(int));
    cudaMalloc((void **)&e,5*sizeof(int));
    cudaMalloc((void **)&f,5*sizeof(int));


 cudaMemcpy(d,a,5*sizeof(int),cudaMemcpyHostToDevice);
 cudaMemcpy(e,b,5*sizeof(int),cudaMemcpyHostToDevice);

addr<<<5,1>>>(d,e,f);


 cudaMemcpy(c,f,5*sizeof(int),cudaMemcpyDeviceToHost);
```

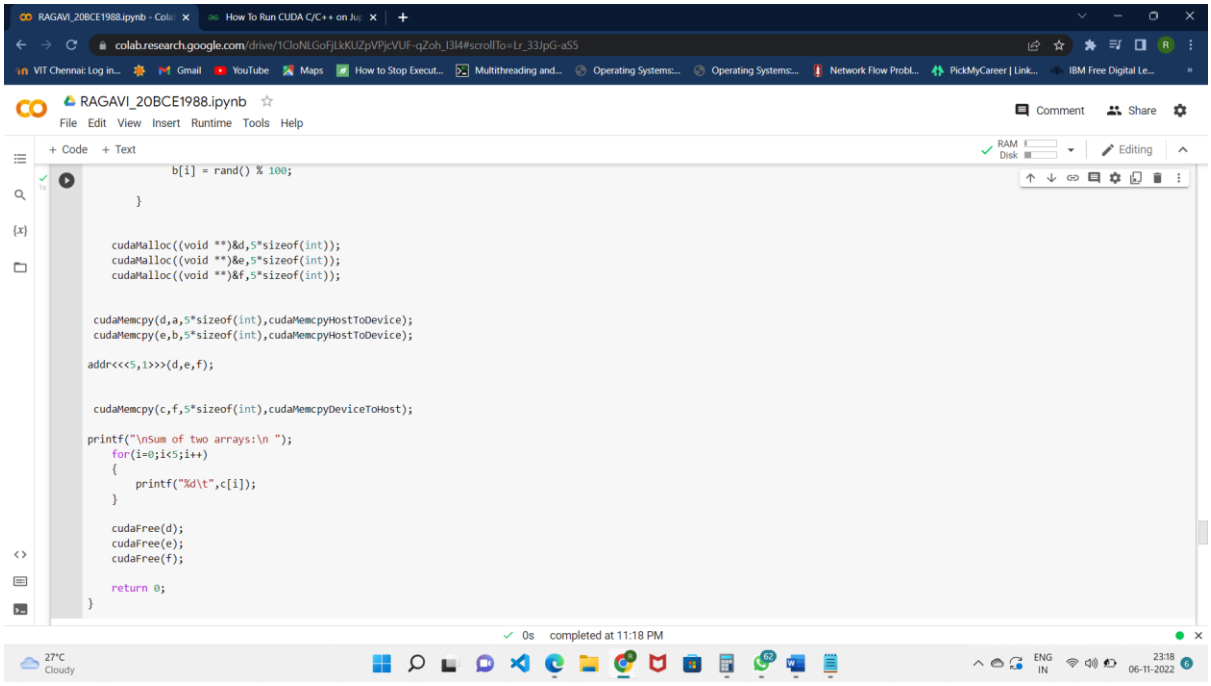
```
printf("\nSum of two arrays:\n ");
    for(i=0;i<5;i++)
    {
        printf("%d\t",c[i]);
    }

    cudaFree(d);
    cudaFree(e);
    cudaFree(f);

    return 0;
}
```

OUTPUT :

```
Sum of two arrays:
5 18 2 56 54
```

ACTUAL CODE SNAPSHOTS:




RAGAVI_20BCE1988.ipynb - Colab
How To Run CUDA C/C++ on Ju
colab.research.google.com/drive/1CloNLGoFjLkKUZpVPjcVUF-qZoh_I3I4#scrollTo=Lr_33JpG-aS5
RAGAVI_20BCE1988.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
RAM
Disk
Editing
b[i] = rand() % 100;
}
cudaMalloc((void **)&d,5*sizeof(int));
cudaMalloc((void **)&e,5*sizeof(int));
cudaMalloc((void **)&f,5*sizeof(int));
cudaMemcpy(d,a,5*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpy(e,b,5*sizeof(int),cudaMemcpyHostToDevice);
addr<<<5,1>>>(d,e,f);
cudaMemcpy(c,f,5*sizeof(int),cudaMemcpyDeviceToHost);
printf("\nSum of two arrays:\n ");
for(i=0;i<5;i++)
{
printf("%d\t",c[i]);
}
cudaFree(d);
cudaFree(e);
cudaFree(f);
return 0;
}
0s completed at 11:18 PM
27°C
Cloudy
ENG
IN
23:18
06-11-2022