

**Veermata Jijabai Technological Institute, Mumbai 400019**

**Experiment No.:** 04

**Aim :** Implementation using OpenMP.

1. Fork Join model,
2. Producer Consumer problem,
3. Matrix Multiplication,
4. find prime number,
5. Largest Element in an array and
6. Pi calculation

**Name :** Kiran K Patil

**Enrolment No.:** 211070904

**Branch:** Computer Engineering

**Batch:** IV

**1. Fork Join Model :**

OpenMP uses a fork-join model of parallel execution. When a thread encounters a parallel construct, the thread creates a team composed of itself and some additional (possibly zero) number of threads

Program :

#include<stdio.h>

#include<omp.h>

int main(){

int nthreads, tid;

/\* Fork a team of threads giving them their own copies of variables \*/

#pragma omp parallel private(nthreads, tid)

{

/\* Obtain thread number \*/

tid = omp\_get\_thread\_num();

printf("Hello World from thread = %d\n", tid);

/\* Only master thread does this \*/

if (tid == 0)

{

nthreads = omp\_get\_num\_threads();

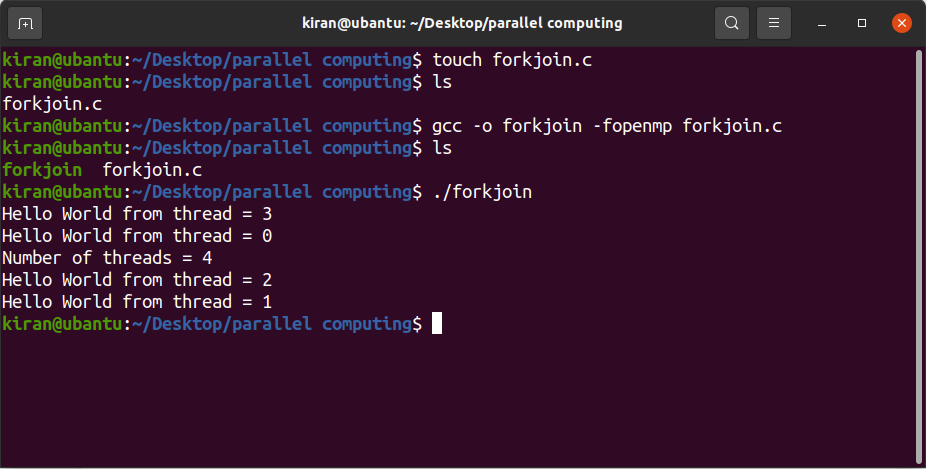
printf("Number of threads = %d\n", nthreads);

}

} /\* All threads join master thread and disband \*/

}

Output:



**2. Producer Consumer problem**

In this program, the master thread will act as a producer while the other threads will wait until the master thread creates buffer, and once added the master notifies the threads using a shared variable and all other threads will consume the data.

Program :

#include<stdio.h>

#include<omp.h>

int main()

{

int i=0;

int x=0;

#pragma omp parallel shared(i)

{

if(omp\_get\_thread\_num()==0)

{

printf("Master thread with Thread ID:%d\n", omp\_get\_thread\_num());

printf("Since it is the producer thread It is adding some data to be consumed by other consumer threads\n");

i+=10;

x=1;

}

else

{

while(x==0)

printf("Waiting for buffer to be filled. Thread ID: %d\n",omp\_get\_thread\_num());

#pragma critical

{

if(i>0){

printf("Data is consumed by Consumer with Thread ID: %d\n",omp\_get\_thread\_num());

i-=5;

} else {

printf("Could not find any data for thread ID: %d\n",omp\_get\_thread\_num());

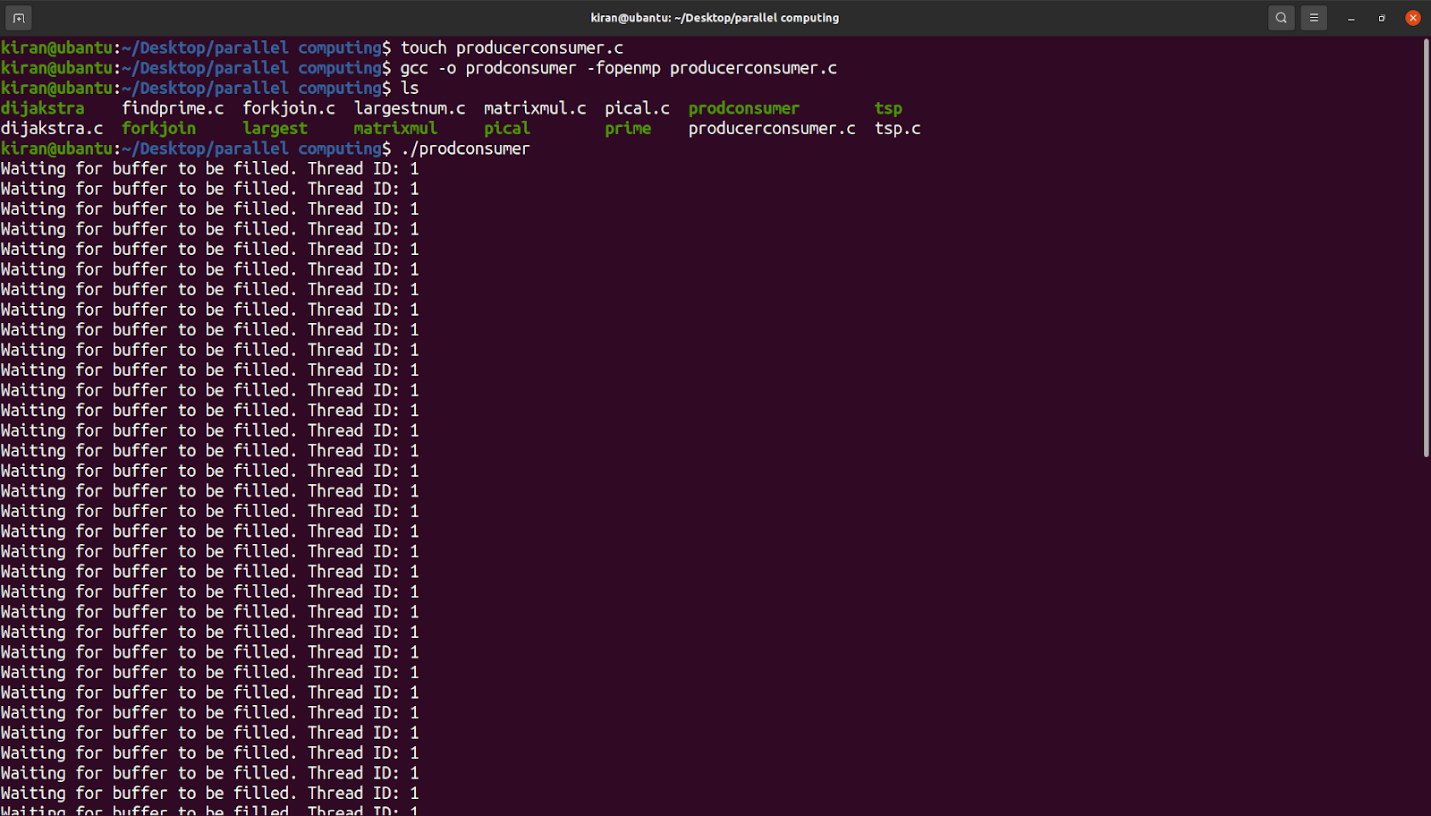
}}

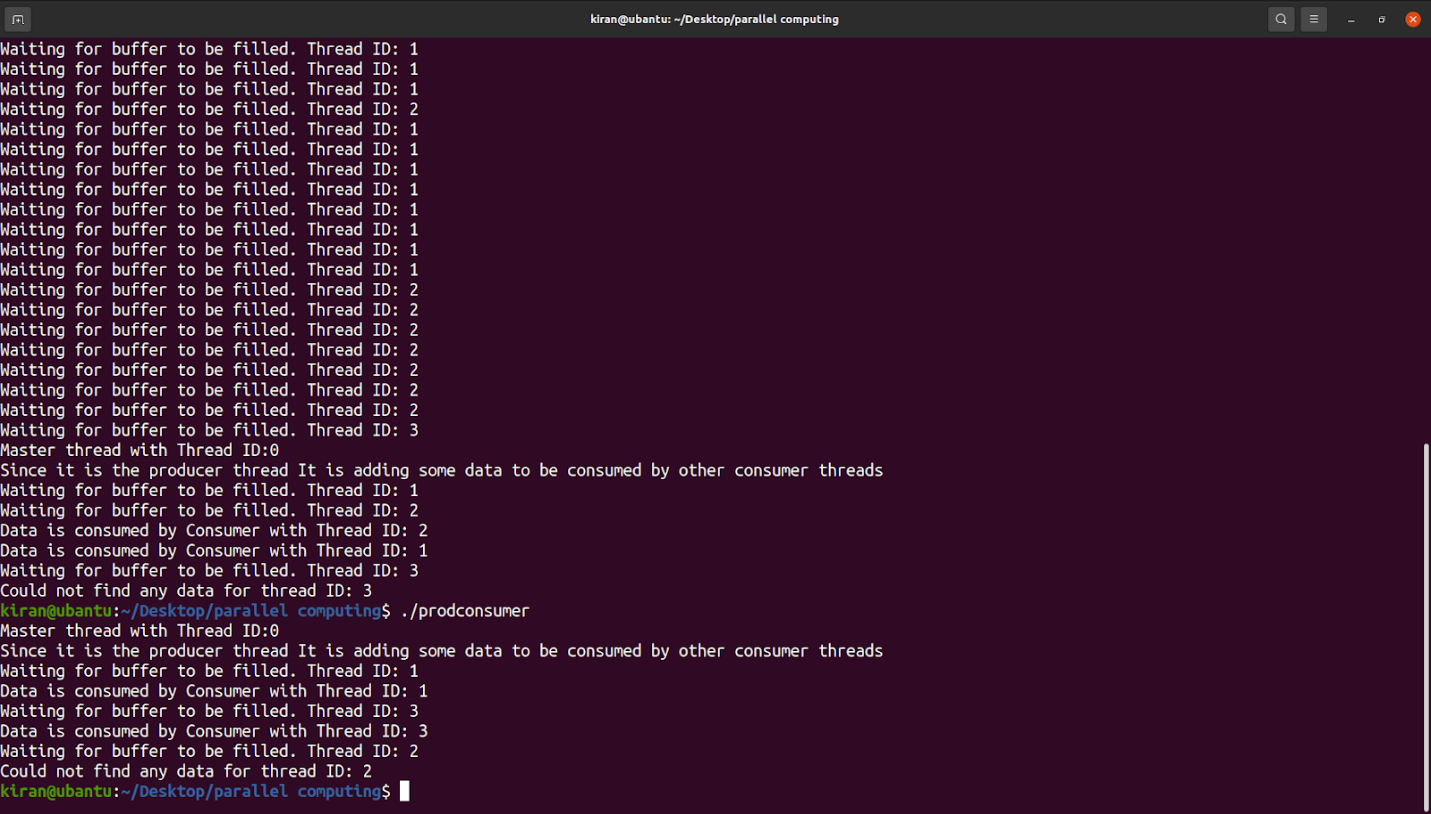
}

}

}

Output :





**3. Matrix multiplication**

Program :

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

int main(){

int i,j,k,m,n,p;

printf("Enter the number of rows in Matrix 1:");

scanf("%d",&m);

int \*matrixA[m];

printf("Enter the number of columns in Matrix 1:");

scanf("%d",&n);

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

matrixA[i] = (int \*)malloc(n\*sizeof(int));

}

printf("<--Now Input the values for matrix 1 row-wise-->\n");

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

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

scanf("%d",&matrixA[i][j]);

}

}

printf("Enter the number of columns in Matrix 2:");

scanf("%d",&p);

int \*matrixB[n];

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

matrixB[i] = (int \*)malloc(p\*sizeof(int));

}

printf("<--Now Input the values for matrix 2 row-wise-->\n");

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

for(j=0;j<p;j++){

scanf("%d",&matrixB[i][j]);

}

}

int matrixC[m][p];

#pragma omp parallel private(i,j,k) shared(matrixA,matrixB,matrixC)

{

#pragma omp for schedule(static)

for (i=0; i<m; i=i+1){

for (j=0; j<p; j=j+1){

matrixC[i][j] = 0;

for (k=0; k<n; k=k+1){

matrixC[i][j]=(matrixC[i][j])+((matrixA[i][k])\*(matrixB[k][j]));

}

}

}

}

printf("The output after Matrix Multiplication is: \n");

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

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

printf("%d \t",matrixC[i][j]);

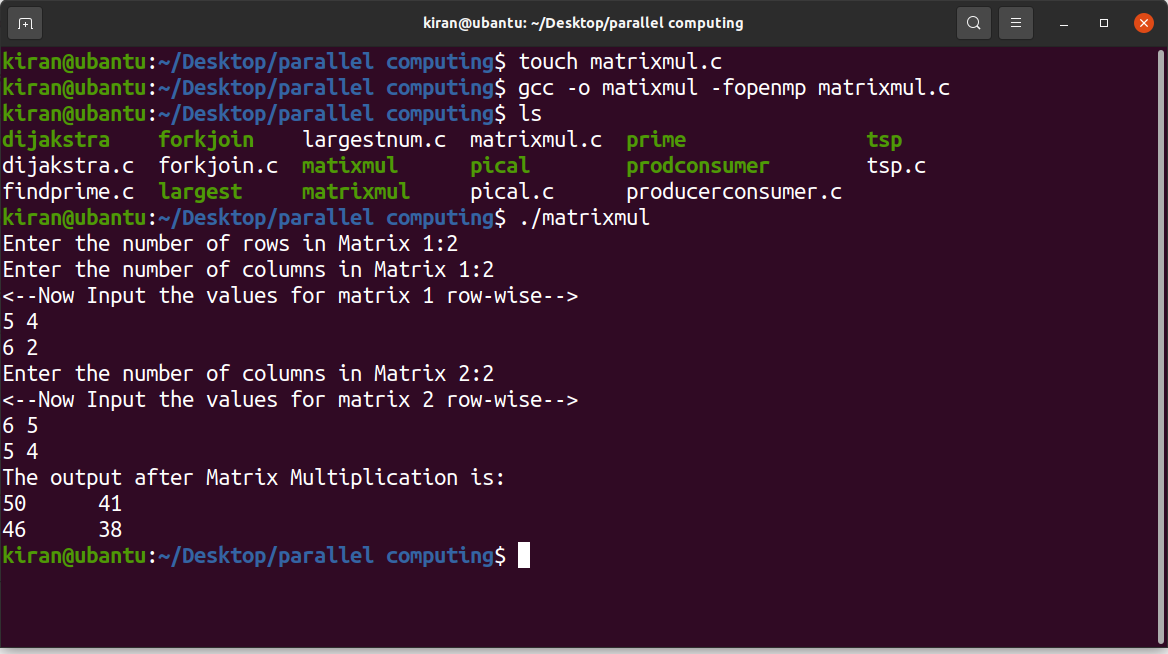
printf("\n");

}

return 0;

}

Output :

****

**4. Find Prime**

Program :

#include<stdio.h>

#include<omp.h>

int IsPrime(int number) {

int i;

for (i = 2; i < number; i++) {

if (number % i == 0 && i != number) return 0;

}

return 1;

}

int main(){

int noOfThreads,valueN,indexCount=0,arrayVal[10000],tempValue;

printf("Enter the Number of threads: ");

scanf("%d",&noOfThreads);

printf("Enter the value of N: ");

scanf("%d",&valueN);

omp\_set\_num\_threads(noOfThreads);

#pragma omp parallel for reduction(+:indexCount)

for(tempValue=2;tempValue<=valueN;tempValue++){

if(IsPrime(tempValue)){

arrayVal[indexCount] = tempValue;

indexCount++;

}

}

printf("Number of prime numbers between 2 and %d: %d\n",valueN,indexCount);

return 0;

}

Output :



**5. Largest Element in an array**

Program :

#include<stdio.h>

#include<omp.h>

int main(){

int numberOfElements,currentMax=-1,iIterator,arrayInput[10000];

printf("Enter the Number of Elements: ");

scanf("%d",&numberOfElements);

for(iIterator=0;iIterator<numberOfElements;iIterator++){

scanf("%d",&arrayInput[iIterator]);

}

#pragma omp parallel for shared(currentMax)

for(iIterator=0;iIterator<numberOfElements;iIterator++){

#pragma omp critical

if(arrayInput[iIterator] > currentMax){

currentMax = arrayInput[iIterator];

}

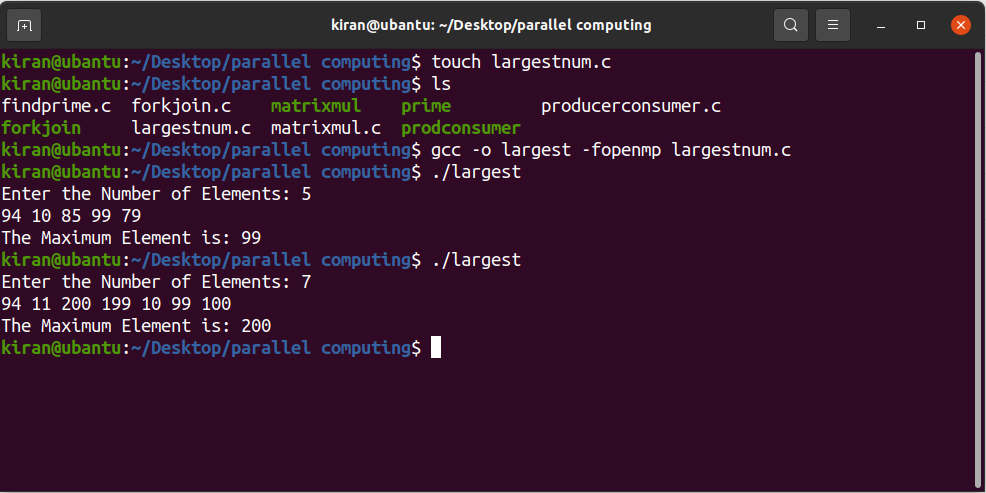
}

printf("The Maximum Element is: %d\n",currentMax);

return 0;

}

Output :

****

**6. Pi calculation**

Program :

#include<stdio.h>

#include<omp.h>

int main(){

int num\_steps=10000,i;

double aux,pi,step = 1.0/(double) num\_steps,x=0.0,sum = 0.0;

#pragma omp parallel private(i,x,aux) shared(sum)

{

#pragma omp for schedule(static)

for (i=0; i<num\_steps; i=i+1){

x=(i+0.5)\*step;

aux=4.0/(1.0+x\*x);

#pragma omp critical

sum = sum + aux;

}

}

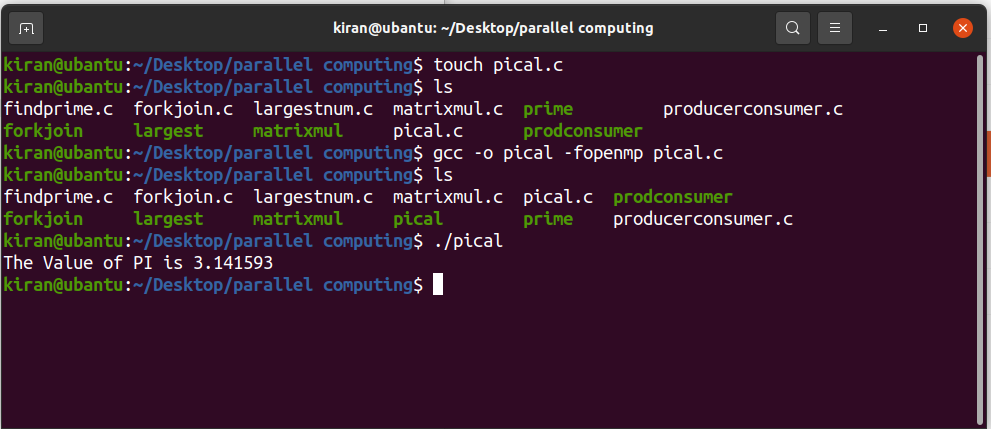
pi=step\*sum;

printf("The Value of PI is %lf\n",pi);

return 0;

}

Output :

****

**Conclusion:**

* In conclusion, OpenMP (Open Multi-Processing) is a widely used application programming interface (API) that enables parallel programming in shared-memory architectures. OpenMP provides a set of compiler directives, runtime libraries, and environment variables that simplify the development of parallel applications.
* By using OpenMP, programmers can explicitly specify which parts of their code can be executed in parallel, and how the parallelism should be managed. This can lead to significant performance improvements, especially in applications that perform intensive computations or data processing.