

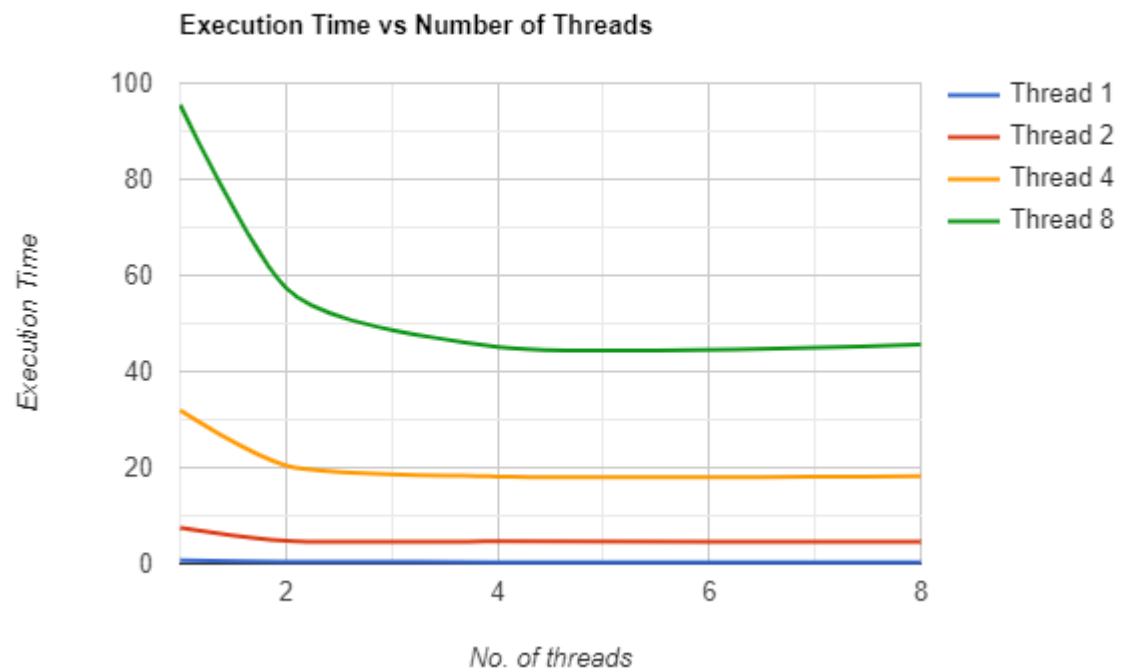
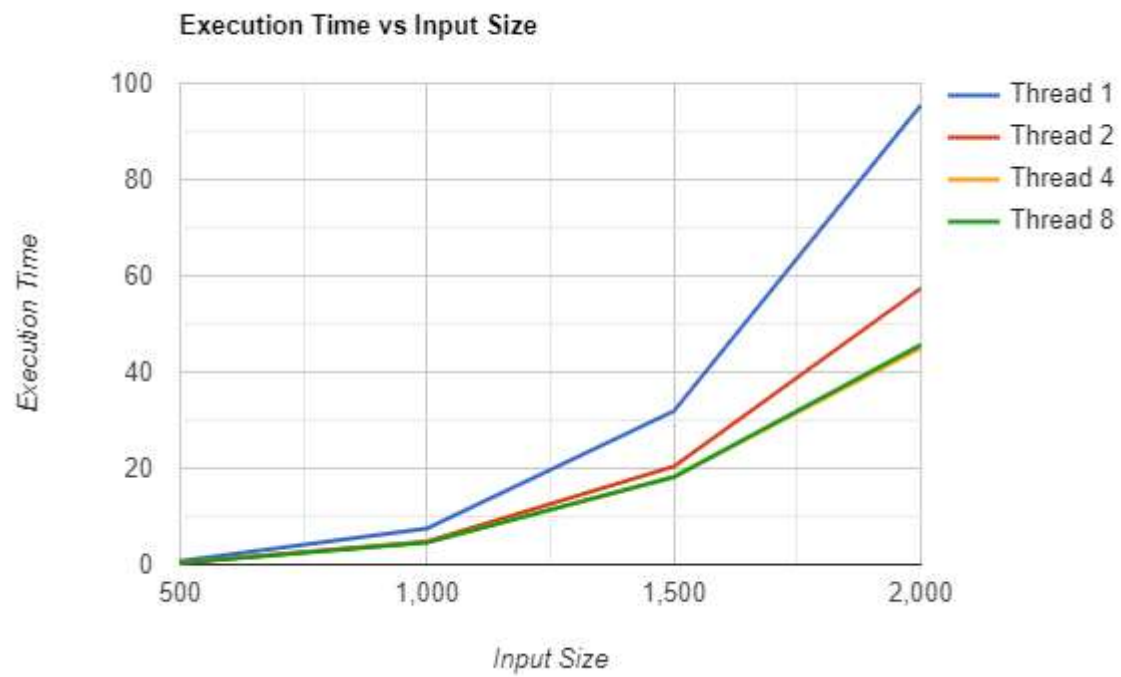
## PADP Lab Program 1

OUTPUT:

```
rohit@Rohit: /mnt/c/Users/rohit/Desktop
rohit@Rohit:/mnt/c/Users/rohit/Desktop$ gcc -fopenmp program1.c
rohit@Rohit:/mnt/c/Users/rohit/Desktop$ ./a.out
The execution time are
Size      1      2      4      8
500      0.730816  0.455516  0.366475  0.355699
1000     7.508006  4.772518  4.644519  4.624295
1500     31.902657  20.430988  18.176569  18.223291
2000     95.489820  57.381326  45.100949  45.635601
rohit@Rohit:/mnt/c/Users/rohit/Desktop$ gcc -fopenmp matrix.c
```

```
rohit@Rohit: /mnt/c/Users/rohit/Desktop
rohit@Rohit:/mnt/c/Users/rohit/Desktop$ gcc -fopenmp matrix.c
rohit@Rohit:/mnt/c/Users/rohit/Desktop$ gcc -fopenmp matrix.c
rohit@Rohit:/mnt/c/Users/rohit/Desktop$ ./a.out
Enter the number of rows and columns of first matrix
3 3
Enter the elements of first matrix
1 1 2
0 2 1
1 2 1
Enter the number of rows and columns of second matrix
3 3
Enter the elements of second matrix
1 0 1
0 1 1
1 2 1
Product of entered matrices:-
3      5      4
1      4      3
2      4      4
rohit@Rohit:/mnt/c/Users/rohit/Desktop$ _
```

## GRAPH:



## CODE:

```
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>
int main(){
int it=1;
printf("The execution time are\nSize\t1\t2\t4\t8\n");
while(it<=4){
int r = 500*it, c = 500*it, i, j, sum =0, k;
//dynamically allocate arrays
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] = rand()/r;
for(i = 0;i < r; i++)
for(j = 0;j < c; j++)
arr2[i][j] = rand()/r;
double x = omp_get_wtime();
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("%d\t",r);
printf("%lf\t", y-x);
for(int p=2;p<=8;p=p*2)
{
double x = omp_get_wtime();
omp_set_num_threads(p);
#pragma omp parallel for private(j, k)
for(i = 0;i < r; i++)
for(j = 0;j < c; j++)
{
arr3[i][j]=0;
for(k = 0;k < r; k++)
arr3[i][j] += arr1[i][k] * arr2[k][j];
}
double y = omp_get_wtime();
printf("%lf\t", y-x);
}
printf("\n");
it++;
}
return 0;
}
```

### Matrix Multiplication:

```
#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;
```

```
}
```