NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY
IT 301 Parallel Computing LAB 4

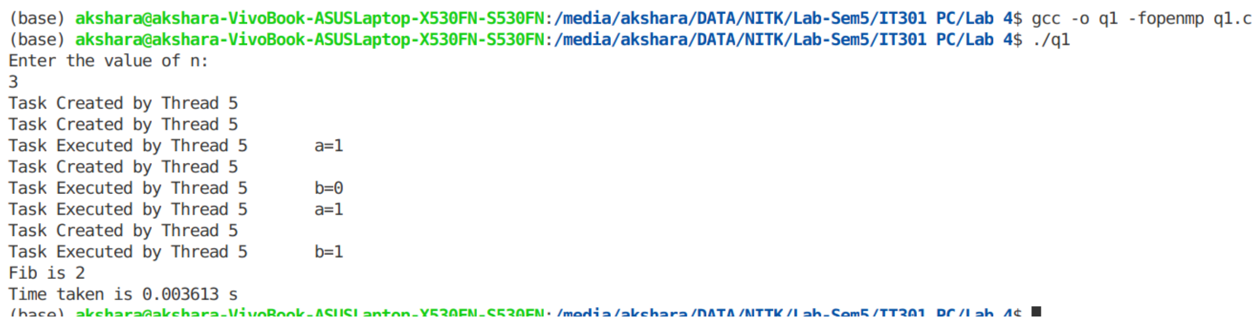
P Akshara - 181IT132

1. Execute following code and observe the working of task directive. Check the result by removing if() clause with task.

```
#include <stdio.h>
#include <omp.h>
int fibo(int n);
int main(void)
{
int n, fib;
double t1, t2;
printf("Enter the value of n:\n");
scanf("%d", &n);
t1 = omp_get_wtime();
#pragma omp parallel shared(n)
{
#pragma omp single
{
fib = fibo(n);
}
}
t2 = omp_get_wtime();
printf("Fib is %d\n", fib);
printf("Time taken is %f s \n", t2 - t1);
return 0;
}

int fibo(int n)
{
int a, b;
if (n < 2)
return n;
else
{
#pragma omp task shared(a) if (n > 5)
{
printf("Task Created by Thread %d\n", omp_get_thread_num());
a = fibo(n - 1);
printf("Task Executed by Thread %d \ta=%d\n", omp_get_thread_num(), a);
}
#pragma omp task shared(b) if (n > 5)
{
printf("Task Created by Thread %d\n", omp_get_thread_num());
b = fibo(n - 2);
printf("Task Executed by Thread %d \tb=%d\n", omp_get_thread_num(), b);
}
#pragma omp taskwait
return a + b;
}
}
```

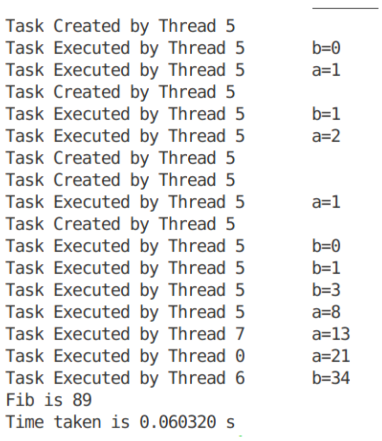
Output

For n=3



```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 4$ gcc -o q1 -fopenmp q1.c
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 4$ ./q1
Enter the value of n:
3
Task Created by Thread 5
Task Created by Thread 5
Task Executed by Thread 5       a=1
Task Created by Thread 5
Task Executed by Thread 5       b=0
Task Executed by Thread 5       a=1
Task Created by Thread 5
Task Executed by Thread 5       b=1
Fib is 2
Time taken is 0.003613 s
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 4$
```

For n=11



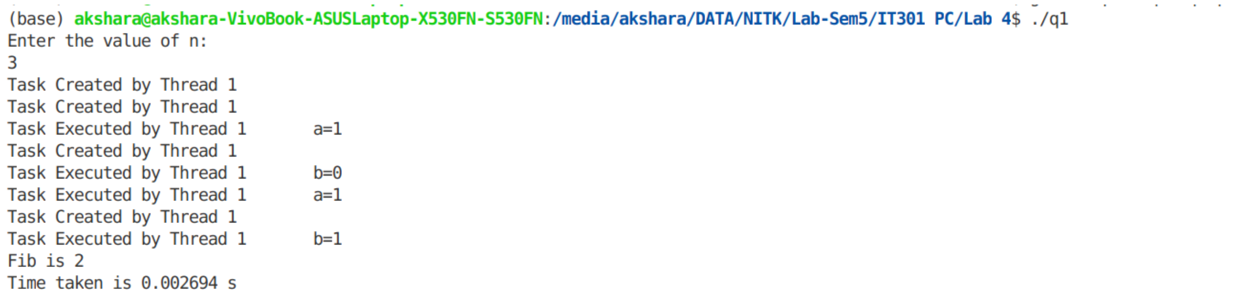
```
Task Created by Thread 5
Task Executed by Thread 5       b=0
Task Executed by Thread 5       a=1
Task Created by Thread 5
Task Executed by Thread 5       b=1
Task Executed by Thread 5       a=2
Task Created by Thread 5
Task Created by Thread 5
Task Executed by Thread 5       a=1
Task Created by Thread 5
Task Executed by Thread 5       b=0
Task Executed by Thread 5       b=1
Task Executed by Thread 5       b=3
Task Executed by Thread 5       a=8
Task Executed by Thread 7       a=13
Task Executed by Thread 0       a=21
Task Executed by Thread 6       b=34
Fib is 89
Time taken is 0.060320 s
```

Analysis

Task scheduling is done for values of n>5 (here when n=11) where tasks are created and executed by the same or different threads for the final computation.

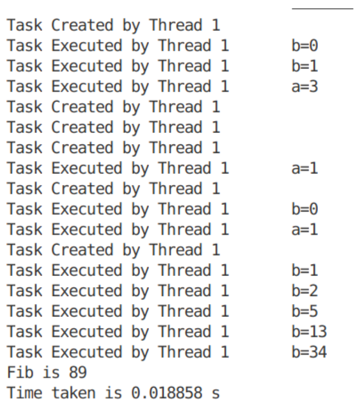
Output after removing if() clause

For n=3



```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 4$ ./q1
Enter the value of n:
3
Task Created by Thread 1
Task Created by Thread 1
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=0
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=1
Fib is 2
Time taken is 0.002694 s
```

For n=11



```
Task Created by Thread 1
Task Executed by Thread 1       b=0
Task Executed by Thread 1       b=1
Task Executed by Thread 1       a=3
Task Created by Thread 1
Task Created by Thread 1
Task Created by Thread 1
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=0
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=1
Task Executed by Thread 1       b=2
Task Executed by Thread 1       b=5
Task Executed by Thread 1       b=13
Task Executed by Thread 1       b=34
Fib is 89
Time taken is 0.018858 s
```

Analysis

Task scheduling is considered for all values of n even when n<5 unlike with the presence of if() clause.

2. Write a C/C++ OpenMP program to find ROWSUM and COLUMNSUM of a matrix a[n][n]. Compare the time of parallel execution with sequential execution.

Sequential

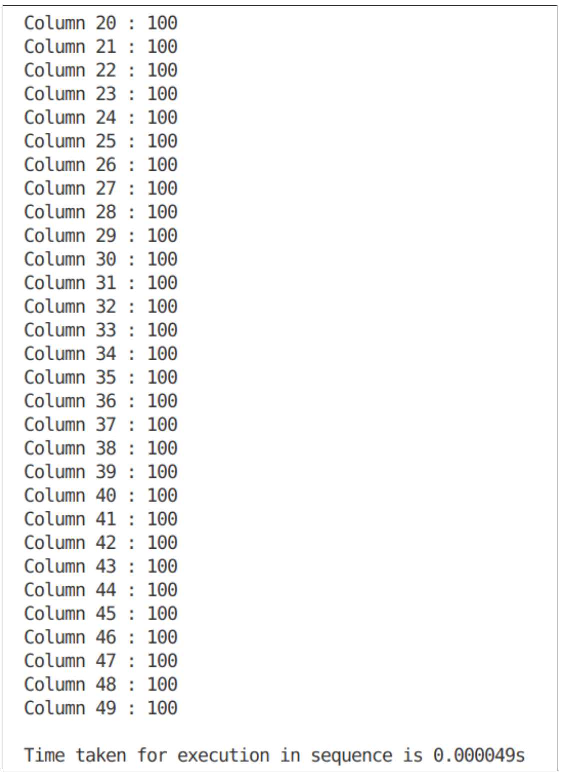
```
#include <stdio.h>
#include <omp.h>
int main(void)
{
unsigned int n, i, j;
double t1, t2;
printf("Enter the value n : ");
scanf("%u", &n);
unsigned int m[n][n];
unsigned int rsum[n];
unsigned int csum[n];
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
if (j % 2 == 0)
m[i][j] = 2;
else if (i % 2 == 0)
m[i][j] = 3;
else m[i][j] = 1;
}
}
t1 = omp_get_wtime();
for (i = 0; i < n; i++)
{
int ans = 0;
for (j = 0; j < n; j++)
//row sum
{
ans += m[i][j];
}
rsum[i] = ans;
}
for (i = 0; i < n; i++)
{
int ans = 0;
for (j = 0; j < n; j++)
//column sum
{
ans += m[j][i];
}
csum[i] = ans;
}
t2 = omp_get_wtime();
printf("Row Sum : \n");
for (i = 0; i < n; i++)
{
printf("Row %u : %u \n", i, rsum[i]);
}
printf("\nColumn Sum : \n");
for (i = 0; i < n; i++)
{
printf("Column %u : %u \n", i, csum[i]);
}
printf("\nTime taken for execution in sequence is %fs \n", t2 - t1);
return 0;
}
```

Parallel

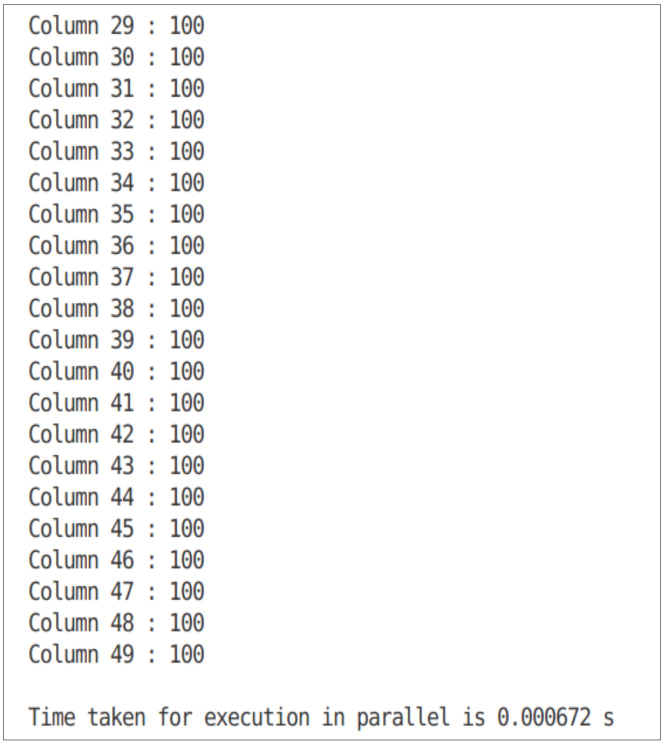
```
#include <stdio.h>
#include <omp.h>
int main(void)
{
unsigned int n, i, j, ans;
double t1, t2;
printf("Enter the value n : ");
scanf("%u", &n);
unsigned int m[n][n];
unsigned int rsum[n];
unsigned int csum[n];
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
if (j % 2 == 0)
m[i][j] = 2;
else if (i % 2 == 0)
m[i][j] = 3;
else
m[i][j] = 1;
}
}
t1 = omp_get_wtime();
#pragma omp parallel shared(n)
//parallel
{
#pragma omp for schedule(static, 5) private(i, j, ans)
for (i = 0; i < n; i++)
{
ans = 0;
for (j = 0; j < n; j++)
//row sum
{
ans += m[i][j];
}
rsum[i] = ans;
}
#pragma omp for schedule(static, 5) private(i, j, ans)
for (i = 0; i < n; i++)
{
ans = 0;
for (j = 0; j < n; j++)
{
ans += m[j][i];
}
csum[i] = ans;
}
}
t2 = omp_get_wtime();
printf("Row Sum : \n");
for (i = 0; i < n; i++)
{
printf("Row %u : %u \n", i, rsum[i]);
}
printf("\nColumn Sum : \n");
for (i = 0; i < n; i++)
//column sum{
printf("Column %u : %u \n", i, csum[i]);

printf("\nTime taken for execution in parallel is %f s \n", t2 - t1);
return 0;
}
```

Outputs

For n=50

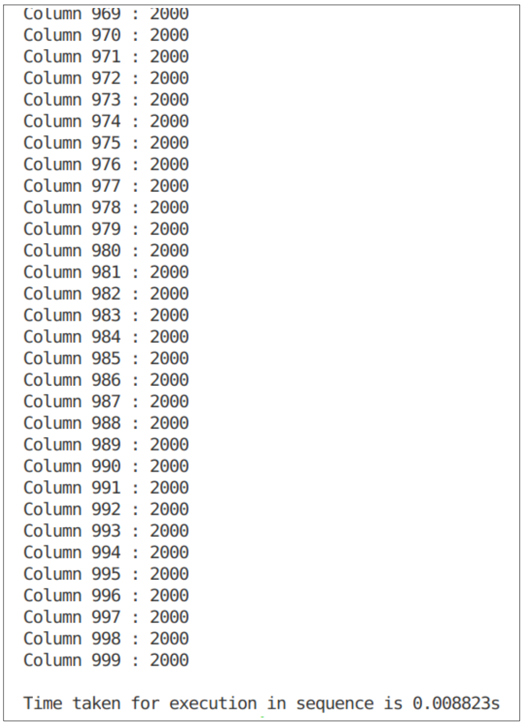


```
Column 20 : 100
Column 21 : 100
Column 22 : 100
Column 23 : 100
Column 24 : 100
Column 25 : 100
Column 26 : 100
Column 27 : 100
Column 28 : 100
Column 29 : 100
Column 30 : 100
Column 31 : 100
Column 32 : 100
Column 33 : 100
Column 34 : 100
Column 35 : 100
Column 36 : 100
Column 37 : 100
Column 38 : 100
Column 39 : 100
Column 40 : 100
Column 41 : 100
Column 42 : 100
Column 43 : 100
Column 44 : 100
Column 45 : 100
Column 46 : 100
Column 47 : 100
Column 48 : 100
Column 49 : 100

Time taken for execution in sequence is 0.000049s
```

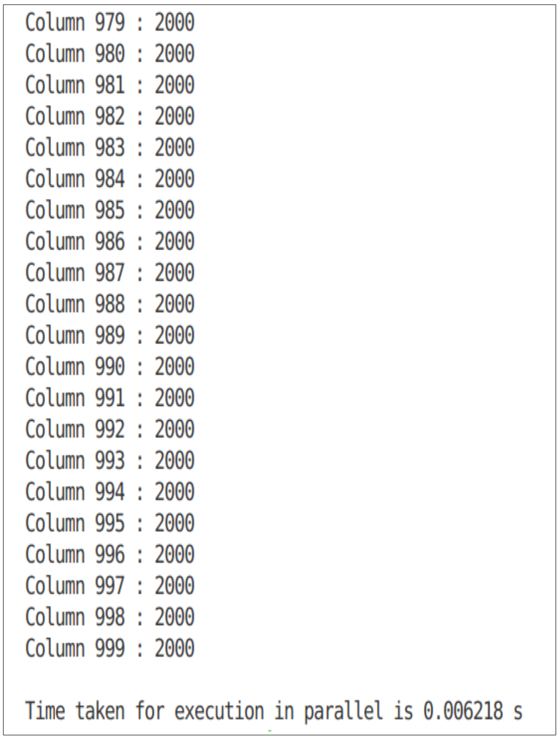


```
Column 29 : 100
Column 30 : 100
Column 31 : 100
Column 32 : 100
Column 33 : 100
Column 34 : 100
Column 35 : 100
Column 36 : 100
Column 37 : 100
Column 38 : 100
Column 39 : 100
Column 40 : 100
Column 41 : 100
Column 42 : 100
Column 43 : 100
Column 44 : 100
Column 45 : 100
Column 46 : 100
Column 47 : 100
Column 48 : 100
Column 49 : 100

Time taken for execution in parallel is 0.000672 s
```

For n=1000

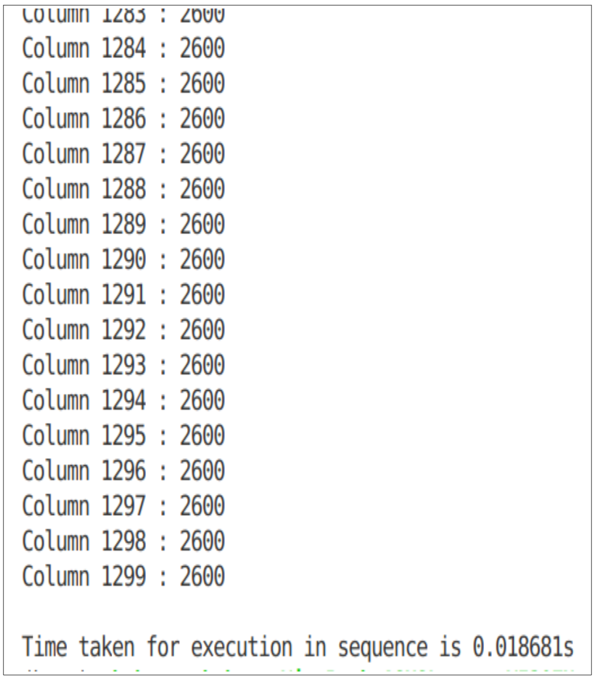


```
Column 969 : 2000
Column 970 : 2000
Column 971 : 2000
Column 972 : 2000
Column 973 : 2000
Column 974 : 2000
Column 975 : 2000
Column 976 : 2000
Column 977 : 2000
Column 978 : 2000
Column 979 : 2000
Column 980 : 2000
Column 981 : 2000
Column 982 : 2000
Column 983 : 2000
Column 984 : 2000
Column 985 : 2000
Column 986 : 2000
Column 987 : 2000
Column 988 : 2000
Column 989 : 2000
Column 990 : 2000
Column 991 : 2000
Column 992 : 2000
Column 993 : 2000
Column 994 : 2000
Column 995 : 2000
Column 996 : 2000
Column 997 : 2000
Column 998 : 2000
Column 999 : 2000

Time taken for execution in sequence is 0.008823s
```

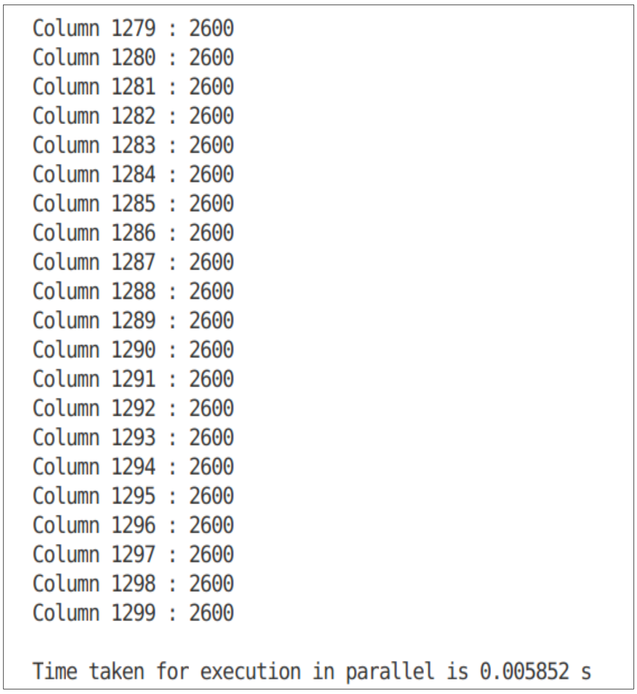


```
Column 979 : 2000
Column 980 : 2000
Column 981 : 2000
Column 982 : 2000
Column 983 : 2000
Column 984 : 2000
Column 985 : 2000
Column 986 : 2000
Column 987 : 2000
Column 988 : 2000
Column 989 : 2000
Column 990 : 2000
Column 991 : 2000
Column 992 : 2000
Column 993 : 2000
Column 994 : 2000
Column 995 : 2000
Column 996 : 2000
Column 997 : 2000
Column 998 : 2000
Column 999 : 2000

Time taken for execution in parallel is 0.006218 s
```

For n=1300



```
Column 1283 : 2600
Column 1284 : 2600
Column 1285 : 2600
Column 1286 : 2600
Column 1287 : 2600
Column 1288 : 2600
Column 1289 : 2600
Column 1290 : 2600
Column 1291 : 2600
Column 1292 : 2600
Column 1293 : 2600
Column 1294 : 2600
Column 1295 : 2600
Column 1296 : 2600
Column 1297 : 2600
Column 1298 : 2600
Column 1299 : 2600

Time taken for execution in sequence is 0.018681s
```



```
Column 1279 : 2600
Column 1280 : 2600
Column 1281 : 2600
Column 1282 : 2600
Column 1283 : 2600
Column 1284 : 2600
Column 1285 : 2600
Column 1286 : 2600
Column 1287 : 2600
Column 1288 : 2600
Column 1289 : 2600
Column 1290 : 2600
Column 1291 : 2600
Column 1292 : 2600
Column 1293 : 2600
Column 1294 : 2600
Column 1295 : 2600
Column 1296 : 2600
Column 1297 : 2600
Column 1298 : 2600
Column 1299 : 2600

Time taken for execution in parallel is 0.005852 s
```

Analysis

For large values of n (matrix dimensions), parallel program executes in less time compared to sequential.

3. Write a C/C++ OpenMP program to perform matrix multiplication. Compare the time of parallel execution with sequential execution.

Sequential

```c
#include <stdio.h>
#include <omp.h>
int main(void)
{
unsigned int n, i, j, k, sum = 0;
double t1, t2;
printf("Enter the value n: ");
scanf("%u", &n);
unsigned int m1[n][n];
unsigned int m2[n][n];
unsigned int m3[n][n];
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
m1[i][j] = 1;
m2[i][j] = 1;
m3[i][j] = 0;
}
}
t1 = omp_get_wtime();
for (i = 0; i < n; i++)
// matrix multiplication
{
for (j = 0; j < n; j++)
{
```

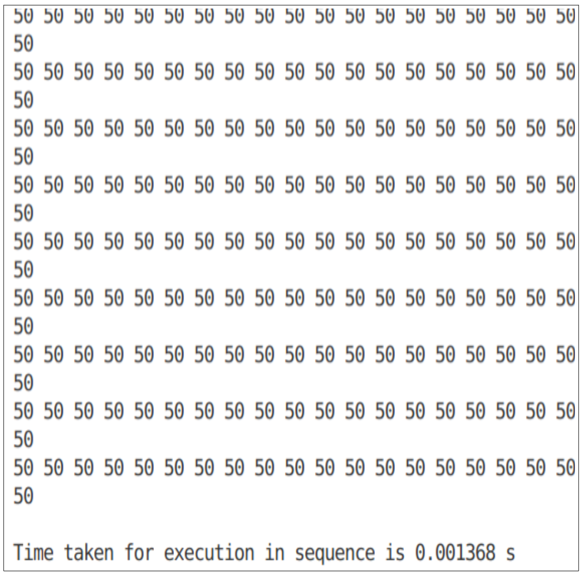
```
sum = 0;
for (k = 0; k < n; k++)
{
sum = sum + (m1[i][k] * m2[k][j]);
}
m3[i][j] = m3[i][j] + sum;
}
}
t2 = omp_get_wtime();
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
printf("%u ", m3[i][j]);
}
printf("\n");
}
printf("\nTime taken for execution in sequence is %f s \n", t2 - t1);
return 0;
}
```

Parallel

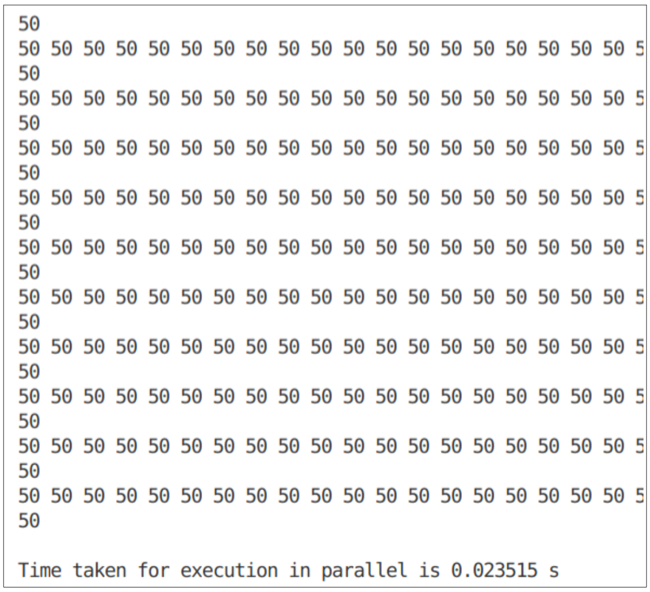
```
#include <stdio.h>
#include <omp.h>
int main(void)
{
unsigned int n, i, j, k, sum = 0;
double t1, t2;
printf("Enter the value n : ");
scanf("%u", &n);
unsigned int m1[n][n];
unsigned int m2[n][n];
unsigned int m3[n][n];
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
m1[i][j] = 1;
m2[i][j] = 1;
m3[i][j] = 0;
}
}
t1 = omp_get_wtime();
#pragma omp parallel shared(n)
{
// parallel
#pragma omp for schedule(static, 10) collapse(2) private(i, j, k, sum)
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
sum = 0;
for (k = 0; k < n; k++)
{
sum = sum + (m1[i][k] * m2[k][j]);
}
m3[i][j] = m3[i][j] + sum;
}
}
}
t2 = omp_get_wtime();
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
printf("%u ", m3[i][j]);
}
printf("\n");
}
printf("\nTime taken for execution in parallel is %f s \n", t2 - t1);
return 0;
}
```

Outputs

For n=50

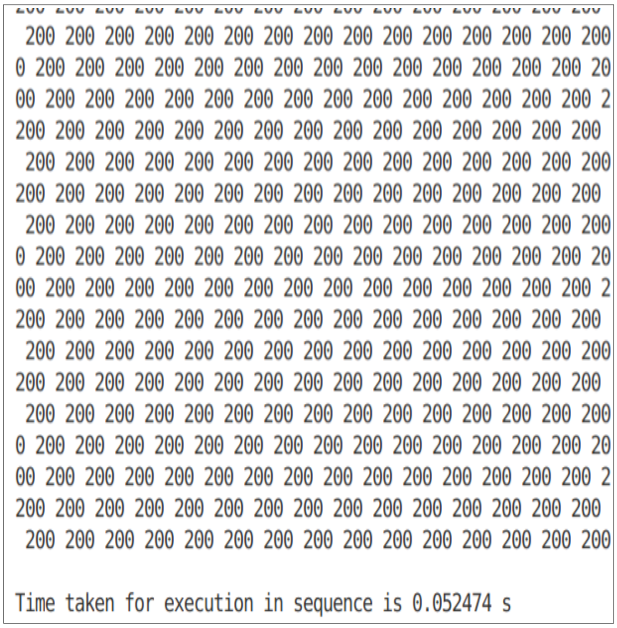


```
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
50

Time taken for execution in sequence is 0.001368 s
```

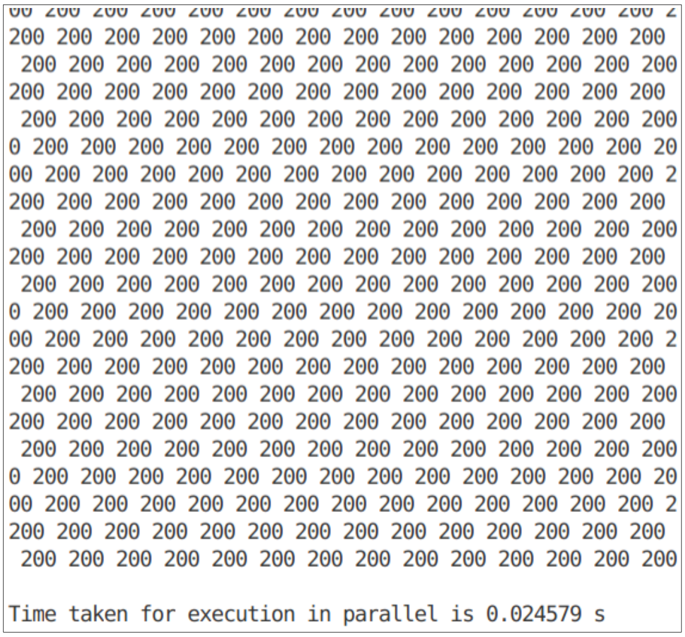


```
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 5
50

Time taken for execution in parallel is 0.023515 s
```

For n=200

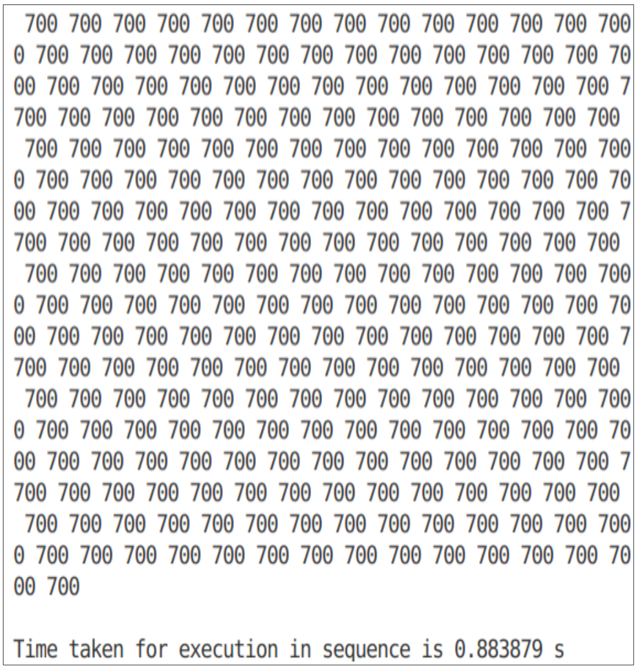


```
200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 20
00 200 200 200 200 200 200 200 200 200 200 200 200 200 200 2
200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 20
00 200 200 200 200 200 200 200 200 200 200 200 200 200 200 2
200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 20
00 200 200 200 200 200 200 200 200 200 200 200 200 200 200 2
200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200

Time taken for execution in sequence is 0.052474 s
```

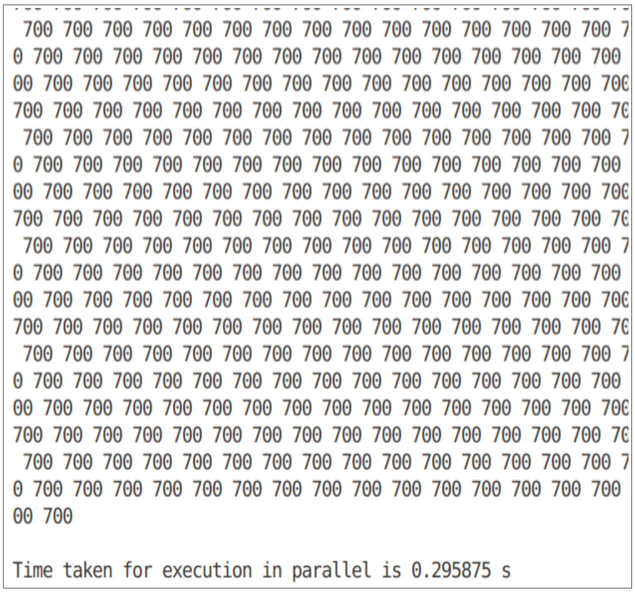


```
00 200 200 200 200 200 200 200 200 200 200 200 200 200 2
200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 20
00 200 200 200 200 200 200 200 200 200 200 200 200 200 2
200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 20
00 200 200 200 200 200 200 200 200 200 200 200 200 200 2
200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 20
00 200 200 200 200 200 200 200 200 200 200 200 200 200 2
200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200

Time taken for execution in parallel is 0.024579 s
```

For n=700



```
 700 700 700 700 700 700 700 700 700 700 700 700 700 700
0 700 700 700 700 700 700 700 700 700 700 700 700 700 70
00 700 700 700 700 700 700 700 700 700 700 700 700 700 7
700 700 700 700 700 700 700 700 700 700 700 700 700 700
 700 700 700 700 700 700 700 700 700 700 700 700 700 700
0 700 700 700 700 700 700 700 700 700 700 700 700 700 70
00 700 700 700 700 700 700 700 700 700 700 700 700 700 7
700 700 700 700 700 700 700 700 700 700 700 700 700 700
 700 700 700 700 700 700 700 700 700 700 700 700 700 700
0 700 700 700 700 700 700 700 700 700 700 700 700 700 70
00 700 700 700 700 700 700 700 700 700 700 700 700 700 7
700 700 700 700 700 700 700 700 700 700 700 700 700 700
 700 700 700 700 700 700 700 700 700 700 700 700 700 700
0 700 700 700 700 700 700 700 700 700 700 700 700 700 70
00 700 700 700 700 700 700 700 700 700 700 700 700 700 7
700 700 700 700 700 700 700 700 700 700 700 700 700 700
 700 700 700 700 700 700 700 700 700 700 700 700 700 700
0 700 700 700 700 700 700 700 700 700 700 700 700 700 70
00 700

Time taken for execution in sequence is 0.883879 s
```



```
 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 7
0 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
00 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 70
 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 7
0 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
00 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 70
 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 7
0 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
00 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 70
 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 7
0 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
00 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 70
 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700 7
0 700 700 700 700 700 700 700 700 700 700 700 700 700 700 700
00 700

Time taken for execution in parallel is 0.295875 s
```

Analysis

For large values of n (matrix dimensions), parallel program executes in less time compared to sequential.