NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL

DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 4

02nd September 2020

Faculty: Dr. Geetha V and Mrs. Tanmayee

---

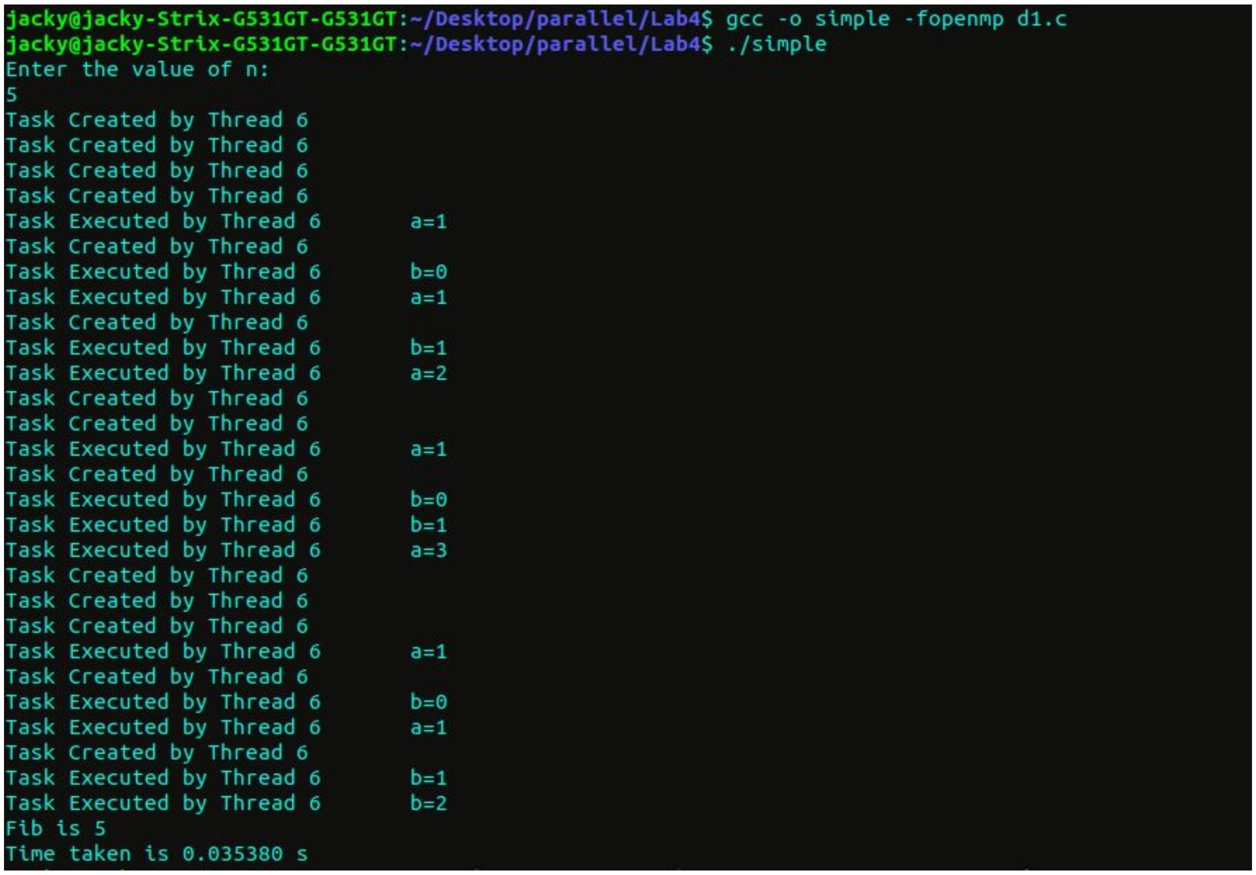
1. Program 1

Execute following code and observe the working of task directive.

```
#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;

}
```



```
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ gcc -o simple -fopenmp d1.c
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ ./simple
Enter the value of n:
5
Task Created by Thread 6
Task Created by Thread 6
Task Created by Thread 6
Task Created by Thread 6
Task Executed by Thread 6       a=1
Task Created by Thread 6
Task Executed by Thread 6       b=0
Task Executed by Thread 6       a=1
Task Created by Thread 6
Task Executed by Thread 6       b=1
Task Executed by Thread 6       a=2
Task Created by Thread 6
Task Created by Thread 6
Task Executed by Thread 6       a=1
Task Created by Thread 6
Task Executed by Thread 6       b=0
Task Executed by Thread 6       b=1
Task Executed by Thread 6       a=3
Task Created by Thread 6
Task Created by Thread 6
Task Created by Thread 6
Task Executed by Thread 6       a=1
Task Created by Thread 6
Task Executed by Thread 6       b=0
Task Executed by Thread 6       a=1
Task Created by Thread 6
Task Executed by Thread 6       b=1
Task Executed by Thread 6       b=2
Fib is 5
Time taken is 0.035380 s
```

OBSERVATION:

Working of task directive:
The task pragma can be used to explicitly define a task. We use the task pragma when we want to identify a block of code to be executed in parallel with the code outside the task region. The task pragma can be useful for parallelizing irregular algorithms such as pointer chasing or recursive algorithms.

In this program task pragma is used for recursive algorithm for fibonacci series.

#pragma omp task shared(a) if(n>5)

The *clause* parameter that is used here is shared (*list*).

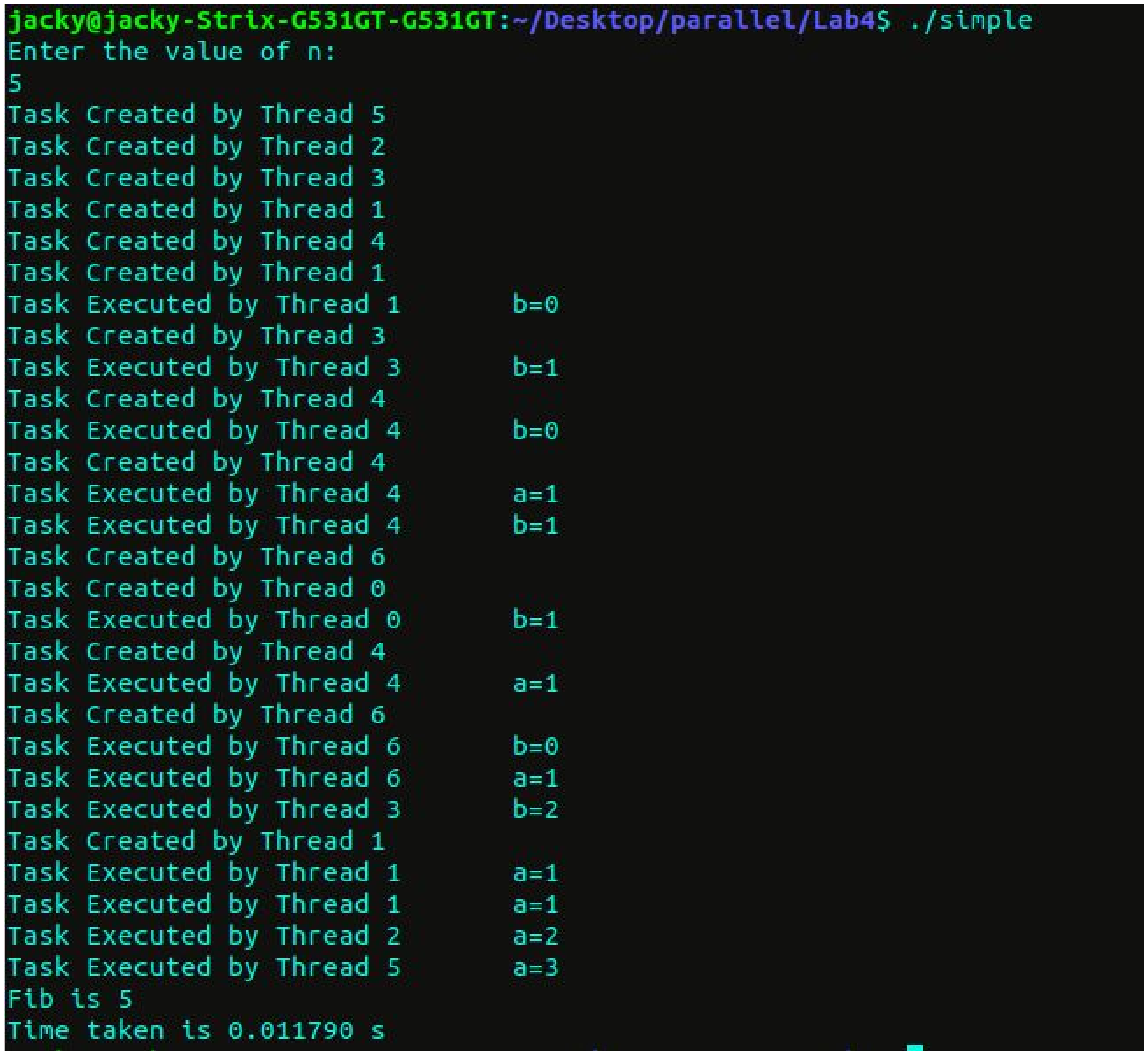
It declares the scope of the comma-separated data variables in *list* to be shared across all threads.

The taskwait pragma is used to specify a *wait* for child tasks to be completed that are generated by the current task.

A final task is a task that makes all its child tasks become final and included tasks. A final task is generated when either of the following conditions is a nonzero value:

- A final clause is specified on a task construct and the final clause expression evaluates to nonzero value.
- The generated task is a child task of a final task.

Check the result by removing if() clause with task.



```
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ ./simple
Enter the value of n:
5
Task Created by Thread 5
Task Created by Thread 2
Task Created by Thread 3
Task Created by Thread 1
Task Created by Thread 4
Task Created by Thread 1
Task Executed by Thread 1       b=0
Task Created by Thread 3
Task Executed by Thread 3       b=1
Task Created by Thread 4
Task Executed by Thread 4       b=0
Task Created by Thread 4
Task Executed by Thread 4       a=1
Task Executed by Thread 4       b=1
Task Created by Thread 6
Task Created by Thread 0
Task Executed by Thread 0       b=1
Task Created by Thread 4
Task Executed by Thread 4       a=1
Task Created by Thread 6
Task Executed by Thread 6       b=0
Task Executed by Thread 6       a=1
Task Executed by Thread 3       b=2
Task Created by Thread 1
Task Executed by Thread 1       a=1
Task Executed by Thread 1       a=1
Task Executed by Thread 2       a=2
Task Executed by Thread 5       a=3
Fib is 5
Time taken is 0.011790 s
```

If we do not use if() clause, Task is executed and created parallelly by different threads in the given range.Here Time taken is less as compared to above program with if() clause.

#pragma omp taskwait

The taskwait pragma is used to specify a *wait* for child tasks to be completed that are generated by the current task.

The task pragma can be useful for parallelizing irregular algorithms such as pointer chasing or recursive algorithms.

In this program task pragma is used for recursive algorithm for fibonacci series.

Programming exercises in OpenMP

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.

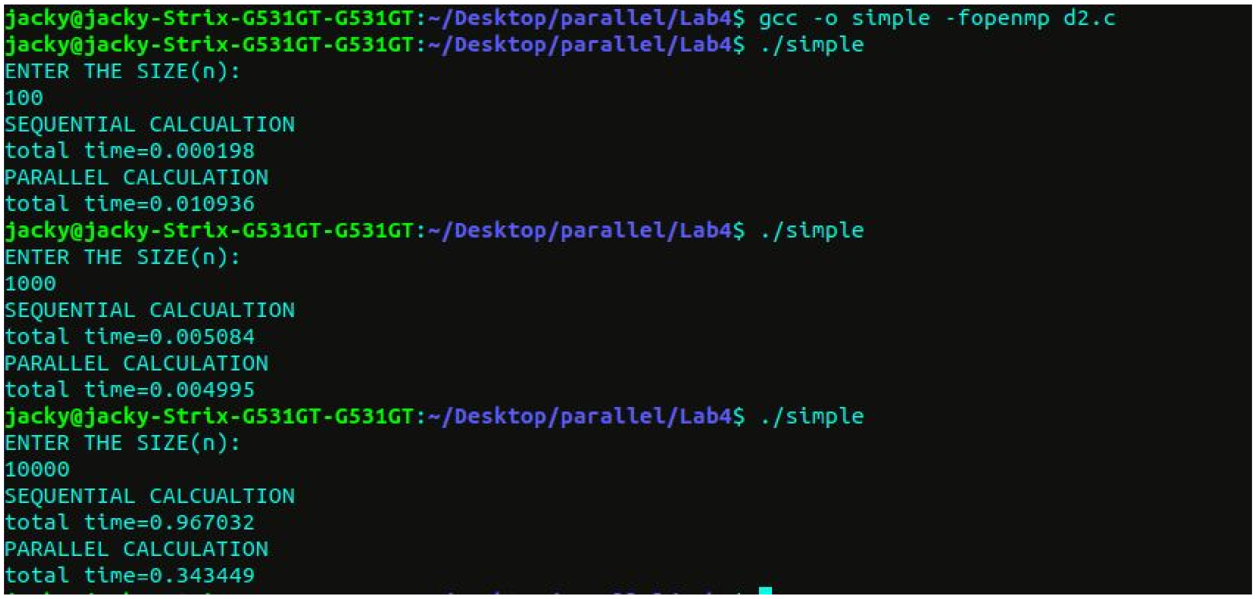
PROGRAM :

```
#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

int main() {

        int SIZE;

        printf("ENTER THE SIZE(n):\n");

        scanf("%d",&SIZE);

        int* rowsum = (int*) malloc (SIZE * sizeof(int));

        int* colsum = (int*) malloc (SIZE * sizeof(int));


        int** matrix = (int**) malloc (SIZE * sizeof(int*));


        for(int i=0; i<SIZE; ++i) {

        matrix[i] = (int*) malloc (SIZE * sizeof(int));

        }
```

```
for(int i=0; i<SIZE; ++i) {

for(int j=0; j<SIZE; ++j) {

matrix[i][j] = rand() % 100;

}

}


printf("SEQUENTIAL CALCUALTION\n");

double st = omp_get_wtime();

for(int i=0; i<SIZE; ++i) {

rowsum[i] = 0;

colsum[i] = 0;

for(int j=0; j<SIZE; ++j) {

rowsum[i] += matrix[i][j];

colsum[i] += matrix[j][i];

}

}

double et = omp_get_wtime();

printf("total time=%f\n", et - st);


printf("PARALLEL CALCULATION\n");

st = omp_get_wtime();

#pragma omp parallel for shared(rowsum, colsum)

for(int i=0; i<SIZE; i++) {

rowsum[i] = 0;

colsum[i] = 0;
```

```
        int tmp1=0, tmp2=0;
        for(int j=0; j<SIZE; ++j) {
                tmp1 += matrix[i][j];
                tmp2 += matrix[j][i];
        }
        #pragma omp critical
        {
                rowsum[i] = tmp1;
                colsum[i] = tmp2;
        }
        }
        et = omp_get_wtime();
    printf("total time=%f\n", et - st);
}
```

OUTPUT :



```
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ gcc -o simple -fopenmp d2.c
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ ./simple
ENTER THE SIZE(n):
100
SEQUENTIAL CALCUALTION
total time=0.000198
PARALLEL CALCULATION
total time=0.010936
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ ./simple
ENTER THE SIZE(n):
1000
SEQUENTIAL CALCUALTION
total time=0.005084
PARALLEL CALCULATION
total time=0.004995
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ ./simple
ENTER THE SIZE(n):
10000
SEQUENTIAL CALCUALTION
total time=0.967032
PARALLEL CALCULATION
total time=0.343449
```

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

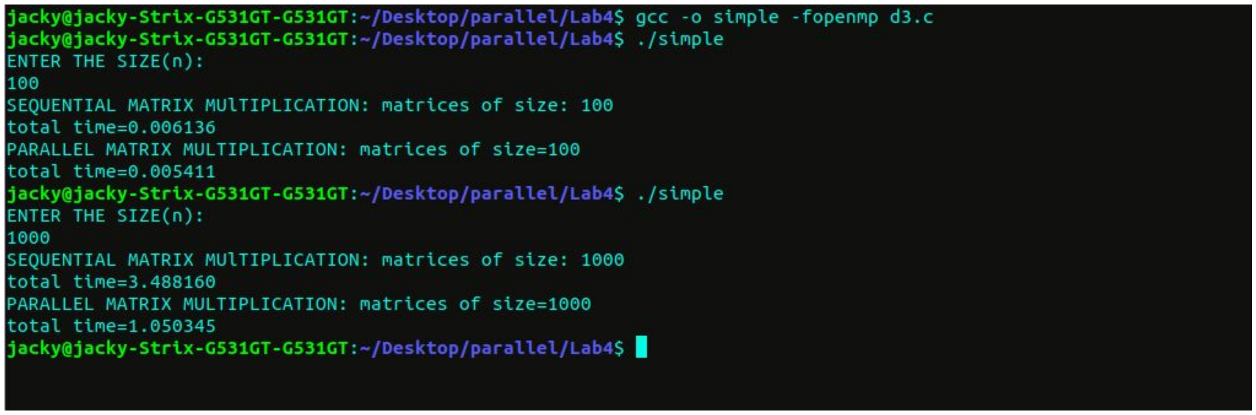
PROGRAM :

```
#include<stdlib.h>

#include<omp.h>

#include<stdio.h>

int main() {

        int SIZE;

        printf("ENTER THE SIZE(n):\n");

        scanf("%d",&SIZE);

        //////////////////////////////////////////////////////Sequencial////////////////////////////////////////////////////////////

        printf("SEQUENTIAL MATRIX MUITIPLICATION: matrices of size: %d\n", SIZE);

        int **mat1 = (int **) malloc (SIZE * sizeof(int*));

        int **mat2 = (int **) malloc (SIZE * sizeof(int*));

        int **res  = (int **) malloc (SIZE * sizeof(int*));

    for(int i=0; i<SIZE; ++i) {

        mat1[i] = (int*) malloc (SIZE * sizeof(int));

        mat2[i] = (int*) malloc (SIZE * sizeof(int));

        res[i]  = (int*) malloc (SIZE * sizeof(int));

  }


  for(int i=0; i<SIZE; ++i) {
        for(int j=0; j<SIZE; ++j) {
        mat1[i][j] = rand()%100;
        mat2[i][j] = rand()%100;
        }
}

//perform matrix multiplication
double st1 = omp_get_wtime();
for(int i=0; i<SIZE; ++i) {
        for(int j=0; j<SIZE; ++j) {
        for(int k=0; k<SIZE; ++k) {
        res[i][j] = mat1[i][k] * mat2[k][j];
        }
        }
}
double et1 = omp_get_wtime();
printf("total time=%f\n", et1 - st1);
        ////////////////////////////////////////////////////////////////////Parallel ///////////////////////////////////////////////////////
        printf("PARALLEL MATRIX MULTIPLICATION: matrices of size=%d\n", SIZE);
        int** mat3 = (int **) malloc (SIZE * sizeof(int*));
        int** mat4 = (int **) malloc (SIZE * sizeof(int*));
        int** res2  = (int **) malloc (SIZE * sizeof(int*));

        for(int i=0; i<SIZE; ++i) {
        mat3[i] = (int*) malloc (SIZE * sizeof(int));
```

```
mat4[i] = (int*) malloc (SIZE * sizeof(int));

res2[i]  = (int*) malloc (SIZE * sizeof(int));

}


//initialize matrices

for(int i=0; i<SIZE; ++i) {

for(int j=0; j<SIZE; ++j) {

mat3[i][j] = rand()%100;

mat4[i][j] = rand()%100;

}

}


//perform parallel matrix multiplication

double st2 = omp_get_wtime();

#pragma omp parallel for shared(mat1, mat2, res)

for(int i=0; i<SIZE; ++i) {

for(int j=0; j<SIZE; ++j) {

int tmp = 0;

for(int k=0; k<SIZE; ++k) {

        tmp += mat1[i][k] * mat2[k][j];

}

#pragma omp critical

{

        res2[i][j] = tmp;

}
```

```
        }

        }

        double et2 = omp_get_wtime();

        printf("total time=%f\n", et2 - st2);

}
```

OUTPUT :



```
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ gcc -o simple -fopenmp d3.c
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ ./simple
ENTER THE SIZE(n):
100
SEQUENTIAL MATRIX MUlTIPLICATION: matrices of size: 100
total time=0.006136
PARALLEL MATRIX MULTIPLICATION: matrices of size=100
total time=0.005411
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$ ./simple
ENTER THE SIZE(n):
1000
SEQUENTIAL MATRIX MUlTIPLICATION: matrices of size: 1000
total time=3.488160
PARALLEL MATRIX MULTIPLICATION: matrices of size=1000
total time=1.050345
jacky@jacky-Strix-G531GT-G531GT:~/Desktop/parallel/Lab4$
```

NAME: BHAJAN KUMAR BARMAN
ROLL NO.: 181IT211