

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 4

02nd September 2020

Submitted By: Harsh Agarwal (181IT117)

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

Code:

```
#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)
        {
            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)
```

```

    {
        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 = 7:

```

MINGW64:/d/Academics/5th Sem/Parallel Computing/Lab/Lab4
Task Created by Thread 5
Task Created by Thread 5
Task Executed by Thread 5      b=1
Task Executed by Thread 1      b=1
Task Created by Thread 2
Task Created by Thread 2
Task Executed by Thread 2      b=0
Task Created by Thread 0
Task Created by Thread 0
Task Created by Thread 0
Task Executed by Thread 0      b=0
Task Created by Thread 0
Task Executed by Thread 0      a=1
Task Executed by Thread 0      b=1
Task Created by Thread 0
Task Created by Thread 0
Task Executed by Thread 0      b=1
Task Created by Thread 0
Task Created by Thread 0
Task Executed by Thread 0      b=0
Task Created by Thread 0
Task Executed by Thread 0      a=1
Task Executed by Thread 0      a=1
Task Executed by Thread 0      a=2
Task Executed by Thread 0      a=3
Task Created by Thread 6
Task Created by Thread 6
Task Executed by Thread 6      b=0
Task Created by Thread 0
Task Executed by Thread 0      a=1
Task Created by Thread 3
Task Executed by Thread 3      a=1
Task Executed by Thread 6      a=1
Task Executed by Thread 2      a=1
Task Executed by Thread 4      b=2
Task Executed by Thread 4      b=5
Task Executed by Thread 5      a=2
Task Executed by Thread 1      b=3
Task Executed by Thread 1      a=8
Fib is 13
Time taken is 0.148000 s
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab4

```

For n = 4:

```
MINGW64/d/Academics/5th Sem/Parallel Computing/Lab/Lab4
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab4
$ gcc Prog1.c -fopenmp
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab4
$ ./a.exe
Enter the value of n:
4
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
Fib is 3
Time taken is 0.012000 s
```

After removing the if() clause:

For n = 4:

```
MINGW64/d/Academics/5th Sem/Parallel Computing/Lab/Lab4
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab4
$ ./a.exe
Enter the value of n:
4
Task Created by Thread 6
Task Created by Thread 6
Task Executed by Thread 6      b=0
Task Created by Thread 0
Task Executed by Thread 0      a=1
Task Created by Thread 5
Task Created by Thread 5
Task Executed by Thread 5      b=1
Task Created by Thread 3
Task Executed by Thread 6      b=1
Task Created by Thread 3
Task Executed by Thread 3      b=0
Task Created by Thread 7
Task Executed by Thread 7      a=1
Task Executed by Thread 3      a=1
Task Executed by Thread 5      a=2
Fib is 3
Time taken is 0.006000 s
```

Analysis:

After removing the if() clause, we are able to achieve Task Scheduling for all the values of n, but in the presence of if() clause, it was not possible for values of $n < 5$.

Therefore in this case of $n = 4$, our execution time reduces.

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.

Code:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#include<time.h>

int main(void){
    int n,i,j;
    double t1_s,t2_s,t1_p,t2_p;
    printf("Enter the value n : ");
    scanf("%u",&n);
    int m[n][n], rsum[n],csum[n];
    srand(time(0));
    printf("matrix:\n");
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            int k=(rand())%10000;
            m[i][j]=k;
            printf("%d ",k);
        }
        printf("\n");
    }

    t1_s=omp_get_wtime();
    for(i=0;i<n;i++){
        int temp=0;
        for(j=0;j<n;j++)temp+=m[i][j];
        rsum[i]=temp;
    }
    for(i=0;i<n;i++){
        int temp=0;
        for(j=0;j<n;j++)
            temp+=m[j][i];
        csum[i]=temp;
    }

    t2_s=omp_get_wtime();
```

```

int temp;

t1_p=omp_get_wtime();
#pragma omp parallel shared(n)
{
#pragma omp for schedule(static,10) private(i,j,temp)
for(i=0;i<n;i++){
    temp=0;
    for(j=0;j<n;j++)
        temp+=m[i][j];
    rsum[i]=temp;
}
#pragma omp for schedule(static,10) private(i,j,temp)
for(i=0;i<n;i++){
    temp=0;
    for(j=0;j<n;j++)
        temp+=m[j][i];
    csum[i]=temp;
}
}
t2_p=omp_get_wtime();

printf("Row Sum : \n");
for(i=0;i<n;i++){
    printf("Row %d: %d \n", i,rsum[i]);
}
printf("\nColumn Sum : \n");
for(i=0;i<n;i++){
    printf("Column %d : %d \n", i,csum[i]);
}
printf("\nTime taken for sequential execution is %f s \n",t2_s-t1_s);
printf("\nTime taken for parallel execution is %f s \n",t2_p-t1_p);
return 0;
}

```

Output:

```

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab4
$ gcc Prog2.c -fopenmp

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab4
$ ./a.exe
Enter the value n : 4
matrix:
6058 5378 2398 4847
8462 6649 1782 9492
5319 2491 4818 6675
3264 7211 5568 5201
Row Sum :
Row 0: 18681
Row 1: 26385
Row 2: 19303
Row 3: 21244

Column Sum :
Column 0 : 23103
Column 1 : 21729
Column 2 : 14566
Column 3 : 26215

Time taken for sequential execution is 0.000000 s

Time taken for parallel execution is 0.002000 s

```

Analysis:

The matrix is generated using random function. For parallel execution, schedule clause is used with parallel for directive which calculates the sum of each row and column.

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

Code:

```

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

    int tmp,i,j,k,M,N,P;printf("Enter the value of M:\n");
    scanf("%d",&M);
    printf("Enter the value of P:\n");
    scanf("%d",&P);
    printf("Enter the value of N:\n");
    scanf("%d",&N);
    int C[M][N],S[M][N],B[P][N],A[M][P];
    srand(time(0));
    printf("Matrix A:\n");

```

```

for(i=0;i<M;i++){
    for(j=0;j<P;j++){
        int k=(rand())%10;
        A[i][j]=k;
        printf("%d ",k);
    }
    printf("\n");
}
printf("Matrix B:\n");
for(i=0;i<P;i++){
    for(j=0;j<N;j++){
        int k=(rand())%10;
        B[i][j]=k;
        printf("%d ",k);
    }
    printf("\n");
}
double start_time = omp_get_wtime();
#pragma omp parallel for private(tmp, j, k)
for (i=0; i<M; i++){
    for (j=0; j<N; j++){
        tmp = 0.0;
        for( k=0; k<P; k++){
            /*C(i,j) = sum(over k) A(i,k) * B(k,j)*/
            tmp += A[i][k] * B[k][j];
        }
        C[i][j] = tmp;
    }
}
double run_time = omp_get_wtime() - start_time;
printf("Multiplied matrix is \n");
for (int i = 0; i < M; ++i){
    for (int j = 0; j < N; ++j){
        printf("%d\t", C[i][j]);
    }
    printf("\n");
}
printf("Time taken by parallel execution %f s \n",run_time);
clock_t t;
t = clock();
for (i=0; i<M; i++){
    for (j=0; j<N; j++){
        tmp = 0.0;
        for( k=0; k<P; k++){
            /*C(i,j) = sum(over k) A(i,k) * B(k,j)*/
            tmp += A[i][k] * B[k][j];
        }
        S[i][j] = tmp;
    }
}

```

```

    }
}
t= clock() - t;
double time_taken = ((double)t)/CLOCKS_PER_SEC;
printf("Time taken by sequential execution %f s \n",time_taken);
return 0;
}

```

Output:

```

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab4
$ gcc Prog3.c -fopenmp
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab4
$ ./a.exe
Enter the value of M:
3
Enter the value of P:
4
Enter the value of N:
5
Matrix A:
6 2 8 5
5 3 9 2
2 2 1 3
Matrix B:
0 4 1 5 2
8 8 6 7 1
7 0 4 9 1
7 4 1 8 3
Multiplied matrix is
107    60    55    156    37
101    52    61    143    28
44     36    21    57     16
Time taken by parallel execution 0.002000 s
Time taken by sequential execution 0.000000 s

```

Analysis:

The matrices are generated using random function. For parallel execution, parallel for directive is used keeping the temporary variables (iterator variables on the rows as well as columns) private.