HIGH PERFORMANCE COMPUTING

LAB ASSIGNMENT – 05

BATCH – 04

2303A51221

T.SAI SATHWIK

TASK PARALLELISM USING OPENMP

EXPERIMENT – 01 PARALLEL FIBONACCI USING OPENMP TASKS

AIM

To implement Fibonacci computation using OpenMP task parallelism and understand recursive task creation and synchronization.

CODE

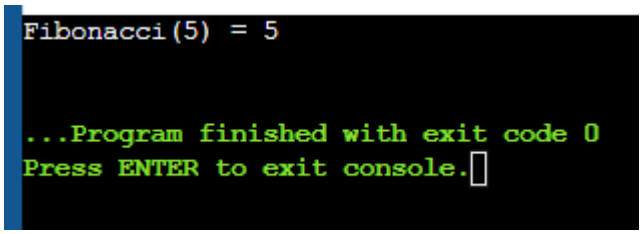
```c
#include <stdio.h>
#include <omp.h>

int fib(int n) {
    int x, y;

    if (n < 2)
        return n;

    #pragma omp task shared(x)
    x = fib(n - 1);

    #pragma omp task shared(y)
    y = fib(n - 2);

    #pragma omp taskwait
    return x + y;
}

int main() {

    int n = 5, result;

    #pragma omp parallel
    {
        #pragma omp single
        result = fib(n);
    }

    printf("Fibonacci(%d) = %d\n", n, result);

    return 0;
}
```

#include <stdio.h>

#include <omp.h>

```
int fib(int n) {
   int x, y;
   if (n < 2)
      return n;
   #pragma omp task shared(x)
   x = fib(n - 1);
   #pragma omp task shared(y)
   y = fib(n - 2);
   #pragma omp taskwait
   return x + y;
}
int main() {
   int n = 5, result;
   #pragma omp parallel
   {
      #pragma omp single
      result = fib(n);
   }
   printf("Fibonacci(%d) = %d\n", n, result);
   return 0;
}
```

OUTPUT



EXPLANATION

- **#pragma omp parallel creates a team of threads.**
- **#pragma omp single ensures only one thread starts the Fibonacci computation.**
- **#pragma omp task creates separate tasks for fib(n-1) and fib(n-2).**
- **Tasks are executed in parallel by available threads.**

- **#pragma omp taskwait ensures both tasks finish before returning result.**
- **Recursive calls generate multiple parallel tasks dynamically.**

OBSERVATION

- **Fibonacci computation is divided into smaller tasks.**
- **Multiple threads execute recursive calls simultaneously.**
- **Task synchronization is handled using taskwait.**
- **Performance improves compared to pure serial recursion (for larger n).**
- **Demonstrates dynamic task creation in OpenMP.**

CONCLUSION

- This experiment demonstrates task-level parallelism using OpenMP. Recursive Fibonacci computation can be parallelized using tasks, allowing multiple threads to execute subproblems simultaneously. Task synchronization is essential to ensure correct results. OpenMP tasks are useful for divide-and-conquer algorithms in HPC.

EXPERIMENT – 2

- **Recursive Sum using OpenMP**

AIM

- To demonstrate parallel region behavior with recursive function execution in OpenMP.

CODE

```c
#include <stdio.h>
#include <omp.h>

int sum(int n) {
    if (n == 0) return 0;

    printf("Thread %d -> sum(%d)\n", omp_get_thread_num(), n);

    return n + sum(n - 1);
}

int main() {
    int result;

    #pragma omp parallel
    {
        #pragma omp single
        result = sum(5);
    }

    printf("Final Sum = %d\n", result);
    return 0;
}
```

OUTPUT

```
Thread 2 -> sum(5)
Thread 2 -> sum(4)
Thread 2 -> sum(3)
Thread 2 -> sum(2)
Thread 2 -> sum(1)
Final Sum = 15
```

CODE

#include <stdio.h>

#include <omp.h>

int sum(int n) {

if (n == 0) return 0;

printf("Thread %d -> sum(%d)\n", omp_get_thread_num(), n);

return n + sum(n - 1);

```
}

int main() {
    int result;

    #pragma omp parallel
    {
        #pragma omp single
        result = sum(5);
    }

    printf("Final Sum = %d\n", result);
    return 0;
}
```

EXPLANATION

- #pragma omp parallel creates multiple threads.
- #pragma omp single ensures only one thread executes the recursive function.
- omp_get_thread_num() prints the thread ID.
- Since no task directive is used, recursion runs serially within one thread.
- Other threads remain idle inside the parallel region.

OBSERVATION

- Only one thread executes recursive calls.
- No parallel task creation occurs.
- Demonstrates difference between parallel region and task parallelism.
- Other threads remain unused.

CONCLUSION

This experiment shows that simply creating a parallel region does not guarantee parallel execution. Without task or work-sharing directives, recursive functions execute serially. Effective parallelization requires explicit task creation in OpenMP.

----------THANKYOU----------