DEPARTMENT OF INFORMATION TECHNOLOGY, NITK, SURATHKAL

Parallel Computing

Lab-5- 9th September 2024

Note: Execute the following programs and take screen shots of the output. Write the analysis of the results and upload the code, results and analysis as a single file in the Moodle.

Total marks: 2+2+6 = 10

1. To understand the concept of task and taskwait. Execute the following code and write the results and observation. **[2]**

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

// Recursive Fibonacci function using OpenMP tasks

int fibonacci(int n)
{
   if (n <= 1) return n;

   int x, y;

   #pragma omp parallel
   {
      #pragma omp single
      {
         #pragma omp task shared(x)
         {
            x = fibonacci(n - 1);
         }

         #pragma omp task shared(y)
```

```
        {
            y = fibonacci(n - 2);
        }

        // Ensure all tasks are completed before combining results
        #pragma omp taskwait
      }
   }

   return x + y;
}

int main()
{
   int n = 15;

   // Set the number of threads to use
   omp_set_num_threads(2);

   // Measure the time taken to compute Fibonacci
   double start = omp_get_wtime();
   int result = fibonacci(n);
   double end = omp_get_wtime();

   printf("Fibonacci of %d is %d\n", n, result);
   printf("Time taken: %f seconds\n", end - start);
   return 0;
}
```

2. Execute the following OpenMP program to implement a linked list traversal in parallel using task and taskwait. Write the results and observation. **[2]**

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
// Define a node in the linked list
typedef struct Node {
    int data;
    struct Node* next;
} Node;
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
// Function to traverse the linked list using OpenMP tasks
void traverseLinkedList(Node* head) {
    if (head == NULL) return;

    #pragma omp parallel
    {
        #pragma omp single
        {
            // Start traversing from the head
            traverseNode(head);
```

```
        }
    }
}
void traverseNode(Node* node) {
    if (node == NULL) return;
    // Print current node's data
    printf("%d -> ", node->data);
    #pragma omp task
    {
        traverseNode(node->next);
    }
    #pragma omp taskwait
}
int main() {
                        // Create this linked list: 10 -> 20 -> 30 -> 40 -> 50 ->
NULL
    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
    head->next->next->next = createNode(40);
    head->next->next->next->next = createNode(50);
    // Print the linked list
    printf("Linked list traversal: ");
    omp_set_num_threads(2);
    // Traverse the linked list
```

```
    traverseLinkedList(head);
    // Finish the output with NULL
    printf("NULL\n");
    // Free the allocated memory
    Node* current = head;
    Node* nextNode;
    while (current != NULL) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
    return 0;
}
```

3. Write a sequential program to add elements of two arrays (C[i]=A[i] +B[i]). Convert the same program for parallel execution. Initialize arrays with random numbers. Consider the array size as 10k, 50k, and 100k. Analyze the results for maximum number of threads and various ***schedule ()*** functions. Based on the observations, carry out the analysis of the total execution time and explain the results (i.e. writing your observations) by plotting the graph.

[6]

[Note 1: Increase the array size until parallel execution time is less than the that of sequential execution].

[Note 2: Plotting the graph is mandatory here. You can use any software for plotting].

[Note 3: Record the results as per the format given below].

Schedule	**Total**	**Total**	**Total**	**Total**

()	Execution time for number of iterations 5k	Execution time for number of iterations 10k	Execution time for number of iterations 50k	Execution time for number of iterations 100k
Sequential execution				
static				
Static, chunksize				
Dynamic, chunksize				
Guided				
Runtime				

****

1. write an OpenMP program to demonstrate

sharing of section work by

2. performing arithmetic operations on one dimensional array by thread