Assignment No.: 02

Title: Implement Concurrent Quick Sort using C++.

Aim: Write a program to implement Concurrent Quick Sort in C++ using Divide and Conquer Strategy.

Objectives:

- To learn the concept of Divide and Conquer Strategy.
- To study the design and implementation of Quick Sort algorithm.

Theory:

Divide and Conquer strategy:

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub- problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Quick sort:

The sorting algorithm in vented by Ho is, usually known as "quick sort "is also based on the idea of divide-and-conquer. To make it concurrent/parallel, we will use OpenMP API. OpenMP is available along with g++ compiler. In Quick sort, input is any sequence of numbers and output is sorted array (here we will consider ascending order).We start with one number, mostly the first number, and finds its position in sorted array. This number is called pivot element. Then we divide the array into two parts considering this pivot elements position in sorted array. In each part, separately, we find the pivot elements. This process continues until all numbers are processed and we get the sorted array. In concurrent Quick sort, each part is processed by independent thread i.e. different threads will find out pivot element in each part recursively. Check in following diagram. Here PE stands for Pivot Element.

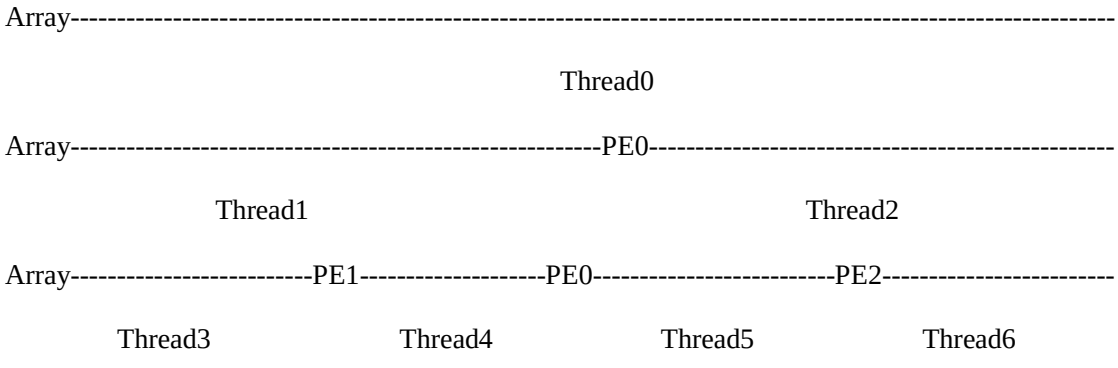


Fig 1 – Concurrently executing Threads

```
void quicksort(int* A,intq,int r)
{
        Int x, s, i;
        if(q<r)
        {       x=A[q];
                s=q;
        for ( i=q+1 ; i<r ; i++)
        {       if (A[i]<=x)
                {       s=s+1;
                        Swap (A[s], A[i]);
                }
        }
}
voidh_quicksort (int* A, int q, int r)
{       int x, s, i ;
        if(q<r)
        {       x=A[q];
                s=q;
                for (i=q+1; i<r; i++)
                {
                        if (A[i]<=x)
                        {
                                s=s+1; swap(A[s],A[i]);
                        }
                }
        swap (A[q], A[s]);
        #pragma ompparallelsections
        {
        #pragma ompsection
        h_quicksort (A, q, s);
        #pragma ompsection
        h_quicksort (A, s+1, r);
        }
}
```

Sections are performed in parallel.

Without omp_set_nested(1) no threds will be created recursively

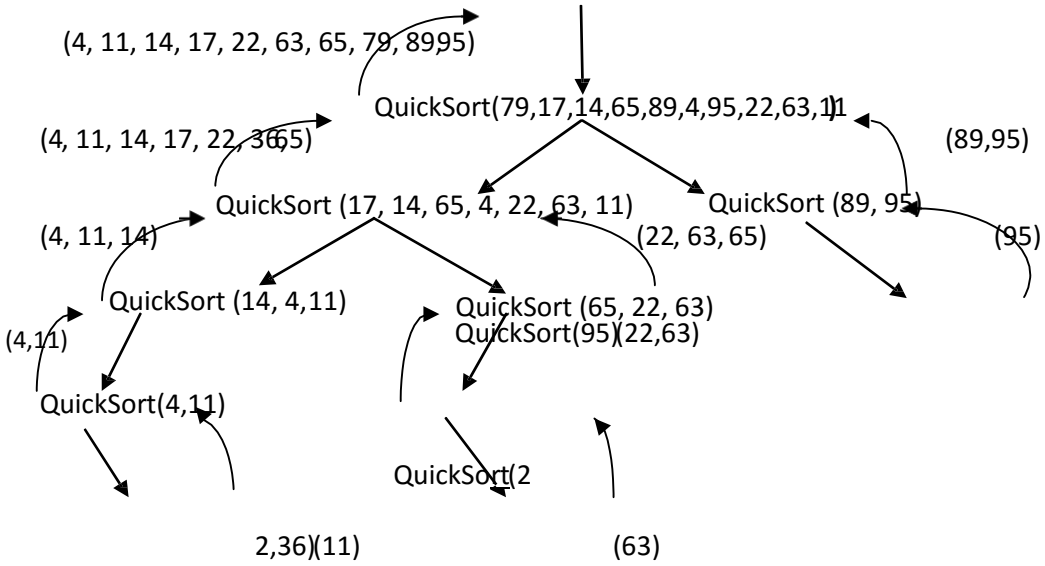
Example:

Given a list of numbers :{ 79, 17, 14, 65, 89, 4, 95, 22, 63,11}
The first number 79 is chosen as pivot

- Low list contains {17, 14, 65, 4, 22, 63, 11}
- High List contains {89,95}

- For sub list {17, 14, 65, 4, 22, 63, 11} choose 17 as pivot
- Low list contains {14, 4,11}
- High List contains {64, 22,63}

{4, 11, 14, 17, 22, 63, 65} is the sorted result of sublist
{17, 14, 65, 4, 22, 63,11}
For sub list {89, 95} choose 89 as pivot
Low list is empty (no need of further recursions)
High list contains {95} (no need of further recursions)
{89, 95} is the sorted
Final sorted list {4, 11, 14, 17, 22, 63, 65, 79, 89, 95}



Mathematical Model :

- Quick Sort is used for sorting of given data.
- Following parameters are used for QuickSort:
- M= {s, e, i, o, F, Success, Failure}
- s = Start state = inserting numbers to array to be sorted.
- e = End state = Sorted list displayed.
- i is the set of input element.
- o is the set of required output.
- F is the set of functions required for QuickSort. = {f1, f2, f3}

- f1 = {Select pivote elemant}
- f2 = {Spliting of array}
- f3 = {Merging of arrays}
- i= {a1, a2, a3,……., an}. = Array of elements to be sorted.
- o={Sorted list of elements by Quick sort}
- Success – Sorted list of elements by Quich sort.
- Failure-ϕ

Output:

Sorted array (in ascending order).

Conclusion:

The concept of divide and conquer strategy is studied and Concurrent Quick Sort algorithm is implemented using C++.

Code

```
#include<iostream>
#include<omp.h>
using namespace std;
int k=0;
class sort
{
        int a[20];
        int n;
public:
        void getdata();
        void Quicksort();
        void Quicksort(int low, int high);
        int partition(int low, int high);
        void putdata();
};
void sort::getdata()
{
        cout<<"Enter the no. of elements in array\t";
        cin>>n;
        cout<<"Enter the elements of array:"<<endl;
        for(int i=0;i<n;i++)
        {
        cin>>a[i];
        }
}
void sort::Quicksort()
{
Quicksort(0,n-1);
}

void sort::Quicksort(int low, int high)
{
if(low<high)
{
        int partn;
        partn=partition(low,high);

cout<<"\n\nThread Number: "<<k<<"  pivot element selected : "<<a[partn];
        #pragma omp parallel sections
        {
        #pragma omp section
                {
                k=k+1;
                Quicksort(low, partn-1);
                }
        #pragma omp section
                {
                k=k+1;
                Quicksort(partn+1, high);
                }
        }//pragma_omp Parallel_end
}

}
int sort::partition(int low ,int high)
{
        int pvt;
        pvt=a[high];
        int i;
        i=low-1;
        int j;
        for(j=low;j<high;j++)
        {
                if(a[j]<=pvt)
```

```
                {
                        int tem=0;
                        tem=a[j];
                        a[j]=a[i+1];
                        a[i+1]=tem;
                        i=i+1;
                }
        }
        int te;
        te=a[high];
        a[high]=a[i+1];
        a[i+1]=te;
        return i+1;
}
void sort::putdata()
{
        cout<<endl<<"\nThe Array is:"<<endl;
        for(int i=0;i<n;i++)
        cout<<" "<<a[i];
}
int main()
{
        int n;
        sort s1;
        int ch;
do
{
                s1.getdata();
                s1.putdata();

                cout<<"\nUsing Quick Sort";
   double start = omp_get_wtime();
                s1.Quicksort();
   double end = omp_get_wtime();
                cout<<"\nThe Sorted  ";
                s1.putdata();
   cout<<"\nExcecution time : "<<end - start<<" seconds ";

cout<<"Would you like to continue? (1/0 y/n)"<<endl;
cin>>ch;
}while(ch==1);
}
```

Machinefile

```
ub0:20
ub1:20
```

Output

```
mpiu@DB21:/mirror/mpiu/CL4 prog/A-2$ g++ qsort.cpp -fopenmp
mpiu@DB21:/mirror/mpiu/CL4 prog/A-2$ ./a.out
Enter the no. of elements in array  5
Enter the elements of array:
7 9 0 -3 2


The Array is:
 7 9 0 -3 2
Using Quick Sort

Thread Number: 0  pivot element selected : 2

Thread Number: 1  pivot element selected : -3

Thread Number: 2  pivot element selected : 7
The Sorted

The Array is:
 -3 0 2 7 9
Excecution time : 0.00439173 seconds Would you like to continue? (1/0 y/n)
0
```