**Lab Assignment 11**

**Title:** Vtune\_Quick\_Sort

**Question:**

Develop a program to analyse the parallel quick sort in VTune Profiler

**Code:**

#include<iostream>

#include<omp.h>

using std::cout;

using std::endl;

class ParallelQuickSort{

//keep count of threads

int k = 0;

private:

//partitioning procedure

int partition(int arr[], int l, int r){

int i = l + 1;

int j = r;

int key = arr[l];

int temp;

while(true){

while(i < r && key >= arr[i])

i++;

while(key < arr[j])

j--;

if(i < j){

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}else{

temp = arr[l];

arr[l] = arr[j];

arr[j] = temp;

return j;

}

}

}

public:

void quickSort(int arr[], int l, int r){

if(l < r){

int p = partition(arr, l, r);

cout << "pivot " << p << " found by thread no. " << k << endl;

#pragma omp parallel sections

{

#pragma omp section

{

k = k + 1;

quickSort(arr, l, p-1);

}

#pragma omp section

{

k = k + 1;

quickSort(arr, p+1, r);

}

}

}

}

//prints array

void printArr(int arr[], int n){

for(int i = 0; i < n; i++)

cout << arr[i] << " ";

cout << endl;

}

//run the whole procedure

void run(){

int arr[] = {9, 6, 3, 7, 2, 12, 5, 1};

int n = sizeof(arr) / sizeof(arr[0]);

quickSort(arr, 0, n-1);

printArr(arr, n);

}

};

int main(){

ParallelQuickSort pqs;

pqs.run();

return 0;

}