NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY
IT 301 Parallel Computing LAB 5

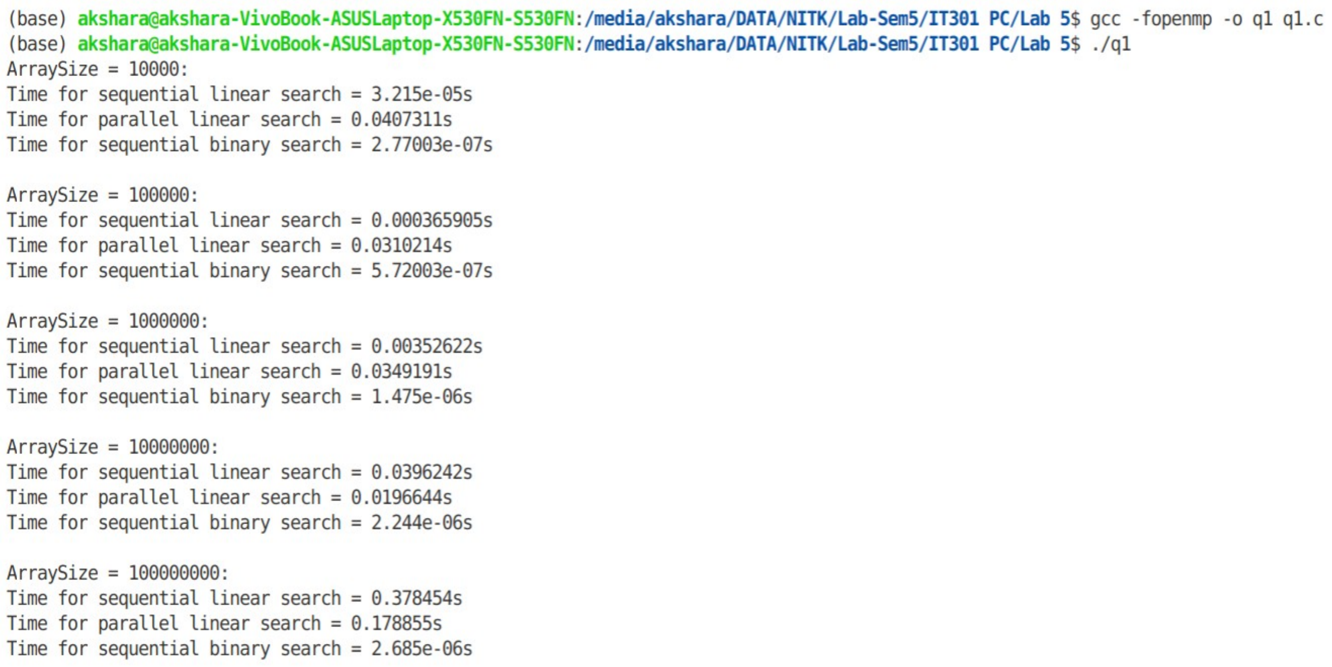
P Akshara - 181IT132

1. Develop a parallel program to find a given element in an unsorted array (a large number of elements starting from 10K can range to 1 lakh and above, based on the memory) using Linear Search. Compare the execution time with the Sequential Linear Search program. Also compare it with the sequential Binary Search program.

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

int size[] = {10000, 100000, 1000000, 10000000, 100000000};

void seq_linear_search(int arr[], int key, int s)
{
double start, end;
int loc = - 1 ;
start = omp_get_wtime();
for(int i = 0; i < size[s]; i ++)
{
if(arr[i] == key)
{
loc = i ;
break;
}
}
end = omp_get_wtime();
printf("Time for sequential linear search = %gs \n",end - start);
}

void par_linear_search(int arr[] , int key, int s)
{
double start, end;
int loc;
start = end = 0 ;
loc = - 1 ;
start = omp_get_wtime();
#pragma omp parallel for
for(int i=0;i<size[s];i++)
{
if(arr[i]==key) loc = i ;
}
end = omp_get_wtime();
printf("Time for parallel linear search = %gs \n",end - start);
}

void seq_binary_search(int arr[],int key,int s)
{
double start, end;
int loc = -1;
int low = 0, high = size[s] - 1;
int mid;
start = omp_get_wtime();
while(low <= high)
{
mid = low + (high - low) / 2 ;
```

```c
if(arr[mid] == key)
{
loc = mid;
break;
}
else if(arr[ mid] > key) high = mid - 1;
else low = mid + 1 ;
}
end = omp_get_wtime();
printf("Time for sequential binary search = %gs \n",end - start);
}

int comp(const void*a,const void*b)
{
return(*(int*)a-*(int*)b);
}

int main()
{
int *arr;
for(int s=0;s<5;s++)
{
srand(time(0));
arr = (int*) calloc( size[s], sizeof(int));
for(int i = 0 ;i < size[s];i++) arr[i] = rand();
int key = arr[size[s]-1];
printf("ArraySize = %d:\n", size[s]);
seq_linear_search(arr,key,s);
par_linear_search(arr,key,s);
qsort(arr,size[s],sizeof(int),comp);
seq_binary_search(arr,key,s);
free(arr);
printf("\n");
}
}
```

Output



```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 5$ gcc -fopenmp -o q1 q1.c
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 5$ ./q1
ArraySize = 10000:
Time for sequential linear search = 3.215e-05s
Time for parallel linear search = 0.0407311s
Time for sequential binary search = 2.77003e-07s

ArraySize = 100000:
Time for sequential linear search = 0.000365905s
Time for parallel linear search = 0.0310214s
Time for sequential binary search = 5.72003e-07s

ArraySize = 1000000:
Time for sequential linear search = 0.00352622s
Time for parallel linear search = 0.0349191s
Time for sequential binary search = 1.475e-06s

ArraySize = 10000000:
Time for sequential linear search = 0.0396242s
Time for parallel linear search = 0.0196644s
Time for sequential binary search = 2.244e-06s

ArraySize = 100000000:
Time for sequential linear search = 0.378454s
Time for parallel linear search = 0.178855s
Time for sequential binary search = 2.685e-06s
```

Analysis

Array sizes from 10^4 to 10^8 was executed.
Upto sizes of the order 10^6, sequential linear search performs better than the parallel version.
Beyond 10^6, there is a significant improvement in the performance of the parallel version over the sequential one as thread fork, join overheads are covered up by the lesser number of computations per thread.
As the asymptotic run time of binary search is O(log n) it runs much faster than both sequential and parallelized version of linear search for all sizes, although the overhead of sorting the array is O(n log n) using quick sort!

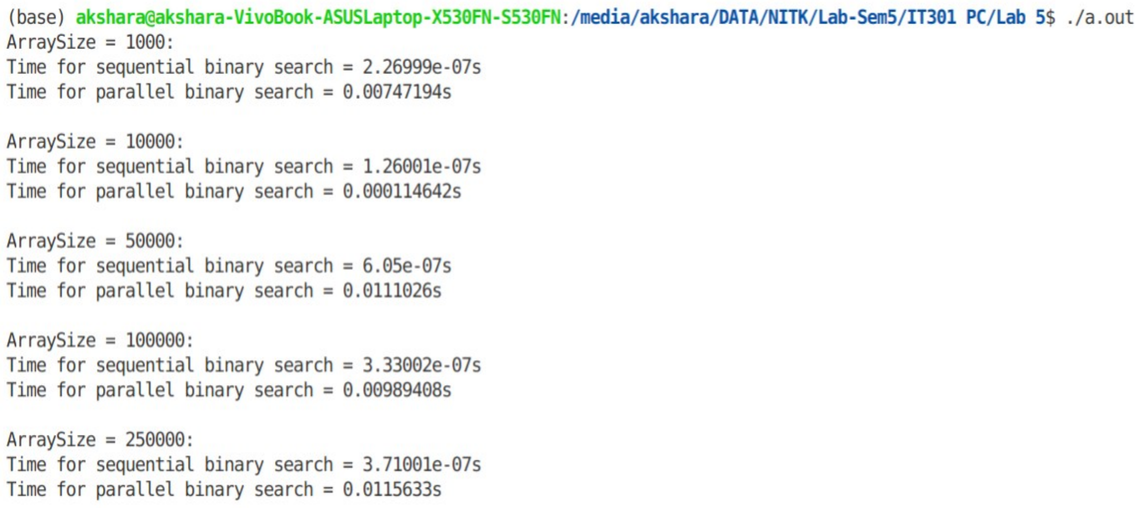
2. Develop a parallel program to find a given element in an unsorted array using Binary Search. Take a large number of elements upto the maximum possible size. Note: Make use of openmp task directive. Also compare the result with the sequential version of Binary Search.

```c
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
int numThreads;
int size[] = {1000, 10000, 50000, 100000, 250000};

int u_bound(int a, int b)
{
if (a % b == 0)
return a / b;
return (a / b) + 1;
}

int abs(int a)
{
return a >= 0 ? a : -a;
}

void swap(long int *a, long int *b)
{
long int t = *a;
*a = *b;
*b = t;
}

int partition(long int arr[], int low, int high)
{
long int pivot = arr[high];
int i = (low - 1);
for (int j = low; j <= high - 1; j++)
{
if (arr[j] <= pivot)
{
i++;
swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);
}

void quickSort(long int arr[], int low, int high, int minSize)
```

```
{
if (low < high)
{
int pi = partition(arr, low, high);
if (high - low > minSize)
{
#pragma omp task firstprivate(arr, low, pi)
{
quickSort(arr, low, pi - 1, minSize);
}
#pragma omp task firstprivate(arr, high, pi)
{
quickSort(arr, pi + 1, high, minSize);
}
}
else
{
quickSort(arr, low, pi - 1, minSize);
quickSort(arr, pi + 1, high, minSize);
}
}
}
void seq_binary_search(long int arr[], long int key, int s)
{
double start, end;
int loc = -1;
int low = 0, high = size[s] - 1;
int mid;
start = omp_get_wtime();
while (low <= high)
{
mid = low + (high - low) / 2 ;
if(arr[mid] == key)
{
loc = mid;
break;
}
else if(arr[ mid] > key) high = mid - 1;
else low = mid + 1 ;
}
end = omp_get_wtime();
printf("Time for sequential binary search = %gs \n",end - start);
}
void par_bin_search_util(long int arr[], int key, int left, int right, int *loc)
{
if (left > right)
{
return;
}
if (right - left <= numThreads)
{
int low = left, high = right, mid;
while (low <= high)
{
mid = low + (high - low) / 2 ;
if(arr[mid] == key)
{
*loc = mid;
break;
}
else if(arr[ mid] > key) high = mid - 1;
else low = mid + 1 ;
}
```

```c
}
else
{
int nums = u_bound(right - left + 1, numThreads);
#pragma omp parallel for schedule(static)
for (int i = 0; i < numThreads; i++)
{
if (arr[left + i * nums] <= key && arr[left + (i + 1) * nums] >= key)
{
#pragma omp task if (0)
{
par_bin_search_util( arr , key , left + i * nums , left + ( i + 1 )* nums , loc);
}
}
}
}
}
void par_binary_search(long int arr[], int key, int s)
{
double start, end;
int loc = -1;
start = omp_get_wtime();
par_bin_search_util(arr, key, 0, size[s] - 1, &loc);
end = omp_get_wtime();
printf("Time for parallel binary search = %gs \n",end - start);
}
void qsort_par(long int *array, int lenArray, int numThreads)
{
int minSize = 500;
#pragma omp parallel num_threads(numThreads)
{
#pragma omp single nowait
{
quickSort(array, 0, lenArray - 1, minSize);
}
}
}
int comp(const void *a, const void *b)
{
return (*(long int *)a - *(long int *)b);
}
int main()
{
long int *arr;
numThreads = omp_get_max_threads();
for (int s = 0; s < 8; s++)
{
srand(time(0));
arr=(long*) calloc(size[s],sizeof(long));
long key = arr[size[s]-1];
printf("ArraySize = %d:\n", size[s]);
omp_set_num_threads(omp_get_max_threads());
qsort_par(arr,size[s],omp_get_max_threads());
seq_binary_search(arr,key,s);
par_binary_search(arr,key,s);
free(arr);
printf("\n");
}
}
```

Output



```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 5$ ./a.out
ArraySize = 1000:
Time for sequential binary search = 2.26999e-07s
Time for parallel binary search = 0.00747194s

ArraySize = 10000:
Time for sequential binary search = 1.26001e-07s
Time for parallel binary search = 0.000114642s

ArraySize = 50000:
Time for sequential binary search = 6.05e-07s
Time for parallel binary search = 0.0111026s

ArraySize = 100000:
Time for sequential binary search = 3.33002e-07s
Time for parallel binary search = 0.00989408s

ArraySize = 250000:
Time for sequential binary search = 3.71001e-07s
Time for parallel binary search = 0.0115633s
```

Analysis

Array sizes from 10^3 to 10^6 was executed.
The sequential version performs better than the parallel version. Beyond 10^6, sorting overhead is very high to test. Possibly the thread tasking operations take more time than the O(log n) sequential algorithm itself for these values of n, due to which sequential is faster.