NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL

DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 5

9th September 2020

Faculty: Dr. Geetha V and Mrs. Thanmayee

---

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>
int cmpfunc (const void * a, const void * b) {
return ( *(int*)a - *(int*)b );
}
int main()
{
int size=(rand()%10000)+10000;
int value=10;
int flag;
int tid;
int A[size];for(int i=0;i<size;i++)
{
A[i]=rand()%10000;
}
double t1,t2;
printf("Total number of element: %d\n\n",size);
printf("\n1.Sequential Linear search:\n");
```

Code for sequential Linear search:

```
printf("\n1.Sequential Linear search:\n");
t1=omp_get_wtime();
flag=0;
for(int i=0;i<size;i++)
{
if(A[i]==value)
{
flag=1;
printf("Element found at pos %d\n",i+1);
}
}
if(flag==0)
{
printf("Element Not found\n");
}
t2=omp_get_wtime();
printf("Time taken %f s\n\n",t2-t1);
```

Code for Parallel Linear search:

```
printf("\n2.Parallel Linear search:\n");
t1=omp_get_wtime();
int i;
#pragma omp parallel num_threads(4) private(i)
{
tid=omp_get_thread_num();
#pragma omp for
for(i=0;i<size;i++)
{
if(A[i]==value)
{
flag=1;
printf("Element found at pos %d by thread %d\n",i+1,tid);
}
}
}
t2=omp_get_wtime();printf("Time taken %f s\n\n",t2-t1);
```

Code for Sequential Binary search:

```
53  qsort(A, size, sizeof(int), cmpfunc);
54  printf("Here we have sorted the given array to perform binary search.\n");
55  printf("\n3.Sequential Binary search:\n");
56  t1=omp_get_wtime();
57  int beg=0,last=size-1,mid;
58  while(beg<=last)
59  {
60  mid=beg+(last-beg)/2;
61  if(A[mid]==value)
62  {
63  flag=1;
64  printf("Element found at pos %d\n",mid+1);
65  break;
66  }
67  else if(A[mid]>value)
68  {
69  last=mid-1;
70  }
71  else
72  {
73  beg=mid+1;
74  }
75  }
76  if(flag==0)
77  {
78  printf("Element not found\n");
79  }
80  t2=omp_get_wtime();
81  printf("Time taken %f s\n\n",t2-t1);
82  return 0;
83  }
84
```

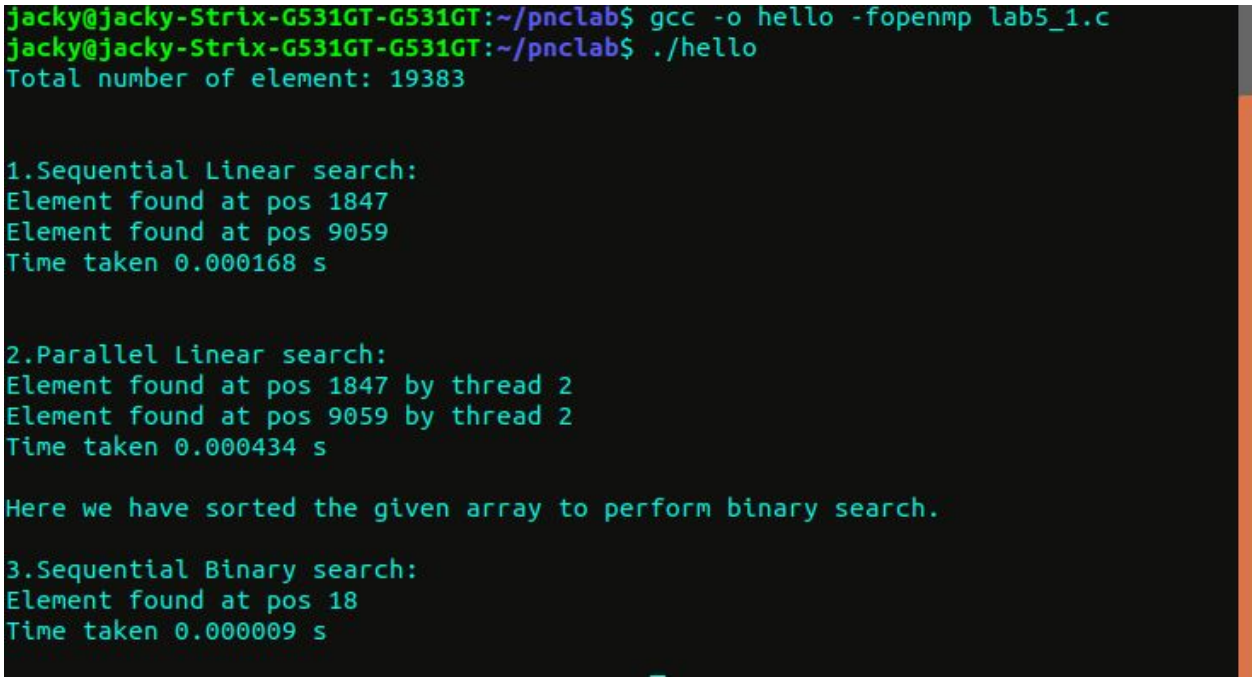
FULL Code:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
int cmpfunc (const void * a, const void * b) {
return ( *(int*)a - *(int*)b );
}
int main()
{
int size=(rand()%10000)+10000;
int value=10;
int flag;
int tid;
int A[size];for(int i=0;i<size;i++)
{
A[i]=rand()%10000;
}
double t1,t2;
printf("Total number of element: %d\n\n",size);
printf("\n1.Sequential Linear search:\n");
```

```
t1=omp_get_wtime();
flag=0;
for(int i=0;i<size;i++)
{
if(A[i]==value)
{
flag=1;
printf("Element found at pos %d\n",i+1);
}
}
if(flag==0)
{
printf("Element Not found\n");
}
t2=omp_get_wtime();
printf("Time taken %f s\n\n",t2-t1);
printf("\n2.Parallel Linear search:\n");
t1=omp_get_wtime();
int i;
#pragma omp parallel num_threads(4) private(i)
{
tid=omp_get_thread_num();
#pragma omp for
for(i=0;i<size;i++)
{
if(A[i]==value)
{
flag=1;
printf("Element found at pos %d by thread %d\n",i+1,tid);
}
}
}
t2=omp_get_wtime();printf("Time taken %f s\n\n",t2-t1);
qsort(A, size, sizeof(int), cmpfunc);
printf("Here we have sorted the given array to perform binary search.\n");
printf("\n3.Sequential Binary search:\n");
t1=omp_get_wtime();
int beg=0,last=size-1,mid;
while(beg<=last)
{
mid=beg+(last-beg)/2;
if(A[mid]==value)
{
flag=1;
```

```
printf("Element found at pos %d\n",mid+1);
break;
}
else if(A[mid]>value)
{
last=mid-1;
}
else
{
beg=mid+1;
}
}
if(flag==0)
{
printf("Element not found\n");
}
t2=omp_get_wtime();
printf("Time taken %f s\n\n",t2-t1);
return 0;
}
```

OUTPUT:



Here we can see that the Time for sequential linear search is more than binary search as binary search (O(log n)) is faster than linear search(O(n)).Time for parallel linear search becomes more than sequential linear search due to small size of array. When the size is increased then the time parallel linear search will be less than sequential linear search but will be less than binary search.

```
jacky@jacky-Strix-G531GT-G531GT:~/pnclab$ gcc -o hello -fopenmp lab5_1.c
jacky@jacky-Strix-G531GT-G531GT:~/pnclab$ ./hello
Total number of element: 109382


1.Sequential Linear search:
Element found at pos 1848
Element found at pos 9060
Element found at pos 37794
Element found at pos 48233
Element found at pos 79710
Element found at pos 79904
Element found at pos 93292
Element found at pos 107594
Element found at pos 109314
Time taken 0.000317 s


2.Parallel Linear search:
Element found at pos 1848 by thread 3
Element found at pos 37794 by thread 3
Element found at pos 9060 by thread 3
Element found at pos 93292 by thread 3
Element found at pos 48233 by thread 3
Element found at pos 79710 by thread 3
Element found at pos 79904 by thread 3
Element found at pos 107594 by thread 3
Element found at pos 109314 by thread 3
Time taken 0.000125 s
```

Here the size of the array is more than 10 lakh so we can see the parallel execution take less time than sequential execution.

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 the openmp task directive. Also compare the result with the sequential version of Binary Search.

CODE :

Note : First we need to sort the array to do binary search.

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
int key=10;
int cmpfunc (const void * a, const void * b) {
return ( *(int*)a - *(int*)b );
}
int binary_search(int arr[],int l,int r){
if(l>=r)return -1;
int mid=(l+r)/2;
if(arr[mid]==key)return mid;
int a = -1,b = -1;
if((r-l)>10000){
#pragma omp task shared(a)
{
a=binary_search(arr,mid+1,r);
}
#pragma omp task shared(b)
{
b=binary_search(arr,l,mid);
}
#pragma omp taskwait
return a>b?a:b;
}
else
{
a=binary_search(arr,mid+1,r);
b=binary_search(arr,l,mid);return a>b?a:b;
}
}
int main()
{
int size=(rand()%10000)+10000;
int value=key;
```

```
int flag,index;
int tid;
int A[size];
for(int i=0;i<size;i++)
{
A[i]=rand()%10000;
}
double t1,t2;
printf("Total number of element: %d\n\n",size);
qsort(A, size, sizeof(int), cmpfunc);
printf("\n1.Sequential Binary search:\n");
t1=omp_get_wtime();
int beg=0,last=size-1,mid;
while(beg<=last)
{
mid=beg+(last-beg)/2;
if(A[mid]==value)
{
flag=1;
printf("Element found at pos %d\n",mid+1);
break;
}
else if(A[mid]>value)
{
last=mid-1;
}
else
{
beg=mid+1;
}}
if(flag==0)
{
printf("Element not found\n");
}
t2=omp_get_wtime();
printf("Time taken %f s\n\n",t2-t1);
printf("\n2.parallel Binary search:\n");
t1=omp_get_wtime();
#pragma omp parallel
{
#pragma omp master
{
index=binary_search(A,0,size-1);
}
```

```
}
printf("Element found at pos %d\n",mid+1);
t2=omp_get_wtime();
printf("Time taken %f s\n\n",t2-t1);
return 0;
}
```

OUTPUT:

```
jacky@jacky-Strix-G531GT-G531GT:~/pnclab$ subl lab5_2.c
jacky@jacky-Strix-G531GT-G531GT:~/pnclab$ gcc -o hello -fopenmp lab5_2.c
jacky@jacky-Strix-G531GT-G531GT:~/pnclab$ ./hello
Total number of element: 19383


1.Sequential Binary search:
Element found at pos 18
Time taken 0.000002 s


2.parallel Binary search:
Element found at pos 18
Time taken 0.023785 s
```

Here we can see that the parallel execution takes more time than sequential execution.As binary search take (log n) time so for a very large value of N the number of iteration will be around order of log(N) which is very small and can be easily done by sequential execution.
But if we use task for the same program because of the thread creation, assignment, scheduling and management overhead will drastically increase for parallel program and with task overhead is more because of the scheduling and other external factor so parallel program with task will perform worse than sequential fo the size of the array that we can construct with c compiler.

Name: **Bhajan Kumar Barman**
Registration no.: **181IT211**