NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 5

09th September 2020

Submitted By: Harsh Agarwal (181IT117)

---

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.

Code:

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

// Function for sequential linear search
int linear_seq(int arr[], int n, int ele){
   for (int a = 0; a < n; a++){
      if (arr[a] == ele){
         return a;
      }
   }
   return -1;
}

// Function for parallel linear search
int linear_par(int arr[], int n, int ele){
   int ret = -1;
   #pragma omp parallel num_threads(4)
   {
      int tid = omp_get_thread_num();
      #pragma omp for // Using the for directive to iterate the array parallely using 4 threads
      for (int a = 0; a < n; a++){
         if (arr[a] == ele){
            #pragma omp critical // Using critical directive to assign the value if we find the element
            {
               ret = a;
            }
         }
      }
   }
```

```
   return ret;
}

// Merge function to sort the array using Merge Sort. This is required in case of binary search
void merge(int arr[], int l, int m, int r){
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
   int L[n1], R[n2];
   for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];
   i = 0;
   j = 0;
   k = l;
   while (i < n1 && j < n2) {
      if (L[i] <= R[j]) {
         arr[k] = L[i];
         i++;
      }
      else {
         arr[k] = R[j];
         j++;
      }
      k++;
   }
   while (i < n1) {
      arr[k] = L[i];
      i++;k++;
   }
   while (j < n2) {
      arr[k] = R[j];
      j++;
      k++;
   }
}

// Recursive Merge Sort function to sort the array in O(nlogn)
void mergeSort(int arr[], int l, int r){
   if (l < r) {
   int m = l + (r - l) / 2;
   mergeSort(arr, l, m);
   mergeSort(arr, m + 1, r);
   merge(arr, l, m, r);
   }
}
```

```
// Function to perform sequential binary search on the sorted array
int binary_seq(int arr[], int n, int ele){
   int l = 0, r = n-1, mid;
   while (l < r){
      mid = l + (r-l)/2;
      if (arr[mid] == ele){
         return mid;
      }
      if (arr[mid] > ele){
         r = mid-1;
      }
      else{
         l = mid+1;
      }
   }
   if (arr[l] == ele){
      return l;
   }
   return -1;
}

int main(){
   srand(time(0));
   long long int n;
   printf("Enter the number of elements: ");
   scanf("%lld", &n);
   int arr[n];
   for (long long int a = 0; a < n; a++){
      arr[a] = rand()%n;
   }
   int ele, ret;

   printf("\nEnter the number to be searched: ");
   scanf("%d", &ele);
   struct timeval TimeValue_Start;
   struct timezone TimeZone_Start;
   struct timeval TimeValue_Final;
   struct timezone TimeZone_Final;
   long time_start, time_end;
   double time_overhead;
   gettimeofday(&TimeValue_Start, &TimeZone_Start);
   ret = linear_seq(arr, n, ele);
   gettimeofday(&TimeValue_Final, &TimeZone_Final);
   time_start = TimeValue_Start.tv_sec * 1000000 + TimeValue_Start.tv_usec;
   time_end = TimeValue_Final.tv_sec * 1000000 + TimeValue_Final.tv_usec;
   time_overhead = (time_end - time_start)/1000000.0;
   printf("TIME TAKEN FOR SEQUENTIAL LINEAR SEARCH: %lf\n",time_overhead);
   if (ret == -1){
```

```
    printf("\tElement not found\n");
  }
  else{
    printf("\tElement found at: %d\n", ret);
  }
  gettimeofday(&TimeValue_Start, &TimeZone_Start);
  ret = linear_par(arr, n, ele);
  gettimeofday(&TimeValue_Final, &TimeZone_Final);
  time_start = TimeValue_Start.tv_sec * 1000000 + TimeValue_Start.tv_usec;
  time_end = TimeValue_Final.tv_sec * 1000000 + TimeValue_Final.tv_usec;
  time_overhead = (time_end - time_start)/1000000.0;
  printf("TIME TAKEN FOR PARALLEL LINEAR SEARCH: %lf\n",time_overhead);
  if (ret == -1){
    printf("\tElement not found\n");
  }
  else{
    printf("\tElement found at: %d\n", ret);
  }
  mergeSort(arr, 0, n-1);
  gettimeofday(&TimeValue_Start, &TimeZone_Start);
  ret = binary_seq(arr, n, ele);
  gettimeofday(&TimeValue_Final, &TimeZone_Final);
  time_start = TimeValue_Start.tv_sec * 1000000 + TimeValue_Start.tv_usec;
  time_end = TimeValue_Final.tv_sec * 1000000 + TimeValue_Final.tv_usec;
  time_overhead = (time_end - time_start)/1000000.0;
  printf("TIME TAKEN FOR SEQUENTIAL BINARY SEARCH: %lf\n",time_overhead);
  if (ret == -1){
    printf("\tElement not found\n");
  }
  else{
    printf("\tElement found at: %d\n", ret);
  }
}
```

Analysis:

As the number of elements in our array can be as large as 10^6 (10 lakh), it is not feasible to take the array elements as input from user. This program takes the number of elements (n) as input and then populates the array randomly with values between 0 and n. This is done using random function.

Now the user gives the input of the number to be searched. That number is searched using all the three different methods:

1. Sequential Linear Search
2. Parallel Linear Search (Achieved using parallel for directive and critical directive)
3. Sequential Binary Search (First the array is sorted using merge sort and then sequential binary search is applied)

The time taken by all the three methods are compared.

Output:

a)

```
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab5
$ ./a.exe
Enter the number of elements: 200000

Enter the number to be searched: 200001
TIME TAKEN FOR SEQUENTIAL LINEAR SEARCH: 0.001148
        Element not found
TIME TAKEN FOR PARALLEL LINEAR SEARCH: 0.000995
        Element not found
TIME TAKEN FOR SEQUENTIAL BINARY SEARCH: 0.000000
        Element not found

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab5
$
```

As the array elements will be between 1 and 20000 (n), the given number, 20001 can't be present in the array and our algorithm searches for it in the entire array and gives the running time.

b)

```
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab5
$ ./a.exe
Enter the number of elements: 200000

Enter the number to be searched: 21
TIME TAKEN FOR SEQUENTIAL LINEAR SEARCH: 0.000000
        Element found at: 581
TIME TAKEN FOR PARALLEL LINEAR SEARCH: 0.001653
        Element found at: 98628
TIME TAKEN FOR SEQUENTIAL BINARY SEARCH: 0.000000
        Element found at: 151
```

In this case, the element 21 may be present in multiple locations in array (array elements are randomized). That is why we get multiple outputs for the position of the element. And here the algorithm will stop as soon as our number is found. Complete array is not searched.

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

Code:

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

```
#include <sys/time.h>
#include <time.h>
#include <omp.h>

// Merge function to sort the array using Merge Sort. This is required in case of binary search
void merge(int arr[], int l, int m, int r){
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
   int L[n1], R[n2];
   for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];
   i = 0;
   j = 0;
   k = l;
   while (i < n1 && j < n2) {
      if (L[i] <= R[j]) {
         arr[k] = L[i];
         i++;
      }
      else {
         arr[k] = R[j];
         j++;
      }
      k++;
   }
   while (i < n1) {
      arr[k] = L[i];
      i++;k++;
   }
   while (j < n2) {
      arr[k] = R[j];
      j++;
      k++;
   }
}

// Recursive Merge Sort function to sort the array in O(nlogn)
void mergeSort(int arr[], int l, int r){
   if (l < r) {
   int m = l + (r - l) / 2;
   mergeSort(arr, l, m);
   mergeSort(arr, m + 1, r);
   merge(arr, l, m, r);
   }
}
```

```
// Function to perform sequential binary search on the sorted array
int binary_seq(int arr[], int n, int ele){
   int l = 0, r = n-1, mid;
   while (l < r){
      mid = l + (r-l)/2;
      if (arr[mid] == ele){
         return mid;
      }
      if (arr[mid] > ele){
         r = mid-1;
      }
      else{
         l = mid+1;
      }
   }
   if (arr[l] == ele){
      return l;
   }
   return -1;
}

// Function to perform parallel binary search on the sorted array
int binary_par(int arr[], int l, int r, int ele){
   if (l > r){
      return -1;
   }
   if (l == r){
      if (arr[l] == ele){
         return l;
      }
      return -1;
   }
   int mid = l + (r-l)/2;  // Calculating the mid term of [l,r]
   if (arr[mid] == ele){ // Checking if element at arr[mid] is equal to reqd element
      return mid;
   }
   int ret = -1;
   #pragma omp parallel num_threads(4)
   {
      #pragma omp single //Using the single directive so that only 1 thread executes this section at a time
      {
         if (arr[mid] > ele){ // Recur for [l,mid-1]
            #pragma omp task // Calculate the ans for left part as a separate task
            {
               ret = binary_par(arr, l, mid-1, ele);
            }
         }
         else{    // Recur got [mid+1,r]
```

```
            #pragma omp task  // Calculate the ans for right part as a separate task
            {
               ret = binary_par(arr, mid+1, r, ele);
            }
         }
         #pragma omp taskwait //Using taskwait so that the process waits for completion of both
                              //the child tasks
      }
   }
   return ret; // Returning the position of the element in the array
}

int main(){
   srand(time(0));
   long long int n;
   printf("Enter the number of elements: ");
   scanf("%lld", &n);
   int arr[n];
   for (long long int a = 0; a < n; a++){
      arr[a] = rand()%n;
   }
   int ele, ret;

   printf("\nEnter the number to be searched: ");
   scanf("%d", &ele);
   struct timeval TimeValue_Start;
   struct timezone TimeZone_Start;
   struct timeval TimeValue_Final;
   struct timezone TimeZone_Final;
   long time_start, time_end;
   double time_overhead;

   mergeSort(arr,0,n-1);

   gettimeofday(&TimeValue_Start, &TimeZone_Start);
   ret = binary_seq(arr, n, ele);
   gettimeofday(&TimeValue_Final, &TimeZone_Final);
   time_start = TimeValue_Start.tv_sec * 1000000 + TimeValue_Start.tv_usec;
   time_end = TimeValue_Final.tv_sec * 1000000 + TimeValue_Final.tv_usec;
   time_overhead = (time_end - time_start)/1000000.0;
   printf("TIME TAKEN FOR SEQUENTIAL BINARY SEARCH: %lf\n",time_overhead);
   if (ret == -1){
      printf("\tElement not found\n");
   }
   else{
      printf("\tElement found at: %d\n", ret);
   }
```

```
    gettimeofday(&TimeValue_Start, &TimeZone_Start);
    ret = binary_par(arr, 0, n-1, ele);
    gettimeofday(&TimeValue_Final, &TimeZone_Final);
    time_start = TimeValue_Start.tv_sec * 1000000 + TimeValue_Start.tv_usec;
    time_end = TimeValue_Final.tv_sec * 1000000 + TimeValue_Final.tv_usec;
    time_overhead = (time_end - time_start)/1000000.0;
    printf("TIME TAKEN FOR PARALLEL BINARY SEARCH: %lf\n",time_overhead);
    if (ret == -1){
        printf("\tElement not found\n");
    }
    else{
        printf("\tElement found at: %d\n", ret);
    }
}
```

Analysis:

As the number of elements in our array can be as large as 10^6 (10 lakh), it is not feasible to take the array elements as input from user. This program takes the number of elements (n) as input and then populates the array randomly with values between 0 and n. This is done using random function.

Now the user gives the input of the number to be searched. That number is searched using all the two different methods:

1. Sequential Binary Search (First the array is sorted using merge sort and then sequential binary search is applied)
2. Parallel Binary Search (First the array is sorted using merge sort and then parallel binary search is applied by recusively calling the function and assigning the results to separate tasks)

The time taken by all the two methods is compared.

Output:

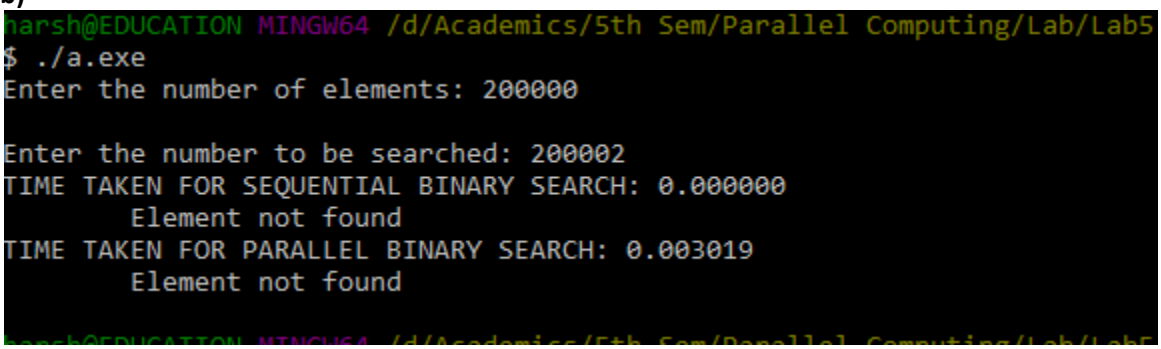
a)

```
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab5
$ ./a.exe
Enter the number of elements: 200000

Enter the number to be searched: 43
TIME TAKEN FOR SEQUENTIAL BINARY SEARCH: 0.000000
        Element found at: 272
TIME TAKEN FOR PARALLEL BINARY SEARCH: 0.001746
        Element found at: 272
```

In this case, our element is found at position 272 in the array and the position is returned by the program.

b)



As the array elements will be between 1 and 20000 (n), the given number, 20002 can't be present in the array and our program searches for it in the entire array and gives the running time of both the algorithms.

NOTE: We observe that the running time of parallel binary search is coming out to be more than sequential binary search. This is because of the overhead time required in assigning the tasks as well as setting up the parallel environment for execution. Asymptotically both the sequential and parallel binary search algorithms have the same time complexity of O(logn)