Parallel Implementation of Sorting Algorithms

Malika Dawra[1], and Priti[2]

[1]Department of Computer Science & Applications
M.D. University, Rohtak-124001, Haryana, India

[2]Department of Computer Science & Applications
M.D. University, Rohtak-124001, Haryana, India

Abstract

A sorting algorithm is an efficient algorithm, which perform an important task that puts elements of a list in a certain order or arrange a collection of elements into a particular order. Sorting problem has attracted a great deal of research because efficient sorting is important to optimize the use of other algorithms such as binary search. This paper presents a new algorithm that will split large array in sub parts and then all sub parts are processed in parallel using existing sorting algorithms and finally outcome would be merged. To process sub parts in parallel multithreading has been introduced.

Keywords: *sorting, algorithm, splitting, merging, multithreading.*

1. Introduction

Sorting is the process of putting data in order; either numerically or alphabetically. It is necessary to arrange the elements in an array in numerical or lexicographical order, sorting numerical values in descending order or ascending order and alphabetical value like addressee key [5, 6]. Many existing sorting algorithms were observed in terms of the efficiency of the algorithmic complexity [9]. All sorting algorithms are appropriate for specific kinds of problems. Some sorting algorithms work on less number of elements, some are used for huge number of data, some are used if the list has repeated values, and some are suitable for floating point numbers. Sorting is used in many important applications and there have been a plenty of performance analysis [7]

Sorting algorithms are classified by:

- Computational complexity in terms of number of swaps. Sorting methods perform various numbers of swaps in order to sort a data.
- System complexity of computational. In this case, each method of sorting algorithm has different cases of performance. They are worst case, when the integers are not in order and they have to be swapped at least once. The term best case is used to describe the way an algorithm behaves under optimal conditions.
- Usage of memory and other computer resources is also a factor in classifying the sorting algorithms.
- Recursion: some algorithms are either recursive or non recursive, while others may be both.
- Whether or not they are a comparison sort examines the data only by comparing two elements with a comparison operator.

There are several sorting algorithms available to sort the elements of an array. Some of the sorting algorithms are:

Bubble sort

In the bubble sort, as elements are sorted they gradually "bubble" (or rise) to their proper location in the array. Bubble sort was analyzed as early as 1956 [1]. The bubble sort compares adjacent elements of an array until the complete list gets sorted.

Selection Sort

The selection sort is a combination of sorting and searching. During each pass, the unsorted element with the smallest (or largest) value is moved to its proper position in the array. The selection sort has a complexity of $O(n^2)$ [3].

Insertion Sort

The insertion sort splits an array into two sub-arrays. The first sub-array is sorted and increases in size as the sort continues. The second sub-array is unsorted, contains all the elements yet to be inserted into the first sub-array, and decreases in size as the sort continues. If an application only needs to sort smaller amount of data, then it is suitable to use this algorithm [2].

Shell Sort

The shell sort repeatedly compares elements that are a certain distance away from each other. The shell sort is a "diminishing increment sort", better known as a "comb sort" [8]. The algorithm makes multiple passes through the list, and each time sorts

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

a number of equally sized sets using the insertion sort [4].

Quick Sort

The quick sort is considered to be very efficient with its "divide and conquer" algorithm. This sort starts by dividing the original array into two sections (partitions) based upon the value of the pivot (can be first element in the array). The first section will contain all the elements less than (or equal to) the pivot and the second section will contain all the elements greater than the pivot. This sort uses recursion – the process of "calling itself". Quick sort was considered to be a good sorting algorithm in terms of average theoretical complexity and cache performance [7].

Merge Sort

The merge sort combines two arrays into one larger array. The arrays to be merged must be sorted first. It follows "divide and conquer" strategy. Firstly divide the n-element sequence to be sorted into two subsequence of n/2 element each. Sort the two subsequences recursively using merge sort and then merge the two sorted subsequences to produce the sorted answer.

2. Proposed Model for Sorting Algorithms

2.1 Divide Array in sub parts

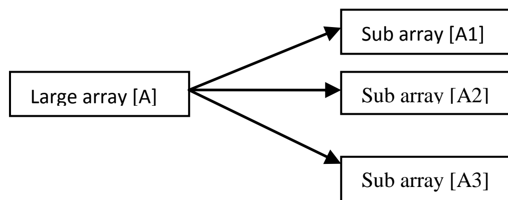

Fig.1 Split the large array into smaller parts and store them in new arrays

2.2 Use of Efficient Algorithms on sub parts

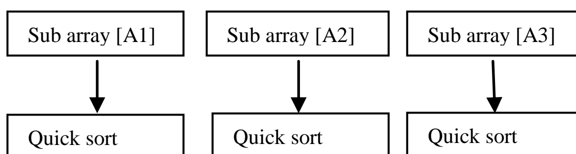

Fig.2 Use of efficient algorithm on sub parts

2.3 Algorithms are implemented in parallel

In order to implement sub arrays in parallel concept of multithreading would be used.

The multithreading paradigm has become more popular as efforts to further exploit instruction level parallelism have stalled since the late-1990s. This allowed the concept of throughout computing to re-emerge to prominence from the more specialized field of transaction processing:

- Even though it is very difficult to further speed up a single thread or single program, most computer systems are actually multi-tasking among multiple threads or programs.
- Techniques that would allow speed up of the overall system throughout of all tasks would be a meaningful performance gain.

The two major techniques for throughout computing are multiprocessing and multithreading.

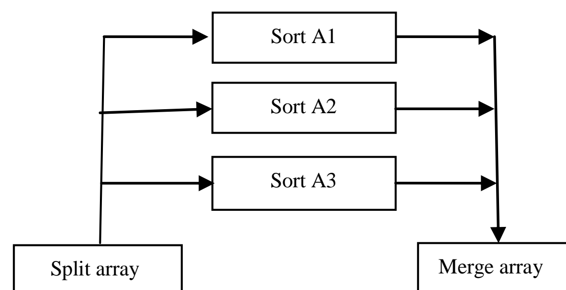

Fig.3 Parallel implementation and merging

2.4 Parallel implementation can be done using multithreading

Multithreading is similar in concept to preemptive multitasking but is implemented at the thread level of execution in modern superscalar processors. Simultaneous multithreading (SMT) is one of the two main implementations of multithreading, the other form being temporal multithreading. In temporal multithreading, only one thread of instructions can execute in any given pipeline stage at a time. In simultaneous multithreading, instructions from more than one thread can be executing in any given pipeline stage at a time. This is done without great changes to the basic processor architecture.

- The main additions needed are the ability to fetch instructions from multiple threads in a cycle.
- A larger register file to hold data from multiple threads.

IJCSI
www.IJCSI.org
Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

2.5 After sorting all sub parts are merged

After merging new array will store sorted information and would be displayed on screen.

3. Implementation

The various steps involved in implementing the above model are:

- Classify the data in separate arrays on the basis of data type and number of digit
- Use sorting techniques on them in parallel. And parallel implementation is done using multithreading.
- Merge sorted array of 1 digit, 2digit, 3 digit ,4 digit, 5 digit , 6 digit and more digit integer and create a large array
- Apply merge sort on that large array and sorted real number and create array of all numerical values.
- Merge all sorted numerical values with sorted string array.

Classifying algorithm works at the insertion time. It will check whether number is string, real or numerical, and if number is numerical then it checks whether it is 1 digit, 2 digit , 3 digit, 4 digit, 5 digit ,6 digit number or more. We have following arrays:

Strarray[] to store string,

Oned[] to store one digit no

Twod[] to store 2 digit no

Threed[]to store 3 digit no

Fourd[] to store 4 digit no

Fived[] to store 5 digit no

Sixd[] to store 6 digit no

Mored[] to store more than 6 digit no.

Realn[] to store real number.

3.1 Classifying algorithm

Step 1

Read the user string a

Step 2

If a is string then put it in strarray[] and exit

Step 3

If a is real number then put it in realn[] and exit

Step 4

i) If a is number then divide a by 1000000 and find remainder r

ii) r = a%1000000

If (r>0) then put it in Mored[];

exit

iii) r = a%100000

If (r>0) then put it in Sixd[];

exit

iv) r = a%10000

If (r>0) then put it in Fived[];

exit

v) r = a%1000

If (r>0) then put it in Fourd[];

exit

vi) r = a%100

If (r>0) then put it in Threed[];

exit

vii) r = a%10

If (r>0) then put it in Twod[];

else put it in Oned[];

exit

Step 5

Exit

Eight arrays will be sorted parallel as all sorting function will be in running as thread

3.2 Algorithm to put element in array

3.2.1 Algorithm for Insertion of an element in an array

Suppose, the array be arr[max], *pos* be the position at which the element *num* has to be inserted. For insertion, all the elements starting from the position

IJCSI
www.IJCSI.org
Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

pos are shifted towards their right to make a vacant space where the element *num* is inserted.

1. FOR I = (max-1) TO pos
2. arr[I] = arr[I-1]
3. arr[I] = num

3.2.2 Algorithm for Deletion of an element in an array

Suppose, now *pos* be the position from which the element has to be deleted. For deletion, all the elements to the right of the element at position *pos* are shifted to their left and the last vacant space is filled with 0.

1. FOR I = pos TO (max-1)
2. arr[I-1] = arr[I]
3. arr[I-1] = 0

3.3 Algorithm to sort array using Quick sort

Quicksort is a divide and conquer algorithm. Quick sort first divides a large list into two smaller sub-lists: the low elements and the high elements. Quick sort can then recursively sort the sub-lists. The steps involved are:

1. Pick an element, called a *pivot*, from the list.
2. Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

```
quick sort('array')
if length('array') ≤ 1
return 'array'  /* an array of zero or one elements is already sorted*/
select and remove a pivot value 'pivot' from 'array'
create empty lists 'less' and 'greater'
for each 'x' in 'array'
if 'x' ≤ 'pivot' then append 'x' to 'less'
else append 'x' to 'greater'
return concatenate(quick sort('less'), 'pivot', quick sort('greater')) // two recursive calls
```

3.4 Algorithm to sort using merge sort

Conceptually, a merge sort works as follows

1. Divide the unsorted list into *n* sub lists, each containing 1 element (a list of 1 element is considered sorted).

```
merge_sort(list m)
/*if list size is 1, consider it sorted and return it*/
if length(m) <= 1
return m
/* else list size is > 1, so split the list into two sublists*/
var list left, right
var integer middle = length(m) / 2
for each x in m before middle
add x to left
for each x in m after or equal middle
add x to right
/* recursively call merge_sort() to further split each sublist until sublist size is 1*/
left = merge_sort(left)
right = merge_sort(right)
/* merge the sublists returned from prior calls to merge_sort() and return the resulting merged sublist*/
return merge(left, right)
```

2. Repeatedly merge sub lists to produce new sub lists until there is only 1 sub list remaining.

```
merge(left, right)
var list result
while length(left) > 0 or length(right) > 0
if length(left) > 0 and length(right) > 0
if first(left) <= first(right)
append first(left) to result
left = rest(left)
else
append first(right) to result
right = rest(right)
else if length(left) > 0
append first(left) to result
left = rest(left)
else if length(right) > 0
append first(right) to result
right = rest(right)
end while
return result
```

3.5 Function to sort String arrays

The C# language and .NET Framework has several collection sorting methods and also there is LINQ query syntax. We benchmark and demonstrate the sort methods on arrays, such as Array.Sort, and Lists. We can call the static Array.Sort method and use it to sort a string array in place. The result is an alphabetical sort. This console program demonstrates how to use the Array.Sort method.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

3.5.1 Program that uses Array.Sort [C#]

```
using System;

class Program
{
static void Main()
{
string[] a = new string[]
{
"Egyptian",
"Indian",
"American",
"Chinese",
"Filipino",
};
Array.Sort(a);
foreach (string s in a)
{
Console.WriteLine(s);
}
}
}
```

Output:
American
Chinese
Egyptian
Filipino
Indian

3.6 Algorithm to merge sorted arrays

In the article we present an algorithm for merging two sorted arrays. One can learn how to operate with several arrays and master read/write indices. Also, the algorithm has certain applications in practice, for instance in merge sort. Assume, that both arrays are sorted in ascending order and we want resulting array to maintain the same order. Algorithm to merge two arrays A[0..m-1] and B[0..n-1] into an array C[0..m+n-1] is as following:

1. Introduce read-indices **i** , **j** to traverse arrays A and B, accordingly. Introduce write-index **k** to store position of the first free cell in the resulting array. By default **i** = **j** = **k** = 0.
2. At each step: if both indices are in range (**i** < m and **j** < n), choose minimum of (A[**i**] , B[**j**]) and write it to C[**k**]. Otherwise go to step 4.
3. Increase **k** and index of the array, algorithm located minimal value at, by one. Repeat step 2.
4. Copy the rest values from the array, which index is still in range, to the resulting array.

4. Result

When we split a large array into equal parts and apply efficient sorting functions on sub arrays in parallel, then parallel execution results in faster processing. It takes less time to merge all sorted arrays that have been processed quickly in separate thread in parallel. Algorithm could be enhanced in many ways. For instance, it is reasonable to check, if $A[m-1] < B[0]$ or $B[n-1] < A[0]$. In any of those cases, there is no need to do more comparisons. Algorithm could just copy source arrays in the resulting one in the right order. More complicated enhancements may include searching for interleaving parts and run merge algorithm for them only. It could save up much time, when sizes of merged arrays differ in scores of times. Merge algorithm's time complexity is $O(n + m)$. Additionally, it requires $O(n + m)$ additional space to store resulting array. One and only limitation is that the systems that does not support multithreading will not be eligible to get benefit. But most of the computers in this era are multithreaded based, so there are negligible technical issues.

References

[1] Astrachanam O., Bubble Sort: An Archaeological Algorithmic Analysis, Duk University, 2003.

[2] Cormen T., Leiserson C., Rivest R. and Stein C., Introduction to Algorithms, McGraw Hill, 2001.

[3] Levitin A., Introduction to the Design and Analysis of Algorithms, Addison Wesley, 2007.

[4] Nyhoff L., An Introduction to Data Structures, Nyhoff Publishers, Amsterdam, 2005.

[5] G. Franceschini and V. Geffert, An In-place Sorting with O (n logn) comparisons and O(n) moves, In Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, pages 242-250, 2003.

[6] Knuth D., The Art of Computer programming Sorting and Searching, 2nd edition, vol. 3. Addison – Wesley, 1998.

[7] J.L. Bentley and R. Sedgewick, Fast Algorithms for Sorting and Searching Strings, ACM-SIAM SODA" 97, 360-369, 1997.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

[8] Box R. and Lacey S., A Fast Sort, Computer Journal of Byte Magazine, vol. 16, no. 4, pp. 315-315, 1991.

[9] Basti Shahzad and Muhammad Tanvir Afzal Enhanced Shell Sorting Algorithm. World Academy of Science, Engineering and Technology 27, 2007.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.