An implementation of sorting algorithm based on java multi-thread technology

Deming Wang, Xiuqiong Zhang, Tao Men, Minrong Wang, Hongying Qin
Laboratory of Intelligent Information Processing and Application,
College of Computer Science, Leshan Normal University
Leshan, China
E-mail: 1486972@qq.com

Abstract—**An efficient implementation of quick sort algorithm based on java muti-thread technology was proposed for multi-core computer system. According to Divide-and-Conquer method, it divided the data into a number of segments, and then merged the segments into one with Merge Algorithm based on multi-thread. The experimental results showed that the new implementation is more effective than the traditional implementation. On the Dual-core computer, performance of the implementation increased by about 40%, the more core of computer, the better performance of the implementation.**

Keywords-java; multi-thread; quick sort; merge sort

I. INTRODUCTION

In a variety of traditional sorting algorithm, quick sort algorithm is often used in applications because quick sort is often faster in practice than other *O(nlog(n))* algorithms[1].Traditional implementation of quick sort uses a recursive strategy, it is meaningful in Simplify to programming. But in using traditional sorting algorithm, program often encounters a stack overflow because too large data and too many recursion level. With the spread of muti-core computer, ordinary desktop computers have strong parallel processing ability. But using traditional serial sort algorithm cannot take full advantage of powerful parallel computing power of the computer. Therefore, how to reduce the recursion level and how to improve the traditional algorithms, made him able to adapt to the development of computer parallel technology, to increase the efficiency of traditional algorithms to a new level, is a worthy subject of study.

This paper proposes a new implementation based on java multi-thread technology. First, the data are divided into several segments, each use a concurrent thread to quickly sort, and then merge all segments into one with merging algorithm, finally all of the data has been sorted.

The paper is organized as follows: In Section 2, the proposed new sort algorithm based on multi-thread is described in detail. In Section 3, describes implementation with java language. The results of experiment are shown in Section 4. In Section 5, further improvement is discussed. We conclude the paper in the last section.

II. ALGORITHM BASED ON MULTI-THREAD

A. Data segmentation algorithm

In order to reduce the recursion level of efficient quick sort algorithm, reducing the amount of data are an effective way. Our strategy is to divided total data into several segments; this is known as data segmentation algorithm. Each data segment is assigned a thread to sort and sort operations have done simultaneously.

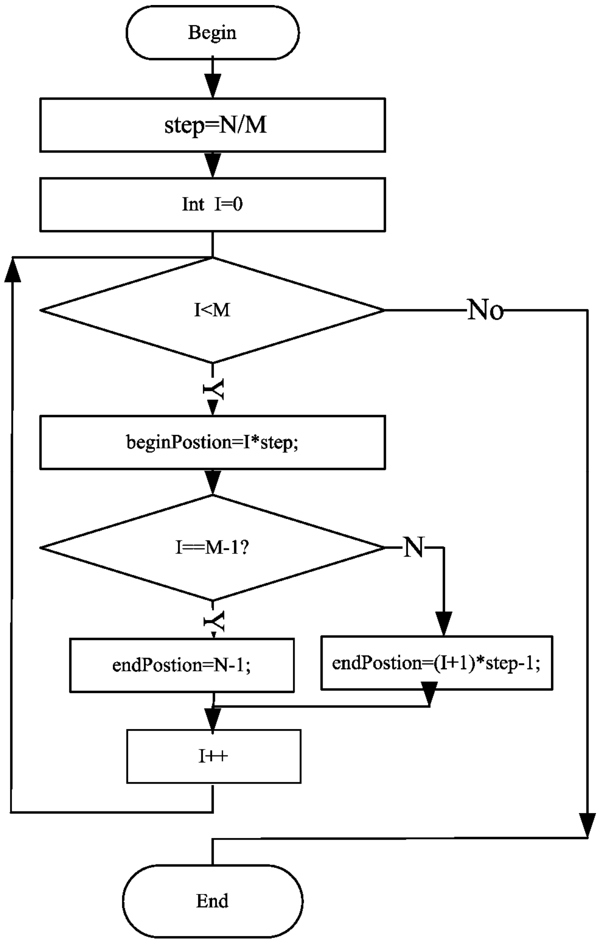

Figure 1. Data segmentation algorithm flowchart

Assuming that there were an array of length N, and number of segments is M, A segmentation algorithm is shown in Figure 1.

Step 1: Calculate the size of segment: step=N/M

Step 2: Set variable I=0

Step 3: if I <M set begin position to I*step, otherwise go to Step 6

Step 4: if I==M-1 set end position to N-1, otherwise, set end position to (i+1)*step-1

Step 5: I plus 1 and go to Step 3

Step 6: algorithm end

B. *Segments sorting algorithm*

After the data was segmented, we have to sort these data. Said above, each data segment is assigned a thread to sort. In order to achieve efficiency, we used the quick sort algorithm for segment to sort and in order to make full use of concurrency power of computer, we use multi-threaded strategy. Sorting algorithm is shown in Figure 2.

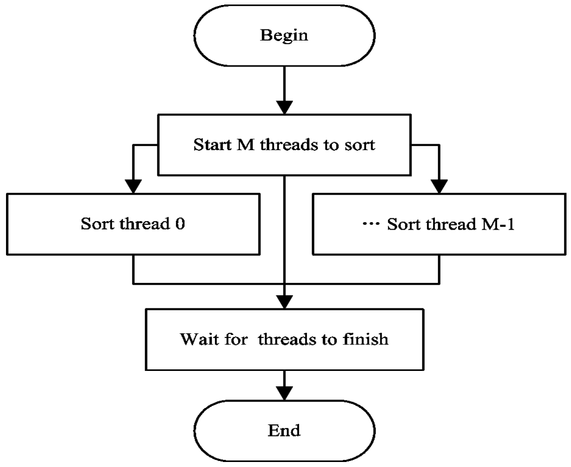

Figure 2. Segments sorting algorithm flowchart

Step 1: start M threads that simultaneously use quick sort algorithm to sort each data segment

Step 2: wait for all threads to finish

Step 3: start merge algorithm

Step 4: algorithm end

C. *Merging segments algorithm*

Algorithm time complexity of Merge Algorithm is O (nlog (n)) and stability of the algorithm is very good, so we chose it as the implementation's merge algorithm. But traditional merge sort algorithm implementation is only executed serially, does not have the ability to execute concurrently, Therefore on this basis we must redesign the concurrent execution algorithm.First,we need a algorithm to get begin position and end position information of segments merged from segments information.

1) *Merge task list generation algorithm*

The role of the algorithm is to generate merge task list from segments information list. The elements of the task list contain position of two adjacent segments. Merge task list is not static, after the completion of a merge work, will regenerate a new merge task list until there are no new merge task. The algorithm is shown in Figure 3.

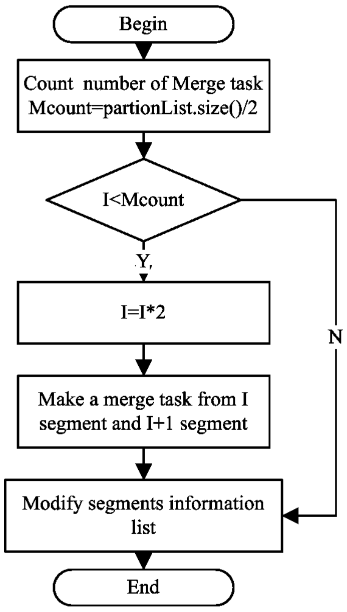
Begin
Count number of Merge task
Mcount=partionList.size()/2
I<Mcount
Y,
N
I=I*2
Make a merge task from I segment and I+1 segment
Modify segments information list
End

Figure 3. Merge task list generation algorithm flowchart

Step 1: Count the number of Merge task depending on the size of segment information table, mcount=partionList.size ()/2, mcount is number of merge task and partionList.size () is number of element of segment information list.

Step 2: From I=0 to mcount-1 do step 3, if I==mcount, then go to step 5

Step 3: get two adjacent segments information from segments information list and construct a merge task object and add to merge list.

Step 4: go to Step 2

Step 5: modify segments information list to remove the elements which added to merge list

Step 6: algorithm end

2) *Merge algorithm base on multi-thread*

After making merge tasks list, each task will start a merge thread which merge two adjacent segments. Number of tasks is usually more than one, depending on number of segments. After a task list merge finished, a new task list will generate, until number of tasks in merge task list is 0.the algorithm is shown in Figure 4.

Step 1: generate task list from segments information list

Step 2: starts a merge thread for each task in task list

Step 3: wait for all merge threads to finish

Step 4: generate task list from segments information list again.

Step 5: if number of task in merge task list is greater than 0, go to Step 2, otherwise go to Step 6

Step 6: algorithm end

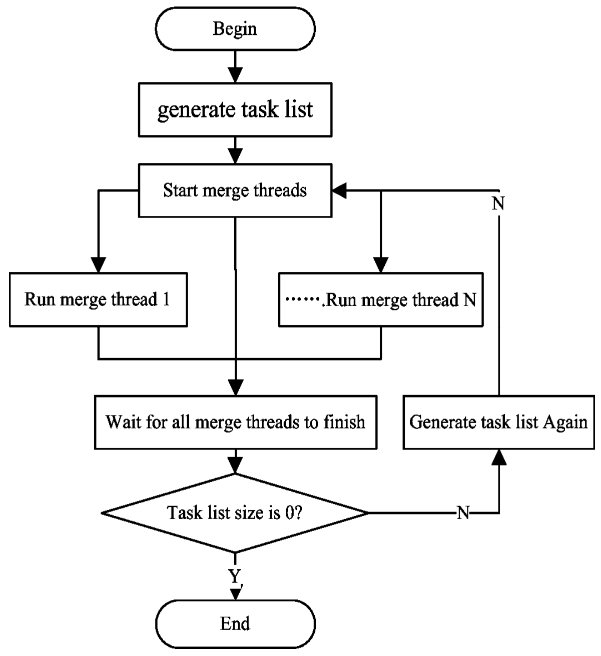

Figure 4. merge algorithm base on multi-thread flowchart

After the above all algorithm steps, sorting of data is finished.

III. IMPLEMENTATION IN JAVA

Java has the characteristics of good object-oriented and good cross-platform capability. To test the performance of the algorithms in different system, so we have chosen Java to implement our algorithm.

In order to complete the sort, we designed the following classes, shown in Figure 5.

A. *Class MergeSort*

This class provides a method, for merging two sorted segmentation into one. It used traditional merging algorithm [2] [3] [4].

B. *Class QuickSort*

This class provides a method, for sorting segmentation. It used traditional quick sort algorithm [5].

C. *Class SortRunner and MergeRunner*

The two classes are thread class for providing sorting and merging work, they have more than one concurrent execution and they are called the QuickSort class and MergeSort class.

D. *Class SortPools*

This is a thread manager class, his role is to manage thread and provides making segments information list and making the Merge task list.

E. *Other Supporting Class*

These class provides some function for supporting sort work, such as some class saving segmentation information, class saving merge tasks, class of stopwatch, and so on.

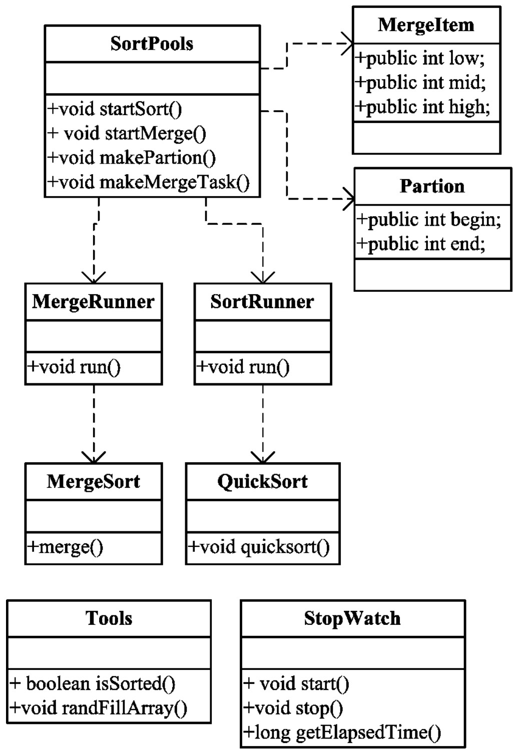

Figure 5. class diagram of implementation in java

IV. EXPERIMENT

We experiment in both of two different operating system on the same computer.

A. *Hardware platform:*

CPU: Intel T7500 Duo CPU (double core inside)

Memory: 2G

CPU Frequency: 2.2G

B. *Software platform:*

Window 7 home

Ubuntu 11.04(Linux)

C. Results

First of all, we have established an array of integers of length 10 billion and then filled random integer in the array, integer range is arbitrary. Next, we use traditional algorithm and our algorithm to sort the same data, record the time they need and Finally we calculated the proportion of reducing time compared to traditional algorithm, the proportion have called Efficiency ratio .

Experimental results have shown in the following Table I and Table II.

TABLE I. RESULTS OF IN WIN7

threads	Results(data size:10000000 integers)		
	Traditional quick sort cost time(ms)	***Our multi-thread sort cost time(ms)***	***Efficiency ratio***
1	2107	2088	1%
2	2167	1306	40%
4	2162	1377	36%
8	2222	1495	33%
500	2232	1823	18%
1000	2214	2092	6%

TABLE II. RESULTS OF IN UBUNTU 11.04

threads	Results(data size:10000000 integers)		
	Traditional quick sort cost time(ms)	***Our multi-thread sort cost time(ms)***	***Efficiency ratio***
1	1859	1824	2%
2	1897	1165	40%
4	2022	1286	36%
8	1862	1271	32%
500	1944	1637	16%
1000	1813	1745	4%

Experimental results show that using our implementation of algorithm with one or more threads, there are different degrees of performance improvement. Degree of performance improvement related to concurrent threads in CPU.That is, if the CPU number of concurrent threads is two, our implementation with two threads will have the fastest results , the more concurrent threads, the more faster. Results in the two platforms are consistent no matter what you use operating system. So our implementation and improvement base on multi-thread technology is effective.

V. DISCUSSION

Due to the current CPU can perform multiple tasks at the same time, so using principle of divide-and-conquer, let some unrelated operations carried out synchronously, can effectively improve the efficiency of implementation. However, if too many threads, since threads frequently switch, it will lead to performance degradation. Our algorithm divides the data into unrelated parts, so you can use the computer's Parallel ability to increase performance, our implementation show the best performance when the number of threads is equal to the number of CPU threads.

In theory, the time complexity of quick sort is *O (nlog (n))*, it is *O ((n/m) log (n/m))* in our implementation because data divided into m segments. the time complexity of merge sort is O(n),merge sort of our implementation is less than *O(n/2)* because concurrent merge operations, So the time complexity of our implementation is less than *O((n/m)log(n/m))+O(n/2)*[6].

In our current implementation, sorting and merging is still serial. But in fact, sort and merge would also be parallel. Therefore, our algorithm can also be further improved, this is our next direction. As long as there are two sorted data block, you can immediately start the merge operation, and this can further improve the performance.

VI. CONCLUSION

In order to make full use of concurrency power of computer, an implementation based on java multi-thread technology is proposed in this paper. It brings advantages of traditional quick sort and merge sort together, and multithreaded improvements makes sorting more efficient than traditional implementation, increased by about 40% on dual-core computer. On the more core computer, speed will increase even more.

This implementation will greatly improve the sorting ability of multi-core computer, allow applications that need sort feature to perform efficiency greatly and the implementation can further improve the efficiency of this algorithm by improve algorithm.

REFERENCES

[1] Wikipedia. Quicksort [EB/OL]. http://en.wikipedia.org/wiki/Quicksort 2011-03-28

[2] Wikipedia. Merge sort [EB/OL]. http://en.wikipedia.org/wiki/Merge_sort 2011-03-28

[3] Knuth, Donald (1998). "Section 5.2.4: Sorting by Merging". Sorting and Searching. The Art of Computer Programming. 3 (2nd ed.). Addison-Wesley. pp. 158–168. ISBN 0-201-89685-0.

[4] Katajainen, Jyrki; Pasanen, Tomi; Teuhola, Jukka (1996). "Practical in-place mergesort". Nordic Journal of Computing 3: pp. 27–40. ISSN 1236-6064. Retrieved 2009-04-04. Also Practical In-Place Mergesort.

[5] R. Sedgwick, Implementing quick sort programs, Comm. ACM, 21(10):847-857, 1978. Implementing Quick sort Programs

[6] Brian C. Dean, "A Simple Expected Running Time Analysis for Randomized 'Divide and Conquer' Algorithms." Discrete Applied Mathematics 154(1): 1-5. 2006.