Parallel Merge Sort Implementation

Using OpenMP

Jaeyoung Park, Kyong-Gun Lee, and Jong Tae Kim
School of information communication engineering,
Sungkyunkwan University, Suwon, GyeongGi-Do,
South Korea

Abstract – One of representative sorting a algorithm, merge sort, is widely used in database system that requires sorting due to its stability. To raise performance of merge sort, it is effective to change parallel method but due to difficulty of changing database system, it requires parallel method that minimizes changing the system. We suggest parallel merge sort that uses OpenMP, capable of parallel, inserted directive to existing program code that improves performance while minimizing change to the system. Also, we examined performance of merge sort depending on k-th number of way and performance of parallel region occupied by sort that can affect sorting process by constructing variety of cases using $2^{20}$ or 1,048,576 data for experimental purposes. As a result, the most important factor that affects parallel method is usage of core, not ways. By considering this factor, experimental results demonstrate a speedup of 1.8 on 2 processor core, and a speed up of 2.8 on 4 processor core.

Keywords: OpenMP, Merge sort, k-way merge sort, parallel processing.

1 Introduction

For database system that processes large amount of information, there are many search algorithms that are fast and accurate for usage. Similarity of all these algorithms is that data are assembled in sort by search requirement [1]. It means that fast and accurate search is fast and accurate sorting. When huge amount of information is updated in a day, this situation demands more frequent data sort. Also, there is trend in which amount of data for sort is getting increased for one process. Therefore, in database system, time and effort are getting increased for sorting. A method is needed to reduce sorting time to fix this problem. A database system that processes huge amount of information needs increase in system effectiveness but since information must be provided continuously, it requires a method that does not alter system greatly while increasing effectiveness. However, a regular comparison-based algorithm can’t go beyond $n \log n$ effectiveness change [2]. There are parallel method that takes care of data simultaneously in hopes of raising effectiveness but those methods arranges road balancing or makes communication between inter- processor faster[3], which makes construction of parallel method difficult and creates great change to the system in the process. Trying to solve these weaknesses, this thesis chose merge sort algorithm due to its characteristic of being stable, which is a reason it is widely used in database system, and OpenMP was used for parallel method. OpenMP would not alter system greatly by inserting directive for parallel code into existing code, and it is supported by most of compilers so it can be used in most of systems [5]. Therefore, parallel merge sort using OpenMP can solve problems discussed earlier. In comparison to other methods, it can be implemented easily so we can expect higher effectiveness without much effort. Along with parallel merge sort algorithm, variety of scenarios will be constructed so that results will be analyzed by examining effectiveness depending on change of k in k-way merge sort and effective of parallel region in applying OpenMP to suggest effective parallel method.

Section 2 would explain k-way merge sort algorithm and OpenMP that was used for parallel sorting. Section 3 would explain which method to divide data domain in attempt to parallel sorting. Next, Section 4 would analyze the results and finally give conclusion.

2 Related Work

2.1 K-way Merge Sort

Merge sort is one of premium sorting algorithms that before sorting, data is separated (divide), each partial component is sorted (conquer), then all sorted components are pasted together in a cycle to sort entire data, which is one of the (divide and conquer) method. During a cyclical call, it goes through three steps; divide, conquer, and paste. Divide is a step where given data is divided into many small data. Conquer is a step where divided data is sorted in orderly manner. Last step is paste where data is pasted and completely sorted. During merge sort’s process, depending on how unit is divided and goes through those three steps are called k-way merge sort. Like Fig 1, if it is divided into two units, it is called 2-way. Like Fig 2, if it is divided into four units, it is called 4-way.
2.2 OpenMP

OpenMP is an API[6] for parallel programming model that allowed to be used by application by inserting compiler directive to allow parallel processing. Even if program code is created in order manner, if directives are added, only specified area is changed for parallel processing. OpenMP can only be used is SMP (shared memory multiprocessor) but after few years, most of CPU provides at least dual core so there is no problem for use it [7]. Its positive trait is that it is de-facto standard, therefore it is provided by many compliers. It can be inserted into given program code so with little effort we can expect high rise in effectiveness and expansion. Fig 3 is the example of the use of OpenMP. When executing the below instruction, the execution time can be reduced by parallel add operation with 4 threads.

```
#pragma omp parallel for shared(n,a,b,c) private(i)
for(i=0;i<n;i++)
c[i]=a[i]+b[i];
```

3 Parallel Method

In this paper, previously mentioned merge sort is paralleled by using OpenMP. In types of merge sort, to find out difference in speed due to number of way, way is changed and implemented to 2-way, 4-way, and 8-way. Also, to find out difference between performance of parallel region and performance due to number of cores, according to number of way, method of dividing and number of cores were changed together.

3.1 2-way Merge Sort

In case of Fig 4, total data is divided into four (1). Divided data are each goes to merge sort in (2). In (3) and (4), it goes through merge sort again and then pasted into sorted data. In a case where four cores are used, in (2), each one goes through merge sort so (2) uses four cores, and (3) and (4) each uses two and one respectively. When two cores are used for parallel processing, in (2), first two data are sorted then next two are sorted. Then it goes to (3). Number of core used in (2) is two, and it was used twice. (3) and (4) used two and one respectively to reduce the time.

![Figure 4](image_url)

In case of Fig 5, it is total process is equal to Fig 4, but the data is divided into 8 units and if four cores are used, in (2), first four goes through merge sorted, then next four goes through merge sort. In (3), eight data are simultaneously pasted together using four cores. In case where two cores are used, in (2), two data goes through merge sort four times. In (3) and (4), two cores are used twice and once respectively. In last step (5), only one core is used. This was for comparing number of division and speed of sorting when total amount of data is equal for merge sort.

![Figure 5](image_url)
3.2 4-way Merge Sort

4-way merge sort, different from 2-way which compares two data at a time, it can compare four data at once for sorting. Therefore, data distribution is increased to multiples of four, four and sixteen, to compare four data at once. Fig 6 shows entire data is divided into four so one through four data goes through merge sort by each core, and it goes through merge sort in the last step. If all four cores are used, it can go to last step at once. If two cores are used, parallel processing goes twice in the second step and one core is used for the last step in the 4-way merge sort. Fig 6 shows order for 4-way merge sort when data is divided into four. In case where two cores are used, dark box shows data pairing where 1 and 2 are processed, then 3 and 4, then when 1 through 4 is done, last 4-way merge sort is done by using only one core.

![Figure 6. 4-way merge sort divided by 4 data](image)

Fig 7 shows 4-way merge sort when data is divided into sixteen. Merge sort is processed in numerical order, and dark box shows data pairing when four cores are used. Four cores each process 1~4, 5~8, 9~12, and 13~16 then process 17~20. After that you get the result in the last step with one core. If only two cores are used, then units should be numerically paired in two and be processed.

![Figure 7. 4-way merge sort divided by 16 data](image)

3.3 8-way Merge Sort

If you ignore the fact that number of units that can be processed at once is eight, then 8-way merge sort is similar to other k-way merge sort. Fig 8 shows method for arranging data by dividing it into eight. Once 1~8 are sorted by merge sort, then 8-way merge sort is processed once more to get final result.

![Figure 8. 8-way merge sort divided by 8 data](image)

4 Experimental Results

Experiment took place in quad core 2.4Ghz environment. 2^30 or 1,048,576 data were randomly generated and measured time it took to sort. Time is measured using CPU internal clock. Table 1 results are averages of 100 times it ran to minimize error that can occur using CPU internal clock. Unit for time is sec. Horizontal line shows number of cores processing merge sort and vertical line shows under what condition merge sort is processed. It also shows number of data divided for 2-way, 4-way, and 8-way. One means one data is used, and it was not divided. For each k-way, one means no parallel process exist, therefore no speed up.

Table 1 results shows that in 4-way and 8-way, even if there were no parallel process, in case of 4-way when data is divided into four and in case of 8-way when it is divided into eight, time was reduced slightly. Through running it 100 times, it showed steady difference so it can’t be said that it is an error. It can be predicted that it happened because experiment was done in quad core system so it can be effect of optimizing compiling stage and effect of multi thread that is automatically done by operating system. It’s increases is too small, and it is in margin of error in 2-way so it clearly does not show improvement so we can’t conclude that dividing earlier data into many pieces leads to improvement.

In looking at method of dividing data for parallel processing in k-way, if two cores are used, when data is split into four you get 0.1749. When data is split into eight you get 0.1753. If four cores are used, when data is split into four you get 0.1162. When data is split into eight you get 0.1192 so it is almost equal. In 4-way, you get 0.2042 and 0.1971 for two cores and 0.1344 and 0.1351 for four cores. Therefore raising number of dividing data does not much affect parallel performance. This is due to four or two core are run with limits of CPU so CPU usage is almost same. If in 2-way, where data is divided by two, CPU that actually runs it can’t use more than two cores so as shown in Table 1, you can’t expect improvement when four cores are used. Also in 4-way, when data is split into five, previously sorted four must wait for the fifth one so it would reduce effectiveness. However, in 2-way where data is divided into two, although one would expect no improvement, it improved effectiveness by 1.17. This is a small profit of work being processed earlier by optimal condition in compiling stage. Therefore, there is no change in conclusion that if there is N core, then data should be divided in multiples of N for most improvement in effectiveness.
Table 1. Execution time of merge sort (sec)

<table>
<thead>
<tr>
<th>Num of cores</th>
<th>Single</th>
<th>Dual</th>
<th>Quad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-way</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.3362</td>
<td>0.3376</td>
<td>0.3377</td>
</tr>
<tr>
<td>2</td>
<td>0.3405</td>
<td>0.2076</td>
<td>0.1768</td>
</tr>
<tr>
<td>4</td>
<td>0.3345</td>
<td>0.1749</td>
<td>0.1162</td>
</tr>
<tr>
<td>8</td>
<td>0.3396</td>
<td>0.1753</td>
<td>0.1192</td>
</tr>
<tr>
<td>4-way</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.3857</td>
<td>0.3849</td>
<td>0.3832</td>
</tr>
<tr>
<td>4</td>
<td>0.3788</td>
<td>0.2042</td>
<td>0.1344</td>
</tr>
<tr>
<td>16</td>
<td>0.3614</td>
<td>0.1971</td>
<td>0.1351</td>
</tr>
<tr>
<td>8-way</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.3936</td>
<td>0.3915</td>
<td>0.3906</td>
</tr>
<tr>
<td>8</td>
<td>0.3669</td>
<td>0.2035</td>
<td>0.1246</td>
</tr>
</tbody>
</table>

Most clearly shown result is when core is increased to take care of many data simultaneously. If two cores are used for parallel processing, in case of 2-way with dividing data into four showed above 1.9 improvement. If four cores were used it showed 2.8x improvement. It used twice as much cores but showed 1.5x improvement from two. This is due to parallel region is limited which all four cores can’t participate in processing at once. In 8-way, from single to dual showed 1.8x improvement and single to quad showed 2.9x improvement similar to 2-way. In case of 4-way, if data is split into four and sixteen, single to quad showed 2.8x and 2.67x respectively. This is due to process where data is increased for single core, some of it were sorted early by operating system. So area available for OpenMP were smaller. Specific number is different but shows similar amount of improvement in performance.

5 Conclusion

For improving performance of database system, performance of sorting algorithms must increase. Among techniques that improve performance, there is parallel processing. However, it is difficult to materialize, unexpected error can occur, and can alter system greatly. Therefore, being effective and easy to use, we paralleled merge sort algorithm using OpenMP. We got the results like table 1 changing the number of cores which participate merge sort process and the region of parallel processing. When looking at the results using the same number of cores, there is no clear relationship with k-way and merge sort. Though changing k-way, it can reduce number of combining but due to computer’s trait, k can’t be all handled at once and internally it uses binary comparator so total number of comparison is the same. It should be distributed to parallel merge sort algorithm using OpenMP and parallel region should be at least be greater than number of core so that it can reduce time that core is not being used. If the number of parallel regions are less than the number of cores, the performance is hardly improved not being able to conjugate the resources. It can be shown by using single core with k-way merge sort and using dual or quad core with 2-way merge sort. For greater effectiveness, area should be in multiples of number of core so that when earlier steps are done, number of core waiting can be reduced, which will result in greater usage of entire core and great performance in parallel processing. Through running merge sort algorithm we implemented with conclusions, in case of dual core, it showed 1.8x improvement, and in case of quad core, it showed 2.8x improvement. Finally, without changing system greatly, data should be divided so that area where OpenMP is applied should be set and parallel that part so great improvement were acquired.

6 References


