VIT

UNIVERSITY

(Estd. u/s 3 of UGC Act 1956)

Vellore - 632 014, Tamil Nadu, India

School of Information Technology and Engineering

B.Tech (IT)

Hybrid Parallel Sorting Algorithm

Sarthak Jain
(11BIT0134)
Govind Sharma
(11BIT0131)
Srimant Mishra
(11BIT0166)

Guide	**Panel member**
Name:	**Name:**
Prof. Ranichandra C	**Prof. Hari Ram Vishwakarma**
Signature:	**Signature:**
Date:	**Date:**

Abstract:

Sorting in database is a common problem in computer science. There are a lot of well-known sorting algorithms created for sequential execution on a single processor. Recently, many-core and multi-core platforms have enabled the creation of wide parallel algorithms.In our project, we plan to develop a hybrid algorithm which uses components of key parallel algorithms in use today. The target is to incorporate best practices of notable algorithms and implement it in an interface via which we can compare our algorithm to existing ones.

Literature Survey

Sorting in database processing is frequently required through the use of Order By and Distinct clauses in SQL . Sorting is also widely known in computer science community at large. Sorting in general covers internal and external sorting. Past published work has extensively focused on external sorting on uni-processors (serial external sorting), and internal sorting on multiprocessors (parallel internal sorting) [1] [2] [4]. External sorting on multi-processors (parallel external sorting) has received surprisingly little attention; furthermore, the way current parallel database systems do sorting is far from optimal in many scenarios. We present a taxonomy for parallel sorting in parallel database systems, which covers jive sorting methods: namely parallel merge-all sort. Parallel binary-merge sort, parallel redistribution binary-merge sort, parallel redistribution merge-all sort, and parallel partitioned sort [1][2].

1. Parallel Merge-All Sort

Parallel Merge-All Sort method is a traditional approach which has been adopted as the basis for implementing sorting operations in several research prototype (Gamma) [4] and some commercial Parallel DBMS. Parallel Merge-All Sort is composed of two phases: ***local sort*** and final ***merge.*** The ***local sort phase*** is carried out independently in each processor. Local sorting in each processor is performed as per normal serial external sorting mechanism. [1][3][10]

2. Parallel Binary-Merge Sort

The first phase of ***Parallel Binary-Merge Sort* is** a *local* ***sort*** as like in parallel merge-all sort. The second phase: the ***merging phase*** is pipelined, instead of concentrating on one processor. The way the merging phase works is by taking the results from two processors, and merging the two in one processors. As this merging technique uses only two processors, this merging is called "Binary Merging". The result of the merging between two processors is passed on to the next level until one processor left; that is the host. Subsequently, the merging process forms a hierarchy. [1][2][10]

The main motivation to use parallel binary-merge sort is that the merging workload is spread to a pipeline of processors, instead of one processor. It is true however that final merging has still to be done by one processor.

3. Parallel Redistribution Binary-Merge Sort

Parallel Redistribution Binary-Merge Sort is motivated by parallelism at all levels in the pipeline hierarchy. Therefore, it is similar to parallel binary-merge sort where both methods use hierarchy pipeline for merging local sort results, but differ in the context of number of processors involved in the pipe. Using parallel redistribution binary-merge sort, all processors are used in each level in the hierarchy of merging.

The steps for parallel redistribution binary-merge sort can be described as follows. First, a local sort is carried out in each processor as like in the previous sorting methods. Second, redistribute the results of local sort to the same pool of processors. Third, do a merging using the same pool of processors. Finally, repeat the above two steps until final merging. The final result is the union of all temporary results obtained in each processor. [1][2][7]

4. Parallel Redistribution Merge-All Sort

Parallel Redistribution Merge-All Sort is motivated by two factors, particularly reducing the height of the tree while maintaining parallelism at the merging stage. This can be achieved by

exploiting the feature of parallel merge-all and parallel redistribution binary-merge methods. In other words, parallel redistribution is a two phase method (local sort and final merging) like parallel merge-all sort, but does a redistribution based on a range partitioning. [1][2]

5. Parallel Partitioned Sort

Parallel Partitioned Sort is influenced **by** the techniques used in parallel partitioned join, where the process is split into two stages: partitioning and independent local work**.** In parallel partitioned sort, first we partition local data according to range partitioning used in the operation. Notice the difference between this method and others. In this method, the first phase is not a local sort. Local sort is not carried out here. Each local processor scans its records and redistribute or repartition according to some range partitioning.

After partitioning is done, each processor will have an unsorted list in which the values come from various processors (places). It is then local sort is carried out. Thus, local sort is carried out after the partitioning, not before. It is also noticed that merging is not needed. The results produced by the local sort are already the final results. Each processor will have produced a sorted list, and all processors in the order of the range partitioning method used in this process are also sorted.[1][2][5]

Requirements

Any language construct can be chosen for the purposes of demonstrating our new algorithm, but we are going to be using C in a parallel environment for the sake of this pedagogy. Hadoop will be used for simulating databases with big data, and Open-Mp will be used for parallel execution of the C implementation of the algorithm.

Module Design

We have witnessed from the existing algorithms that the key issues to be targeted are

1. Height of the tree should be small
2. Bottleneck should not exist
3. All processors should be used in every step of the pipeline hierarchy
4. Skew should be eliminated as far as possible
5. Final merging should be as trivial as possible
6. Processors should get equitable work

And from the demonstration in the slideshow we can show that the parallel partitioned sort is successful in being a counter to all the challenges except skew and equitable distribution of work. It produces a one level tree, no bottleneck, has optimum usage of all processors in the pipeline, and makes the final merging trivial.

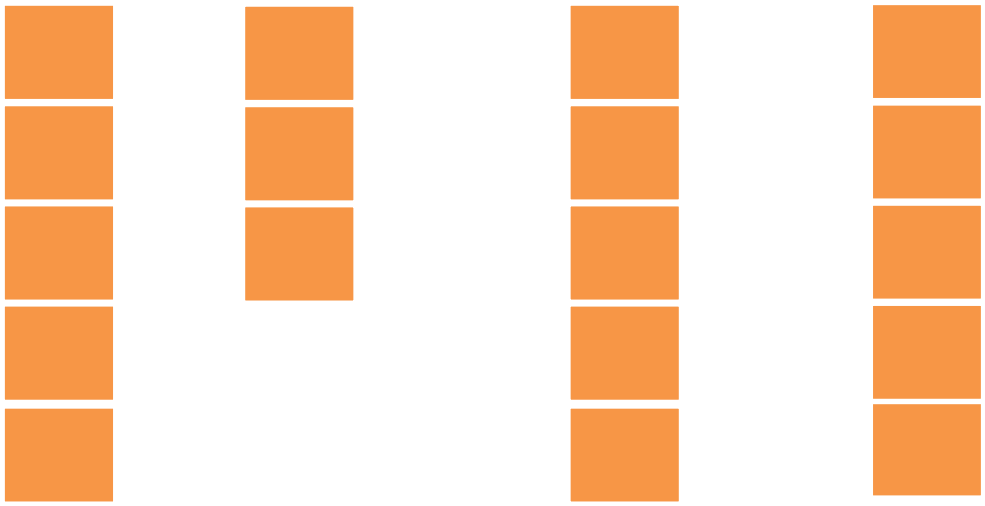
A method to overcome these issues are using buckets to distribute equitable work and eliminate skew to a great degree.

There already exists the concept of buckets, but the nature of distribution is not optimal in it. We shall be using one of two methods, namely probability distribution or regression to optimally distribute elements within the buckets.

The concept of buckets will be described in detail in a further section of this project, but here we provide a brief overview to what it is and how we plan to implement it.
While using buckets to eliminate skew, we use the analogical bucket to serve as a container for elements that we obtain from processors. We do this so as to maintain an equitable distribution of work among processors where all of them are allotted an approximately even load.

The general tendency is to make the more buckets than there are processors. This way we can quantize the load into chunks approximately the same size.

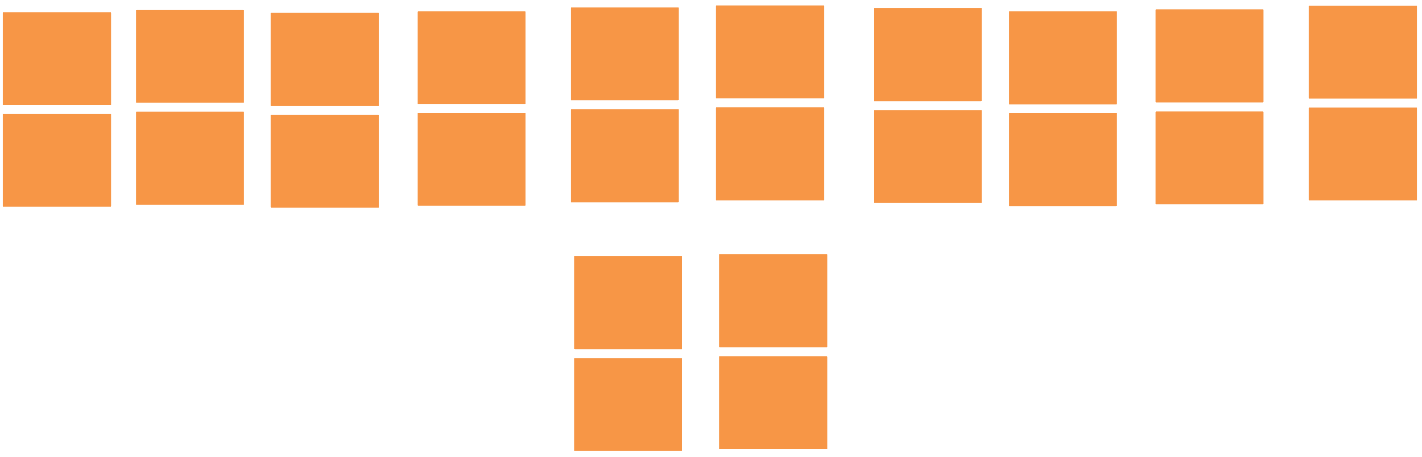
Diagrammatical representation



Here all the boxes will contain some values. They will need to be merged and then be sorted.

If we implemented the concept of buckets into this, we would have to distribute these 28 boxes into a configuration that 4 processors would be able to optimally use. One example would be to use 7 boxes with 4 elements each. This would make one processors load higher by 4 items. Another method could be to use 12 buckets of 2 elements each and then 2 buckets of 2 elements each. Since we aim to maximise the parallelism of this process, we would use the latter combination.

The combination of buckets would be:



While we used a random guessing method to generate nearly equal work for all the buckets, for larger sets of data, we would use advanced mathematical methods to carry out the process of bucketing more scientifically. These methods would put the probability distribution or regression models to use.

References

1. Bergsten, **B.,** Couprie, M., and Valduriez, P., "Overview of Parallel Architecture for Databases", The Cornpiiter Journal
2. Bitton, D., et al, "A Taxonomy of Parallel Sorting", ACM Conip. Surv.
3. High-Performance Parallel Database Processing and Grid Databases by David Taniar, Clement H.C Leung, Wenny Rahayu and Sushant Goel
4. Bergsten, B., Couprie, M., and Valduriez, P., "Overview of Parallel Architecture for Databases", The Cornpiiter Journal
5. DeWitt, D.J. and Gray, J., "Parallel Database Systems: The Future of High Performance Database Systems", Cornm. of the ACM
6. Xipeng Shen and Chen Ding., "Adaptive Partitioning for Sorting Using Probability Distribution"
7. Hua, K.A., Lee, C. and Hua, C.M., "Dynamic Load Balancing in Multicomputer Database Systems Using Partition Tuning", lEEE TKDE
8. OpenMP: OpenMP website – www.openmp.org.
9. Kitsuregawa, M. and Ogawa, Y., "Bucket Spreading Parallel Hash: a New, Robust, Parallel Hash Join Method for Data Skew in the Super Database Computer (SDC)"
10. Sequential and parallel algorithms – http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/algoen.htmL