Performance Evaluation of Apriori on Dual Core with Multiple Threads

Anuradha.T
ANU,Guntur
A.P.,INDIA

Satya Pasad R
ANU,Guntur
A.P.,INDIA

S.N.Tirumalarao
NEC,Guntur
A.P.,INDIA

ABSTRACT
The complexity of handling the scalability problem of huge data can be reduced with parallel processing. The efficiency of parallel processing changes as the number of processors or number of threads change. Parallel processing is more appropriate for the field like data mining as it is the technique of analyzing large quantities of data to extract useful knowledge. Data mining is very much essential to the modern society as more and more data is being collected from various fields. Experiments are conducted to test the run time efficiency of the apriori algorithm on dual core processor by changing the number of threads for different databases at different support counts. This paper also present the comparison of real time, user time and system time with multiple threads on dual core compared to sequential implementation.

General Terms
DataMining, Parallel Processing

Keywords
apriori, data mining, efficiency, multi core,openMP, parallel processing

1. INTRODUCTION
Data is being collected in the society from various fields regularly for different purposes because of which lot of data is being accumulated at different sources. Data mining is a technique developed by information technology people to make use of this data for getting some useful knowledge (1, 2). Data mining consists of different functionalities in which frequent itemset mining (FIM) or association rule mining (ARM) (3) is an important one. As data mining is evolved to analyze massive data, sequential algorithms can not deal with them properly in terms of scalability (4, 5).

Various parallel FIM approaches were proposed to tackle the problem of scalability using different approaches. (5,6,7,8,9) In this paper we use an easy and efficient technique by utilizing the technological improvement in processor technology (10,11) to tackle the problem of scalability in terms of time. We use Pentium dual core and multi threading concept for parallelizing the apriori algorithm (12) which is a popular FIM algorithm and present the performance results of real time, user time and system time by changing the number of threads. We compare the efficiency of the algorithm for 2,3,4 threads.

2. RELATED WORK
Various sequential algorithms were proposed for finding the frequent itemsets. Algorithms that used horizontal data format where data is represented as transaction ID versus items sold in each transaction were proposed (12,13,14).The algorithm proposed by Zaki uses vertical data format consisting of each item versus transaction Ids in which the item is sold (15).To reduce the scalability problem, Count distribution and data distribution were the first two parallel ARM algorithms based on apriori proposed by Rakesh Agrawal and John C.Shafer (7). The parallel algorithm for finding the frequent itemsets using FP-growth algorithm was proposed by Zaıane et al (9). A survey on parallel and distributed association rule mining is done by Mohammed J.Zaki (5). Finding sub trees from a tree structured data on multi core is proposed by S.Tatikonda and S.Parthasarathy (16). Frequent itemset mining on multi core was proposed by Li Liu et al. and Anuradha et al. (17, 18). Efficiency of parallel processing was discussed in (19, 20).

3. THEORETICAL BACKGROUND
3.1 Frequent itemset mining
Frequent itemsets are the sets of items. If these sets occur more than a specified number of times (which is known as minimum support count) in the transactional database, then we call them frequent. It was first proposed by Agrawal et al. and it helps in finding the associations between different sets of items in the given database. The concept of association rule mining (3) proposed for market basket analysis identifies the probability for a set of items to be purchased by the customers when ever they are purchasing another sets of items. For example if the customer is buying {milk, sugar}, what is the probability that he may also buy {bread}? This information is more useful for the retailers to increase their sales by keeping those products available in the stores or for arranging racks in the stores. It is based on two parameters support and confidence.

1. Support= percentage of transactions in D that contain XUY

$$Support(X \Rightarrow Y) = P(X \cup Y) \quad (1)$$

2.Confidence=percentage of transactions in D containing X that also contain Y.

$$Confidence(X \Rightarrow Y) = P(Y/X) \quad (2)$$

Where X and Y are the subsets of the set of all given items in the transactional database D.

3.2 Apriori algorithm
There are many algorithms proposed for FIM of which apriori algorithm is the simple and most popular algorithm. It is based on the apriori property that all nonempty subsets of a frequent itemset must also be frequent. In the Apriori algorithm, each different item is placed in candidate 1-itemset

C_1. we count in how many transactions each item C_1 is occurring. We compare this count with a predefined number called as minimum support count, min_sup. If the count of item is more than min_sup, that item is placed in frequent 1-itemset L_1. The set L_1 is natural joined with itself to find candidate 2-itemset C_2. The process of finding the count of each itemset in C_k for finding k-frequent itemset L_k and joining L_k with itself to get C_{k+1} is repeated until L_k is a null set.

The execution of this algorithm requires scanning the entire database for finding the count of each item in every candidate k-itemset. To find the set of all frequent itemsets, it requires more time if the database size is more.

3.3 Multi core Architectures

A multi core processor is a single computing component with two or more actual processors (21, 22).A multi core processor implements multiprocessing in a single physical package. In multi core architectures, degradation of signals is less as they have to travel less distances when compared to normal shared memory parallel processors because the processors are placed on the same chip. These enhanced quality signals allow more data to be sent in a given time period. The largest boost in performance will likely be noticed in improved response-time while running CPU- intensive processes. Applications that are designed for a multiprocessor or multithreaded environment can take advantage of multi core processor architectures (23).

3.4 OpenMP

OpenMP stands for open multi processing (24, 25,26,27). It is a very convenient API for parallelizing programs in a shared memory environment because it provides a set of pragmas which, when used in a program, instructs an OpenMP - capable compiler to parallelize it. If the compiler is not OpenMP aware, the OpenMP pragmas are silently ignored. OpenMP uses the fork-join model of parallel execution. As long as the application is running sequentially, the master thread will execute it and when ever a parallel region construct is encountered, master thread forks child threads and all the child threads run parallelly until they complete executing the statements in the parallel construct. Again master thread continues to run the remaining part of the application. We can set the number of threads by using omp_set_num_threads () library function or setting the OMP_NUM_THREADS environment variable. OpenMP supports two basic kinds of parallelism with two work sharing constructs. The #pragma omp for is used for loops. It is a type of data parallelism and #pragma omp section is used for sections and can be used to implement a type of functional parallelism. A work sharing construct must be enclosed with in a parallel region in order for the directive to execute in parallel.

4. PARALLELIZING APRIORI ON DUAL CORE USING OPENMP

Our proposed algorithm [18] for parallelizing apriori on dual core is based on the count distribution algorithm proposed by Agrawal and Shafer [4]. This method follows a data parallel strategy and runs in two modes- sequential and parallel. We have partitioned the database into that many horizontal partitions that are equal to number of threads. Each thread will find the local count of all the candidate itemsets in that partition. We have used #pragma omp sections clause to parallelly run the threads. After finding the local counts, the algorithm goes into sequential mode to get the global count of all the items. We find the global count by summing up all the local counts. We have compared this count with minimum support count which gives the frequent itemsets; L_k. L_k is joined with itself to get C_{k+1}. Again the same process of finding the local counts and global count is repeated until there is no more frequent itemsets that is L_k is a null set.

4.1 Pseudo code for 2 threads

1.divide the database into 2 partitions.
2.select a minimum support count ,min_sup.
3.SET_ OMP_NUM_THREADS =2
//All the items I1 to I10 in the database, D will be in candidate 1-itemset , C_1.
//For each item [i] in C_1, the following procedure will be followed.

```
4. /* start of parallel code
#pragma omp parallel
#pragma omp sections
{
omp section
{ //partition1
find_count1 (i)
}
omp section
{   //partition2
find_ count2 (i)

}
```

}//The first section calculate the count of items in partition1 and second section calculate the count of items in section2 .

5.count[i]=count1[i]+count2[i]

if count[i]>min_sup ,place item[i] in frequent 1-itemset,L_1.

6. Join L1 with itself to get C2 where the items are of the type (i , j).

7.Go to step 4 for finding count of items in c2.

8. Repeat this process until $L_{k=} \phi$.

5. EXPERIMENTAL SETUP

To test the performance of apriori with multithreading on multi core processor , we have used Intel Pentium Dual-core with processor speed 1.6GHz and 3GB RAM .To test how

the performance varies with multiple threads, 2 ,3 and 4 threads are created at the time of parallelization. To get openMP compatibility, we have used Fedora 9 Linux (Kernel 2.6.25-14, Red Hat nash version 6.0.52) equipped with GNU C++ (*gcc* version 4.3) for our experimentation. Different randomly generated datasets with 1 to 10lakh records consisting of any items from I1 to I10 are used. Our algorithm is also tested with the standard accidents dataset from UCI repository [28]. We have used all 3,40,183 transactions but only 1 to 10 items of the accident dataset for testing purpose. The experimentation is done at different support counts.

6. EXPERIMENTAL WORK

First the algorithm is run in sequential mode. Then the algorithm is run in parallel mode with 2, 3 and 4 threads. This process is repeated for all the datasets and support counts. Run time, user time and system time are noted down using the time command of Linux. Efficiency is calculated using the run time values with the following formula :

$$E(n) = \frac{Execution time\ using\ one\ processor}{N \times Execution time\ using\ N\ processors} = \frac{t_s}{Nt_n}$$

Where t_s is the time for sequential execution,

t_n is time for parallel execution

and n is the number of processors (3)

7. EXPERIMENTAL RESULTS

7.1 Efficiency of apriori on dual core using different threads

From our experimentation, we have observed the following points:

- The efficiency of parallelizing the algorithm on dual core is more using two threads than using three or four threads.
- For any dataset, efficiency is decreasing as the support count is increasing. So, our implementation is more scalable at lower support counts compared to higher support counts (Fig 1 to Fig 11)

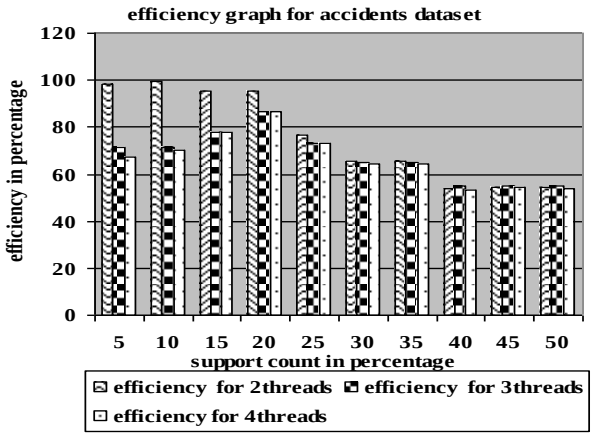

Figure 1. Efficiency graph of accidents dataset for different threads at different support counts

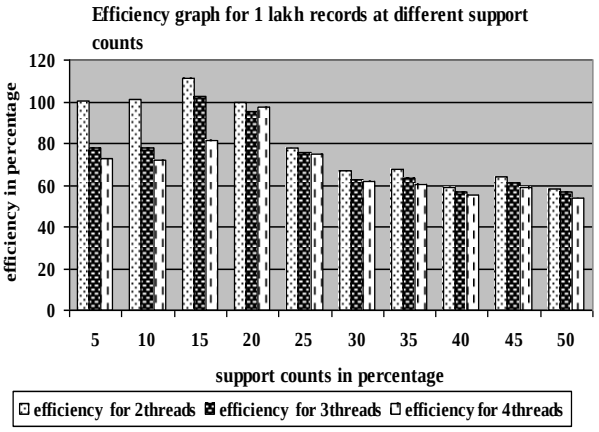

Figure 2. Efficiency graph of 1 lakh dataset for different threads at different support counts

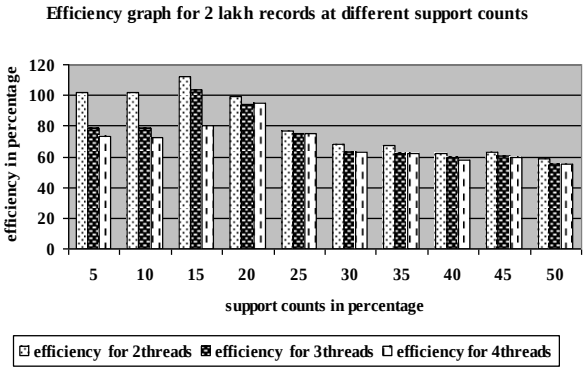

Figure 3. Efficiency graph of 2 lakh dataset for different threads at different support counts

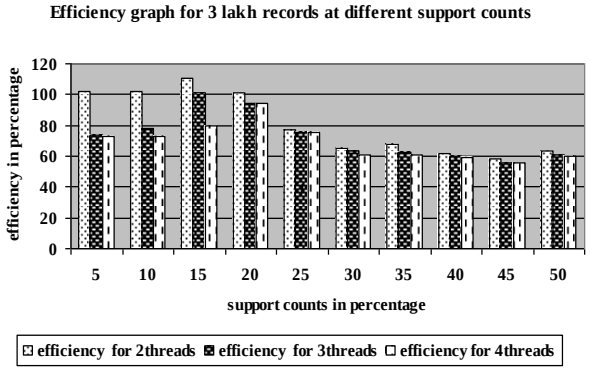

Figure 4. Efficiency graph of 3 lakh dataset for different threads at different support counts

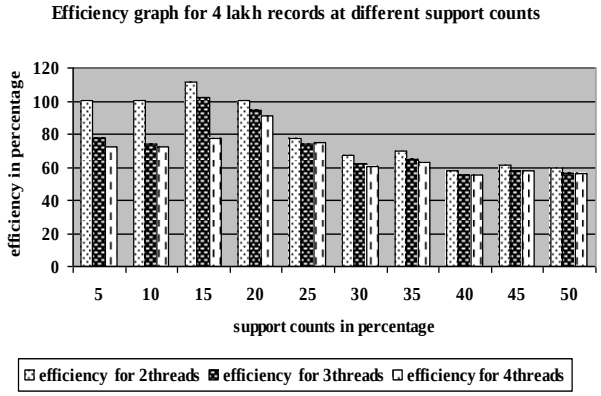

Figure 5. Efficiency graph of 4 lakh dataset for different threads at different support counts

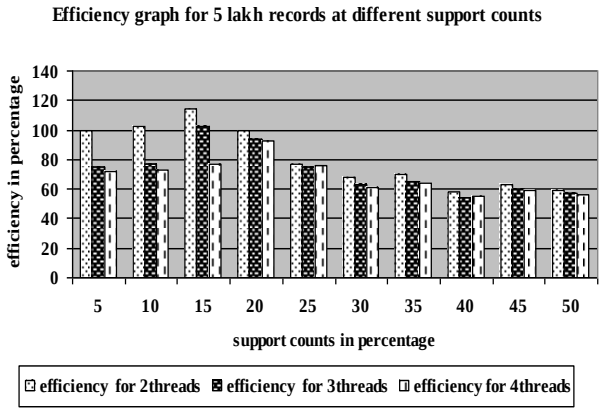

Figure 6. Efficiency graph of 5 lakh dataset for different threads at different support counts

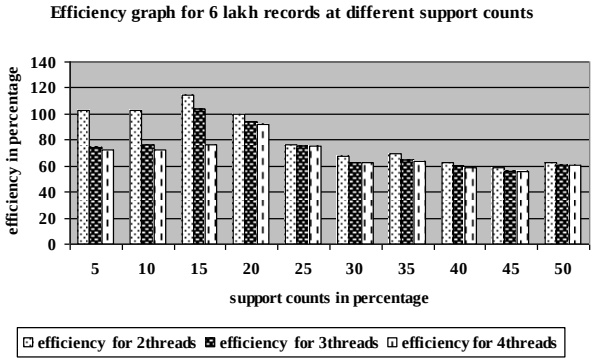

Figure 7. Efficiency graph of lakh dataset for different threads

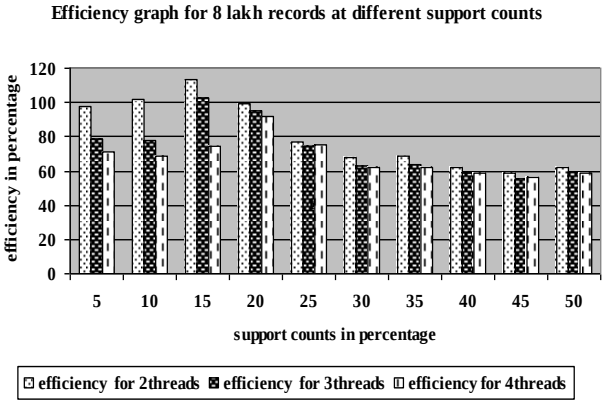

Figure 9. Efficiency graph of 8 lakh dataset for different threads at different support counts

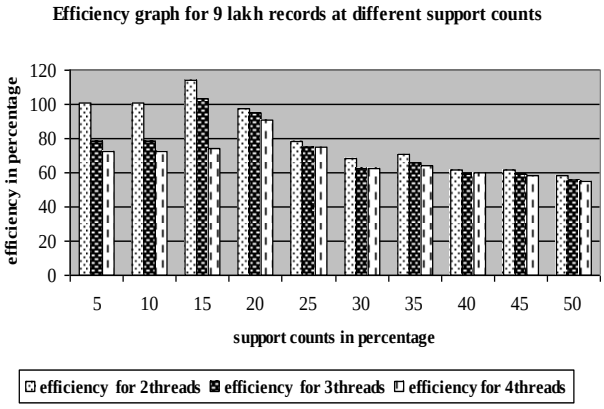

Figure 10. Efficiency graph of 9 lakh dataset for different threads at different support counts

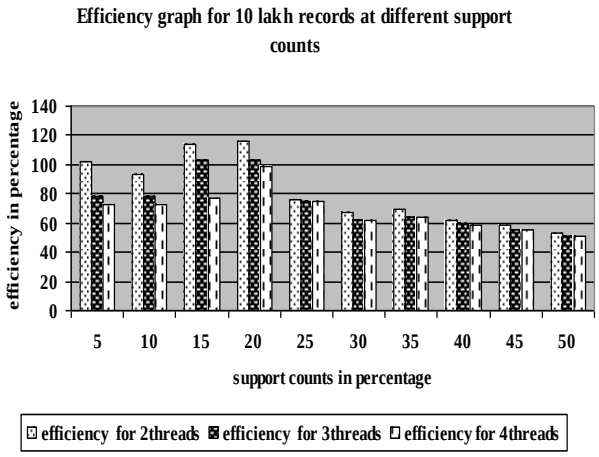

Figure 11. Efficiency graph of 10 lakh dataset for different threads at different support counts

7.2 Comparing the sequential versus parallel results of real time, user time and system time

We have derived the following points from real time , user time and system time values:

1. We can observe the performance benefit in real time with parallel implementation. We can have more benefit when the number of threads is equal to two and it is decreasing by increasing the number of threads. (Fig 12 to Fig 15).

2. We cannot find much difference in user time with parallelization on dual core using different threads compared to sequential execution. (Fig 16 to fig 18)

3.System time increases with parallel processing compared to sequential execution. It is increasing as the number of threads are increasing (Fig 19 to Fig 21).

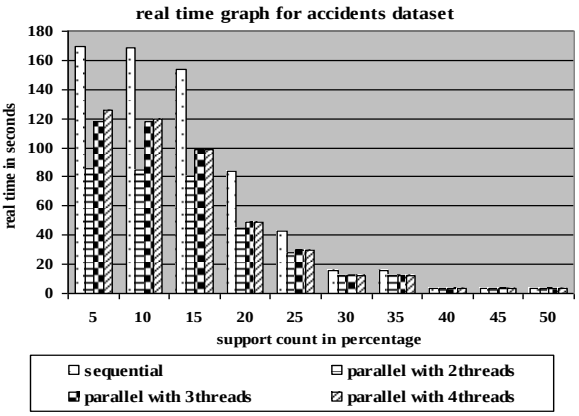

Figure 12. Real time graph for accident dataset at different support counts using sequential and 2, 3, 4 threads

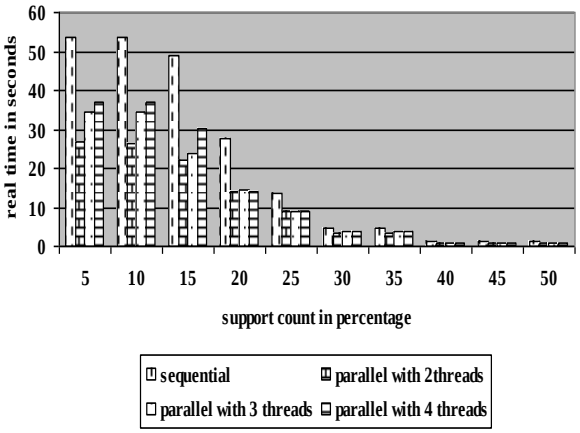

Figure 13. Real time graph for 1lakh dataset at different support counts using sequential and 2, 3, 4 threads

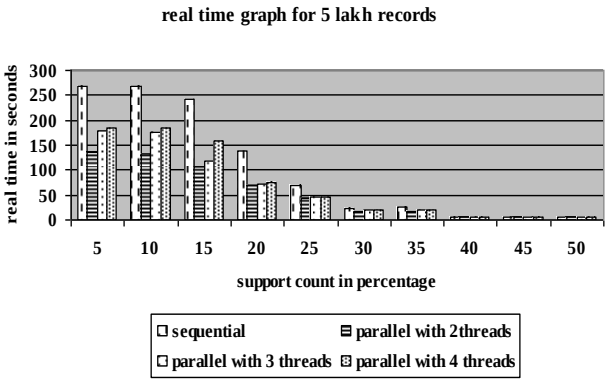

Figure 14. Real time graph for 5 lakh dataset at different support counts using sequential and 2, 3, 4 threads

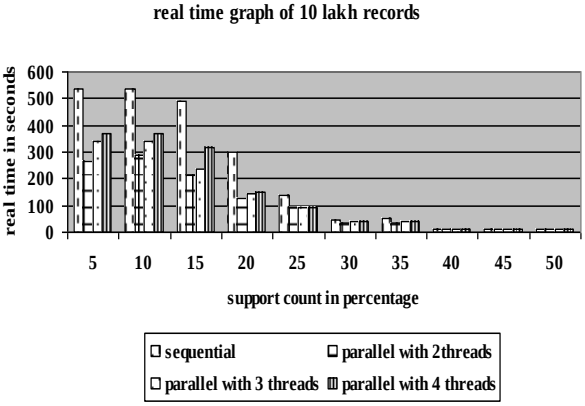

Figure 15. Real time graph for 10 lakh dataset at different support counts using sequential and 2, 3, 4 threads

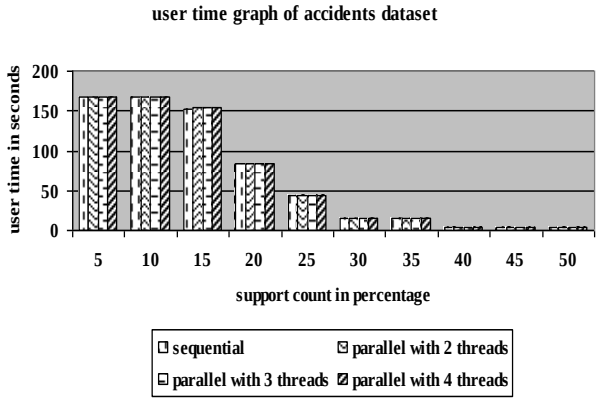

Figure 16. User time graph for accidents dataset at different support counts using sequential and 2, 3, 4 threads

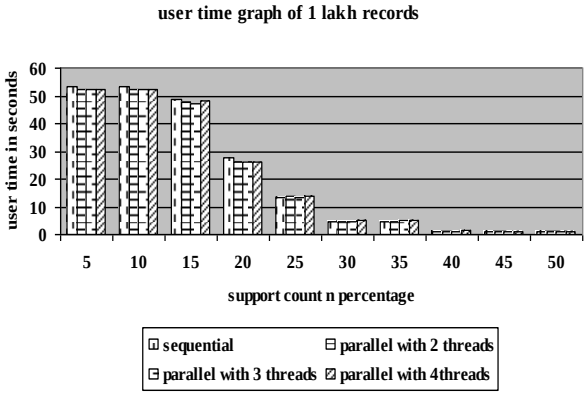

Figure 17. User time graph for 1 lakh dataset at different support counts using sequential and 2, 3, 4 threads

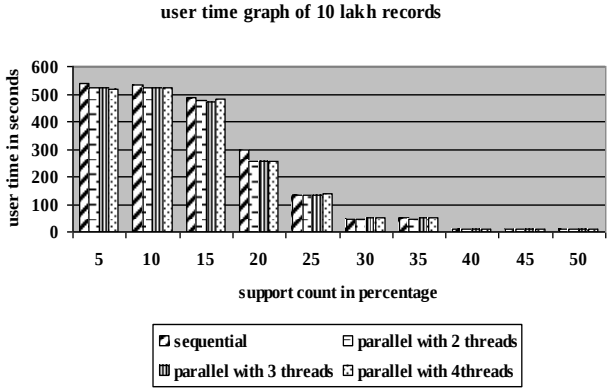

Figure 18. User time graph for 10 lakh dataset at different support counts using sequential and 2, 3, 4 threads

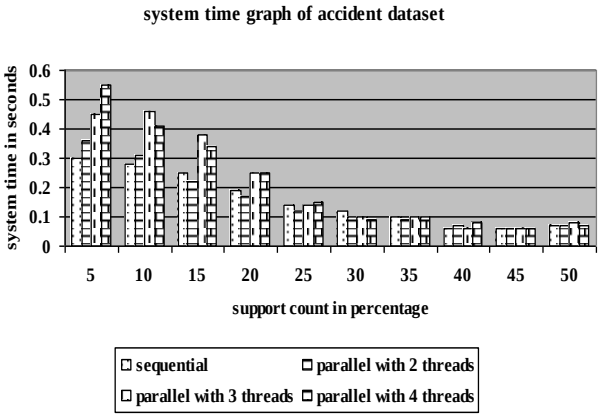

Figure 19. System time graph for accidents dataset at different support counts using sequential and 2, 3, 4 threads

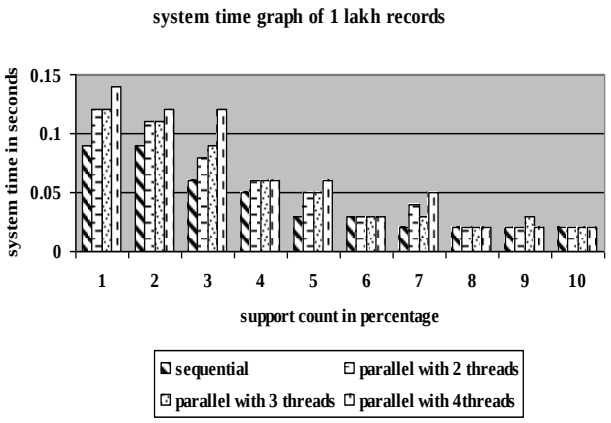

Figure 20. System time graph for 1 lakh dataset at different support counts using sequential and 2, 3, 4 threads

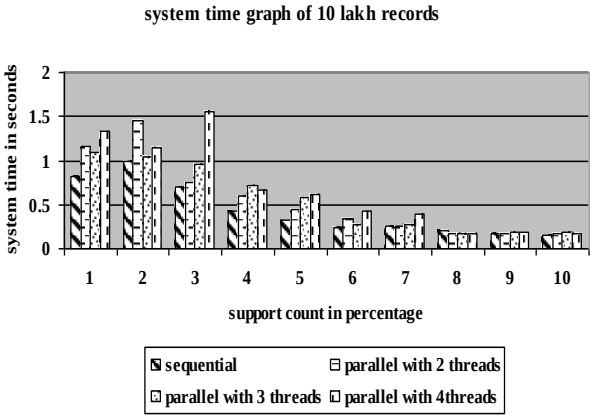

Figure 21. System time graph for 10 lakh dataset at different support counts using sequential and 2, 3, 4 threads

8. CONCLUSIONS AND FUTURE WORK

Parallelization of apriori is implemented on dual core using openMp in Linux environment. The efficiency of the algorithm is measured by changing the number of threads. Efficiency of the algorithm is more on dual core with 2 threads than using 3 or 4 threads. At lower support counts 100% efficiency is observed with 2 threads and it is decreased with the increase of support count and number of threads. Real time ,user time and system time is also compared on dual core for different databases with different support counts using different threads. It is observed that the real time decreased with parallelization than serial execution and it is increased with the increase of number of threads. There is only a slight reduction in the user time with parallelization compared to sequential execution but the benefit of parallelization is not observed in user time, system time as in the case of real time and the system time is slightly increased in parallel execution. In our future work, we run this algorithm on quad core and compare the performance of this algorithm on dual core and quad core and we also want to implement the algorithm using memory mapping concept.

9. REFERENCES

[1] Jiawei Han and Micheline Kamber, Data Mining concepts and Techniques, Morgan Kaufmann Publishers, San Francisco 2006.

[2] Fayyad, Usama,Gregory Piatetsky-Shapiro, and Padhraic Smyth, From Data Mining to Knowledge Discovery in Databases, AI Magazine Volume 17 Number 3(1996).

[3] R Agrawal, T Imielinski, A Swami, "Mining association rules between sets of items in large databases", In: Proceedings of the 1993ACM-SIGMOD international conference on management of data (SIGMOD'93), Washington, DC, pp 207–216.

[4] R. Agrawal and J. Shafer, "Parallel mining of association rules", IEEE Trans. Knowl. Data Eng., vol. 8, pp. 962–969, Dec. 1996

[5] M.J. Zaki, "Parallel and distributed association mining:A survey", IEEE Concur, vol. 7, pp. 14–25, Dec. 1997.

[6] JS Park, MS Chen MS, PS Yu, "Efficient parallel mining for association rules", In: Proceeding of the 4th international conference on information and knowledge management, Baltimore, MD, 1995, pp 31–36.

[7] R Agrawal , JC Shafer , "Parallel mining of association rules: design, implementation, and experience", IEEE Trans Knowl Data Eng 8:962–969 1996.

[8] DW Cheung , J Han , V Ng , A Fu , Y Fu , "A fast distributed algorithm for mining association rules", In: Proceeding of the 1996 international conference on parallel and distributed information systems, Miami Beach, FL, 1996 , pp 31–44 .

[9] O. R Zaiane, M. El-Hajj, and P. Lu, "Fast parallel association rule mining without candidacy generation", in Proc. ICDM, 2001, [Online].Available: citeseer.ist.psu.edu/474 621.html, pp. 665–668.

[10] Herb Sutter, "The Free Lunch Is Over A Fundamental Turn Toward Concurrency in Software", This article appeared in Dr. Dobb's Journal, 30(3), March 2005.

[11] N.Karmakar, "The New Trend in processor Making Multi-Core Architecture", www.scribd.com, 15th may2011.

[12] R Agrawal, R Srikant, "Fast algorithms for mining association rules", In: Proceedings of the 1994 international conference on very large data bases (VLDB'94), Santiago, Chile, pp 487–499.

[13] Jiawei Han, Hong Cheng, Dong Xin, Xifeng Yan, "Frequent pattern mining: current status and future directions", In the Journal of Data Min Knowl Disc (2007) 15:55–86,Springer Science+ Business Media, LLC 2007.

[14] J Han , J Pei , Y Yin , "Mining frequent patterns without candidate generation", In: Proceeding of the 2000 ACM-SIGMOD international conference on management of data (SIGMOD'00),Dallas, TX, pp 1–12.

[15] MJ Zaki , "Scalable algorithms for association mining", IEEE Trans Knowl Data Eng 12:372–390, 2000.

[16] Shirish Tatikonda, Srinivasan Parthasarathy, "Mining Tree Structured Data on Multicore Systems", VLDB '08, August 2430, 2008, Auckland, New Zealand.

[17] Li Liu, Eric Li, Yimin Zhang, Zhizhong Tang, "Optimization of Frequent Itemset Mining on Multiple-Core Processor", VLDB '07, September 23-28, 2007, Vienna, Austria.

[18] T Anuradha , R Satya Prasad ,S.N. Tirumalarao, "Parallelizing Apriori on Dual Core using OpenMP" , International Journal of Computer Applications 43(24):33-39, April 2012.

[19] Amdahl, Gene, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings (30): 483–485, 1967.

[20] "Parallel Computing", CFD Online. [Online]. Available: http://www.cfdonline.com/Wiki/Parallel_computing. [Accessed: May24, 2012].

[21] "Multi-core Processor", From wikipedia ,the free encyclopedia.Available:en.wikipedia.org/wiki/Multi-core_processor [Accessed: May 24,2012.

[22] Jernej Barbic, "Multi-core architectures", ppt

[23] www-bcf.usc.edu/~jbarbic/multi-core-15213-sp07.ppt.

[24] JOHN FRUEHE, "Planning Considerations for Multicore Processor Technology", Available:www.cefriel.it/brandole/pc/14-dell-multocore.pdf.

[25] "OpenMP Architecture, OpenMP C and C++ Application Program Interface", Copyright © 1997-2002 OpenMP Architecture Review Board.http://www.openmp.org/.

[26] Kent Milfeld, "Introduction to Programming with OpenMP", September 12th 2011, TACC.

[27] Ruud van der pas, "An Overview of OpenMP", NTU Talk January 14 2009.

[28] S.N.Tirumala Rao, E.V.Prasad, N.B.Venkateswarlu, "A Critical Performance Study of Memory Mapping on Multi-Core Processors: An Experiment with k-means Algorithm with Large Data Mining Data Sets", International Journal of Computer Applications number 9:211-358, 2010.

[29] K Geurts, G Wets, T. Brijs and K. Vanhoof, "Profiling High Frequency Accident Locations Using Association Rules", Electronic Proceedings of the 82th Annual Meeting of the Transportation Research Board, Washington, January 12-16, USA, 2003, 18p.