I.J. Information Technology and Computer Science, **2017, 4, 18-23**
Published Online April 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2017.04.03

Improved Parallel Apriori Algorithm for Multi-cores

Swati Rustogi
Banasthali Vidyapith, Rajasthan, India
E-mail: swati.rustogi@gmail.com

Manisha Sharma
Banasthali Vidyapith, Rajasthan, India
E-mail: manishsharma8@gmail.com

Sudha Morwal
Banasthali Vidyapith, Rajasthan, India
E-mail: sudha_morwal@yahoo.co.in

***Abstract*—**Apriori algorithm is one of the most popular data mining techniques, which is used for mining hidden relationship in large data. With parallelism, a large data set can be mined in less amount of time. Apart from the costly distributed systems, a computer supporting multi core environment can be used for applying parallelism. In this paper an improved Apriori algorithm for multi-core environment is proposed.

The main contributions of this paper are:

- An efficient Apriori algorithm that applies data parallelism in multi-core environment by reducing the time taken to count the frequency of candidate item sets.
- The performance of proposed algorithm is evaluated for multiple cores on basis of speedup.
- The performance of the proposed algorithm is compared with the other such parallel algorithm and it shows an improvement by more than 15% preliminary experiment.

***Index Terms*—**Multi-core, data mining, parallelism, Apriori.

I. Introduction

Data mining comprises of many techniques, out of which association rule mining is very popular. Association mining involves finding out hidden relationships It has been used in many diverse areas like searching for relevant articles from a document repository [Ji, Y., Ying, H., Tran, J., Dews, P., & Massanari, R. M., 2015], predicting adverse drug reactions after taking inputs from social media [Yang, H., & Yang, C. C. ,2015], identifications of diseases [Sengupta, D., Sood, M., Vijayvargia, P., & Naik, P. K. ,2013], emergency management [Fan, B., & Luo, J. ,2013], finding out relation between music genre and regions [Narberth, K., Goienetxea , I., Johnson, C., & Conklin, D. ,2012]. Till now for data mining multiprocessors [Yu, K. M., & Zhou, J., 2010], clouds or grids [Aflori, C., & Craus, M., 2007] have been used Limitations with these systems are 1) increased cost due to increased hardware; 2) Decreased throughput of the system due to time taken in communication between multiple processors. The existing association mining techniques should be enhanced to take advantage of commonly available multi core systems. These systems provide the advantage of reduced cost as they are based on shared memory architecture.

Algorithms have been developed to take the advantage of multi core environment [Yu, K. M., & Wu, S. H., 2011]. In this paper Apriori algorithm has been redesigned to gain better performance in multi-core environment. The paper has been organized as following: In section 2 Apriori algorithm, Apriori algorithm with TID table and parallel versions of Apriori are discussed. In section 3, the proposed algorithm and its working with an example is discussed. In section 4 simulation results are explained using two datasets. In section 5 conclusions of this work and future work is presented.

II. Related Work

There are many algorithms which fall in the category of association mining. Of all the algorithms Apriori and FP growth are the most popular algorithms. First we are going to look into the popular algorithms used for association, namely Apriori and FP-growth. After this we are going to study the parallel implementations of Apriori algorithm.

Apriori: The first serial algorithm of Apriori was proposed in [Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami, 1993]. Main parts of the algorithm are: generating itemsets and calculating their count in the database. The existing itemsets are merged with each other to generate new itemsets. This process is iterative and it is continued till no more new candidate keys can be generated. Also the itemsets, whose count is less than the support, are not considered for candidate generation in any of the iterations.

To generate candidate itemsets, transactions are read from database. The multiple reads of database reduces the efficiency of algorithm. To remove this disadvantage TID table is prepared which maps each item to the transaction Identification (TID) for the transaction in which it is occurring [Yu, K. M., Zhou, J., Hong, T. P., & Zhou, J. L., 2010].

FP-Growth: This algorithm is based on FP-tree data structure [Han, J., Pei, J., Yin, Y., & Mao, R., 2004]. It reduces execution time as it scans transaction database only twice. For applying parallelism Apriori algorithm is more appropriate than FP growth, because tree structures in FP growth are more difficult to balance [Yu, K. M., Zhou, J., Hong, T. P., & Zhou, J. L., 2010].

Parallel versions of Apriori designed so far are based on processors connected in a network or grid. Authors of [Jian, L., Wang, C., Liu, Y., Liang, S., Yi, W., & Shi, Y., 2013] used GPUs for achieving parallelism. Authors of [Lin, M. Y., Lee, P. Y., & Hsueh, S. C., 2012] developed Apriori algorithm based on map reduce. Map Reduce has the disadvantage of heavy disk IOs at every map and reduce operation [Rathee, S., Kaul, M., & Kashyap, A., 2015]. Authors of [Ravi, V. T., & Agrawal, G., 2009] designed a middleware to perform parallel execution of Apriori, K-means and E-M algorithm. Authors in [Ye, Y., & Chiang, C. C., 2006] proposed a parallel algorithm for Apriori. But disadvantage with this algorithm are the multiple database reads required. Authors in [Yu, K. M., & Wu, S. H., 2011] have proposed Apriori algorithm based on multi core architecture. This algorithm aims at reducing the time for candidate generation.

III. Proposed Algorithm

To take advantage of the multicore architecture of computers, the existing algorithm needs to be modified. And for this parallelism has to be applied. To apply parallelism, data or task parallelism can be used [Garg, R., & Mishra, 2011]. Data contains a set of transactions. Each transaction contains a set of items. In this work data parallelism and scatter gather approach [Holmes, D. W., Williams, J. R., & Tilke, P., 2010] has been used. Data parallelism means splitting the data among multiple cores. Then each core will perform same processing on the data assigned to it.

In the proposed algorithm there are three phases: initial, scatter and gather phase. In initial phase, TID table will be generated. In scatter phase, each core performs its task and generates output. In gather phase, the output generated by each core is gathered and further actions are taken.

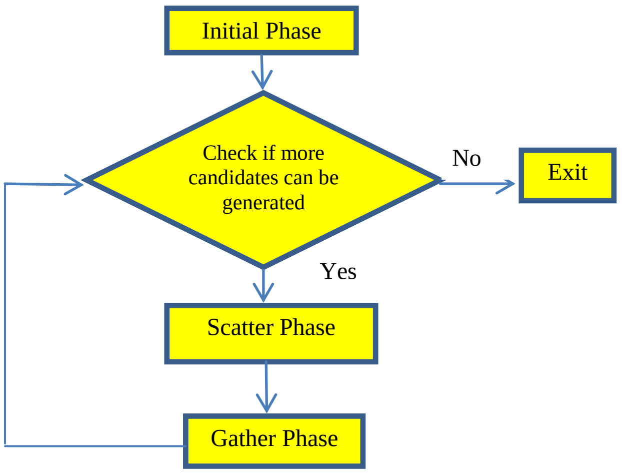

Fig.1. Phases in the proposed algorithm

1. **Initial Phase:** TID table is populated with keys. Table 1 is the database example which will be used to explain the proposed algorithm.

Table 1. Database Example

Transaction ID (TID)	Transaction Detail
1	1,2,5
2	2,4
3	2,3
4	1,2,4
5	1,3
6	2,3
7	1,3
8	1,2,3,5
9	1,2,3

TID table is populated with TIDs after scanning database. For this database is divided equally among all the cores, so that TID table is populated in less time. Table 2 displays the contents of TID table after initial phase.

Table 2. TID table after Initial Phase

Item	TID List
1	1,4,5,7,8,9
2	1,2,3,4,6,8
3	3,5,6,7,8,9
4	2,4
5	1,8

TID List of Item 1 contains the transaction IDs in which Item 1 is present.

2. **Scatter Phase:** In scatter phase candidate keys are generated. Each core generates itemsets for equal number of items. After this count is computed for each new generated candidate key. Fig. 2 summarizes the workings of scatter phase.

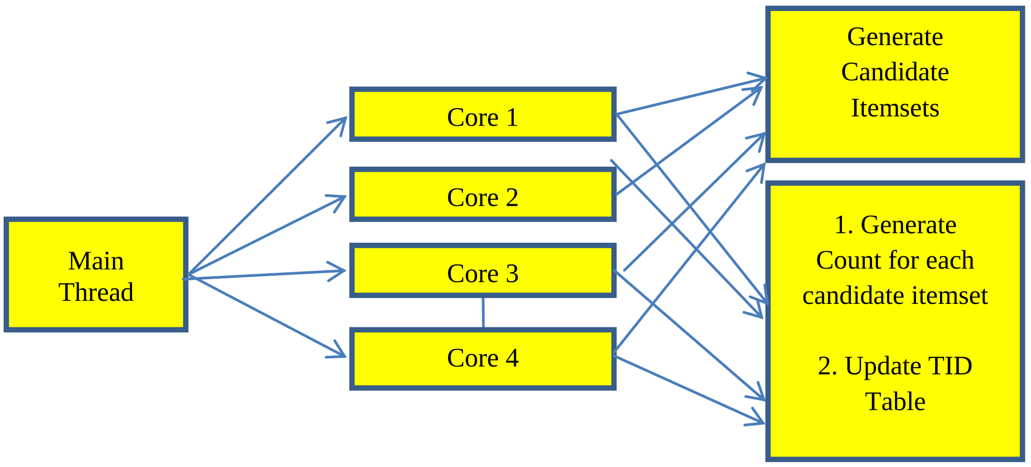
Main Thread
Core 1
Core 2
Core 3
Core 4
Generate Candidate Itemsets
1. Generate Count for each candidate itemset
2. Update TID Table

Fig.2. Scatter Phase

Due to following strategies new itemsets are generated in a lesser amount of time:

1. Each core generates itemsets for equal number of items.
2. To generate new candidate key, existing candidate keys are merged with themselves.
3. To compute count of new candidate key, the key is split into two parts. First part contains the new candidate key except the last item. The second part contains the last item present in the new candidate key. TID list is fetched for both the parts. Intersection of the TID list gives the TID list of the new candidate key. Number of TIDs in the generated TID list gives the count for the new TID list. Given below are the pseudocodes of merge function and compute count function.

```
Merge Function:
For each I1 € Item_index1 do
        For each I2 € Item_index2 do
                CI1 = Candidate Item at index I1
                CI2 = Candidate Item at index I2
                If (CI1[0]==CI2[0] &&

              CI1[n-2] == CI2[n-2] &&
                CI1[n-1] !=CI2[n-1]
                Then
                New_candidate_key = merge of CI1 and CI2
```

```
Compute Count Function:
Split the new candidate key in two parts
Split_key_1 and Split_key_2
TID1 = TID List of Split_key_1
TID2 = TID List of Split_key_2
If (TID1 ⊅ TID2 && TID1⊄TID2)
Then
        Return
Else
        Compare elements of TID1 and TID2
End If
Add new_candidate_key, new_TID_list to TID table
Count of new_candidate_key= number of TIDs in new_TID_list
```

Table 3 is the TID table after candidate-2 itemsets has been generated. After this candidate-3 itemsets will be generated using candidate-2 itemsets.

Table 3. TID table after candiate-2 itemset generation

Item	TID List
1	1,4,5,7,8,9
2	1,2,3,4,6,8
3	3,5,6,7,8,9
4	2,4
5	1,8
1,2	1,8
1,3	5,7,8,9
1,4	4
1,5	1,8
2,3	3,6,8
2,4	2,4
2,5	1,8
3,5	8

Next we show the generation of TID list and count generation for candidate-3 itemset with two examples.

Case **I:** New candidate key 1,2,3
Split_key_1 = 1,2
Split_key_2 = 3
TID1 = 1,8
TID2 = 3, 5, 6, 7, 8, 9
TID1∩TID2 = 8
So TID list of 1, 2, 3 is {8} and count is 1.

Case II: New candidate key 1, 3, 4
Split_key_1 = 1, 3

Split_key_2 = 4
TID1 = 5,7,8,9
TID2 = 2,4

Since TID1 $\not\subset$ TID2 and TID1 $\not\supset$ TID2 so TID list not generated for 1,3,4.

3. **Gather Phase:** The main thread prunes the candidate itemsets. If the count of the candidate key is less than the minimum support then the candidate key is discarded.

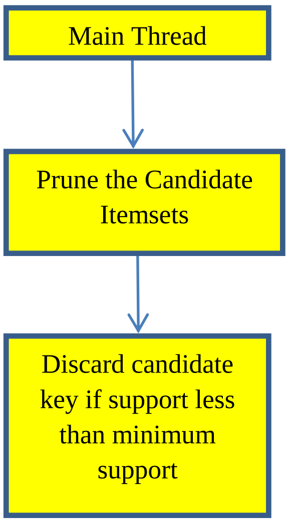
Main Thread
Prune the Candidate Itemsets
Discard candidate key if support less than minimum support

Fig.3. Gather Phase

IV. Experimental Results

Algorithm is implemented in multicore environment and compared with the algorithm given by author in [Yu, K. M., & Wu, S. H., 2011]. Following are the specifications of the system: Intel(R) Core (TM) i3-4030 CPU @ 1.90GHz, RAM 4GB. Two datasets are used to compare the algorithms. Dataset A has been generated by using IBM data generator [17]. Dataset B is a retail dataset [Tank, D. M., 2014].

Table 4 shows the characteristics of the data set A.

Table 4. Dataset Characteristics of Dataset A

Dataset	Transactions	Distinct Items	Average Transaction Size
T10I4D10K	10000	1000	10

Table 5 shows the execution time of MATI algorithm in seconds [Yu, K. M., & Wu, S. H., 2011] and the improved parallel algorithm for dataset A. For this experiment minimum support is varied and then execution time is observed. Fig. 4 is graphical representation of table 5. From the figure it can be observed that execution time is less in case of improved parallel algorithm.

Table 5. Execution time comparison (Dataset A)

Minimum Support	Execution Time (seconds)	
	MATI	Improved parallel algorithm
.01	71.96	38.127
.02	14.76	2.40
.03	2.94	0.69
.04	1.28	.37
.05	0.53	.39
.005	227.5	171.2

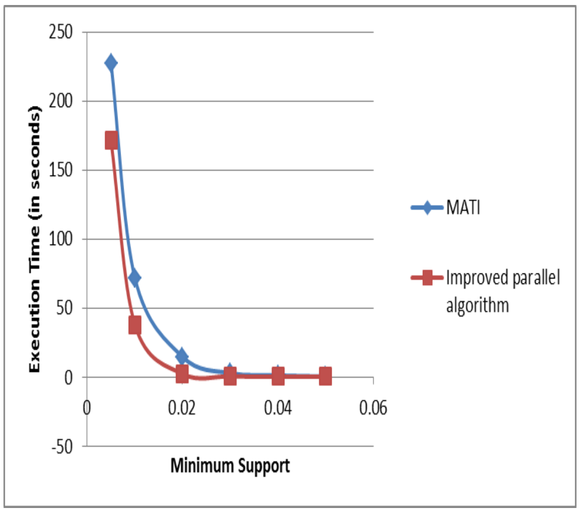

Fig.4. Execution time comparison of MATI and Improved Algorithm (Dataset A)

The algorithm is evaluated for speedup for multiple cores.

Speedup = T(1)/T(n), where T(1) is the time taken in serial execution and T(n) is the time taken by using n processors. It indicates the impact of using multiple processors or cores on the execution time of the algorithm. For computing speedup, dataset is T10I4D10K and support is .03.This experiment is conducted by using serial Apriori algorithm and the improved parallel algorithm. From table 6 it is clear that execution time decreases when using parallel algorithm and speedup increases with increase in number of cores. Fig. 5 is the graphical representation of table 6.

Table 6. Speedup with multiple cores (Dataset A)

No. of cores	T(1)	T(n)	Speedup
1	4.16	1.03	4.03
2	4.04	0.92	4.39
3	3.99	0.73	5.46
4	3.88	0.68	5.70

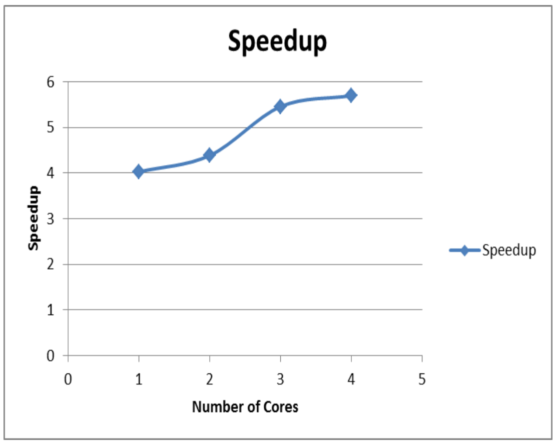

Fig.5. Speedup with multiple cores (Dataset A)

As shown in Fig. 5, speedup increases with increase in number of cores. This indicates that as the number of cores increases the speedup increases.

Table 7 shows the characteristic of dataset B.

Table 7. Dataset Characteristics of Dataset B

Dataset	Transactions	Distinct Items	Average Transaction Size
Retail	5000	200	3

Table 8 shows the execution time of MATI and improved parallel algorithm, for Retail dataset. Fig. 6 is the graphical representation of table 8. Fig. 6 shows that execution time is less in case of the improved parallel algorithm.

Table 8. Execution time comparison (Dataset B)

Minimum Support	Execution Time (seconds)	
	MATI	Improved parallel algorithm
.01	1.47	0.50
.02	0.73	0.31
.03	0.66	0.31
.04	0.65	0.30
.05	0.48	0.27
0.005	2.84	2.02

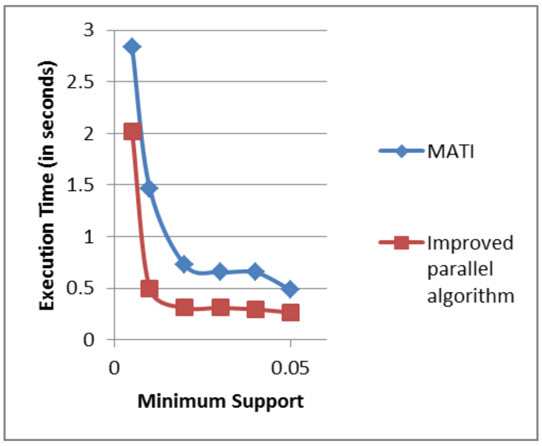

Fig.6. Execution time comparison of MATI and Improved Algorithm (Dataset B)

Table 9 and fig. 7 shows speedup for dataset B. Experiment is conducted with minimum support as 0.03. As per the table and figure, it is evident that speedup increases with increase in number of cores.

Table 9. Speedup with multiple cores (Dataset B)

No. of cores	T(1)	T(n)	Speedup
1	0.799	0.415	1.92
2	0.749	0.384	1.95
3	0.697	0.341	2.04
4	0.657	0.312	2.10

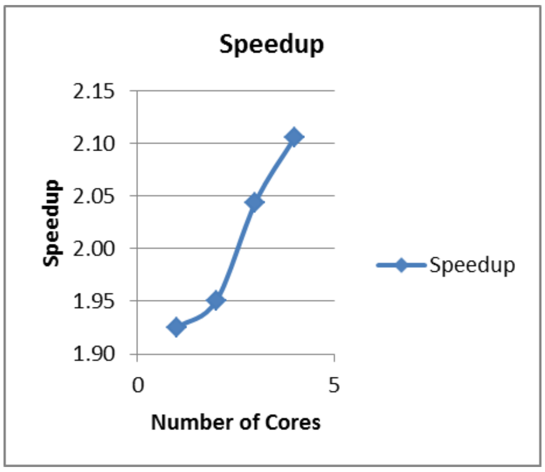

Fig.7. Speedup with multiple cores (Dataset B)

V. Conclusion and Future Work

Apriori algorithm is one of the popular algorithms used for association mining. The proposed algorithm reduces the execution time of the algorithm by using data parallelism in multi-core environment, which is evident from the experiment results. It has been observed that the proposed algorithm executes in a lesser time in comparison to other such parallel algorithm and serial algorithm. Also it has been observed that speedup increases with increase in number of cores. Thus it can be concluded that with increase in number of cores, execution time of the proposed algorithm will decrease and speedup will increase.

Although an efficient Apriori algorithm for multi-core has been proposed, it can be analyzed from the point of view of load on the cores. Also the load can be managed dynamically, while the algorithm is in execution.

References

[1] Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases." ACM SIGMOD Record 22.2 (1993), 207-216.
[2] Aflori, C., & Craus, M. (2007). Grid implementation of the Apriori algorithm. Advances in engineering software, 38(5), 295-300.
[3] Asha, P., & Jebarajan, T. (2015). Association Rule Mining and Refinement Using Shared Memory Multiprocessor Environment. In Artificial Intelligence and

Evolutionary Algorithms in Engineering Systems (pp. 105-117). Springer India.

[4] Fan, B., & Luo, J. (2013). Spatially enabled emergency event analysis using a multi-level association rule mining method. Natural hazards, 67(2), 239-260.

[5] Holmes, D. W., Williams, J. R., & Tilke, P. (2010). An events based algorithm for distributing concurrent tasks on multi-core architectures. Computer Physics Communications, 181(2), 341-354.

[6] Garg, R., & Mishra, P. K. (2011). Exploiting parallelism in association rule mining algorithms. International Journal of Advanced Technology, 2, 222-232.

[7] Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data mining and knowledge discovery, 8(1), 53-87.

[8] Ji, Y., Ying, H., Tran, J., Dews, P., & Massanari, R. M. (2015, November). Integrating association mining into relevance feedback for biomedical literature search. In International Conference on Bioinformatics and Biomedicine (BIBM), 2015 IEEE (pp. 531-536). IEEE.

[9] Jian, L., Wang, C., Liu, Y., Liang, S., Yi, W., & Shi, Y. (2013). Parallel data mining techniques on graphics processing unit with compute unified device architecture (CUDA). The Journal of Supercomputing, 64(3), 942-967.

[10] Neubarth, K., Goienetxea, I., Johnson, C., & Conklin, D. (2012). Association Mining of Folk Music Genres and Toponyms. In International Society for Music Information Retrieval Conference, Vol. 2012, p. 13th.

[11] Ravi, V. T., & Agrawal, G. (2009, May). Performance issues in parallelizing data-intensive applications on a multi-core cluster. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (pp. 308-315). IEEE Computer Society.

[12] Rathee, S., Kaul, M., & Kashyap, A. (2015, October). R-Apriori: an efficient apriori based algorithm on spark. In Proceedings of the 8th Workshop on Ph.D. Workshop in Information and Knowledge Management (pp. 27-34), ACM.

[13] Sengupta, D., Sood, M., Vijayvargia, P., & Naik, P. K. (2013). Association rule mining based study for identification of clinical parameters akin to occurrence of brain tumor. Bioinformation, 9(11), 555-9.

[14] Tank, D. M. (2014). Improved Apriori Algorithm for Mining Association Rules. International Journal of Information Technology and Computer Science (IJITCS), 6(7), 15.

[15] Tanna, P., & Ghodasara, Y. (2014). Using Apriori with WEKA for Frequent Pattern Mining. arXiv preprint arXiv:1406.7371.

[16] Yang, H., & Yang, C. C. (2015). Using Health-Consumer-Contributed Data to Detect Adverse Drug Reactions by Association Mining with Temporal Analysis. ACM Transactions on Intelligent Systems and Technology (TIST), 6(4), 55.

[17] Ye, Y., & Chiang, C. C. (2006, August). A parallel apriori algorithm for frequent itemsets mining. In Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06) (pp. 87-94). IEEE.

[18] Yu, K. M., Zhou, J., Hong, T. P., & Zhou, J. L. (2010). A load-balanced distributed parallel mining algorithm. Expert Systems with Applications, 37(3), 2459-2464.

[19] Yu, K. M., & Zhou, J. (2010). Parallel TID-based frequent pattern mining algorithm on a PC Cluster and grid computing system. Expert Systems with Applications, 37(3), 2486-2494.

[20] Yu, K. M., & Wu, S. H. (2011, November). An efficient load balancing multi-core frequent patterns mining algorithm. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on (pp. 1408-1412). IEEE.

[21] http://fimi.ua.ac.be/data/

Authors' Profiles

Swati Rustogi is currently pursuing Ph.D. (CS) from Banasthali Vidyapith, Rajasthan, India. She is working as an Assistant Professor with Amity University, Noida. She completed her graduation from Delhi University and MCA from Birla Institute of Applied Sciences.

She has seven years of teaching and research experience and five years of industry experience. In research she has been focusing on distributed systems.

Dr. Manisha is currently working as Associate Professor (Computer Science), Banasthali University. She obtained her Ph. D. degree in Computer Science from Banasthali University. She is life member of CSI and many academic bodies of various universities.

She has presented many papers in international and national conferences and has many publications in journals to her credit. Current research interests of Dr. Manisha are Artificial Intelligence, Intelligent Systems, Data Mining, and Natural Language Processing. She has more than 17 years of teaching experience. She is an active research supervisor and guiding Ph. D. students in the area of Artificial Intelligence and Data Mining.

Dr. Sudha Morwal: Sudha Morwal has done research in the field of Natural Language Processing. Currently she is working as Associate Professor in the Department of Computer Science at Banasthali Vidyapith (Rajasthan), India.

She is NET qualified and has done M.Tech (Computer Science) and M.Sc. (Computer Science) from Banasthali University (Rajasthan), India.

How to cite this paper: Swati Rustogi, Manisha Sharma, Sudha Morwal,"Improved Parallel Apriori Algorithm for Multi-cores", International Journal of Information Technology and Computer Science(IJITCS), Vol.9, No.4, pp.18-23, 2017. DOI: 10.5815/ijitcs.2017.04.03