IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-10 E-ISSN: 2347-2693

# Hybrid Parallel Programming Using Locks and STM

**Ryan Saptarshi Ray [1*], Parama Bhaumik [2], Utpal Kumar Ray [3]**

[1*]Dept. of Information Technology, Jadavpur University, Kolkata, India
[2] Dept. of Information Technology, Jadavpur University, Kolkata, India
[3] Dept. of Information Technology, Jadavpur University, Kolkata, India

[*]*Corresponding Author: ryan.ray@rediffmail.com, Tel.: 9831520613*

**Available online at: www.ijcseonline.org**

Received: 17/Sep/2017, Revised: 30/Sep/2017, Accepted: 13/Oct/2017, Published: 30/Oct/2017

***Abstract*—** Software Transactional Memory (STM) is a new alternative approach to locks which solves the problem of synchronization in parallel programs. In STM users have to identify the critical sections in the program and enclose them within transactions by using appropriate STM function calls. Then STM automatically by its internal constructs ensures synchronization in the program. This paper shows how to solve the problem of synchronization in parallel programs by using a hybrid programming approach using both locks and STM. Locks use pessimistic approach to solve the problem of synchronization in parallel programs. STM uses optimistic approach to solve the problem of synchronization in parallel programs. Both the optimistic and pessimistic approaches have some advantages and disadvantages. The disadvantage of optimistic approach is that transactions are aborted when validation cannot be done. This approach works well when there are no conflicts (hence the term optimistic) but wastes work when there are conflicts. Aborting of transactions is a severe problem when the transactions are long and interactive. The disadvantage of pessimistic approach is that large number of locks in the program will lead to very slow execution speed which may cancel out the gains made by solving the problem in parallel. The hybrid approach combines the advantages of the optimistic and pessimistic approaches removing their disadvantages without any degradation of performance.

***Keywords*—** Multiprocessing, Parallel Processing, Locks, Software Transactional Memory, Hybrid Parallel Programming

## I. INTRODUCTION

Ensuring synchronization is a very important problem in parallel programs. Currently locks are used to solve this problem. Locks use pessimistic approach. Software transactional memory (STM) is a promising alternative approach for parallel computation which does not have most of the limitations of the locks-based approach. STM uses optimistic approach. Both the optimistic and pessimistic approaches have some disadvantages.

In the pessimistic approach it is always assumed that results will surely be erroneous if multiple threads execute the critical section simultaneously which may not always be the case. The more the number of critical sections in a program the more will be the number of locks. Large number of locks in the program will lead to very slow execution speed which may cancel out the gains made by solving the problem in parallel **[1]**.

The main disadvantage of the optimistic approach is that in those types of problems where simultaneous execution of critical sections by multiple threads leads to inconsistency the same critical sections have to be executed again and again until the values are consistent. This may lead to drastic degradation of performance and may overset all the gains achieved by parallel execution **[2]**.

In this paper we present a hybrid approach using both locks and STM to solve the problem of synchronization in parallel programs. The programming example considered is finding out the minimum element in an array. It is a small prototype of a real-life example in which different areas of a database are accessed in real time in parallel.

In the example which we have used the simultaneous execution of multiple critical sections by multiple threads will lead to inconsistency. Thus in this case the use of pessimistic approach is more advantageous. When the optimistic approach (STM) was used the transactions were aborted a large number of times as simultaneous execution of multiple critical sections by multiple threads was frequently leading to inconsistency. Thus the same critical sections were executed again and again. This resulted in large execution time (24 sec). The disadvantage of optimistic approach is that transactions are aborted when validation cannot be done. This approach works well when there are no conflicts (hence the term optimistic) but wastes work when there are conflicts. Aborting of transactions is a severe problem when the transactions are long and interactive **[3].** When the pessimistic approach (locks) was used the execution time was 5 seconds. In the hybrid approach the pessimistic approach was used in one half of the array and optimistic approach in the other half. The execution time was 5 seconds. Thus we can say that the hybrid approach combines the advantages of the optimistic and pessimistic approaches removing their disadvantages without any degradation of performance.

In the ideal case hybrid approach may also lead to

 

further performance improvement. This is because in our example we have used optimistic approach and pessimistic approach in one half of the array each. But in the ideal case optimistic and pessimistic approaches are applied where each is advantageous. Pessimistic approach is applied in those portions of the program where simultaneous execution of multiple critical sections by multiple threads may lead to inconsistency and optimistic approaches where simultaneous execution of multiple critical sections by multiple threads may not lead to inconsistency.

Earlier also hybrid programming was used in different applications which is described in detail in Section II (related work) **[4]**, **[5]**, **[6]**, **[7]**, **[8]**. In our work we have used STM for the optimistic approach. That is we have shown a combination of locks and STM which has not been used earlier.

In this paper Section III finds out minimum element in an array using locks (pessimistic approach). Section IV finds out minimum element in an array using STM (optimistic approach). Section V finds out minimum element in an array using hybrid approach (combination of optimistic and pessimistic approaches) using both locks and STM. The performances of the different approaches are compared.

## II. RELATED WORK

In 1992 Philip S. Yu and Daniel M. Dias published a paper entitled "Analysis of Hybrid Concurrency Control Schemes For a High Data Contention Environment" **[4]**. This paper developed a hybrid approach for transaction processing in databases exploiting the property that with sufficient buffer, data blocks referenced by aborted transactions can continue to be kept in memory and be available for access during rerun, thus greatly reducing the abort probability during rerun. Also in 1992 Sang H. Son and Juhnyoung Lee published a paper entitled "A New Approach to Real-Time Transaction Scheduling" **[5]**. This paper presented new real-time transaction scheduling algorithms which employed a hybrid approach. The protocols made use of a new conflict resolution scheme called dynamic adjustment of serialization order, which supported priority-driven scheduling, and avoided unnecessary aborts. Then in 1997 Sung Ho Cho, JongMin Lee, Chong-Sun Hwang and WonGyu Lee published a paper entitled "Hybrid Concurrency Control for Mobile Computing" **[6]**. This paper proposed a hybrid concurrency control scheme which alleviated the effects of frequent disconnections and limitation of bandwidth in mobile computing environments. In 2003 SungHo Cho published a paper entitled "A Hybrid Concurrency Control with Deadlock-free Approach" **[7]**. This paper suggested an efficient hybrid concurrency control scheme that used a deadlock-free approach and ensured that transactions were re-started at most once. After 2009 Jan Lindstrom published a paper entitled "Hybrid Concurrency Control Method in Firm Real-Time Databases" **[8]**. This paper proposed a concurrency control method where transactions accessed tables by pessimistic concurrency control, optimistic concurrency control or nocheck concurrency control based on application needs.

In our work we have also used hybrid concurrency control to simultaneously access the elements of an array. The new thing which our work has done is that for the optimistic approach we have used STM. That is we have shown a combination of locks and STM which has not been used earlier.

## III. SOLUTION USING LOCKS

Synchronization of multiple threads is the hardest problem that has to be overcome when writing parallel programs. Two or more threads can try to access the same locations in memory. So if careful measures are not taken the results will be erroneous. If multiple threads try to modify the same variable(s) at the same time, data can become corrupt. Nowadays this problem is solved by using locks in parallel programs.

In programming with locks programmers first have to identify the critical sections in the program. Critical section is a block of code that contains variable(s) that may be accessed simultaneously by two or more threads. Then when a thread will try to enter a critical section, it has to first acquire that section's lock. If another thread is already holding the lock, the former thread must wait until the lock-holding thread releases the lock, which it does when it leaves the critical section .

This type of approach is called pessimistic approach. In this approach it is always assumed that results will surely be erroneous if multiple threads execute the critical section simultaneously which may not always be the case. The more the number of critical sections in a program the more will be the number of locks. Large number of locks in the program will lead to very slow execution speed which may cancel out the gains made by solving the problem in parallel.

However a significant advantage of the pessimistic approach is that it ensures that the same critical section never has to be executed again and again which may be the case with optimistic approach. This approach is advantageous in situations where really simultaneous execution of critical sections by multiple threads leads to erroneous results.

The following code shows a parallel program using threads and locks which finds out the minimum element in an array. Thus pessimistic approach is used to solve the problem of synchronization in the program.

 

```
#include<stdlib.h>
#include<stdio.h>
#include<pthread.h>
#include<sys/time.h>
#include<time.h>

#define ARRAY_SIZE 100000000
#define NUM_THREAD 2
unsigned long n=ARRAY_SIZE;
unsigned char arr[ARRAY_SIZE],global_min=255;
void *thread_search_min1( );
unsigned char local_min_array[NUM_THREAD],global_min_lock_check=255;
pthread_mutex_t mutex1=PTHREAD_MUTEX_INITIALIZER;
main(int argc,char ** argv)
    {
    pthread_mutex_init(&mutex1,NULL);
    pthread_t tid[NUM_THREAD];
    int i,k,p;
    struct timeval ini_tv,final_tv;
    int arg_to_be_passed[NUM_THREAD];
    arr[0]=8;
    arr[1]=16;
    arr[2]=3;
    arr[3]=5;
    arr[4]=23;
    arr[5]=7;
    printf("Enter number of times code is to be executed\n");
    scanf("%d",&p);
    gettimeofday(&ini_tv,NULL);
    for(k=0;k<p;k++)
    {
    global_min=arr[0];
    for(i=0;i<NUM_THREAD;i++)
    {
    arg_to_be_passed[i]=i;
    pthread_create(&tid[i],NULL,thread_search_min1,&arg_to_be_passed[i]);
    }
    for(i=0;i<NUM_THREAD;i++)
    {
    pthread_join(tid[i],NULL);
    }
    }
    gettimeofday(&final_tv,NULL);
     printf("\nTotal Time Taken = %ld\n", final_tv.tv_sec - ini_tv.tv_sec);
    return 0;
    }
    void *thread_search_min1(int * num_ptr)
    {
    unsigned long j;
    unsigned char byte_under_stm;
    int num,*number_ptr;
    number_ptr=num_ptr;
    num=*number_ptr;
    local_min_array[num]=255;
    for((j=(((num*n)/NUM_THREAD)));j<(((num+1)*n)/NUM_THREAD);j++)
    {
    pthread_mutex_lock(&mutex1);
       if(arr[j]<global_min)
       {
          global_min= arr[j];
       }
       pthread_mutex_unlock(&mutex1);
    }

    pthread_exit(0);
                                    }
```

The following statements in the code are used for thread creation:

```
for(i=0;i<NUM_THREAD;i++)
{
arg_to_be_passed[i]=i;
pthread_create(&tid[i],NULL,thread_search_min1,&arg_to_be_passed[i]);
}
```

The following statements ensure that even if any thread completes its execution early it has to wait for completion of all the other threads.

```
for(i=0;i<NUM_THREAD;i++)
{
pthread_join(tid[i],NULL);
}
```

In the thread function the critical section of the program is the portion where the array elements are accessed and the global variable global_min is being updated. It is enclosed using locks in the following fashion:

```
    for((j=(((num*n)/NUM_THREAD)));j<(((num+1)*n)/NUM_THREAD);j++)
    {
    pthread_mutex_lock(&mutex1);
       if(arr[j]<global_min)
       {
          global_min= arr[j];
       }
       pthread_mutex_unlock(&mutex1);
}
```

The lock calls which have been used in the program are described below:-

 

**pthread_mutex_init(&mutex1,NULL)**- It is used for lock initialization.

**pthread_mutex_lock(&mutex1)**- It is used for locking. Any thread which wants to access the critical section has to first acquire the lock on mutex1.

**pthread_mutex_unlock(&mutex1)**- It is used for unlocking.

In this program the critical region is enclosed within locks. So the code faces no synchronization problems.

## IV. SOLUTION USING STM

STM can also be used to solve the synchronization problem. In case of STM also the programmer has to first identify the critical section of the program and then has to enclose it within a transaction. But the difference with locks is that STM uses optimistic approach for synchronization.

Optimistic approach means that STM allows multiple threads to execute the critical section at the same time. After the executions are over the values of the edited variables are checked to see if the values are consistent. If the values are inconsistent then the critical sections of the threads are executed again. Optimistic approach is advantageous and very suitable for those problems where execution of critical sections by multiple threads simultaneously does not lead to any inconsistency. In those cases due to parallel execution of critical sections by multiple threads execution times improve considerably.

But the main disadvantage of the optimistic approach is that in those types of problems where simultaneous execution of critical sections by multiple threads leads to inconsistency the same critical sections have to be executed again and again until the values are consistent. This may lead to drastic degradation of performance and may overset all the gains achieved by parallel execution.

Some code snippets of solution of the parallel program to find out minimum element in an array using pthreads and STM are shown below. The overall structure of the code is same as that of the code using locks shown in Section III. The only difference is instead of locks STM is used in the code for synchronization. The extra header files which must be declared are:-

```
#include <stm.h>
#include<atomic_ops.h>
```

The following macros must also be declared.

```
#define RO                  1
#define RW                  0
#define START(id, ro)       { stm_tx_attr_t _a = {id, ro}; sigjmp_buf *_e = stm_start(&_a); if (_e != NULL) sigsetjmp(*_e, 0)
#define LOAD(addr)          stm_load((stm_word_t *)addr)
#define STORE(addr, value)  stm_store((stm_word_t *)addr, (stm_word_t)value)
#define COMMIT              stm_commit(); }
```

All the operations in the main function must be enclosed within the statements stm_init() and stm_exit().

In the thread function the critical section is identified and enclosed within a transaction in the following manner:

```
for((j=(((num*n)/NUM_THREAD)));j<(((num+1)*n)/NUM_THREAD);j++)
{
START(0,RW);
byte_under_stm=(unsigned char)LOAD(&global_min);
if(arr[j]< byte_under_stm)
{
byte_under_stm=arr[j];
}
STORE(&global_min, byte_under_stm);
COMMIT;
}
```

The STM functions and calls which have been used in the program are described below:

**stm_init** initializes the TinySTM library at the outset. It must be called from the main thread before accessing any other functions of the TinySTM library.

**stm_init_thread** initializes each thread that performs transactions. Each thread that performs transactional operations should call it once before it calls any other function of the TinySTM library. In this program the thread function **thread_search_min1** calls it.

**stm_exit** is the corresponding shutdown function for stm_init. It is used to clean up the TinySTM library. It should be called once from the main thread after all transactional threads have completed execution.

**stm_exit_thread** is the corresponding shutdown function for stm_init_thread. It is used to clean up the transactional thread. It should be called once from each thread that performs transactional operations upon exit. In this program it is used to clean up the thread function **thread_search_min1**.

**START(0,RW)** starts a transaction. In this program the thread function **thread_search_min1** uses it.

 

**COMMIT** closes a transaction. In this program the thread function **thread_search_min1** uses it.

**byte_under_stm=(unsigned char) LOAD(&global_min);** is used to store the value of global_min in byte_under_stm. In this program the thread function **thread_search_min1** uses it.

**STORE(&global_min,byte_under_stm);** is used to store the value of byte_under_stm in global_min. In this program the thread function **thread_search_min1** uses it.

In this program the critical section is enclosed within a transaction using TinySTM which is a type of STM. So the program faces no synchronization problem.

## V. SOLUTION USING HYBRID APPROACH

In this paper we have developed and demonstrated a hybrid approach using locks and STM which is a combination of the optimistic and pessimistic approaches to solve the problem of synchronization in parallel programs. The hybrid approach combines the advantages of the optimistic and pessimistic approaches and does not suffer from their disadvantages.

The problem we have considered here is finding out minimum element in an array. This is a small prototype of a real-life example in which multiple parts of the same database are accessed and modified simultaneously in real time. In our example we have ensured that the elements of one half of the array are accessed in parallel using optimistic approach and the other half using pessimistic approach. The program is shown below. The overall structure of the code is same as that of the code in Section III. The only difference is that the hybrid approach is used in the program.

```
#include<stdlib.h>
#include<stdio.h>
#include<pthread.h>
#include<sys/time.h>
#include<time.h>
#include<atomic_ops.h>
#include<stm.h>

#define RO                    1
#define RW                    0
#define START(id, ro)         { stm_tx_attr_t _a = {id, ro}; sigjmp_buf *_e = stm_start(&_a); if (_e != NULL) sigsetjmp(*_e, 0)
#define LOAD(addr)            stm_load((stm_word_t *)addr)
#define UNITLOAD(addr, timestamp) stm_unit_load((stm_word_t *)addr, (stm_word_t *)timestamp)
#define STORE(addr, value)    stm_store((stm_word_t *)addr, (stm_word_t)value)
#define UNITSTORE(addr, value, timestamp) stm_unit_store((stm_word_t *)addr, (stm_word_t)value, (stm_word_t *)timestamp)
#define COMMIT                stm_commit(); }

#define ARRAY_SIZE 100000000
#define NUM_THREAD 2
unsigned long n=ARRAY_SIZE;
unsigned char arr[ARRAY_SIZE],global_min=255;
void *thread_search_min1( );
unsigned char local_min_array[NUM_THREAD],global_min_lock_check=255;
pthread_mutex_t mutex1=PTHREAD_MUTEX_INITIALIZER;
main(int argc,char ** argv)
  {
  stm_init( );
  pthread_mutex_init(&mutex1,NULL);
  pthread_t tid[NUM_THREAD];
  int i,k,p;
  struct timeval ini_tv,final_tv;
  int arg_to_be_passed[NUM_THREAD];
  arr[0]=8;
  arr[1]=16;
  arr[2]=3;
  arr[3]=5;
  arr[4]=23;
  arr[5]=7;
  printf("Enter number of times code is to be executed\n");
  scanf("%d",&p);
  gettimeofday(&ini_tv,NULL);
  for(k=0;k<p;k++)
  {
  global_min=arr[0];
  for(i=0;i<NUM_THREAD;i++)
  {
  arg_to_be_passed[i]=i;
  pthread_create(&tid[i],NULL,thread_search_min1,&arg_to_be_passed[i]);
  }
  for(i=0;i<NUM_THREAD;i++)
  {
  pthread_join(tid[i],NULL);
  }
  }
  gettimeofday(&final_tv,NULL);
   printf("\nTotal Time Taken = %ld\n", final_tv.tv_sec - ini_tv.tv_sec);
  return 0;
  stm_exit( );
  }
  void *thread_search_min1(int * num_ptr)
  {
```

 

```
    unsigned long j;
    unsigned char byte_under_stm;
    int num,*number_ptr;
    number_ptr=num_ptr;
    num=*number_ptr;
    local_min_array[num]=255;
    stm_init_thread( );
    for((j=(((num*n)/NUM_THREAD)));j<(((num+1)*n)/NU
M_THREAD);j++)
    {
    if(j<(n/2))
    {
    START(0,RW);
    byte_under_stm=(unsigned char)LOAD(&global_min);
    if(arr[j]< byte_under_stm)
    {
    byte_under_stm=arr[j];
    }
    STORE(&global_min, byte_under_stm);
    COMMIT;
    }
    else
    {
    pthread_mutex_lock(&mutex1);
       if(arr[j]<global_min)
       {
          global_min= arr[j];
       }
       pthread_mutex_unlock(&mutex1);
    }

    }

    stm_exit_thread( );
    pthread_exit(0);
    }
```

In the above code in the thread function the critical section is the portion where the array elements are accessed and the global variable global_min is being updated. One half of the array elements are accessed using STM (optimistic approach) and the other half using locks (pessimistic approach) in the following manner.

```
    for((j=(((num*n)/NUM_THREAD)));j<(((num+1)*n)/NU
M_THREAD);j++)
    {
    if(j<(n/2))
    {
    START(0,RW);
    byte_under_stm=(unsigned char)LOAD(&global_min);
    if(arr[j]< byte_under_stm)
    {
    byte_under_stm=arr[j];
    }
    STORE(&global_min, byte_under_stm);
    COMMIT;
    }
    else
    {
    pthread_mutex_lock(&mutex1);
       if(arr[j]<global_min)
       {
          global_min= arr[j];
       }
       pthread_mutex_unlock(&mutex1);
    }
}
```

## VI. PERFORMANCE COMPARISON OF OPTIMISTIC, PESSIMISTIC AND HYBRID APPROACHES

The execution times for the different approaches are shown in the table below (Table 1).

Table 1. Execution Times for the different approaches

| **Locks (seconds) (Pessimistic Approach)** | **STM (seconds) (Optimistic Approach)** | **Locks + STM (seconds) (Hybrid Approach)** |
|---|---|---|
| 5 | 24 | 5 |

From the above table (Table 1) we can see that the execution time of the programs in case of locks and the hybrid approach was 5 seconds and in case of STM was 24 seconds. So we can say that the hybrid approach removes the disadvantages of the optimistic and pessimistic approaches and combines their advantages without any degradation of performance.

In our work we have divided the array into 2 equal parts and have applied optimistic approach in one part and pessimistic approach in the other. In the ideal case hybrid approach may also lead to performance improvement as optimistic and pessimistic approaches are applied where each is advantageous. Pessimistic approach is applied in those portions of the program where simultaneous execution of multiple critical sections by multiple threads may lead to inconsistency and optimistic approaches where simultaneous execution of multiple critical sections by multiple threads may not lead to inconsistency.

## VII. SYSTEM SPECIFICATIONS

The specifications of the system in which we compiled and executed the codes are given below:

**SYSTEM DESCRIPTION**

**1. Hardware Configuration**

Model Name: Intel® Xeon ® CPU E5645 2.40 GHz

 

Number of CPU cores: 6
Total Memory Space: 4.008 GB
Cache: 12288KB

**2. Operating System**
Fedora 11

**3. Software Configuration**
1) The language used in the programs is C.
2) gcc compiler version 4.4.0.

## VIII. CONCLUSION AND FURTHER WORK

In this paper we have first shown how to solve the problem of synchronization in parallel programs using optimistic approach (STM) and pessimistic approach (locks). Then we have developed and demonstrated a hybrid approach which removes the disadvantages of the optimistic and pessimistic approaches and combines their advantages without any degradation of performance. The problem considered in this paper is finding out minimum element in an array. It is a small prototype of a real life example in which multiple parts of a database are accessed and edited simultaneously in real time.

In our work we have divided the array into 2 equal parts and have applied optimistic approach in one part and pessimistic approach in the other. In the ideal case hybrid approach may also lead to performance improvement as optimistic and pessimistic approaches are applied where each is advantageous. Pessimistic approach is applied in those portions of the program where simultaneous execution of multiple critical sections by multiple threads may lead to inconsistency and optimistic approach where simultaneous execution of multiple critical sections by multiple threads may not lead to inconsistency. We are considering doing further work in this area.

## REFERENCES


[1] Ryan Saptarshi Ray,"*STM:Lock-Free Synchronization*", Special Issue of IJCCT, ISSN (*ONLINE*): 2231 – 0371, ISSN (*PRINT*): 0975 – 7449, Volume- 3, Issue-2, pp. 19-25, February 2012

[2] Ryan Saptarshi Ray and Utpal Kumar Ray, "*Writing Lock-Free Code*", In the Proceedings of International Conference on Computer Science and Engineering(ICCSE),Kolkata, pp. 19-25, 24th March 2012

[3] Pascal Felber, Christof Fetzer, Torvald Riegel, "*Dynamic Performance Tuning of Word-Based Software Transactional Memory*" In the Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, pp. 237-246 ,2008

[4] Philip S.Yu, Daniel M. Dias, "*Analysis of Hybrid Concurrency Control Schemes For a High Data Contention Environment*", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,Volume-18, Issue-2, pp. 118-129, 1992

[5] Sang H. Son, Junhyoung Lee, "*A New Approach to Real-Time Transaction Scheduling*", In the Proceedings of the Fourth Euromicro workshop on Real-Time Systems, pp. 177-182, June 1992

[6] Sung Ho Cho, JongMin Lee, Chong-sun Hwang, WonGyu Lee, "*Hybrid Concurrency Control for Mobile Computing*" In the Proceedings of High Performance Computing on the Information Superhighway, pp. 478-483, April 1997

[7] SungHo Cho, "*A Hybrid Concurrency Control with Deadlock-free Approach*", In the Proceedings of the International Conference on Computational Science and Its Applications, pp. 517-524, 2003

[8] Jan Lindstrom, "*Hybrid Concurrency Control Method in Firm Real-Time Databases*"

[9] Debendranath Das, Ryan Saptarshi Ray, Utpal Kumar Ray, "*Implementation and Consistency Issues in Distributed Shared Memory*", International Journal of Computer Sciences and Engineering (IJCSE) E-ISSN:2347-2693 Volume- 4, Issue-12, pp 125-131, 2016


## Authors Profile

**Ryan Saptarshi Ray** received the degree of B.E. in I.T. from School of Information Technology, West Bengal University of Technology, India in 2007. He received the degree of M.E. in Software Engineering from Jadavpur University, India in 2012. Currently he is PhD Scholar in the Department of Information Technology, Jadavpur University, India. He was employed as Programmer Analyst from 2007 to 2009 in Cognizant Technology Solutions. He has published 2 papers in International Conferences, 9 papers in International Journals and also a book titled "Software Transactional Memory: An Alternative to Locks" by LAP LAMBERT ACADEMIC PUBLISHING, GERMANY in 2012 co-authored with Utpal Kumar Ray.

**Parama Bhaumik** received B.Sc Phy(Hons.), B.Tech and M.Tech in Computer Science & Engineering from Calcutta University, India in 1996,1999 and 2002 respectively. She has done her Ph.D in Engineering from Jadavpur University, India in 2009. Currently she is working as Associate Professor in the Department of Information Technology, Jadavpur University, India. She has more than 32 research publications in Journals of repute, Book chapters and International Conferences.

**Utpal Kumar Ray** received the degree of B.E. in Electronics and Telecommunication Engineering in 1984 from Jadavpur University, India and the degree of M.Tech in Elecrical Engineering from Indian Institute of Technology, Kanpur in 1986. He was employed in different capacities in WIPRO INFOTECH LTD., Bangalore, India; WIPRO INFOTECH LTD., Bangalore, India, Client: TANDEM COMPUTERS, Austin, Texas, USA; HCL America, Sunnyvale, California, USA, Clent: HEWLETT PACKARD, Cupertino, California, USA; HCL

 

Consulting, Gurgaon, India; HCL America, Sunnyvale, California, USA; RAVEL SOFTWARE INC., San Jose, California, USA; STRATUS COMPUTERS, San Jose, California, USA; AUSPEX SYSTEMS, Santa Clara, California, USA and Sun Micro System, Menlo Park, California, USA for varying periods of duration from 1986 to 2002. From 2003 he is working as Assistant Professor in the Department of Information Technology, Jadavpur University, India. He has published 22 papers in different conferences and journals. He has also published a book titled "Software Transactional Memory: An Alternative to Locks" by LAP LAMBERT ACADEMIC PUBLISHING, GERMANY in 2012 co-authored with Ryan Saptarshi Ray.

 