# A Proposed Mechanism Using Parallelization Techniques in Steganography and Digital Watermarking for Multi Processor Systems

**Samir B Patel, Shrikant N. Pradhan**

Department of Computer Science & Engineering, Institute of Technology, Nirma University, Ahmedabad.

*Abstract*-- **Parallel programming techniques used in the paper are for a shared memory multiprocessor computer for performing Steganography and digital watermarking. Steganography and digital watermarking is the art of hiding information into the cover object taking care that is imperceptible with the naked eye. There are computer with many central processing units (CPUs), all of which have equal access to a common pool of main memory. Multiprocessor computers can be used for general-purpose time sharing as well as for compute intensive applications. If the user creates only single stream applications, then a multiprocessor will not be any more or less difficult to use than a uniprocessor i.e. it will be quite sensible to run such a program on a uniprocessor (one CPU).**

**This paper focuses on a technique through which optimization is possible for embedding and extracting the information in Steganography and digital watermarking techniques exploiting the parallelization techniques. The parallelization techniques are expected to be implemented on almost all the applications running on Laptops, PDA, Desktop and mobile phone. The demand for resource hungry applications on mobile phones is expected to grow exponentially hence the future of mobile phone is likely to be with multicore embedded processors on mobile phones too. Laptops and Desktops are already available in market having dual-core computing power. The paper draws an attention to highlight the possible parallelization in Steganography and digital watermarking techniques to get superior performance**.

*Index Terms*—**fork, join, spin_lock, barriers, steganogprahy, watermarking.**

## I. Introduction to Parallel Programming

Parallel programming is a kind of approach involving apportioning the work what is normally thought of as an indivisible calculation among many processors resulting into more rapid completion of work than if it were done by a single CPU. The type of parallelism that involves nearly independent tasks, such as database management, parallel I/O and Monte Carlo simulations of trajectories, are examples of coarse-grained parallelism. On the other hand, the example in which different iterations of a loop are executed by different processors is called fine-grained parallelism. In coarse grained parallelism, each calculation is conceptually nearly independent of the others and normally involves relatively infrequent communication among the individual calculations.

In fine grained parallelism, what is normally thought of as a single, indivisible calculation is partitioned among processors. This commonly involves subdividing a loop and requires relatively frequent communication between programs running on different CPUs. Fine grained parallel programming is generally more difficult to do than coarse-grained parallel programming. Our approach in this paper will be a combination of coarse grained and fine grained parallelism.

Our paper stresses simplicity and focuses on fundamentals. It is possible to create a tremendous variety of parallel programs with just five library functions. One for Sharing memory, one for creating processes, one for destroying processes, and two for interprocess synchronization (locks and barriers).

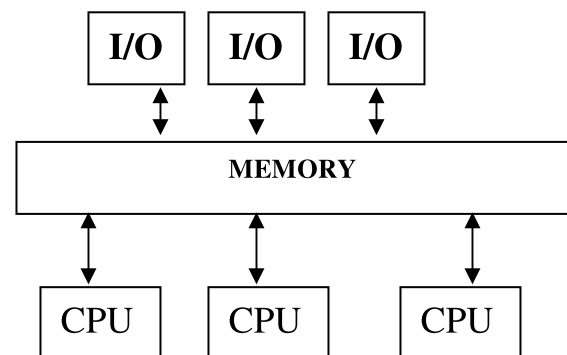



Fig. 1: Schematic model of a shared memory multiprocessor computer

Schematic model of a shared memory multiprocessor computer is shown in figure1. This is a model of a shared memory multiprocessor, so called because the processors share a single pool of memory. In this model there are many

CPUs and I/O modules. The memory contains the instructions executed by each CPU, as well as the data on which the program operates.

The model in figure 1 is meant to imply that any CPU can access any memory location at any time, with one exception that no single memory location can be accessed by two CPUs simultaneously. In the model, different processes can access different memory locations simultaneously. Although this schematic model is an idealization, fast overlapped access to the memory can be implemented, so memory access is almost simultaneous. The closer the implementation gets to the idealized model of figure 1 the more effective the parallel program will be in comparison with a comparable sequential one.

When the programs are loaded from disk, the data and instructions for such program go into the region of main memory resulting into different sections of physical memory. These regions are called processes. In UNIX operating system, which is time shared, all that is necessary to know is that it acts as an autonomous switch, allowing different processes to execute on the single processor at different times, and it protects the memory occupied by a given process from being accessed by any of the other process.

## II. LIBRARY MODULES

There are number of different methods of implementing parallelism in UNIX. We propose to use Library modules developed using various system calls available in UNIX to achieve that objective.

### *1) Forking - Creating processes*

```
int id, nproc, process_fork
……
id=process_fork(nproc)
```

The function process_fork(nproc), when called from an executing process, creates nproc-1 additional processes. Each additional process is an exact copy of the original (spawner) process.

After returning from the process_fork function, there are nproc-1 additional and identical processes to time share or to execute in parallel. The original process, which called process_fork(nproc) in the first place, is still running, so now there are nproc processes in all. Each of the nproc processes has a private copy of each of the variables of the parent, unless the variables are explicitly shared. The process which made the call is called the parent process and the nproc-1 additional processes are called the child processes. After the return from fork function each process continues executing at the next executable statement following the process_fork function. The process_fork function returns an integer, called the process-id, which is unique to each process. It returns 0 to the parent and the integers 1, 2,..,nproc-1 to the children, each child getting a different number. So apart from the value of ID, the processes are identical.

- **Joining Processes:** The join is carried out by the statement process_join( ). The join is the opposite of the fork - it destroys all the child processes, leaving only the parent running. Whenever a process makes the process_join call, if the process is a child process, it is destroyed. If the process is the parent process i.e. id = 0, it waits until all the child processes are destroyed. The parent process continues with the statement following the call to process_join. In general for the parent process, the joint is a wait and for the child process the join is a kill. Hence, it becomes mandatory for all the processes to execute the process_join function as shown in figure 2.

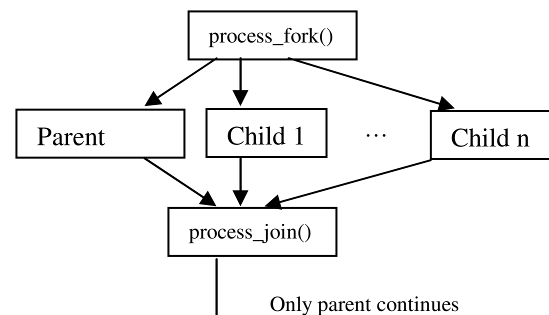



Fig. 2: All processes must execute process_ join function.

### *2) Shared Memory*

```
int num_of_bytes
any_declaration var_name e.g. int DCT,LSB
shared(var_name, num_of_bytes)
```

Where var_name is the name of the variable. It is actually a pointer to the variable. The term num_of_bytes is the number of bytes which the variable occupies in memory. The subroutine shared is called by the parent, before forking. This call arranges for var_name to be shared among all the children created by the calling process. That is, when the children are created, the shared variables occupy the same physical region of memory. When the parent process calls process_fork and spawns a number of children, each child is an identical copy of the parent. All the variables and values are copied, and, unless memory is declared to be shared changes in the data of one process do not appear in the data of another process.

The behaviour of parallel programs can be extremely sensitive to shared memory. Very common errors in parallel programming are not sharing variables which should be shared, sharing variables which should not be shared, or misusing shared variables. In analyzing shared-memory parallel programs, it is essential to assume that all processes are randomly scheduled. That is, any process can be idled and restarted at any time. It can be idle for any length of time. It is incorrect to assume that two processes will proceed at the

same rate or that one process will go faster because it has less work to do than another process.

*3) Spin-Locks, Contention, and self scheduling*

One problem of loop splitting is that it could lead to an uneven distribution of work among the processes. This may lead to less than ideal speedup. Where,

Ideal speedup = Time required for sequential execution / Time required for parallel execution.

The general form of loop splitting is

**for (i=id ; i < n ; i += nproc) { work section }.**

If the value of n is small (i.e. <5) then there is an unequal distribution of work among processes resulting into less than ideal speedup. However, in our implementation process, the value of n will be very large, distribution will be fairly even and hence we will be able to achieve near ideal speedup.

The alternative is self scheduling-each process chooses its next index value only when it is ready for one. Self-scheduling allows some processes to execute only a few iterations, while others may execute many iterations. The situation in which two or more processes try to alter a shared variable in parallel is called contention. The solution to avoid contention is to have indivisible unit i.e. statements to be executed in protected region or has mutual exclusion.

In order to enforce protection, it is necessary that one process communicate to all other processes that it is in a protected portion of the program and all other processes must behave in a responsible manner by staying away. Such communication is an example of interprocess communication (IPC) or synchronization. This communication normally involves the processes sharing one or more variables. The required structure for implementing such synchronization is the spin-lock. Spin lock internally makes use of a semaphore which is nothing but a shared variable.

Following is a sample code showing addition of a constant to each element of an array. This also illustrates the use of spin-lock.

```
#include <stdio.h>
#include "fork_join.h"
#include "sharedmemory.h"
#include "semaphore.h"


int main()
{
int *arr, nproc, num, *nextindex,*lock,shmid,id,i=0,c;
   printf("\nEnter the no of processes:");
   scanf("%d",&nproc);
// lock variable is declared as shared
   lock=(int *)shared(sizeof(int),&shmid);
   nextindex = (int *)shared(sizeof(int),&shmid);
   arr = (int *)shared(sizeof(int)*num,&shmid);
   setbuf(stdout,NULL);

printf("\nEnter the no of elements u want to enter in an array
:");
   scanf("%d",&num);
   arr = (int *)malloc(sizeof(int)*num);
   for(i=0;i<num;i++)
   {
      printf("\nEnter the value of an array %d :",i);
      scanf("%d",&arr[i]);   }

   printf("\nEnter the constant value :");
   scanf("%d",&c);
          // initialization of spinlock
   spin_lock_init(lock);

   *nextindex=num-1;
   i=*nextindex;

   id=process_fork(nproc);
   while(i > 0)
   {spin_lock(lock); // lock is acquired
    i =*nextindex;
    *nextindex = *nextindex - 1;
   spin_unlock(lock); // lock is released
   if(i < 0)
      break;
printf("\n The old array[%d] : %d",i,arr[i]);
   arr[i]=arr[i]+c;
printf("\n The new array[%d] : %d",i,arr[i]);
printf("\n The value of nextindex:%d",*nextindex);
  }
   process_join(nproc,id);
   cleanup_memory(&shmid);
}
```

The program for an operating system may involve thousands of different spin-locks. When the function spin_lock is called by a process, the logical action is that a check is made to see if the lock is unlocked. If it is, then the lock is locked and the process that called spin_lock is allowed to continue executing instructions. Thus, when a process finds the lock as unlocked, it locks it and then proceeds into the protected region. However if the lock was already locked when the process called the function spin_lock, then the calling process must wait (spin its wheels) until the lock becomes unlocked. It is also occupying a processor completely, executing a set of nonproductive instructions. As a result spin_locks should be used sparingly, and the protected regions should be as small as possible, so as to minimize the overhead of processes spinning unproductively at the locks.

The function spin_lock takes care of the details of checking the state of the lock, allowing the process to continue or causing it to spin. Spin-lock is used to eliminate contention and is used to enforce that only one process should ever be able to update a shared variable at a time. Using spin_locks inefficiently can make a parallel program actually run slower

than the sequential version. If there were a large number of processes (nproc is large), then the processes could pile up at the lock, waiting to enter the protected region. This can cause the self-scheduling program to execute more slowly than the loop-splitting version. A general feeling is that, if the calculation are long and involved, then the self-scheduling technique will be more efficient that the loop-splitting version. So it is upto the programmer to decide which technique is appropriate for the particular situation.

*4) Barriers*

A barrier causes processes to wait and allows them to proceed only after a predetermined number of processes are waiting at the barrier. It is used to ensure that one stage of a calculation has been completed before the processes proceed to a next stage which requires the results of the previous stage. Barriers are used to eliminate race conditions, in which the result of a calculation depends on the relative speed at which processes execute. Along with spin_locks barriers are the most important synchronization mechanism for fine grained parallel programming.

The following section illustrates the use of barriers
int bar_array[4], blocking_no

- Declared a bar_array as shared
  shared(bar_array, size)
- Initialize the barrier
  barrier_init(bar_array, blocking_no)
- nproc processes starts their work i.e. id=process_fork(nproc)
- Logic to compute partial data will start from this place.
- No process can continue past the barrier until all the processes have executed the barrier call.
  barrier(bar_array)
- All processes will continue past the barrier call
- At the end all the processes will have to execute the process_join function
- At this place, all the child processes get terminated, only parent process remains and ultimately it also dies.

The barrier function has two phases: a trapping phase and a release phase. At trapping phase the subroutine checks to see how many processes have already made this call. If the number of processes, including the newly arrived one, is less than the blocking number, the newly arrived process must wait (along with all the processes that have already arrived at the barrier). On the other hand, if the number of processes which have made the call, inclusive of the new one, equals the blocking number, then all the processes that are waiting are allowed to proceed, including the last one to make the call. This later releasing of processes must occur before any other process can be trapped further.

## III. INTRODUCTION TO STEGANOGRAPHY AND DIGITAL WATERMARKING

Steganography and Digital watermarking as mentioned earlier is the ability of sending message within the image such that the existence of the message is not known to the user. The objective is to avoid the awareness of hidden message within the image during the relay. If there is suspicion then the goal is not satisfied. Steganalysis is the ability of identifying and mining such covert messages.

Cryptography and Steganography form the foundation for a large number of digital watermarking concepts. The stego system is conceptually similar to the crypto system.

Figure 3 shows the overall representation of the stego system whereby a key is additional data needed for embedding and extracting. The Embedding function and the Extracting function are opposite to each other in the sense that reverse operation will take place in extracting the message than that of embedding the message in the cover object.

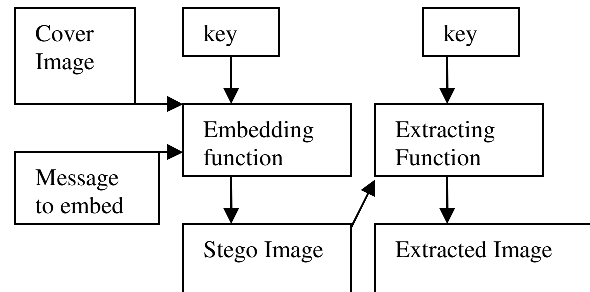



Fig. 3: Block diagram of Stego System

Watermarking is very similar to steganography in a number of respects. Both seek to embed information inside a cover object with little to no degradation of the cover object. Watermarking however adds the additional requirement of robustness. An ideal steganography system would embed a large amount of information, perfectly securely with no visible degradation to the cover object. An ideal watermarking system however would embed an amount of information that could not be removed or altered without making the cover object entirely unusable. As a side effect of these different requirements, a watermarking system will often trade capacity and perhaps even some security for additional robustness. Some methods of steganography and watermarking are as under.

- LSB (Least Significant Bit)
- Transformation based schemes [9]

A major advantage of LSB algorithm is that it is quick and easy, whereas using transformation techniques like Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT) takes a large amount of time to embed and the embedding capacity is also less. There are number of other ways in which embedding can be carried out like redundant

pattern encoding, spread spectrum method etc.

Applications: There is a growing importance of steganography and watermarking in intelligence work, as what is viewed as a serious threat to some governments. Even the spying agencies can use it for the secret data transmission. Most researchers believe that steganography's niche in security is to supplement cryptography, not to replace it. Description like place, person's name, time, event, ownership, accessibility, etc. can be piggybacked with the original cover image/video/audio and retrieved at the destination end.

## IV. STEGANOGRAPHY ON IMAGE USING PARALLEL PROGRAMMING APPROACH

Steganography is the art of hiding information inside the cover object like image, audio or video, whereas adaptive steganography - an intelligent approach to hide messages through the techniques like LSB, Matrix Encoding and PN-Sequences - serves as a capable solution to latest security assurance concerns. Incorporating the above data hiding concepts with established cryptographic protocols in wireless communication would greatly increase the security and privacy of transmitting sensitive or non-sensitive information.

Here, we propose a technique through which ASCII gets converted to Base64. These Base64 bits get inserted into every pixel of an RGB image, so that, each pixel will have one character to carry, as a result if the image is of 256 x 256 pixels then it can carry as many as 65536 characters in an image for us. On the sender and receiver sides there will be Base64 encoding table of our choice. This is how this technique is adaptive. It can even convert non ASCII values (image) to our Base64 table values. A simple example of converting capital 'A' to base64 is resulting into 'QQ=='.

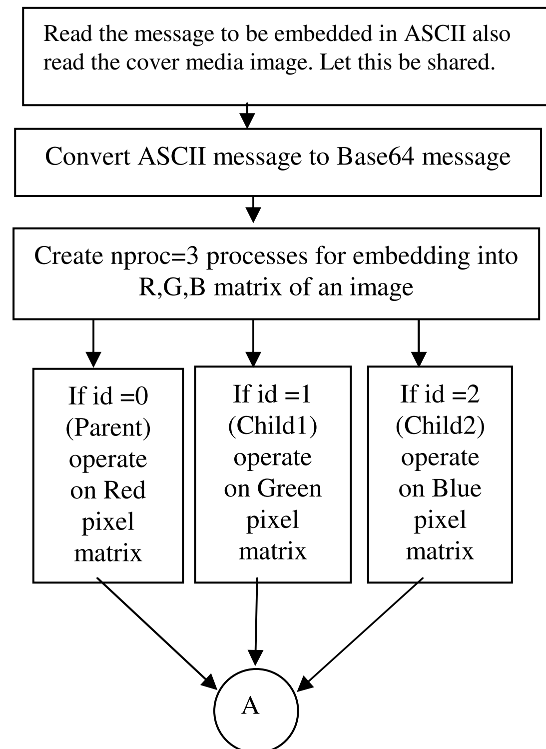
The first thing to note is the '=' at the end of the Base64 encoded string. A Base 64 encoded string will have zero, one or two '='s at the end. As '=' is not part of the Base 64 encoding, it can only ever appear at the end and has a special meaning. If there is one = then there are 2 inserted zeros and if there are two = characters then there are 4 inserted zeros at the end.

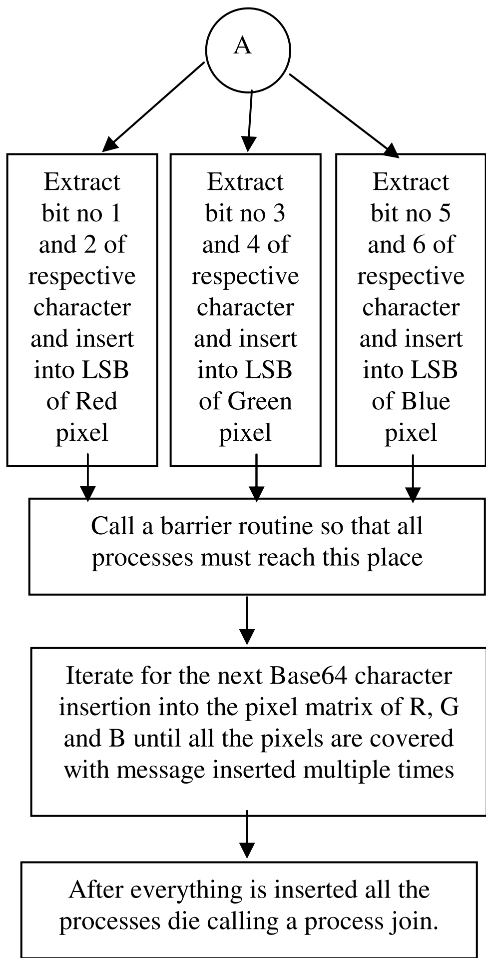
ASCII of A is 01000001 and gets converted to 010000 01*0000* i.e. 16 in our table and that is QQ and four zeros at the end results into ==. The reverse will happen at the decoder side and will combine all the pixel values to form the actual data.

The concept is as follows:

- Let the cover image be the shared image.
- Convert the message to be embedded into Base64 using the conversion table.
- Let there be three processes in the system. Operating upon Red, Green and Blue pixel values.
- Fix the insertion pattern for Red as bits 1 and 2, for Green as bits 3 and 4, and Blue as bits 5 and 6.
- Each process is having identical work to do for insertion. Let process id=0 (Parent) read two bits 1,2, process id=1(Child 1) read next two bits 3,4 and process id=2 (Child 2) read the remaining two bits 5,6 of the Base64 value of the first character. Each process replaces the Least Significant Bits (LSB) of Red, Green and Blue pixels respectively.
- There will be a barrier call after insertion of each character by all the processes, so that all the 3 processes will read the next character for further insertion and will continue till all the message is inserted into the image (probably multiple times).
- The embedding algorithm which is applied at this point has to be very fast and the capacity of embedding also has to be considered. It must be possible to embed the information multiple times so that even though if some attack takes place intentionally/un-intentionally at the destination or in between then the information can be extracted error free.
- The embedding of information in to the cover object i.e. image, movie or audio should not have any artifacts.
- There are other data hiding techniques which focus on robustness of the hidden data rather than capacity of data. If the robustness is to be considered then such a technique can be classified as Digital watermark or else a simple steganography technique.

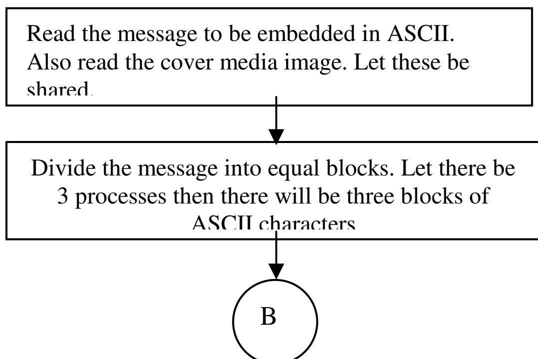
**Layout for Inserting Message in parallel Technique 1:**

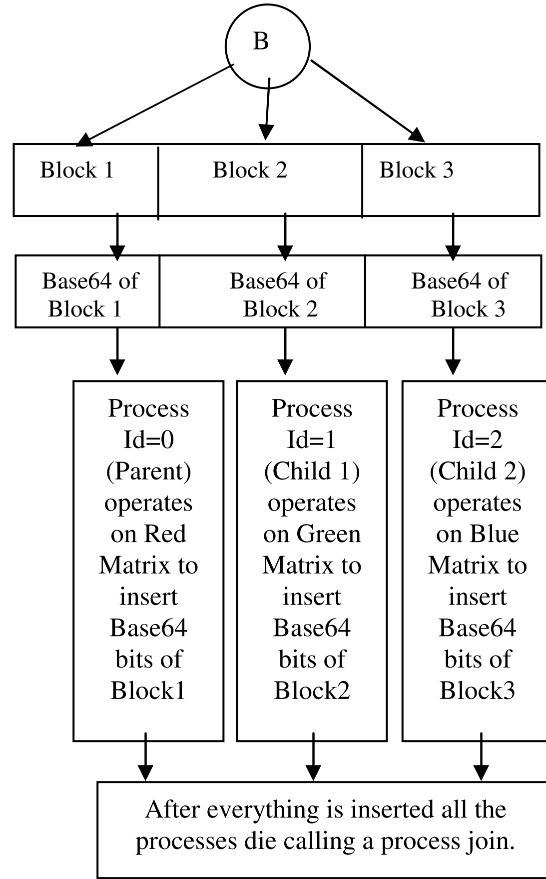





- In the earlier discussion we assumed the storing of the message will be in row wise. However, the data can be inserted column wise using a loop splitting technique. Example if there are three processes then process 0 (Red), process 1 (Green) and process 2 (Blue) will iterate over column number 1,4,7…, 2,5,8…. and 3,6,9…. respectively. Using this technique the data is more scattered and improves the level of security.
- This technique is quite capable of defending attacks. Since data is inserted multiple times, we can extract the contents from the other remaining un-attacked pixels.

**Layout for Inserting Message in parallel Technique 2:**




- This technique may not be acceptable if there is an attack since the data is inserted in block scattered manner. Here, if there is an attack on just one pixel three characters are affected. Whereas in the earlier technique only one character was affected.

## V. ATTACKS ON EMBEDDED MESSAGE/WATERMARKS

A watermarked image is likely to be attacked intentionally or unintentionally. Some intentional attacks include cropping, filtering, rotation, scaling etc. and unintentional attacks include compression, transmission noise etc. Summarization of these different types of attacks [10]:

## VI. CONCLUSION

In this paper the author has identified an area whereby parallelization techniques can be used for performing Steganography and Digital watermarking techniques. A simple idea about loop splitting, self scheduling, barriers and spin lock has to be used to perform the operations in parallel. These techniques if implemented on a time shared single processor system will not gain any computation advantage but if implemented on a multi core will have tremendous advantage. The speed up will be very good and the whole operation will be performed faster. Next generation of PCs are going to be only with multi core processors. Even the mobiles are also expected to be with multi core embedded

devices. Hence designing a technique to perform watermarking in parallel is of very high significance.

## VII. ACKNOWLEDGEMENT

We would like to thank all the members of NIRMA UNIVERSITY for providing continuous support and inspiration.

## REFERENCES

[1] Rajmohan, ”Watermarking of Digital Images”, ME Thesis Report, Dept. Electrical Engineering, Indian Institute of Science, Bangalore, India, 1998.

[2] S.P.Mohanty, ”Watermarking of Digital Images”, Masters Project Report, Dept. of Electrical Engineering, Indian Institute of Science, Bangalore - 560 012, India, Jan 1999.

[3] B.Pfitzmann, ”Information Hiding Terminology”, Proc. of First Int. Workshop on Information Hiding, Cambridge, UK, May30-June1, 1996, Lecture notes in Computer Science, Vol.1174, Ross Anderson(Ed.), pp.347-350.

[4] W. Bendor, et. al., ”Techniques for Data Hiding”, IBM Systems Journal, Vol.35, No.3 and 4, pp. 313-336, 1996.

[5] B.M.Macq and J.J.Quisquater, ”Cryptography for Digital TV Broadcasting”, Proc. of the IEEE, Vol.83, No.6, June 1995, pp. 944-957.

[6] David Kahn, ”The History of Steganography”, Proc. of First Int. Workshop on Information Hiding, Cambridge, UK, May30-June1 1996, Lecture notes in Computer Science, Vol.1174, Ross Anderson(Ed.), pp.1-7.

[7] R.J. Anderson and Fabien A.P. Petitcolas, ”On the Limits of Steganography”, IEEE Journal on Selected Areas in Comm., Vol.16, No.4, May 1998, pp.474-481.

[8] R.J. Anderson, ”Stretching the Limits of Steganography”, Proc. of First Int. Workshop on Information Hiding, Cambridge, UK, May30-June1 1996, Lecture notes in Computer Science, Vol.1174, Ross Anderson( Ed.).

[9] Samir B. Patel, “Image Based Watermarking and Authentication mechanism” of the “National Conference on Current Trends in Technology” NUCONE 2007, Ahmedabad, India, pp 295 - 300, 29th Nov. -1st Dec., 2007.

[10] Samir B. Patel, “Proposed secure mechanism for identification of ownership of undressed photographs captured using camera based mobile phones” of the 2nd IEEE International conference on Digital Information Management, pp 442 - 447 , 28-31 October 2007.

[11] C.Cachin, ”An Information-Theoritic Model for Steganography”, Proc. of the 2nd International Workshop on Information Hiding, Portland, Oregon, USA, 15-17 Apr 1998, Lecture notes in CS, Vol.1525, Springer-Verlag.

[12] S.Craver, ”On Public-Key Steganography in the Presence of an Active Warden”, Proc. of the 2nd International Workshop on Information Hiding, Portland, Oregon, USA, 15-17 Apr 1998, Lecture notes in Comp Sc, Vol.1525, Springer-Verlag.

[13] N.F.Johnson and Sushil Jajodia, ”Exploring Steganography: Seeing the Unseen”, IEEE Computer, Vol.31, No.2, pp.26-34, feb.1998.

[14] J. M. Acken, ”How Watermarking Value to Digital Content?”, Communications of the ACM, July 1998, Vol.41, No.7, pp.75-77.

[15] S. Craver, et. al., ”Technical Trials and Legal Tribulations”, Communications of the ACM, July 1998, Vol.41, No.7, pp.45-54.

[16] I. J. Cox and M. Miller, ”A Review of Watermarking and Importance of Perceptual Modelling”, Proc. SPIE Human Vision and Imaging, SPIE-3016, Feb 1997.

[17] F. Mintzer, et. al., ”Opportunities for Watermarking Standards”, Communications of the ACM, July 1998, Vol.41, No.7, pp.57-64.

[18] R. Mehul and R. Priti, “Discrete Wavelet Transform based Multiple Watermarking Scheme,” Proceedings of IEEE Region 10 Technical Conference on Convergent Technologies for the Asia-Pacific, Bangalore, India, October 14-17, 2003.

[19] A Survey of Digital Image Watermarking Techniques Vidyasagar M. Potdar, Song Han, Elizabeth Chang presented at 2005 3rd IEEE International Conference on Industrial Informatics (INDIN)