International Journal of Grid and Distributed Computing
Vol. 9, No. 1 (2016), pp.127-134
http://dx.doi.org/10.14257/ijgdc.2016.9.1.13

Performance Evaluation and Analysis of Parallel Computers Workload

M.Narayana Moorthi[1] and R.Manjula[2]

[1]Assistant Professor, (SG), SCOPE-VIT University, Vellore-14

[2]Associate Professor, SCOPE-VIT University, Vellore-14
mnarayanamoorthy@vit.ac.in, rmanjula@vit.ac.in

Abstract

The current and future computer industry is changing from single core CPU to Multicore Processors. The next generation computing systems are focusing on parallel computing to solve the problem in fast and high speed using parallel programming concepts. By running more than one task at the same time with multiple processors concurrently or parallel we can achieve high speed in our computing applications. Here the performance improvement is measured in terms of increase in the number of cores per machine and is analyzed for better optimal work load balance. The parallel computers follow different workload scheduler. In this paper we investigate how to tune the performance of threaded applications with balanced load for each core or processor. The focus here is the process of comparative analysis of single core and multicore systems to run an application program for faster execution time and optimize the scheduler for better performance.

***Keywords**: Multicore scheduler, Parallel Computer, Parallel Program, High speed Computing*

1. Introduction

The processing capability of Microprocessor [7-9] is increasing in terms of clock speed and inclusion of more and more execution units with pipeline support in a chip to support parallelism. According to Moore's law [1-2] the number of transistors are doubled in a Microprocessor in every 18 months. Now a day's the parallel computers are used in all places like schools, colleges, organization's and scientific computing centers to solve large problems in less time. The parallel computer is defined as the collection of more than one processing element or cores to solve a large size problem in fastest rate. The Multicore [4-6] and Many core processors are replacing the current world of computing systems. Processing the task in parallel is the present and future computing challenge [1] [2,3,6,11]. The efficient way of allocating the task to the available cores is the goal of any work load scheduler. There are many job scheduling algorithms exist for workload balancing. There is not a single algorithm which is efficient for all scenarios. The way we measure the performance is the running time or execution time of the program or application. The need to reduce the running time of a program is to get high speed computing which can be done through parallel programming [4,5,6] and parallel computing machines. The Amdahl's Law is often used in parallel computing to predict the theoretical maximum speed up using multiple processors. In case of parallelization, Amdahl's Law [1,2,3,10,11] states that if P1 is the proportion of a program that can be made parallel and (1-P1) is the proportion that cannot be parallelized, then the maximum speedup that can be achieved by using Ni Processors is 1/(1-P1)+P1/Ni. Other factor to be analyzed in threaded applications is load imbalance. Balancing the workload among threads or cores or processors is critical to application performance. The key objective for load balancing is to minimize idle time on threads and share the workload equally among all cores and threads with minimal work sharing overheads.

2. Parallel Programming Languages

The fundamental computer machine is identified as Von-Neumann Architecture. It has only one processor or single core to solve the given problem. The program is executed sequentially. To improve the run time of program parallelism is introduced with different levels. The Multicore Processor Architectures consist of one or more processor cores to provide higher levels of parallelism.

2.1. Overview of Open MP

Open MP [10-17] Open Multiprocessing is a parallel programming Language. It has the collection of compiler directives, library functions and environment variables. Open MP follows the Fork-Join Model for the execution of parallel program. This Open MP can be used for shared memory parallelism in C /C++ [8,9] Programs. A program written in Open MP executes as a master or single thread as same as sequential program. When the parallel construct is encountered, the master thread creates the number of threads and each thread computes the task simultaneously or concurrently. The sample structure of open MP parallel programming is as follows.

```
//without Open MP
#include < stdio.h >
#include <conio.h>
void main(void) { printf("sample");}
```
Output: sample

The above c-program code will print sample as output in a single core machine. If the same program we want to run in multicore machines then the following modifications are to be made for Open MP structure.

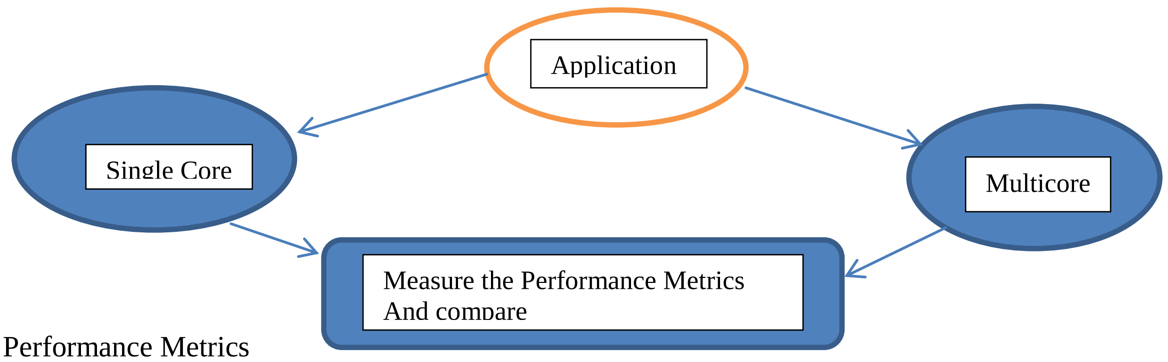
```
//with Open MP
#include < stdio.h >
#include <conio.h>
#include<omp.h>
void main(void) {#pragma omp parallel
{printf ("sample");}}
```
Output: sample, sample   ----   if number of cores is 2
Output: sample, sample, sample, sample ----   if number of cores is 4.

The following steps illustrate the proposed work for Performance Study, Evaluation and analysis of single core and Multicore Machines.

Step1: Identify and define the large size problem to be solved
Step2: Identify the sequential algorithm to solve the above problem
Step3: Run this sequential algorithm code in a single core machine
Step4. Measure the various performance parameters (Like Total Execution time or Run time, Cache memory size, hit ratio, CPU utilization, *etc*)
Step5: Identify the concurrency in the above application code (Independent sub tasks)
Step6: Write the parallel code using parallel programming languages (Open MP)
Step7: Run the above code using multicore machines
Step8: Measure the various performance parameters (Like Total Execution time or Run time, Cache memory size, hit ratio, CPU utilization, *etc*)
Step9: Rerun the above code using various thread size or processors cores (2, 4, 8, 16, 32, n) and make a comparative analysis table.

2.2. Performance Evaluation and Analysis

The following Figure -01 illustrates how the performance of single core and multicore architectures [18] differs from various applications.



Performance Metrics

Figure 01. Performance Analysis Flow Diagram

The following are few performance metrics we are interested to study and analyze while running our application on single core and multicore machines. They are as follows,

(1) Speed up : It is the ratio of the sequential execution time to parallel execution time
(2) Iso Efficiency: This defines the utilization of CPU
(3) Power consumption $P = VI = CV^2F$; P-Power / V-Supply Voltage / I- Current / F – operating frequency
(4) E=PT ; E-Energy, T-Execution Time
(5) T=Nt ; N-No of Clock Cycles , t- clock duration

The following assumptions are made to make a comparative analysis of applications on single core and multicore machines.

S-Computer System
R-List of Computer Resources
S= {R1, R2, R3…Rn}
A-Application Program
T-Task
ST-Sub Tasks
I –Input Devices
O-Output Devices
M-Memory
P-Processor / Core
I.e. Ai $\rightarrow \sum$STi; i=1 to n;
Si = {Ii, Oi, Pi, Mi}; i=1 to n;
Ai $\rightarrow$Request {Si};
Si $\rightarrow \in$Ri

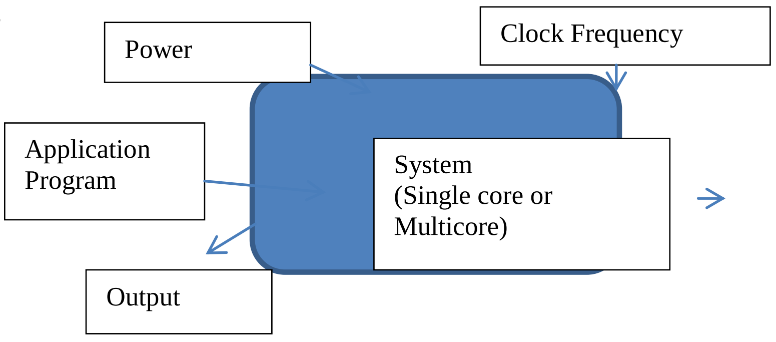


Figure 02. Application Running on the System

To run an application each sub task of an application is in need of resources like processor time, memory and I/O Operations. Ai –Sequential or Parallel. If Ai is sequential run on single core machine and convert into parallel by using parallel programming languages to run on multicore machines

Workload Balance Issue:

If ST (A) i = Pj; Work load is balanced

If ST (A) i >Pj; Processor core selection and Assignment of additional task to which core plays an important role, If ST (A) i <Pj; Few processor cores are idle
(i is the number of tasks and j is the processor cores)\

When evaluating application performance execution time and power consumption are the main metrics to consider.

Execution Time (Single core) = ∑ Time (STi); i=1 to n;
Execution Time (Multicore) = Max (Time (STi); i =1 to n;
Speedup = Sequential execution time/Parallel execution time

Complexity Analysis: An algorithm is a finite sequence of steps well defined to solve a given problem [19]. In many case there will be more than one method or algorithm to solve the given problem. Here the choice or selection of an algorithm depends on efficiency or complexity of an algorithm. The complexity of an algorithm measures the running time and or storage space required for the algorithm. The following notations are used to measure the complexity of an algorithm. Given M1 is an algorithm, n1 is the input data size, f1(x) is the output

Time Complexity of sequential algorithm (vector operations) =O (n)
Time Complexity of parallel algorithm (vector operations) = O (n/p);
P is the number of processor cores

The sequential algorithm runs on only one core or processor. The parallel algorithm runs on many cores or processors. The two measures of complexity of an algorithm are as follows. Worst case (Maximum time or space) and Average case (Expected Value) plays an important role to select the algorithm. The probability theory is a mathematical modeling of phenomenon of chance or randomness. The set S of all possible outcomes of a given experiment is called the sample space. The particular outcome i.e. an element in S is called a sample point. An event A is a set of outcomes or a subset of sample space S. Let S= {a1, a2, a3, ---, an}. A finite probability space or probability model is obtained by assigning to each point a_i in S a real number P_i called the probability of a_i. Suppose E is an event in a sample space with P(E)>0.The probability that an event A occurs once E has occurred is known as the conditional probability of A given E, written P(A\E) is defined as follows. P (A\E) =P (A ∩ E) / P (E). If X is given by the distribution

X1	X2	X3	-	Xn
P1	P2	P3	-	Pn

Then the Mean or expectation of X is μ=E(X) = x1p1+x2p2+x3p3+---+xnpn = ∑xipi
The variance =Var(X) = $(x_1-\mu)^2 p_1+(x_2-\mu)^2 p_2+---+(x_n-\mu)^2 p_n = \sum (xi-\mu)^2 p_i$
The standard deviation σ = √VAR(X)

2.3. Work load Scheduler

The scheduler allocates the next pending job or task [21] [22] or work load to the processor or processor cores. The scheduler follows various rules, regulations and policies to allocate the job to the system. Here the goal of any scheduler is to optimize the utilization CPU and other system resources.

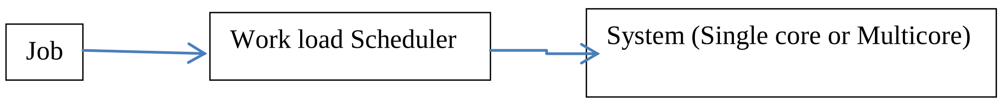


Figure 03. Work Load Scheduler

If the machine is a single core machine the jobs are executed sequentially or one by one. Whereas the same machine is multicore machine the jobs are executed simultaneously of parallel [20] to all cores in the machine taking the workload balance. The following parallel programming approaches are followed.

Temporal / Data Parallelism:

Let the number of jobs = n1
The time taken to do the job = p1
If the jobs are completed with single core machine then the time taken is n1p1
Assuming that the jobs are independent and each job is divisible by k1 sub tasks.
Thus the time taken for each sub task is p1/k1
Thus the time taken by multicore processors is n1p1/k1
Speedup = Sequential execution time / parallel execution time = n1p1/n1p1/k1 = k1
Here k1 refers the number of cores.
Assume the following: - The machine is 4 core machines
The number of task and their execution time is as follows.

Task	T1	T2	T3	T4	T5
Time	4	7	12	4	6

Total task = 5
Total number of cores (Processors) = 4
Total execution time = 4+7+12+4+6 =33
Sequential execution time = 1 x 33 = 33 units
Parallel assignment = 33/4 = 8 units
Core 1 = T1, T4
Core 2 = T2
Core 3 = T3
Core 4 = T5
Speed up = 33 / 8 = 4 (Equal to the number of cores)

Experimental Case study: The following vector operations are performed on single core and multicore machines. The sample code is shown below.

```
//Vector operations sample sequential code
#include<stdio.h>
#include<conio.h>
void main(void)
{          int a1[5],a2[5],a3[5],a4[5],a5[5],i;
           for(i=0;i<5;i++)
           { a1[i]=i;a2[i]=i*i;        }
           for(i=0;i<5;i++)
           {printf("%d\t%d\n",a1[i],a2[i]);}
           for(i=0;i<5;i++)
           {a3[i]=a1[i]+a2[i];a4[i]=a1[i]-a2[i];a5[i]=a1[i]*a2[i];      }
           for(i=0;i<5;i++)
           {printf("%d\t%d\t%d\t%d\t%d\n",a1[i],a2[i],a3[i],a4[i],a5[i]);     }}
//Vector operations parallel code using open MP Parallel programming Language
#include<stdio.h> #include<conio.h> #include<omp.h>
void main(void)
{int a1[5],a2[5],a3[5],a4[5],a5[5],a6[5],i;
float s,e,t;          s=omp_get_wtime();
#pragma omp parallel for
                     for(i=0;i<100;i++)
           {a1[i]=i;a2[i]=i;}
#pragma omp parallel for
           for(i=0;i<100;i++)
           {printf("%d\t%d\n",a1[i],a2[i]);}
#pragma omp parallel for
           for(i=0;i<100;i++)
           {a3[i]=a1[i]+a2[i];a4[i]=a1[i]-a2[i];a5[i]=a1[i]*a2[i];      }
#pragma omp parallel for
           for(i=0;i<100;i++)
           {printf("%d\t%d\t%d\t%d\t%d\n",a1[i],a2[i],a3[i],a4[i],a5[i]);      }
           e=omp_get_wtime();      t=e-s;    printf("total time is %f\t\n",t);}
```

Dependency graph: The following picture illustrates the dependence analysis. In Fig 04a-Dependency graph of sequential running of code is illustrated. In Fig 04b dependency graph of Parallel running of code is illustrated.

V- Read Vector, A-Vector Addition, B-Vector Subtraction, C-Vector Multiplication.

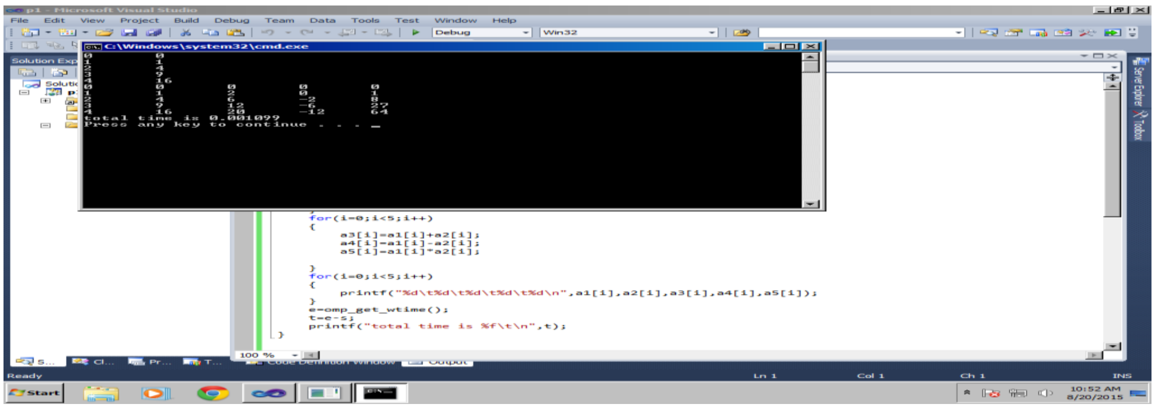
$$\begin{tikzcd} V \arrow[d] \\ A \arrow[d] \\ B \arrow[d] \\ C \arrow[d] \\ {} \end{tikzcd}$$

$$\begin{tikzcd} & V \arrow[dl] \arrow[d] \arrow[dr] & \\ A \arrow[dr] & B \arrow[d] & C \arrow[dl] \\ & O & \end{tikzcd}$$

O

Figure 04.a-Sequential Task **Figure 04. b-Parallel Task**

Output: To run the application code the following machine configuration with the installed software is used. Fig 04 illustrates the snap shot of sample output. Quad core machine with Windows 7 Operating system, Microsoft Visual studio 2010 Software, Intel C++ Compiler, Intel VTune Performance Analyser



```
0           0
1           1
2           4
3           9
4           16
0           0           0           0           0
1           1           2           0           1
2           4           6           -2          8
3           9           12          -6          27
4           16          20          -12         64
total time is 0.001099
Press any key to continue . . . _
```

```
    for(i=0;i<5;i++)
    {
        a3[i]=a1[i]+a2[i];
        a4[i]=a1[i]-a2[i];
        a5[i]=a1[i]*a2[i];

    }
    for(i=0;i<5;i++)
    {
        printf("%d\t%d\t%d\t%d\t%d\n",a1[i],a2[i],a3[i],a4[i],a5[i]);
    }
    e=omp_get_wtime();
    t=e-s;
    printf("total time is %f\t\n",t);
}
```

Figure 04. Snap Shot of Sample Output

Table 02. Time Measurement

Size of Vector	Sequential Time Sec(Single core)	Parallel Time Sec(2 cores)	Speedup
05	0.001099	0.001099	1
50	0.050781	0.066284	0.7661

100	0.118774	0.121460	0.97788
150	0.187988	0.189575	0.99162
175	0.184937	0.254517	0.72662
200	0.202148	0.232461	0.86956
250	0.207031	0.255127	0.81148

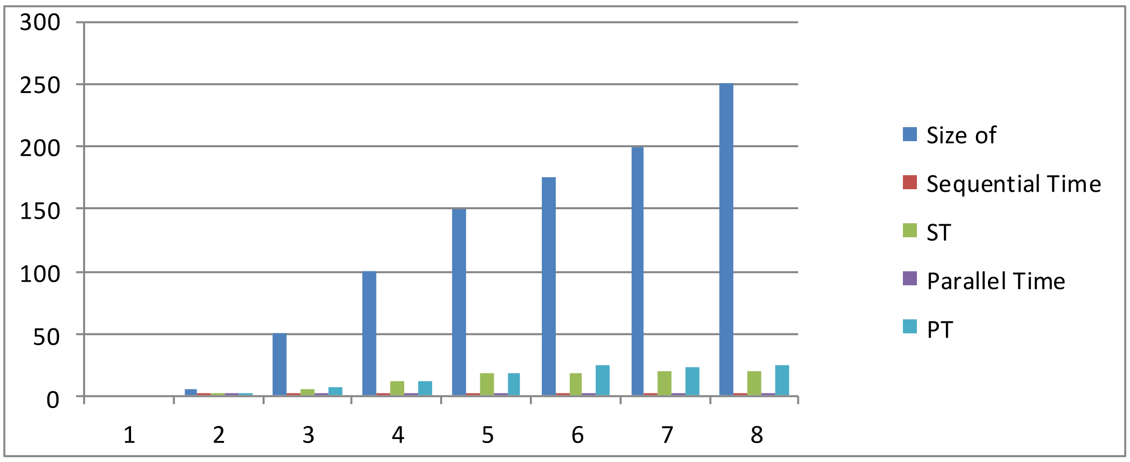


Figure 05. Analysis of Sequential and Parallel Run Time

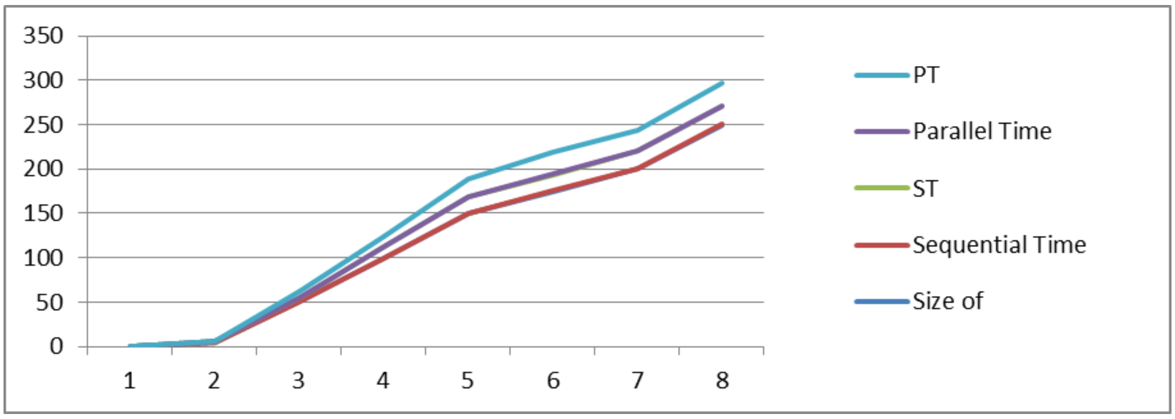


Figure 06. Analysis of Single Core and Multicore Execution Time

Conclusion and Future work:

Here in this paper we have studied the single core and multicore machine operations. We also analyzed a sample application code vector operation on single core and multicore machines. It is observed that for small "n" value we do not see any difference in serial and parallel computations. If the data size is more we get the benefit of parallel computing by running multiple tasks continuously with multiple cores. The future and or next generation computers are equipped with many core and GPGPU Systems. It has more benefits in terms of processing cores but it is also giving open challenge for the software developers to make use of multiple cores simultaneously for parallel computing. So it is time for us to learn the parallel programming model to address the increasing processor cores in the next generation computers.

References

[1] J. L. Hennessy, D. A. Patterson, "Computer Architecture: A Quantitative Approach", 2nd Edition, Morgan Kaufmann Publishing Co. (**1996)**

[2] K. Hwang, "Advanced Computer Architecture parallelism, scalability, programmability", Tata McGraw hill edition **(2006)**

[3] D. E.Culler, J. Pal Singh , A.Gupta, "Parallel Computer Architecture a hardware / software approach" , Elsevier .

[4] Frank Schirrmeister, "Multi-core Processors: Fundamentals, Trends, and Challenges', *Embedded Systems Conference* 2007 ESC351, Imp eras, Inc.

[5] Muti-Core Processors, The Next Evolution in Computing http://multicore.amd.com/Resources/33211A_Multi-Core_WP_en.pdf

[6] Intel Multi-core technology, http://www.intel.com/multicore/

[7] B.Guy,"An Analysis of Multicore Microprocessor capabilities and their suitability for current day application Requirements", Bowie University, Maryland in Europe, Nov **(2007)**.

[8] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software", Dr.Dobb's Journal, vol.30, no. 3, March **(2005)**.

[9] H. Sutter," The concurrency revolution" in C /C++ users Journal, vol. 23, no. 2, February **(2005)**.

[10] M. J.Quinn,"Parallel programming in C with Open MP", Mc Graw Hill Edition

[11] https://software.intel.com/en-us/blogs/2008/12/31/top-10-challenges-in-parallel-computing

[12] Overview of Performance Measurement and Analytical Modeling Techniq... http://www1.cse.wustl.edu/~jain/cse567-11/ftp/multcore/index.html Garrison Prinslow, gprinslow@gmail.com (A paper written under the guidance of Prof. Raj Jain)

[13] [www.openmp.org

[14] www.intel.com

[15] M.Narayana Moorthi, P.Mohan Kumar, Dr.J.Vaideeswaran.,"Overview of application performance on Multicore environment".,IJARCS, Volume 1,No.4,Nov-Dec **(2010)**

[16] R.Dashora, H. P. Bajaj, A. Dube, Narayanamoorthy M, "ParaRMS Algorithm:A Parallel Implementation of Rate Monotonic Scheduling Algorithm Using OpenMP", School of Computing Sciences and Engineering,VIT University,Vellore, India,dashora.rajnish@gmail.com, School of Computing Sciences and Engineering,VIT University,Vellore, India,harshpbajaj@yahoo.co.in, School of Electrical Engineering,VIT University,Vellore, India,akshatdube.28@gmail.com , School of Computing Sciences and Engineering,VIT University,Vellore, India,mnarayanamoorthy@vit.ac.in, IEEE Explore

[17] P. Samui,"Handbook of Research on Computational Techniques for Simulation based Engineering" IGI Global Publications – **(2015).**

[18] V.Rajaraman, C.Sivaram Murthy, "Parallel computers Architecture and Programming ", prentice hall of india(p) Ltd **(2003).**

[19] Seymour Lipschutz, Marc Lars Lipson, " Theory and Problems of Discrete Mathematics", second edition, Tata Mcgraw Hill Publishing company Ltd

[20] Vijay Anand Korthikarti,Gul Agra, "Analysis of parallel Algorithms for Energy Conservation in Scalable Multicore Architectures" , International conference on parallel processing **(2009)**.

[21] E. Seo, J. Jeong, S. park and J. Lee, "Energy Efficient scheduling of Real time tasks om Multicore Processors", IEEE Transactions on parallel and distributed systems,Vol-19,No.11,November **(2008).**

[22] W. yeen Lee, "Energy Efficient scheduling of periodic Real time tasks on lightly loaded Multicore Processors", IEEE Transactions on parallel and distributed systems, vol 23,No3,March (**2012)**