Simulated Performance Analysis of Multiprocessor Dynamic Space-Sharing Scheduling policy

Amit Chhabra[1], Gurvinder Singh[2] and Gaurav Kumar[3]

[1, 2&3] Department of Computer Science & Engineering,
Guru Nanak Dev University,
Amritsar-143001, Punjab, INDIA

Summary
Multiprocessor systems are the wave of the future rightly said because they offer tremendous potential to solve inherently parallel and complex computation intensive applications. In order to exploit the full potential of such computing systems, job scheduling or processor allocation (both are considered synonyms here) decisions plays a great role. Such scheduling decisions involves determining number of jobs to execute simultaneously as well as the number of processors to be allocated to each running application in a manner so as to minimize job's execution time and/or maximizing throughput. The growth of such multiprocessor systems has in turn paves the way for creation of efficient processor allocation policies in order to reduce job response time and make efficient utilization of system's processors. When we submit jobs or applications to multiprocessor system which in turn relies on job scheduling policies to allocate processors to such incoming jobs, we are really interested to know how well such policies are performing. Performance evaluation methodologies like actual experimental setup i.e. multiprocessor or parallel system, Theoretical/Analytical modelling and Simulation can be used to evaluate the performance of scheduling policies. Actual experimentation on multiprocessor or parallel system is still a costly and complex approach and moreover these systems are still out of reach to young researchers even doing research in higher education institutes like universities or technical colleges in developing countries India, Pakistan, Malaysia and Bangladesh etc. There may be several reasons for the non-availability of these systems. One can very well evade out theoretical/analytical modelling to be used for the same purpose due to their inaccuracy.
All these drawbacks have motivated us to switch towards virtualization or simulation of multiprocessor environment for the performance measurement of processor allocation policies. Simulation provides the powerful way to measure performance before the system under study has not actually been implemented. Such simulation can capture the dynamic interaction between applications and parallel architectures. Also it offers flexibility as one can make modifications to the simulation model and check their effect easily.

This paper is an effort to provide a GUI based simulated multiprocessor framework/environment for the performance measurement of dynamic space sharing scheduling policy. Virtualization of multiprocessor environment is carried out with the help of simulated program which simulates all components of actual multiprocessor so as to give best possible outcome. Such simulated framework will provide the stage for the young researchers to model and evaluate their scheduling policies on virtual multiprocessor environment. The intention behind this multiprocessor simulation environment is the necessity to facilitate the research of multiprocessor systems and performance measurement of scheduling algorithms in developing countries.

Key words:
Multiprocessor environment, dynamic scheduling policy, processor allocation, Simulation, and Performance evaluation.

1. Overview of Scheduling Mechanism in Multiprocessor Systems

The goal of maximizing the system performance has led to the development of the job schedulers in multiprocessor environment that match the requirements and workload with resource availability in terms of basic architecture and processors. Full benefits of parallelizing a problem will only be achieved if tasks of an application are properly scheduled to the available processors.

Scheduling or processor allocation in the light of multiprocessors involves determining the number of jobs to be executed simultaneously as well as the number of processors to be allocated to active jobs. An efficient processor allocation policy can result into proper utilization of system resources i.e. processors and also helps to achieve a considerable execution time for parallel jobs. In fact inefficient processor allocation policy will definitely leads to underutilization of system resources.
The processor allocation policy must be such that user submitted jobs would get services from the system resources without being hindered by the overall problems associated with the policy itself [1]. Thus there are two important properties of a policy. Firstly, user jobs should be able to efficiently access the resources and secondly, the overhead incurred for implementing the policy should be well utilized in terms of overall performance.

1.1 Objectives of Scheduling Algorithms

In the design of scheduling algorithms for efficient parallel processing, there are four fundamental aspects[2]:

Manuscript received February 5, 2009
Manuscript revised February 20, 2009

Performance, Time-complexity, Scalability and Applicability.

By high performance we mean the scheduling algorithms should produce high quality solutions. The algorithms must be robust so that they can be used under a wide range of input parameters. Scheduling algorithms should have low time-complexity. The time-complexity of an algorithm is an important factor so far as the quality of solution is not compromised. Parallel scheduling algorithms must be scalable. On the one hand, the problem should possess problem-size scalability, that is, the algorithms consistently give a good performance even for large input. On the other hand, the algorithms should possess processing-power scalability, that is, given more processors for a problem, the parallel scheduling algorithms produce solutions with almost the same quality in a shorter period of time. Scheduling algorithms could be used in practical environments. To achieve this goal one must take into account realistic assumptions about the program and multiprocessor models.

1.2 Classification of policies

In a broader way processor allocation policies [3][4] are classified into two categories:

1.2.1 Time sharing and space sharing

1.2.2 Static and dynamic scheduling

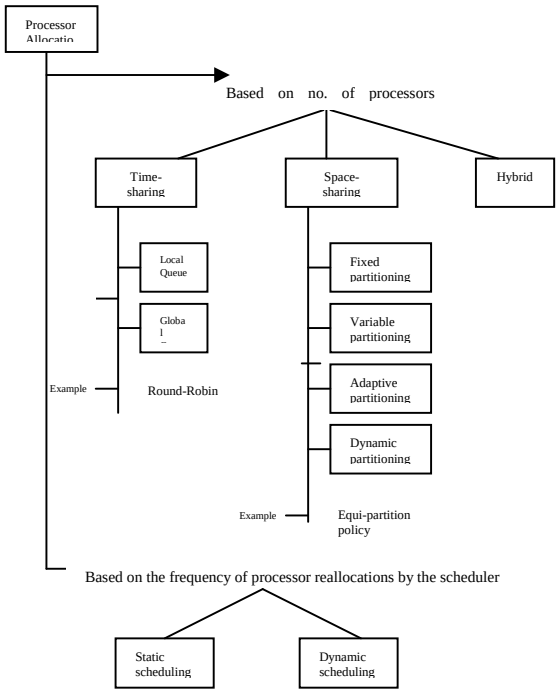

Fig. 1: Processor Scheduling Policies

1.2.1 Time Sharing and Space Sharing

These policies as shown in the Fig. 1 & Fig. 2 are differentiated among each other on the basis of number of processors to be allocated among the contending jobs.

In time sharing approach different applications are executed on same processors during different time intervals. Processors are time-shared among applications. In this approach

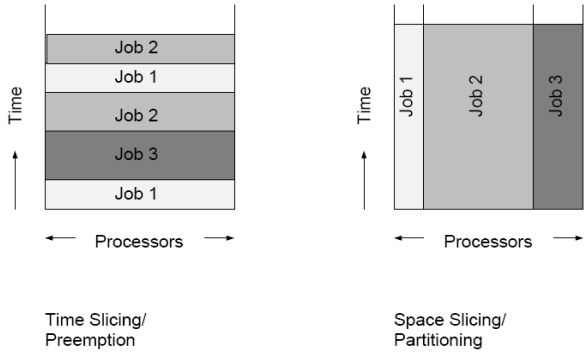

Fig. 2: Time sharing and Space Sharing Policies

1.2.2 Static and Dynamic Scheduling

Another classification of multiprocessor schedulers as shown in Fig.1 is based on the frequency of processor reallocations performed by the scheduler. The two flavours of such classification are the static and dynamic scheduling approaches. In *static* scheduling processors are allocated for the lifetime of the application. The allocated processors are not relinquished until the job is completed. Hence processors may not be effectively utilized especially when the variability in system load is quite high. These algorithms are simple to implement and have low scheduling overhead.

In *dynamic* scheduling, processors can be reallocated at any point during a job's execution. Changes in processor allocation usually occurs due to events such as the completion of an application (released processors are redistributed among other applications), arrival of an application (processors may be taken from other running applications in order to facilitate the immediate start of a new job), or due to changes in the parallelism of the job (changes in parallelism may result in an increase or decrease in the number of allocated processors). The frequent reallocation of processors introduces extra overhead due to context switches and the resultant loss of processor cache context. However the advantage of dynamic scheduling lies in its ability to adapt itself to changing system conditions.

2. Simulated Framework for Performance Measurement of Dynamic Scheduling Policy

Simulation[5] plays a vital role in multiprocessor studies. While the analytical modelling is often inadequate and hardware prototyping is costly and inflexible, software simulation has certain benefits that make it the dominant method for evaluating processor allocation policies and directing the development of new ideas. Software simulators are easier to make work, moderately accurate and less expensive than their hardware counterparts. They are more flexible allowing enhancements with new measuring features and real-world behaviour capturing within a matter of days.

One of the biggest advantages of simulation is that it offers flexibility as one can make modifications to the simulation model and checks their effect easily. We are also proposing simulated framework for the performance analysis of dynamic processor allocation policy. Such a simulated framework as shown in Fig.3 will be consisting of two main components or layers.

2.1 Simulated Multiprocessor Environment

First layer will be a simulator program developed using Visual Basic for the virtualization of actual multiprocessor environment. This simulated system environment is expressed in the terms of real multiprocessor system parameters (like no. of processors, memory and interconnection network), workload specification for dynamic scheduling policy and measures for execution time specification. This simulation provides effective and flexible environment for the evaluation of scheduling policy by thoroughly parameterising the system and its environment and uses random number generator for generating average arrival rates of jobs. It is being assumed here that arriving jobs are parallel in nature and can be easily broken in smaller parts. Following system and scheduling policy parameters have been simulated and taken care in the proposed framework

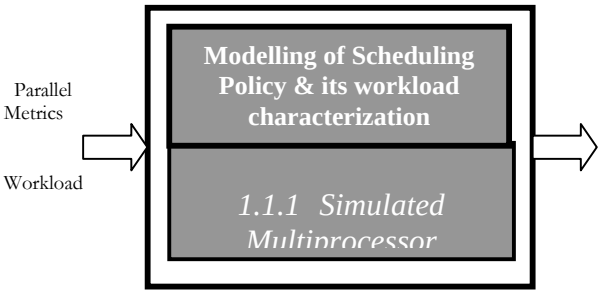

Fig 3: Showing main components of the proposed framework

System and scheduling policy parameters:

1. Incoming jobs of varying computational requirement
2. Number of processors in the system
3. Arrival rates of incoming jobs(Simulated using Poisson distribution)

2.2 Modelling of dynamic scheduling policy

Second layer deals with modelling of the dynamic scheduling policy in the simulated multiprocessor environment. Dynamic Equipartition is the policy that is being modelled in this environment for its performance measurement. Average completion time will be used as performance metric to evaluate the performance.

2.2.1 Equipartition- Equipartition [6][7] is a dynamic space-sharing policy proposed by McCann *et al*. The main goal of the Equipartition is to perform an equal allocation among running applications. Then the allocation number of each job is increased by one in turn, and any job whose allocation has reached the number of requested processors drops out. This algorithm continues until either there are no remaining jobs or until all P processors have been allocated. The only information provided by the application is the maximum number of processors (p^{max}) that it can use.

The behaviour of this algorithm is shown by the following Fig. 4:

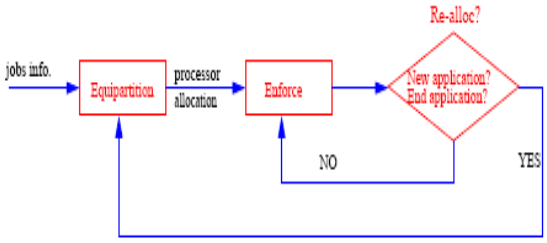

Fig. 4: Showing the behaviour of Equipartition algorithm

Once decided the processor allocation, it is maintained until a new application starts its execution or a running application finishes its execution. In that case, the Equipartition algorithm is re-applied. The problem [8] of Equipartition is that in most cases an equal allocation neither means *equal performance* nor *good performance*.

Some of the assumptions and policy parameters are as shown in Table 1:

Table 1: Showing various policy parameters and assumptions

Algorithm	Parameters	
Equipartition	p^{max}	Maximum no. of processors job can use
	P	Total no. of processors in the system
	J_1 & J_2	Jobs in the system
	P_1 & P_2	Maximum parallelism of job1 & job2
		Arrival time of incoming jobs generated using random number function using Poisson distribution
		Completion time is generated on the basis of execution time function which takes care of job's computational requirement and its arrival time
Assumptions		The computational requirement of each job is known a priori
		Process and Jobs are considered as synonyms
Performance evaluation metric to be used	*Avg. Completion time*	It is obtained by dividing the sum of completion times of all jobs at particular time by total number of active jobs at that time

Processor allocation for a job J_i in equipartition(*DEQUI*) will be min (p_i^{max},P/N). At any instant of time more than one process can enter into the scheduling system.

2.2.2 Relationship between Application and Simulator

There exists an interface program written in Visual Basic 6.0 as shown in Fig.5 that grabs the workload characteristics of the submitted application and passes this information to the simulated multiprocessor environment which also contains a scheduler of desired scheduling policy.

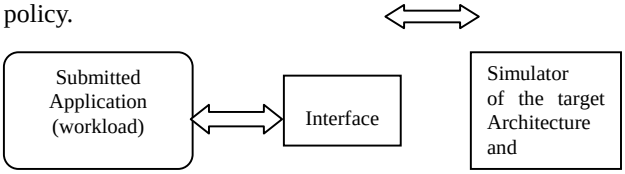

Fig.5 Relationship between Application and Simulator

3. Snapshots and Details about working of the Simulator

3.1 Characteristics of the simulator

1. **GUI based –** Easy to use & understand.
2. **Menu based-** Simulator has RUN menu which further have options to start/resume or pause the simulation program for the sake of capturing data from it.
3. **Data Backup-** Data generated by simulation is stored in MS-Access and can be used whenever needed.
4. **Graph generator**-By just clicking on Graph Generator button in simulator it generates the performance graphs in Excel worksheet. With the help of VBA Macro coded in Excel worksheet, data from MS-Access is passed to excel worksheet and hence graphs are generated from this data.
5. **Status Window** – Simulator has a status window at bottom left which tells the current status of simulator i.e. whether it is paused, resumed or initialized.
6. **Various parameters shown in List boxes and text boxes of simulator**- Various Parameters generated by Simulator after scheduling policy are being modelled in it are shown in various list and text boxes.
7. **Run-time performance measurement**-It measures the completion time as well as average completion time during run-time at regular intervals.

3.2 How simulator and modelled scheduling policy works:

1. Initially information about number of processors available is fed to the simulator as shown in snapshot Fig.6.
2. Then number of jobs/processes as shown in snapshot by Label **No. of processes arrived** keeps on coming at some time instant as shown in **AT Time** text box.
3. On the basis of number of process arrived it equally divides the number of processors among all the available processes with the constraint that no processes will get processors more than its computational requirement (p^{max}).
4. When processors are divided among processes they started giving response and at run time their completion times are measured at regular intervals. As information about completion times of various jobs at any instant is available, average completion time of the system is generated by dividing the sum of completion of all jobs by the number the active jobs.
5. Each process has got a unique process ID as indicated by label **Process ID**.
6. Processes which got terminated are shown by label **Terminated Processes.**

Various snapshots of the simulator are shown in Fig.6, Fig.7 & Fig.8

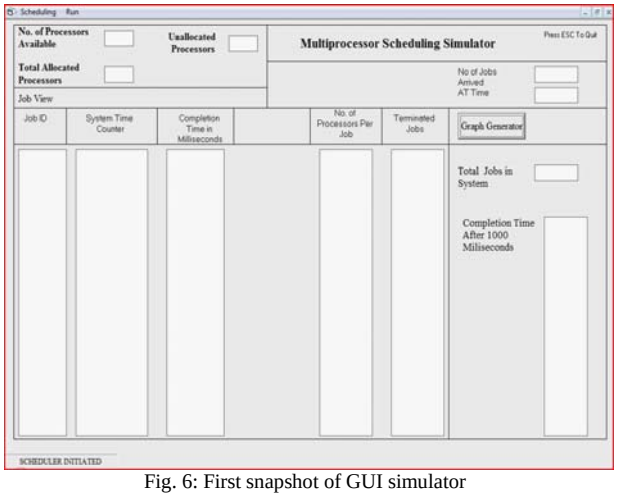

Fig. 6: First snapshot of GUI simulator

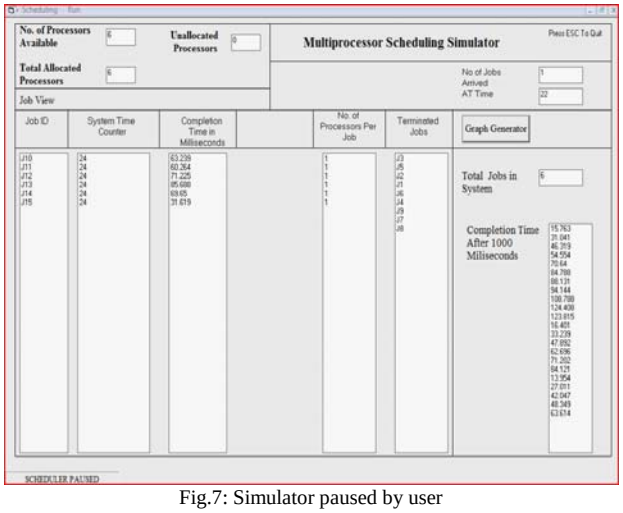

Fig.7: Simulator paused by user

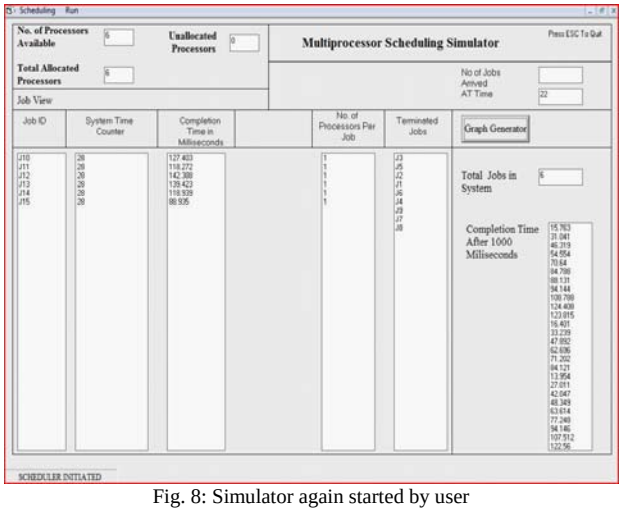

Fig. 8: Simulator again started by user

4. Results of the simulated experimental setup

Graph Generator button is responsible for capturing data from simulator as well as storing this data into MS-Access and with the help of a VBA Macro coded into MS-Excel worksheet ,the captured data is passed to Excel worksheet and later on graphs are generated from this data.

Different test cases are simulated by varying the number of processors available and accordingly data is collected as shown in Table 2.

Test case 1: Number of processors available (NOP) =4
Test case 2: Number of processors available (NOP) =6
Test case 3: Number of processors available (NOP) =8

Table 2 :Showing data captured from all the three test cases

Arri-val Time	Curr-ent Total Jobs (NOP=4)	Avg. Completi-on Time (NOP=4)	New Jobs Arri-ved (NOP=4)	Curr-ent Total Jobs (NOP=6)	Avg. Completi-on Time (NOP=6)	New Jobs Arri-ved (NOP=6)	Curr-ent Total Jobs (NOP=8)	Avg. Completi-on Time (NOP=8)	New Jobs Arri-ved (NOP=8)
1	4	15.231	4	3	18.229	3	8	13.256	8
2	4	31.865	0	3	33.135	0	8	26.174	0
3	4	48.067	0	3	51.610	0	8	40.924	0
4	4	63.526	0	3	65.337	0	8	57.250	0
5	4	80.069	0	2	71.044	0	8	72.908	0
6	4	96.180	0	1	67.905	0	8	86.979	0
7	4	108.695	0	6	14.778	6	8	102.296	0
8	3	120.162	0	6	30.426	0	7	117.046	0
9	2	129.940	0	6	46.202	0	6	131.014	0
10	1	136.813	0	6	62.152	0	7	126.583	1
12	1	12.563	1	6	76.310	0	5	125.429	0
13	1	29.310	0	6	92.392	0	5	142.574	0
14	4	17.065	4	5	103.887	0	5	115.164	0
15	4	31.071	0	4	115.789	0	4	108.158	0
16	4	45.690	0	4	128.890	0	3	100.211	0
17	4	59.085	0	5	117.409	1	4	63.603	0
18	4	74.421	0	4	125.900	0	3	34.837	0
19	4	89.706	0	4	143.387	0	3	48.165	0
20	3	96.298	0	4	110.919	0	4	48.936	1
21	3	112.033	0	3	92.561	0	3	58.144	0
22	2	122.584	0	2	56.440	0	2	55.870	0
23	1	118.015	0	2	68.698	0	1	49.574	0
25	2	15.272	2	1	55.742	0	7	16.744	7
26	2	29.963	0	2	11.706	2	7	31.571	0
27	2	43.944	0	2	27.229	0	7	46.729	0

Here NewJobsArrived =0 means no new job arrived at that particular instance of time.
CurrentTotalJobs = All active jobs in the system

Various Graphs generated from 3 test cases are shown in Fig 9, Fig.10 & Fig.11.

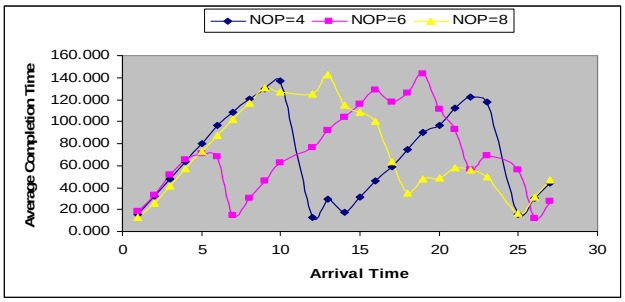

Fig.9: Showing Average Completion time for 3 test cases at any time instance

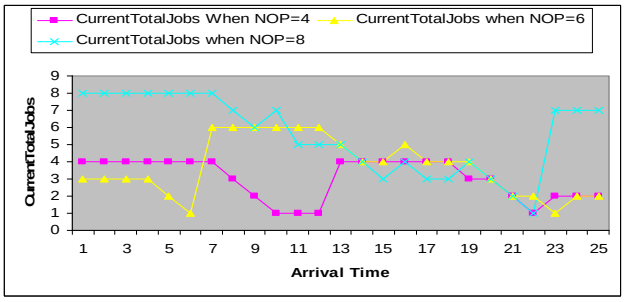

Fig. 10: Showing CurrentTotalJobs(Active jobs) for 3 test cases at any time instance

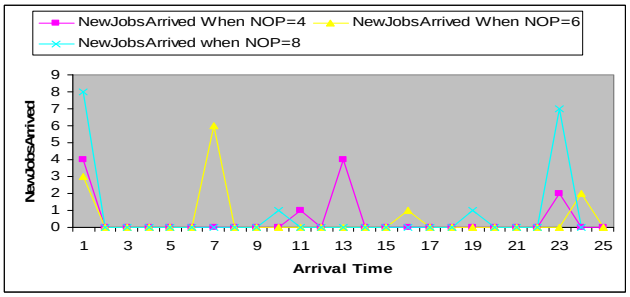

Fig.11: Showing NewJobsArrived for 3 test cases at any time instance

5. Conclusion and Future directions

Work done in this paper was an effort to design and develop a simulated multiprocessor environment so as to virtualize the actual multiprocessor system. This paper presents a multiprocessor simulation environment developed with the aim to facilitate the research of multiprocessor systems as well as performance measurement of scheduling algorithms in developing countries. A simulator program was coded in Visual Basic 6.0 to fulfil this purpose. Later on dynamic space sharing policy (dynamic equipartition) was modelled in this virtual environment and its performance was analysed by taking 3 different test cases. In future the work done in this paper can be extended by modelling many more dynamic processor allocation algorithms in the developed environment. Effort will be done in future to validate the data captured by simulator with actual experimental setup.

REFERENCES:

[1] T.L.Casavant and J.G.Kuhl, "A taxanomy of scheduling in General-purpose Distributed Computing Systems", in *IEEE transactions on software engineering*, February 1988, vol 14, No.2, pages 141-154.

[2] Yuen Chung Kwong, Series on scalable computing Vol 1, 2 & 3 Annual reviews on scalable computing, Singapore University press, World scientific.

[3] Timothy B. Brecht and Kaushik Guha, "Using Parallel Program Characteristics in Dynamic Multiprocessor Allocation Policies", *Performance Evaluation*, Volume 27-28, (October 1996), Pages: 519 – 539.

[4] Raghu Subramaniam, "A framework for parallel job scheduling", Ph.D thesis submitted of university of California, Irvine.

[5] Davor Magdic, "Limes: **A** Multiprocessor Simulation Environment for PC platforms", in *PROC. 21st INTERNATIONAL CONFERENCE ON MICROELECTRONICS (MIEL'97)*, VOL.2, YUGOSLAVIA, 14-1 7 SEPTEMBER, 1997.

[6] C. McCann, R. Vaswani and J. Zahorjan "A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors", in *ACM Trans. on Computer Systems*, 11(2), pp. 146-178, May 1993.

[7] A. Tucker and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors",In *Proceedings of the 12th ACM Symposium on OperatingSystems Principles*, pages 159–166, Dec. 1989.

[8] Julita Corbalán, Xavier Martorell and Jesús Labarta, "Performance-Driven Processor Allocation", in *IEEE Transactions on Parallel and Distributed Systems*, Volume 16, Issue 7 (July 2005), Pages 599 – 611.

Amit Chhabra has received his bachelor's and Master's degree in Computer Science from Guru Nanak Dev University, Amritsar. Author is working as a Lecturer in the Department of Computer Science & Engineering, Guru Nanak Dev University, Amritsar.

His research areas include Distributed Systems and Parallel Computing.

Gurvinder Singh has received his master's in computer applications and Ph.D from Guru Nanak Dev University, Amritsar. Author is currently working as a Reader in the Department of Computer Science & Engineering, Guru Nanak Dev University, Amritsar. His research areas include Distributed Systems, Parallel Computing and Grid Computing.

Gaurav Kumar has received his master's degree in computer applications from Guru Nanak Dev University,Amritsar. Author is working as a Lecturer in the Department of Computer Science & Engineering, Guru Nanak Dev University, Amritsar. His research areas include Advanced Computer Architecture and Object Oriented Systems Engineering