ISSN 2047-3338

Efficient Thread Mapping in Multicore Architecture with Laxity Based Algorithms

Prerana B. Jaipurkar[1] and Kapil N. Hande[2]
[1,2]Department of Comp. Science & Engg. Smt. Bhagwati Chaturvedi College of Engg., Nagpur
[1]preranajai@gmail.com, [2]kapilhande@gmail.com

***Abstract*– In real-time systems, a task needs to be performed correctly and timely. The correctness of each computation depends on both the logical results of the computation and the time at which results are produced. So "time" is most important in real-time application systems. Multicore and multithreaded CPUs becomes the new approach in real time system to achieve system performance, power efficiency, and software concerns in relation to application and workload characteristics. Multiprocessor Real time system requires an efficient algorithm to determine when and on which processor a given task should execute. The proposed work presents a comparative study of different customized Multiprocessor scheduling algorithms which maximizes system performance and decides the real time tasks that can be processed without violating timing constraints. A major advantage of the simulation is that it provides a fast and easy way to evaluate the system performance in Real-time system and consider tasks priorities which cause higher system utilization and lowers deadline miss time. To overcome the run-time scheduling and the prioritized-partitioned problems, accomplish a multiprocessor system on chip simulator which is capable of accurately simulating a variety of processor, memory, multiprocessor system on chip configurations and evaluate their effect on real-time system to improve the system performance with the help of different algorithms i.e. FPZL (Fixed Priority until Zero Laxity) & DPZL (Dynamic Priority until Zero Laxity).**

***Index Terms*– Real Time Operating System, Multi-Processor, Scheduling Algorithm, FPZL and DPZL**

I. INTRODUCTION

REAL-TIME (computing, communication, and information) systems have become increasingly important in everyday life. A real-time system is required to complete its work and deliver its services on a timely basis. Examples of real-time systems include digital control, authority and control, signal processing, and telecommunication systems. A real time - system provides a support on which to build and organize the features of the system. A real time system should provide services such as inter-task communication and time management and can solve a variety of problems that can occur in application code, logical correctness, and execution of various tasks delivered by the user since it provides multitasking capability and allows the application to be broken down into smaller pieces. Each task is assigned its own priority based on its importance, and pre-emptive scheduling tries to ensure that the CPU runs the task that has the highest priority among those that are ready-to-run. Real time systems are required to provide results within a specific time period. The correctness of the system depends not only on the logical results of the computations but also on physical instants at which these results are produced. Scheduling of tasks in a multiprocessor is to execute no of tasks for effective scheduling and high performance. The multiprocessing scheduling is done to minimize total time of program execution and also maximize processors utilization at the same time. The performance of each and every scheduling algorithm depends on performance parameters. The following are some performance parameters of the Scheduling algorithms.

- CPU Utilization
- Task Migration
- Number of Preemption
- Less Execution Time
- Resource Utilization

In addition to the CPU Utilization, there should be the best possible use of system resources in multi-processor environment. The rest of this paper is organized as follows. Section II describes related work and scheduling algorithms. Section III describes literature survey and existing methodologies. Section IV introduces the problem definition and proposed work. Finally section V concludes the paper.

II. RELATED WORK

In real-time systems, produced output is equally important as the logical correctness. That is, real-time systems must not only perform correct operations, but also perform them at correct time. A logically correct operation performed by a system can result in either an invalid, completely a waste of time, or degraded output depending upon the strictness of time constraints. Based on the level of strictness of timing constraints, real-time systems can be classified into three broad categories: hard real-time, soft real-time, and firm real-time systems.

[1]Prerana B. Jaipurkar, III Semester M.Tech, Department of Computer Science & Engineering, Smt. Bhagwati Chaturvedi College of Engineering, Nagpur University, India, (Email: preranajai@gmail.com)

[2]Assistant Professor Kapil N. Hande, Department of Computer Science & Engineering, Assistant Professor Smt. Bhagwati Chaturvedi College of Engineering, Nagpur University, India (Email: kapilhande@gmail.com)

In Hard Real-Time System requires that fixed deadlines must be met otherwise disastrous situation may arise whereas in Soft Real-Time System, missing an occasional deadline is undesirable, but nevertheless tolerable. System in which performance is degraded but not destroyed by failure to meet response time constraints is called soft real time systems. Such systems must be predictable and temporally correct. The designer must verify that the system is correct prior to runtime –i.e., for instance, for any possible execution of a hard real-time system, each execution results in all deadlines being met. Even for the simplest systems, the number of possible execution scenarios is either infinite or prohibitively large. Therefore, simulation or testing can be used to verify the temporal correctness of such systems. In the proposed work, following are the standard parameters that characterize tasks of real-time applications.

A Processor: A processor performs the major number of critical situation that drives any computer's operation. Processor plays such an important role that computers are often defined and described exclusively on the type of processor. Processors work by performing calculations based on specific instructions that software running on the computer. These instructions, which are loaded into the processor when an application runs, tell the processor how to manipulate amount of data stored in the computer's memory (RAM).In other words, processors are constantly merged through instructions and data that are loaded into it from the computer's memory.

A Multiprocessor: Multiprocessor system contains more than one such CPU, allowing them to work in parallel. This is called SMP, or Simultaneous Multiprocessing. As the multiprocessor architectures are already widely used, it becomes more and more clear that future real-time systems will be deployed on multiprocessor architectures. Multiprocessor architectures have certain new features that must be taken into consideration. For that application programs executing on different cores usually shared caches, interconnect networks, and shared memory bandwidth, making the conventional design practices not suitable to multi-core systems.

Cache: A small amount of high-speed memory residing on or close to the CPU as shown in Figure-1. In addition to working with the main memory, processors also work with a special type of high-speed memory referred to as *cache*. In fact, most of the time processors work directly with various types of cache memory and this cache memory, in turn, works with the main memory. Essentially, the cache memory acts as a high-speed buffer in between the processor and main memory, shuffling data into the processor as it needs it, or requests it. As a result, the processor takes advantage of the high-speed cache memory and therefore works faster, which, in turn, makes the computer that the processor drives, operate faster.

Cache memory supplies the processor with the most frequently requested data and instructions. Level 1 cache (primary cache) and Level 2 cache (secondary cache) is the cache second closest to the processor and is usually on the system board.

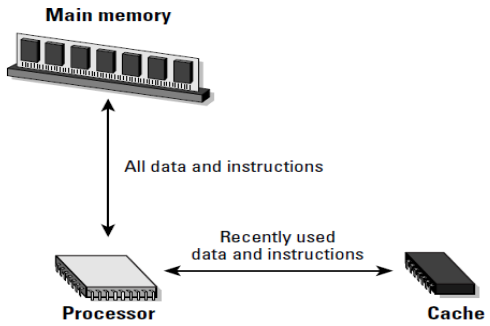

Fig. 1: Cache memory

Multitasking: In computing, multitasking is a method by which multiple tasks, shares common processing resources such as a CPU. Multitasking refers to the ability of the OS to quickly switch between each computing task to give the impression that different applications are executing simultaneously. As CPU clock speeds have increased steadily over time, not only do applications run faster, but OSs can switch between applications more quickly. This provides better overall performance. Many actions can happen at once on a computer, and individual applications can run faster.

Single Core: In a single CPU core, as shown in Fig. 2 tasks runs at any point in time, meaning that the CPU is actively executing instructions for that task. Multitasking solves this problem by scheduling which task may run at any given time and when another waiting task gets a turn.

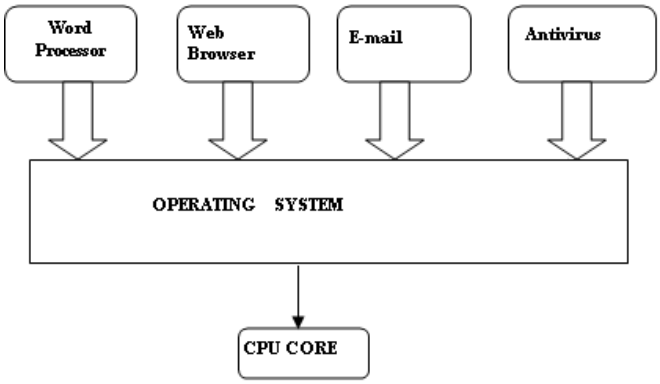

Fig. 2: Single-core systems schedule tasks on 1 CPU to multitask

Multicore: When running on a multicore system, multitasking OSs can truly execute multiple tasks concurrently. The multiple computing engines work independently on different tasks. For example, on a dual-core system, as shown in Figure-3,four applications - such as word processing, e-mail, Web browsing, and antivirus software - can each access a separate processor core at the same time and can multitask by checking e-mail and typing a letter simultaneously, thus improving overall performance for applications.

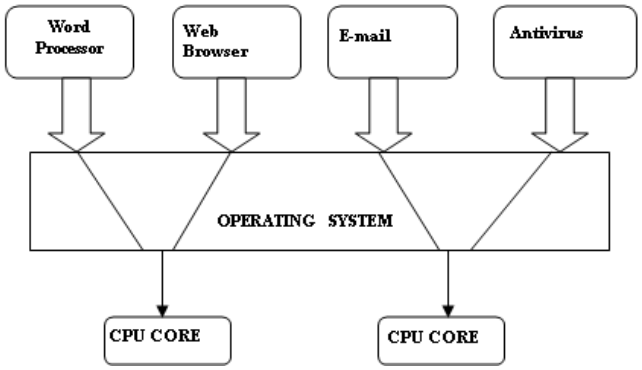

Fig. 3: Dual-core systems to execute two tasks simultaneously

The OS executes multiple applications more efficiently by splitting the different applications, or processes, between the separate CPU cores which shown in Fig. 4. The computer can spread the work - each core is managing and switching through half as many applications as before - and deliver better overall throughput and performance. In effect, the applications are running in parallel.

A *thread* is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers. Traditional processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time. Multi-threaded applications have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files. Multithreading extends the idea of multitasking into applications, so subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

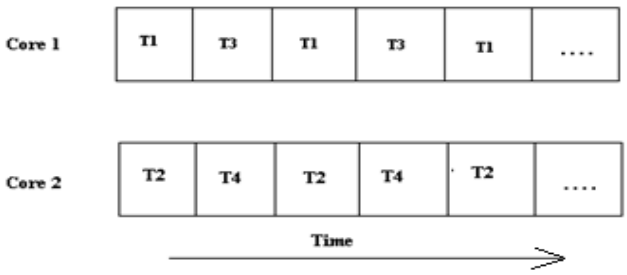

Fig. 4: Parallel Execution on Multicore System

In a multithreaded, an example that application might be divided into four threads - a user interface thread, a data acquisition thread, network communication, and a logging thread. All can prioritize each of these so that they operate independently which shown in Fig. 5. Thus, in multithreaded applications, multiple tasks can progress in parallel with other applications that are running on the system.

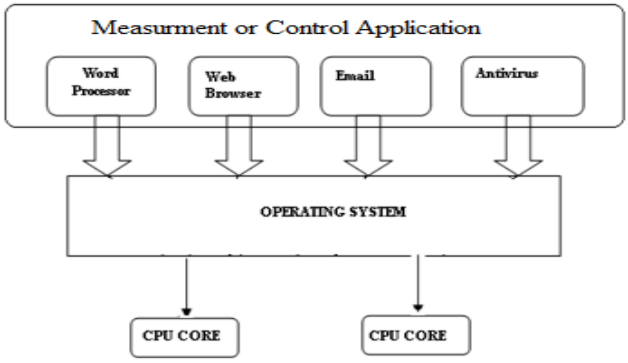

Fig. 5: Dual-core system enables multithreading

Applications that take advantage of multithreading have numerous benefits, including the following:

- More efficient CPU use
- Better system reliability
- Improved performance on multiprocessor computers

In many applications, a single-threaded request, a synchronous call effectively blocks, or prevents, any other task within the application from executing until the operation completes. Multithreading prevents this blocking.

While the synchronous call runs on one thread, other parts of the program that do not depend on this call run on different threads. Execution of the application progresses instead of stalling until the synchronous call completes. In this way, a multithreaded application maximizes the efficiency of the CPU because it does not idle if any thread of the application is ready to run.

A. Scheduling Algorithms

Earliest Deadline First Scheduling (EDF) algorithm assigned the highest priority if it is having the shortest deadline. The highest priority belongs to the task with the closest deadline while the task with the longest deadline has the lowest priority. Deadline of a task plays an important role in earliest deadline first scheduling and schedule the number of tasks on the processor.

Earliest Deadline First until zero laxity (EDZL) Scheduling algorithms is a hybrid preemptive priority scheduling scheme in which jobs with zero laxity are given highest priority and other jobs are ranked by their respective deadlines that a number of jobs missing their deadline are significantly reduced if scheduled by EDZL on m identical processors.

III. LITERATURE SURVEY

Jian Chen and Lizy K.John [1] proposed a scheduling model that heterogeneous multicore processors promise high execution efficiency under diverse workloads, and program scheduling is critical in exploiting this efficiency. This work presents a novel method to leverage the inherent characteristics of a program for scheduling decisions in heterogeneous Multicore processors. The method projects the core's configuration and the program's resource demand to a unified multi-dimensional space.

Zheng Wang Michael F.P.O'Boyle [2] describes that the thread mapping has been extensively used as a technique to efficiently exploit memory hierarchy on modern chip-multiprocessors. It places threads on cores in order to amortize memory latency and/or to reduce memory contention.

Kumar et al. [3] proposed a straightforward scheduling policy uses trial-and-error approach to find the match between programs and cores and a dynamic program scheduling approach.

Julian Bui, Chenguang Xu [4] states that cache memories are widely used in microprocessors to improve the system performance and several works have been done in cache fields. Cache size, cache protocols, associate numbers, etc. are all important parameters for performance.

Sherry Joy Alvionne [5] proposed a technique which to be used in multiple processors executing in parallel. Also, because of embedded systems have limited memory size, adding more functions in the system will limit the data that can be stored in the memory.

Chen and John [6] employ fuzzy logic to calculate the program-core suitability, and use that to guide the program scheduling. However, their method is not scalable since the complexity of fuzzy logic increases exponentially as the number of characteristics increases.

C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun [7] specifies that, the problem of reliability that is a serious threat to the current computer industry. While recent advances have embraced low-cost reliability solutions as a replacement for traditional high-cost full redundancy techniques, focused on single threaded workloads running on a single core.

Gulati et al. [8] uses efficiency threshold to dynamically allocate processor for the given task. All of these methods exploit intra-program diversity, and could adapt to program phase changes. In scheduling scheme exploits inter-program diversity and statically allocates programs to cores by analyzing inherent program characteristics.

M. Diener, F. Madruga, E. Rodrigues, M. Alves [9] in this, focused on how to improve barrier performance by either reducing memory contentions introduced by accessing shared flags within a barrier or by reducing the critical path of a barrier.

M. Bertogna, M. Cirinei, G. Lipari [10] presents the Fixed Priority until Zero Laxity (FPZL) scheduling algorithm for multiprocessor real-time systems. FPZL is similar to global fixed priority preemptive scheduling; however, whenever a task reaches a state of zero laxity it is given the highest priority.

A. Existing Methodologies

1) A Semi-Partitioned Fixed-Priority Scheduling (*SPFPS*)

The multiprocessor platform consists of M identical processors and N independent tasks. Each task is characterized by its Worst Case Execution Time (WCET), period and deadline. Tasks are sorted in non decreasing order. Task indices are used to represent the task priorities. A Semi-Partitioned Fixed-Priority Scheduling (SPFPS) algorithm with a linear utilization, selects the processor with the least workload assigned so far, among all processors to assign the next task. It applies Rate Monitoring Scheduling (RMS) algorithm on each processor to schedule the tasks. The constraint on the utilization of each task is removed by introducing an extra task pre-assigning mechanism.

2) Extended Boundary Fair (E-BFAIR)

Extended Boundary fair (E-Bfair) scheduler has best possible periodic tasks to examine in boundary fair (Bfair) scheduling algorithm. The next boundary time is determined by considering the irregular arrivals of sporadic tasks, to minimize the scheduling points. Since a periodic real-time task can only miss its deadline at its period boundary, an optimal discrete time based boundary fair (Bfair) scheduling algorithm which makes scheduling decisions and ensures fairness for tasks only at their period boundaries. A set of *n* sporadic real-time task is used where each task is characterized by its worst case computation requirement and minimum inter-arrival time with implicit deadlines. The allocation of processors to tasks should satisfy the following two constraints: a processor is allocated to only one task at any time and a task is allocated to at most one processor at any time.

3) Ant Colony Optimization (ACO)

ACO is a branch of Swarm Intelligence. The advantages of ant-based systems are inherent parallelism, robustness & scalability along with simplicity of individual agent.

ACO Based Scheduling Algorithm**:** The ACO based algorithms are computational models motivated by the collective foraging behavior of ants. Each ant is an autonomous agent that constructs a path. There might be one or more ants concurrently active at the same time. Ants do not need synchronization. Forward ant moves to the good-looking neighbor from the current node, probabilistically. A probabilistic choice is biased by Pheromone trails previously deposited and heuristic function. Without heuristics information, the algorithm tends to converge towards initial random solution. In backward mode, ants lay down the pheromone. Pheromone intensity of all the paths decreases with time, called pheromone evaporation. It helps in unlearning poor quality solution. ACO based scheduling algorithm is given as per following:

- Construct the tour of different ants
- Analyze the results of ant's journeys
- Update the value of Pheromone and find probability of each task and select the task for execution.

In ACO based scheduling algorithm, each schedulable task is considered as a node and all the ants will start their journey from different nodes. Number of ants are taken same as number of schedulable tasks at that time. The ants will traverse depending on the value of pheromone and some heuristic function. Pheromone value will be updated on each node depending on the performance of the journey and finally the task is selected with maximum probability of the best performance.

IV. PROBLEM DEFINITION AND PROPOSED WORK

A Real-Time System responds in a (timely) predictable way to unpredictable external inspiration arrivals. In short, a Real-Time System has to fulfill under extreme load conditions that is required to finish certain tasks within the time boundaries it has to respect. Also simultaneous processing that is more than one event may happen simultaneously, all deadlines should be met. For a given task set it is useful to know what changes can be made to complete a task that will result in a system which beneficial to know the speed of a processor that delivers a schedulable system. For scheduled systems on a uniprocessor: task execution time, speed of the processor, and also the performance does not satisfactory. So, proposed a scheduling approach to arrange real-time periodic and non-periodic tasks in Multiprocessor systems with static and dynamic optimal scheduling algorithms. In contrast, main approach is to balances task loads of the processors successfully and higher priority tasks running properly and overcome the following problems with multiprocessor system.

• *The run-time scheduling problem:* a set of tasks with real-time, find a schedule that meets simultaneous execution of multiple processes.

• *The prioritized-partitioned problem:* For a program to run on processor that must be split in a series of tasks which depends on each another or can be interdependent. The split implies partitioning and mapping, that distributes to each processor and can be static or dynamic.

A. Algorithms

To overcome multiprocessor run-time scheduling and prioritized-partitioned problems the following algorithms can be employed: 1) FPZL, 2) DPZL

- *FPZL (Fixed Priority until Zero Laxity)*

FPZL is based on the state of zero laxity. Zero laxity is the state where execution time is equal to its deadline. In this algorithm the task reaches a state of zero laxity it is given the highest priority. The priorities of all the tasks are considered as fixed until it reaches to a state of zero laxity.

Phase-1: This phase defines the fixed priorities. These priorities should not be changed any further.

1) Tasks are ordered in the increasing order of user priority that is from highest user priority to least user priority.

2) The first N task is added to the critical set such that the CPU load factor does not exceed maximum utilization & for the tasks present in the critical set critical priority is assigned 1 else critical priority is 0.

3) The tasks present in the critical set must be scheduled before the tasks in the non-critical set. This concept of scheduling reduces the irregularity of the system.

- *DPZL (Dynamic Priority until Zero Laxity)*

DPZL is a preemptive priority scheduling scheme in which jobs with zero laxity are given highest priority and other jobs are ranked by their respective deadlines.

Phase-2: This phase defines the dynamic priority. Dynamic priority is calculated at each clock cycle. Dynamic priorities are set according to the following steps;

1) When there is only one critical task, schedule it at any free processor without pre-emption.

2) When more than one critical task is present in the ready queue, schedule the tasks which assigns a higher priority to a task with a smaller laxity also split task is constrained to have a deadline equal to its computation time. The second part of the task has the maximum time available to complete its execution on a different processor.

3) Laxity at each clock cycle is computed for all the remaining processes in the ready queue as laxity= deadline - (execution time + current clock cycle)

4) Processes with zero laxity are available then these processes are assigned with higher priority over other process having non-zero laxity.

5) The process with zero laxity is scheduled at any available free processor without preemption.

6) When no free processor is available, and zero laxity process is present in the ready queue then preemption occurs. The process with the longest deadline is preempted and the zero laxity process is assigned to that processor.

7) After the execution of all zero laxity processes the remaining processes in the ready queue are assigned to the processor as per EDF scheduling policy.

B. Proposed Model

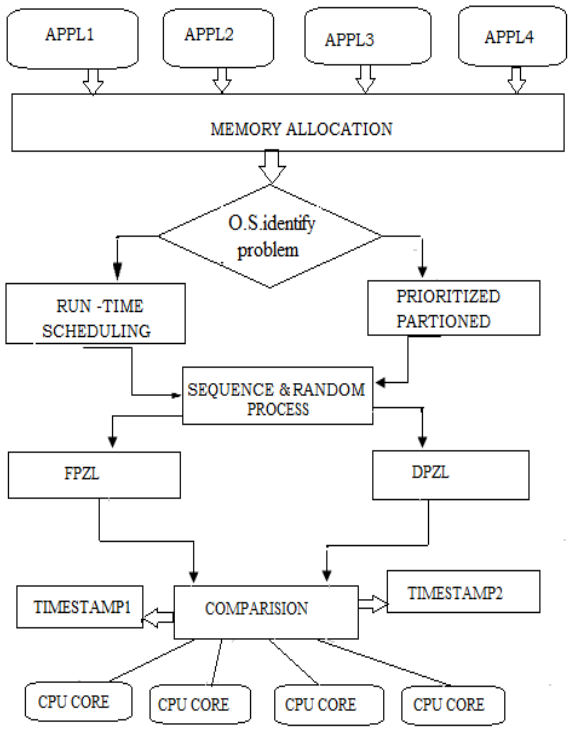

Fig. 6: Proposed Architecture

As shown in Fig. 6, the major factors consi**d**ered in approach to determine the scheduling of task priority, deadline, required computation time, and used CPU time. The notion of laxity is used in the proposed approach to facilitate the computation. Laxity is the maximum time that a task can wait before being executed (i.e., laxity = deadline - computation time). A task's priority shows the importance of the task. The inputs of these parameters are justified and represented to compute the level value for deciding which task to select to schedule next.

C. Simulation Approach for the System

A simulation of a system is the operation of a model of the system. Generally, a model proposed for a simulation study for that a mathematical model developed with the help of simulation software. The operation of the model can be studied, and hence, properties concerning the behavior of the actual system or its subsystem can be secondary. Simulation is a tool to evaluate the performance of a system, existing or proposed, under different configurations of interest over long periods of time. The steps involved in developing a simulation model.

- Identify the problem.
- Formulate the problem.
- Collect and process real system data.
- Formulate and develop a model.
- Document model for future use.
- Select appropriate experimental design.
- Establish experimental conditions for runs.
- Perform simulation runs and present results.

D. Simulation Model

Simulation models consist of the following components: system entities, input variables, performance measures, and functional relationships. For instance in a simulation model , the server and the queue are system entities, arrival rate and service rate are input variables, mean wait time and maximum queue length are performance measures, and 'time in system = wait time + service time' is an example of a functional relationship.

Real-time system (RTS) simulator for modeling task

1. Task creation module
2. Processor creation module
3. Task allocation module
4. Priority estimation module
5. System module to combine 1, 2, 3 & 4

E. Steps for the Working of Simulator

The working of simulator follows the steps shown in Fig. 7:

- Initially values for the tasks are assigned.
- Then information about number of processors is fed to the simulator.
- On the basis of number of tasks arrived divides the load among different processors.
- When tasks are divided among processors, starts to give response and at run time choose the scheduling algorithm.
- Run-time measures performance of the completion time as well as average completion time during run-time at regular intervals.
- Then start the simulation where jobs start running and get the response time, waiting time and computation time.
- When there is only one critical task, schedule it at any free processor without pre-emption.
- When more than one critical task, schedule the tasks which assigns a higher priority to a task with a smaller laxity and split tasks which is controlled to have a deadline equal to its computation time.
- The process with zero laxity is scheduled at any available free processor without preemption.
- The process with the longest deadline is preempted and the zero laxity process is assigned to that processor and the executions of all zero laxity processes in the ready queue are assigned to the processor.

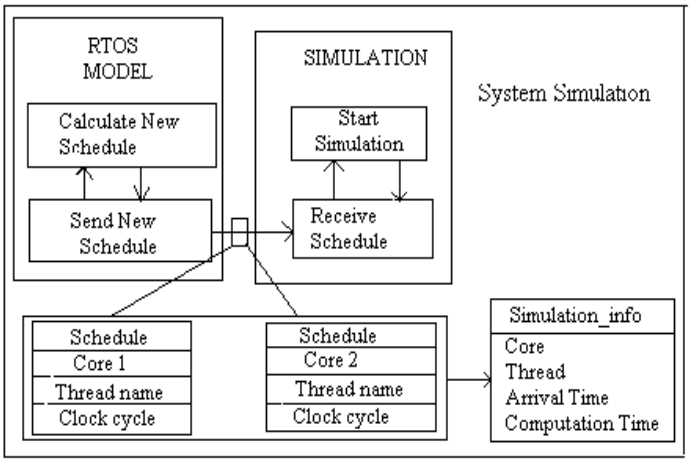

Fig. 7: Simulation Model

In modern operating systems, a Scheduling is a key concept to schedule no of tasks and running them on available CPUs for maximum utilization, which is carried out by a scheduler. Performance of scheduling algorithms in simulated condition deals with the best possible assignment of a set of tasks to the multiprocessor system and inform to their execution for minimizing the total completion time. When submits the no of tasks to the multiprocessors systems then really want to know how well such tasks are performing. So that switches towards a simulation and proposed work provides a simulated multiprocessor environment for the performance measurement and analysis of scheduling algorithms in Simulated Parallel environment. With this improve processor utilization, decrease the total system cost and power consumption, as well as improve fault tolerance. The proposed system, design and develop a simulated multiprocessor environment so as to virtualize the actual Scheduling system and to facilitate the study of multiprocessor systems as well as performance measurement of scheduling algorithms.

V. CONCLUSION

In real-time system, to overcome the run-time scheduling problem and the prioritized-partitioned problem implement a multiprocessor system on chip simulator which is capable of accurately simulating a variety of processor, memory, multiprocessor system on chip configurations and evaluate their effect on real-time system to improve the system performance with the help of different algorithms i.e. FPZL (Fixed Priority until Zero Laxity) & DPZL (Dynamic Priority until Zero Laxity). The main objectives of the proposed system to Simulate and evaluate effect of algorithm on real time system to improve performance and to increase efficiency and maximum utilization of the processor.

REFERENCES

[1] Jian Chen and Lizy K. John"Efficient Program Scheduling for Heterogeneous Multi-core Processors", *IEEE Micro*, pp 17-25, May 2008.

[2] M´arcio Castro, Lu´ıs Fabr´ıcio Wanderley G´oesy, Christiane Pousa Ribeiro, "A Machine Learning-Based Approach for Thread Mapping on Transactional Memory Applications", *IEEE*, pp. 978-1-4577-1950-2011.

[3] R. Kumar, et al, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction", *Micro-36, pp.* 81-92, Dec. 2009.

[4] Julian Bui, Chenguang Xu" Understanding Performance Issues on both Single Core and Multi-core Architecture" *IEEE Transaction Parallel Distribution System*, pp. 599–611, 2009.

[5] Jinkyu Lee, Arvind Easwaran, Insik Shin, Insup Lee, 2011. "Zero-Laxity based Real-Time Multiprocessor Scheduling", Journal of Parallel and distributed computing, 84, pp. 2324-2333.

[6] J. Chen and L. K. John,"Energy aware program scheduling in a heterogeneous multicore system", *IISWC'08*, pp.1-9, Sept. 2008.

[7] C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "Stamp: Stanford transactional applications for multi-processing," *in IEEE International Symposium on Workload Characterization*, 2008.

[8] D.P.Gulati et al., "Multitasking Workload Scheduling on Flexible Core Chip Multiprocessors", *PACT'08*, pp187-196, Oct. 2008.

[9] M. Diener, F. Madruga, E. Rodrigues, M. Alves, J. Schneider, P. Navaux, and H.-U. Heiss, "Evaluating thread placement based on memory access patterns for multi-core processors," *in IEEE International Conference on High Performance Computing and Communications,sept*., pp. 491–496, 2010.

[10] M. Bertogna, M. Cirinei, G. Lipari. "FPZL Schedulability analysis of global scheduling algorithms on multiprocessor platforms". *IEEE Transactions on parallel and distributed systems*, 20(4): 553-566.April 2009.

[11] H. S. Behera, Naziya Raffat, Minarva Mallik "A Modified Maximum Urgency First Scheduling Algorithm with EDZL for Multiprocessors in Real Time Applications" International *Journal of Advanced Research In Computer Science and Software Engineering, Volume 2, Issue 4,* April 2012.

[12] Komal S. Bhalotiya "Customized Multiprocessor Scheduling Algori for Real time Systems" *Proceedings published by International Journal of computer Applications,* 7-8 April, 2012.

[13] Sumedh.S.Jadhav & C.N. Bhoyar, "FPGA Based Embedded Multiprocessor Architecture", *International Journal of Electrical and Electronics Engineering (IJEEE) ISSN (PRINT)*: 2231 – 5284, Vol-1, Issue-3, 2012.

[14] Parisa Razaghi, Andreas Gerstlauer, "Host-Compiled Multicore RTOS Simulator for Embedded Real-Time Software Development," *Software Engineering, IEEE Transactions on*, 978-3-9810801-7-9, 2011.

[15] Sherry Joy Alvionne V. Sebastian," Implementation of Phase-II Compiler for ARM7TDMI-S Dual-Core processor" *In Proc. RTSS*, pp. 398-409, 2011.

[16] I-Yao Chuang,Tso-Yi Fan, Chi-Hung Lin, Chun-Nan &Jen-Chieh Yeh", HW/SW Co-design for Multi-core System on ESL Virtual Platform", pp.978-1-4244-8499-7, 2011.

[17] Abhinandan Majumdar, Srihari Cadambi, and Srimat T. Chakradhar," An Energy-Efficient Heterogeneous System for Embedded Learning and Classification" *IEEE embedded systems letters*, Vol. 3, No. 1, March 2011.

[18] M. Diener, F. Madruga, E. Rodrigues, M. Alves, J. Schneider,P. Navaux, & H.-U. Heiss, "Evaluating thread placement based on memory access patterns for multi-core processors," *in IEEE International Conference on High Performance Computing and Communications,sept.*, pp. 491–496, 2010.

[19] Julian Nita, Adrian Rapan, Vasile Lazarescu, "Efficient Program Scheduling for Heterogeneous Multi-core Processors", *IEEE Micro*, pp 17-25, May 2010.

[20] Antonino, Tumeo, Marco, "A Dual-Priority Real-Time Multiprocessor System on FPGA for Automotive Applications", 2008.