A Comparitive Study of Different Customized Multiprocessor Scheduling Algorithms on Multicore Architecture

Prof. Prerena Jaipurkar
Assi. Prof., Computer Engineering Department
SRPCE, NAGPUR (Maharashtra), India
preranajai@gmail.com

Prof. Pranali D. Tembhurne
Assi. Prof., Computer Engineering Department
SRPCE, NAGPUR (Maharashtra), India
pranali.tembhurne06@gmail.com

Prof.Sweta R Bhelonde
Assi. Prof., Computer Engineering Department
SRPCE, NAGPUR (Maharashtra), India
shwetabhelonde@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 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 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.

Index Terms *– Real Time Operating System, Uniprocessor, Multi-Processor, Scheduling Algorithm, Context Switch, Multicore.*

*****

I. INTRODUCTION

The Central Processing Unit (CPU) is the heart of the computer system so it should be utilized efficiently. For this purpose CPU scheduling is very necessary. CPU Scheduling is one of the fundamental concepts of Operating System Sharing of computer resources between multiple processes is called scheduling [1].

A. Scheduling on Uniprocessor

Uniprocessor platforms include one processor on which number of jobs can be executed. In uniprocessor scheduling, central state information of the entire task states accurately. In a single processor multi-programming system, multiple processes/tasks are contained within memory. Processes survive between Running, Ready, waiting, Blocked, and Suspend. A main goal is to keep the processor busy, by allocating task to the processor to execute, and always having at least one process able to execute. To keep the processor busy is the main purpose of process scheduling. Uniprocessor scheduling is categorized as follows:

- Long-term scheduling: To add to the processes those are fully or partially in memory.
- Short-term scheduling: The decisions as to which process to execute next or in future.

The goal of the scheduling in Uniprocessor is to achieve high processor utilization, high throughput, number of processes completed per unit time and low response time. But Uniprocessor scheduling affects the performance of the system, because it determines which process will wait and which will progress.

B. Scheduling on Multiprocessors

Multiprocessors platforms include more than one processor on which jobs can get executed. The approaches to multiprocessor real-time scheduling can be categorized into two classes: partitioned and global. Under partitioning, the set of tasks is statically partitioned among processors, that is, each task is assigned to a unique processor upon which all its jobs execute. In contrast to partitioning, under global scheduling, a single system, priority is used, and a global ready queue is used for storing ready jobs. The performance of each and every scheduling algorithm depends on performance parameters like deadline of a task, release time of a task, execution time of a task, laxity of a task, CPU utilization of a task, number of preemption, resource utilization etc and all the tasks will be scheduled according to their unique assigned priority scheduling. It takes decisions to introduce new processes for execution or re-execution.

The various CPU scheduling algorithms are
•FCFS (First Come, First Serve) CPU Scheduling :

In this scheduling the process that request the CPU first is allocated to CPU first.

•SJF (Shortest Job First) CPU Scheduling :

In this scheduling the process with the shortest CPU burst time is allocated to CPU first.

•Priority Scheduling :

In this scheduling the process with high priority is allocated to CPU first.

•Round Robin Scheduling :

RR scheduling is used in timesharing systems. It is same as FCFS scheduling with preemption is added to switch between processes. A static Time Quantum (TQ) is used in this CPU Scheduling

The various scheduling parameter for the selection of the scheduling algorithm are :

•Context Switch :

A context switch is process of storing and restoring context (state) of a preempted process, so that execution can be resumed from same point at a later time. Context switching is wastage of time and memory that leads to the increase in the overhead of scheduler, so the goal of CPU scheduling algorithms is to optimize only these switches.

•Throughput :

Throughput is defined as number of processes completed in a period of time. Throughput is less in round robin scheduling. Throughput and context switching are inversely proportional to each other.

•CPU Utilization :

It is defined as the fraction of time cpu is in use. Usually, the maximize the CPU utilization is the goal of the CPU scheduling

•Turnaround Time :

Turnaround time is defined as the total time which is spend to complete the process and is how long it takes the time to execute that process.

•Waiting Time :

Waiting time is defined as the total time a process has been waiting in ready queue.

•Response Time :

Respond Time is better measure than turnaround time. Response time is defined as the time used by the system to respond to the any particular process. Thus the response time should be as low as possible for the best scheduling.

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.

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.

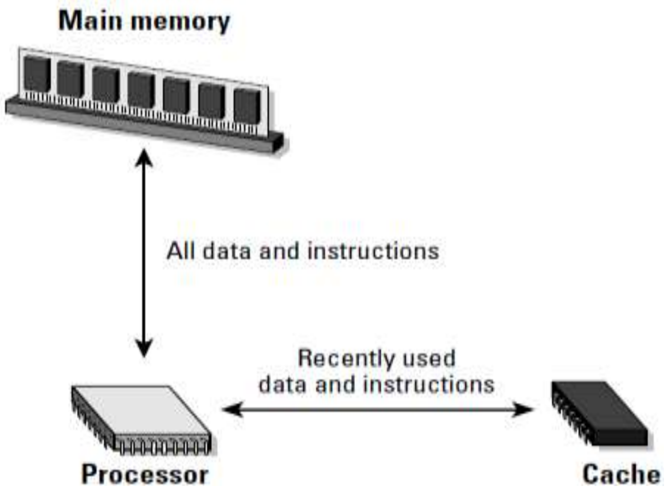

Main memory
All data and instructions
Recently used data and instructions
Processor
Cache

Figure. 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.

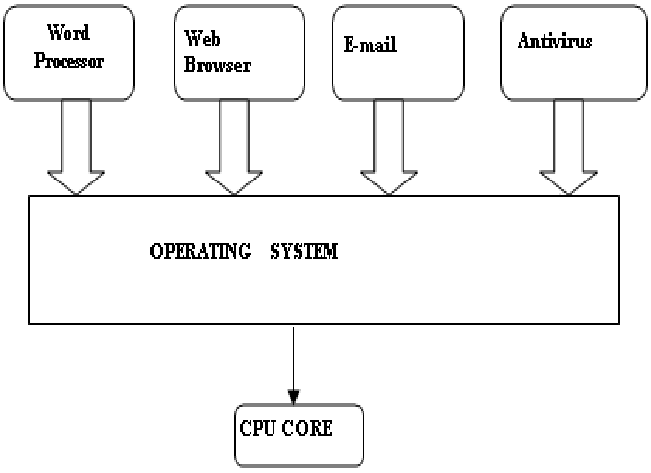


Figure. 2: Single-core systems schedule tasks on 1 CPU to multitask

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.

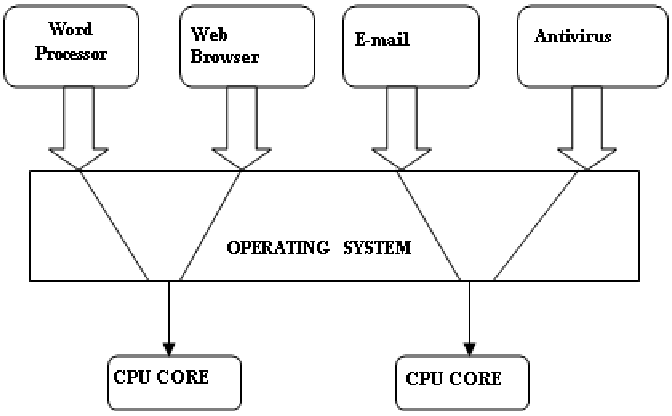
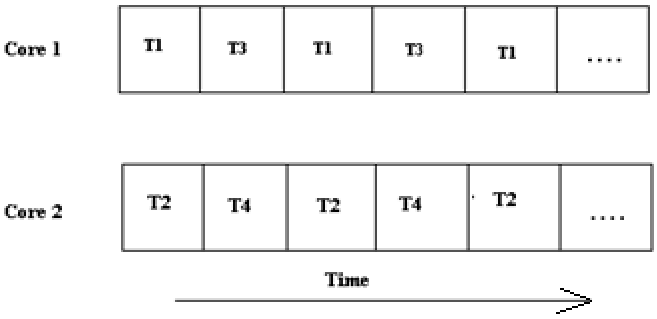


Figure. 3: Dual-core systems to execute two tasks simultaneously

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.

Thread **:** 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.



Core 1	T1	T3	T1	T3	T1	....
Core 2	T2	T4	T2	T4	T2	....

Figure. 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.

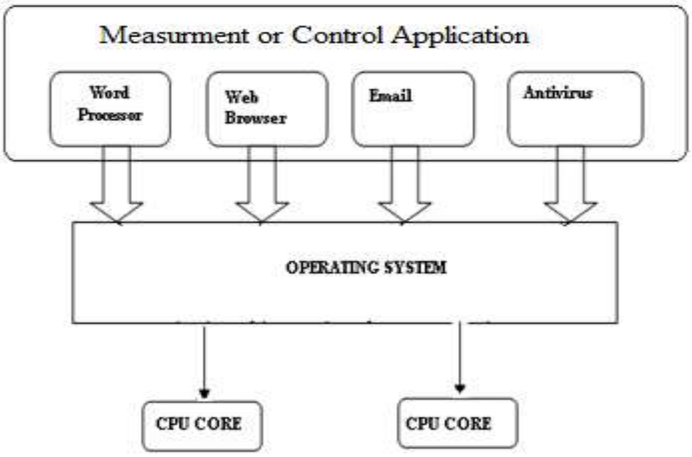


Figure 5: Dual-core system enables multithreading

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.

IV. 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] Prerana B. Jaipurkar1 and Kapil N. Hande2, " Efficient Thread Mapping in Multicore Architecture with Laxity Based Algorithms", ***I***nternational ***J***ournal of ***C***omputer ***S***cience and ***T***elecommunications [Volume 3, Issue 12, December 2012]

.