A Review Of Fault Tolerant Scheduling In Multicore Systems

Shefali Malhotra, Parag Narkhede, Kush Shah, Samanth Makaraju, M. Shanmugasundaram

Abstract: In this paper we have discussed about various fault tolerant task scheduling algorithm for multi core system based on hardware and software. Hardware based algorithm which is blend of Triple Modulo Redundancy and Double Modulo Redundancy, in which Agricultural Vulnerability Factor is considered while deciding the scheduling other than EDF and LLF scheduling algorithms. In most of the real time system the dominant part is shared memory.Low overhead software based fault tolerance approach can be implemented at user-space level so that it does not require any changes at application level. Here redundant multi-threaded processes are used. Using those processes we can detect soft errors and recover from them. This method gives low overhead, fast error detection and recovery mechanism. The overhead incurred by this method ranges from 0% to 18% for selected benchmarks. Hybrid Scheduling Method is another scheduling approach for real time systems. Dynamic fault tolerant scheduling gives high feasibility rate whereas task criticality is used to select the type of fault recovery method in order to tolerate the maximum number of faults.

Keywords: multicore processor, fault tolerant, dynamic scheduling, check-pointing, Earliest Deadline First, Task Graph, etc.

1. INTRODUCTION

As the Semiconductor Technology is growing day by day it's now possible to have a billion of transistors on a small chip. These small transistors are more susceptible to both transient and permanent faults. Consequently, by increasing the budget of the chip, Multicore design is popular for enhancing the framework throughput. By expanding the quantity of transistors and contracting their sizes, the rate of Software blunders on processors can't be disregarded; consequently, the dependability of such frameworks has been a standout amongst the most essential difficulties as littler transistors are more susceptible to faults.Time constraints, energy efficiency throughput are the important criterion in the design process of real time systems [1]. A multi-core processor has two or more independent cores into a single package. A dual core processor has two independent cores and quad core has four cores. The advantages of multicore processor unit over the many single core processors unit are (1) higher throughput, (2) linear power consumption, (3) efficient utilization of processor cores, (4) high performance per unit cost [2].

- *Shefali Malhotra, is currently pursuing master's degree program in Embedded Systems in VIT University, Vellore, India. E-mail: shefali.malhotra2014@vit.ac.in*
- *Parag Narkhede, is currently pursuing master's degree program in Embedded Systems in VIT University, Vellore, India. E-mail: narkhede.paragsuresh2014@vit.ac.in*
- *Kush Shah, is currently pursuing master's degree program in Embedded Systems in VIT University, Vellore, India. E-mail: 3shahkush.prakashkumar2014@vit.ac.in*
- *SamanthMakaraju, is currently pursuing master's degree program in Embedded Systems in VIT University,Vellore, India. E-mail: makaraju.samanth2014@vit.ac.in*
- *M. Shanmugasundaramis currently working as Assistant Professor (Sr) in Embedded Systems Division, School of Electronics Engineering in VIT University, Vellore, India. E-mail: mshanmugasundaram@vit.ac.in*

ARM MPCore and IBM Cell are the examples of multicore processors employed in the real time embedded systems. Multicore processors are classified into two parts, (1) homogenous or heterogeneous [3]. [4] states that most of the existing multicore processors are homogenous. The multicore processor is mainly concern for managing the tasks in such a way to utilize the cores effectively. The scheduler in the operating system is responsible for keeping all the cores in the processor busy during the execution of the real time tasks to improve the total execution time. Faults can be categorised into these main categories: permanent, transient and intermittent faults [5]. Permanent Fault such as wear off of any part which require replacement from the spare part to restore the system functionality. Transient fault are the short term faults and can be distinguished from others with their duration of occurrence and causes. These may occur due to external noise and other sources Intermittent fault occurs happened at interims on account of some inward tedious glitch of the segments like temperature vacillations, power supply and noise . To have fault tolerance in multicore systems task scheduling is done. These fault tolerance scheduling algorithms increase the system reliability. In different fault tolerant system, software which is running on a single core employ redundant execution at different level of abstraction, at instruction and virtual machine level. Methods which operate at instruction level have low error detection latencies compare to hardware level. But methods which work at process level allow error propagation. In multi-threaded programs which are running on multicore processors, Shared memory access is frequent than interrupts or signals. For that achieve efficient execution of replicas is very much difficult. For that different deterministic languages are used. We can perform fault tolerance using redundant execution of software in which replicated copies give same output for given input. This method can be implemented using a user level library so it does not require modification in kernel. The error detection mechanism is optimized to perform memory comparisons of the replicas efficiently in user space [6]. Depends on the use of multiple threads performance can be further increase. For scheduling of soft real time tasks with non-real time tasks two level hierarchical scheduling is used. This method decreases average deadline miss ratio and also support the real time requirements for other tasks. The principle objective of hardware based algorithm is to discover a fitting assignments task on the cores and tolerate processing component failures which could either be rogeneous or heterogeneous. The heterogeneity of the processors implies that they have diverse velocities or transforming abilities. The transient flaw likelihood that may happen in transistors, entryways and even a bit, is called Architectural helplessness variable (AVF) [5]. By averaging over some time, this component can characterize the rate of soft errors that can show up on a core; while it runs a task. This algorithm when compared with other techniques we found that the proposed fault tolerant method outflanks both TMR and DMR methods about 35% in average. In computing most common topologies in N-Modulo Redundancy are TMR and DMR. TMR method is used when reliability of task is important as it can only mask the faults. The problem is that voter can also be faulty. The other method DMR includes two cores running parallel and checks for the similarity in the output thus only indicate the mismatch doesn't tolerate the fault so have to use separate mechanism for fault recovery [5]. In software based fault tolerance based approach replica of process are used so it is necessary that replicas use same memory addresses. We also need to ensure that leader and follower use same replica copies. Replica can be created using fork system call in which process generates same process as follower process and it works same as leader process. If the result of both leader and follower process is same then we can say that there is no error. But if there is difference between leader and follower process then error is there. We can use check points for efficiency. If after execution of leader and follower process result is same then previous checkpoint is eliminated and if result is different than process is again started from previous checkpoint. We can also use regular interval methods for error detection. In that leader and follower processes are compared after fixed interval of time say 100ms. If we find difference it means fault detected. In multicore architecture, no comparator hardware is used, therefore task replication plays a key role in their architecture. Choosing a check pointing with rollback recovery, increases the probability of completing the task on time [7]. In the memory allocation, malloc can be used for memory allocation. Malloc can be non-deterministic because it uses mmap for allocation of memory blocksof large sizes and mmap itself is non-deterministic. So we have to make sure that leader and follower use same memory using mutex, which is locked and unlocked deterministically.Many researches are going on scheduling of periodic and aperiodic task. One of the classical scheduling mechanism is on the basis of priority of the task [8]. Authors in [9] introduces scheduling method which uses the rate motonic scheduling in which the priority of the task is defined on the basis of the time required. The task having shorter time period have the highest priority and assigned the core first. The Earliest Deadline First algorithm in which the task with earlier deadline has the highest priority. An alternate scheduling approach is Primary Backup (PB) that is a standout amongst the other methodologies, in which two version of the task is schedule on two different cores. An acceptance test is required for validation of results.

2. SCHEDULING METHODOLOGIES

This paper discuss about various scheduling algorithm for multi core system.

A. Hybrid Scheduling

The algorithm discussed in [5] is the combination of TMR and DMR fault tolerance scheduling algorithm in which by using the Agricultural Vulnerability Factor (AVF) of each task the scheduler can predict the occurrence of fault when the task is running on the core. Each task is defined as a 4-tuple, $T_i = (a_i, d_i, c_i, AVF_i)$ where a_i, d_i, c_i and AVF_i denote arrival time, deadline, execution time and architectural vulnerability factor of each task. For effective scheduling all the tasks must meet their deadline and each core should have one task assigned to it. Thus the efficiency of algorithm is evaluated by total execution time of the task and the utilisation of the core. Core utilisation can be calculated as:

$$\text{Utilisation} = \frac{\sum_{i=1}^{n} \text{Pi}}{\text{N} \times \text{T}} \qquad (1)$$

Where, P_i denotes the time each core is busy with running different tasks, N determines number of cores and T is total execution time of the task group. As per [10], in Hybrid Scheduling Method approach one can partition application in to maximum number of parallel tasks so that best result can be achieved. Each parallel task can run concurrently with other tasks. Here It may occur that real time tasks and non-real time task can run concurrently. In this hybrid scheduling method, two level scheduling policy is used. At the top level sporadic server is used for scheduling policy and at bottom level, a rate-monotonic OS scheduler is used.

B. Implementation Method

First, sorting of the tasks in waiting queue are done on the basis of their arrival time. Than the number of cores required for task to run in safe mode is determined by comparing AVF of each task to the $AVF_{threshold}$. If the task AVF is greater than or equal to $AVF_{threshold}$than it will executed in TMR mode or else in DMR mode. If any faultoccurs than task is re executed in DMR mode [5].

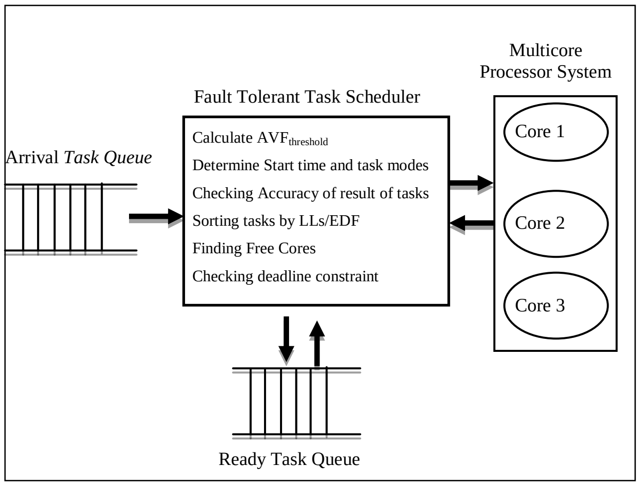
Multicore Processor System
Fault Tolerant Task Scheduler
Arrival Task Queue
Calculate AVFthreshold
Determine Start time and task modes
Checking Accuracy of result of tasks
Sorting tasks by LLs/EDF
Finding Free Cores
Checking deadline constraint
Core 1
Core 2
Core 3
Ready Task Queue

Figure 1. *The Schematic of Fault tolerant task scheduling algorithm for multicore systems.*

C. Non-preemptive EDF Scheduling

There are several dynamic priority scheduling algorithmssuch as Earliest-Deadline-First (EDF), Least-Laxity (LL), LeastSlack-Time-First (LST), and Minimum-Laxity-First (MLF), where in each one tasks are prioritized and scheduled according to specific parameter. As per [1] EDF is an optmal algorithm for single processor system with a set of preemptiveindepedent tasks.[11] proves that the sufficient but not the necessary condition to have a feasible scheduling for a number of preemptive task with total utilization U on a set of M single core processor is

$$U \ll \frac{M}{2M-1} \qquad (2)$$

This can be considered as a boundary for multicore processor with M cores. Whenever a task is released, it will be added toReadyList, and its absolute deadline (AD) will be calculated. When there is only one idle core, the system checks whether the task with the earliest deadline can be scheduled on this idle core or not. If the task can be scheduled, it will be assigned to the core, and the core will be assumed busy during the execution. Otherwise, missing the task deadline means that scheduling of this application on the given architecture is not feasible. Consider an example of a task set, consisting of six tasks is shown in figure 2. Itshows the EDF scheduling of this task set on a quad-core processor. From the figure, it can be seen that,T2 has least AD so T2 is scheduled firston P1, following that T1is scheduled on P2, T3 onP3. The competition is observed between T4 and T5 where both are released at the same time, but the absolute deadline of T5 is earlier than T4,soT5is scheduled first. The other tasks will be scheduled on idle cores in the same manner [1].

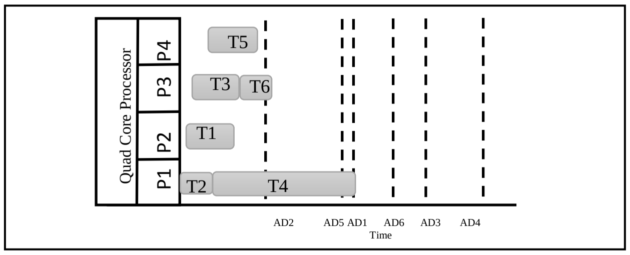

Figure 2: *EDF task scheduling of a task set on a quad-core processor*

Reexecution and recovery with checkpoint are more appropriate to be utilized with soft real time systemswhereas time redundancy methods are appropriate for hard real time systems with tight deadlines.However, check pointing and re-execution method increases the total execution time.In [12],real-time scheduling refers to the problem in which there is a due date or time connected with the execution of an undertaking. Author's present a scheduling scheme, called earliest deadline with energy guarantee (EDeg) that combinely accounts for characteristics of the energy source, capacity of the energy storage as well as energy consumption of the tasks, and time.

D. Check Pointing Optimization

There are trade-offs between applying frequent- and infrequent checkpoints for tasks in a system. Frequent check pointing decreases re-execution time in the presence of faults, while task execution time is increased. On the other hand, infrequent check pointing has lower time overhead in the absence of faults, whereas the amount of re-execution will be increased if a fault is detected. In [13], it is shown that the optimalnumber (m) of checkpoints considering k faults in the task is given by:

$$m = \left|\left| \sqrt{\frac{K \times C}{Cs}} \right|\right| \qquad (3)$$

Where, Cs is time overheads of saving a checkpoint, Cr is time overhead of recovering from a checkpoint. The worst-case response time of a task forCheckpointing with rollback recovery (Cc) can be given by

$$Cc = (C + m \times Cs) + K \times (Cs + Cr) + \frac{K \times C}{m+1} \qquad (4)$$

Where (C+m x Cs) gives the execution time of a task using checkpointing without any faults, and (Cs+Cr)+C/m+ 1 is the cost offault recovery for single fault, (see also[p18]).If the rate of occurrence of a fault on one core is high, then a task needs more time to recover from faults, this may lead to miss its deadline. In such cases hardware replication methods are used as they have the ability of parallel execution of the replicated copies of original tasks on the other processing cores.

E. Harvesting Aware Real-Time Scheduling Algorithm

In [14] in real time embedded system along with scheduling of the tasks power management is also an issue that should be undertaken. In this paper, authors propose a harvesting aware real-time scheduling algorithm which aims to lessen the energy consumption while attainably plan the arrangement of intermittent tasks inside their due date (deadline).This can be done by Dynamic Voltage and frequency selection, executing the task within the speed such that it can consume as much energy as required for its completion meeting its deadline.

F. Dynamic Voltage and Frequency Selection Algorithm

In [15] this paper Author's propose a low-multifaceted nature and task assignment mapping, scheduling, and power management method for multi-core real-time embedded systems with energy harvesting. This method is based on the CPU utilisation. This method combine new dynamic voltage and frequency selection algorithm along with the energy harvesting awareness form the proposed utilisation based algorithm. On multi core system this algorithm gives effective utilisation of harvested energy.

G. Dynamic Fault Tolerant Scheduling

[1] describes the dynamic fault tolerant scheduling (DFTS) algorithm. This algorithm uses task criticality which is based on "utilization" and "time of resource allocation". Task utilization is used to dynamically select the type of fault recovery method in order to tolerate the maximum number of faults. Based on the task criticality parameters, task is categorized into critical and non-critical. Critical tasks will be replicated on separate cores to increase the probability of on time task completion in faulty condition. Non critical tasks are scheduled on a single core and check pointing

with rollback recovery fault tolerant technique is applied on them. The scheduling feasibility rate of the DFTS algorithm is higher than other fault tolerant scheduling methods.

H. Fault-Tolerant Scheduling Based On Task Criticality

The algorithm mentioned in [1] selects a suitable fault tolerance technique assignment for each task when the resources for the task are available. During scheduling it is to be taken care that no tasks will be scheduled until all other tasks with higher priorities are scheduled and the previously occurred faults in the system are tolerated. The scheduler calculates the criticality threshold for the task which is present at the first position of the ready list as soon as ideal core is available in the system. The time when an ideal core is allocated to each task is defined as Resource Allocation Time (RAi).

$$\text{Delay}_i = RA_i - R_i \qquad (5)$$

Delay indicates the time wasted between the task entered into Ready list and time when the scheduler assigns hardware resources to that task and applies fault tolerant policies. Increasing delay for a task may lead a noncritical task becomes critical. So in order to calculate the criticality threshold and tolerate the expected faults in each task, the scheduler applies check-pointing technique to non-critical tasks whereas hardware replication technique to critical tasks.

I. Task Graph Scheduling Using NABBIT

As discussed in [16], this algorithm depends on someof the information from the user about the task graph like Task key, Sink task, Predecessors and Successors. After providing this information the task graph scheduling algorithm captures the structure of the task graph. This algorithm is built on NABBIT task graph scheduler which is based on the work stealing. In the task graph the tasks are referred by keys and the runtime through a concurrent hash map will control the execution. A created task is inserted into the hash map using the INSERT TASK IF ABSENT routine and later with a call to GETTASK. For every task the runtime holds the join, notify array and status to proceed further. The task execution begins with creation and insertion of the sink task into the hash map, followed by an invocation of the INITANDCOMPUTE function. INITANDCOMPUTE starts the task and forwards its immediate predecessors through calls to TRYI NITCOMPUTE. Invoking INITANDCOMPUTE and TRYINITCOMPUTE in a recursive fashion, the execution expands the task graph and reaches one of the source tasks with no incoming dependencies after executing the task updates its status as Computed and starts notifying the successors involved in its notify array. After the last successor in the notify array, the task changes its status to COMPLETED.

3. ANALYSIS OFSCHEDULING TECHNIQUES

Scheduling Algorithm	Characteristics
Hybrid Scheduling	This task scheduling algorithm can promise the right execution of running task and decline aggregate time of task execution by expanding the cores while considering AVF of each task.
Non-Pre-Emptive EDF Scheduling	EDF is an optimal algorithm for single processor system with a set of pre-emptive independent tasks
Check Pointing Optimization	check pointing decreases re-execution time in the presence of faults, while task execution time is increased
Harvesting Aware Real-Time Scheduling Algorithm	This algorithm decreases the energy consumption while attainably plan the arrangement of intermittent tasks
Dynamic Voltage And Frequency Selection Algorithm	This is the low-multifaceted nature and task assignment mapping, scheduling.
Dynamic Fault Tolerant Scheduling	scheduling feasibility rate of the DFTS algorithm is higher than other fault tolerant scheduling methods
Fault-Tolerant Scheduling Based on Task Criticality	Here, task criticality is used to select the type of fault recovery method in order to tolerate the maximum number of faults.
Task Graph Scheduling Using NABBIT	This algorithm can efficiently recover from an arbitrary faults that are proportional to the amount of work lost

4. CONCLUSION

In this paper, numerous fault tolerant scheduling algorithms applicable for multicore processor systems are discussed. According to the system requirements the scheduling technique should be choosen. But all algorithm of real time scheduling have some restriction so it is future's need to suggest some hard real-time scheduling algorithm which will decrease the energy and also timing overhead by utilizing speed in such a way that response time of task is less than or simply equivalent to the existing approach despite the fact on the cost of lesser energy consumption.

REFERENCES

[1] Mohammad H. Mottaghi, Hamid R. Zarandi, "DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors", Microprocessors and Microsystems 38 (2014) 88–97.

[2] F. Kong, W. Yi, Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multi-cores", in: Design Automation and Test in Europe, 2011, pp. 1–6.

[3] Saifullah, K. Agrawal, C. Lu, C. Gill, “Multi-core real-time scheduling for generalized parallel task models”, in: 32nd IEEE Real-Time Systems Symposium (RTSS), 2011, pp. 217–226.

[4] Chen, L.K. John, “Efficient program scheduling for heterogeneous multi-core processors”, in: Design Automation Conference (DAC), 2009, pp. 927–930.

[5] ShamimShiravi* and Mostafa E. Salehi, “Fault Tolerant Task Scheduling Algorithm for Multicore Systems”,The 22nd Iranian Conference on Electrical Engineering (ICEE 2014), May 20-22, 2014,pp. 885–890.

[6] Hamid Mushtaq, Zaid Al-Ars, Koen Bertels, “Efficient Software-BasedFault Tolerance Approach on Multicore Platforms”, EDAA, 2013

[7] Ying Zhang and KrishnenduChakrabarty, “Fault Recovery Based on Checkpointing for Hard Real-Time Embedded Systems”, Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’03) , 2013

[8] S. Gotoda, M. Itoa and N. Shibata, “Task scheduling algorithm for multicore processor system for minimizing recovery time in case of single node fault,” 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp.260,267, 13-16, 2012

[9] Ching-Chih Han, K.G. Shin and J. Wu, “A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults,” IEEE Transactions on Computers, vol.52, no.3, pp.362,372, 2003

[10] Pengliu Tan, Jian Shu and Zhenhua Wu, A Hybrid Real-Time Scheduling Approach on Multi-Core Architectures, JOURNAL OF SOFTWARE, VOL. 5, NO. 9, SEPTEMBER 2010, pp 958-965.

[11] R.I. Davis, A. Burns,”A survey of hard real-time scheduling for multiprocessor systems”, ACM Comput. Surv. 43 (4) (2011) (Article 35).

[12] Agrawal, S., Yadav, R. S. and Ranvijay. “A Pre-emption Control Approach for Energy Aware Fault Tolerant Real Time System”, International Journal of Recent Trends in Engineering, 381-386,2009.

[13] Y. Zhang, K. Chakrabarty,”A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems”, IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 25 (1) (2006) 111–125.

[14] Bertogna, M. and Baruah, S., “Limited Preemption EDF Scheduling of Sporadic Task Systems”, IEEE Transactions on Industrial Informatics, 579 – 591, 2010.

[15] Dehghan and Maryam, 2010. “Adaptive checkpoint placement in energy harvesting real-time systems”, 18th Iranian Conference on Electrical Engineering (ICEE), 932 - 937.

[16] Mehmet Can Kurt, SriramKrishnamoorthy, Kunal Agrawal and Gagan Agrawal, “Fault-Tolerant Dynamic Task Graph Scheduling”, SC14, November 16-21, 2014, New Orleans IEEE, 2014.