Schedulability of Real-Time Fault-Tolerant Systems Using Multi-cores

Risat Mahmud Pathan, Jan Jonsson, Johan Karlsson
Chalmers University of Technology, SE-412 96, Göteborg, Sweden
{risat, janjo, johan}@chalmers.se

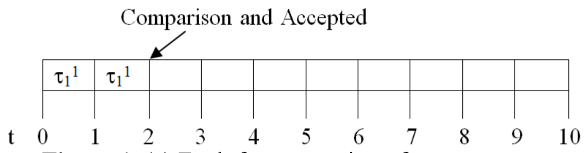
Abstract: *Fault-tolerance is a crucial aspect of safety critical systems. When such systems need to meet strict timing requirements, achieving fault-tolerance is more challenging. Considering the highly frequent transient faults, we propose redundant execution of real-time tasks to mask at most F transient errors in safety critical systems. Rate-Monotonic scheduling of a set of n preemptive real-time periodic tasks in multi-cores tolerating transient fault is the focus in our work.*

Introduction: Use of computers has made the automotive and aerospace industry to contemplate shifting from mechanical to electrical control by introducing brake-by-wire and fly-by-wire systems. In such safety-critical *hard real-time systems* missing the deadline of a task can pose threat to human lives. Moreover, when faults occur in such system, recovery from the error before deadline of a task is must. We consider tolerating *transient faults* in this work. Transient faults are temporary malfunctioning of a computing unit have shown to occur at much a higher rate than permanent faults [1]. In this paper, the well known *duplication and comparison* paradigm, and redundant execution of task are used for transient error detection and recovery [2]. If the number of faults and task execution time is relatively large, use of multi-cores having more computational power can provide better schedulability of real-time tasks. In this paper, we propose the rate-monotonic scheduling of a set of *n* preemptive periodic tasks that mask the effect of F transient errors in multi-cores. For brevity, the detail equations to generate the redundant tasks and the scheduling algorithm for multiple cores are not outlined here. Interested readers can find the details in [3].

Motivation: Building more powerful uniprocessors with increasing transistor counts has ceased due to limited instruction level parallelism, increased wire delay and latency to main memory access. Now the trend in processor industry is to accommodate many processing cores in the same die area, called chip multi processor (CMP) [4]. CMPs can provide better performance to many applications in terms of throughput. For example, to application software, a Sun's Niagara processor will appear as 32 discrete processors [5]. If such a processor is used for real-time task scheduling, a total of 32 real-time tasks can be scheduled in parallel. Using our proposed error detection and recovery technique, redundant copies of real-time tasks can run in parallel. Such inherent parallelism of real-time periodic tasks tolerating transient faults for safety critical real-time systems is the best target for execution in multi-core processors.

Task model: The real-time task set we consider consists of *n* periodic tasks, $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$. Each task τ_i has a period T_i, and a relative deadline D_i, worst case execution time C_i and priority P_i. According to rate-monotonic static priority assignment, task with shorter period has higher priority. The relative deadline is considered to be equal to its period in this work, that is T_i=D_i. The length of the Planning Cycle (PC) in which the task schedule repeats iteratively is the least common multiple of all task periods, that is PC=lcm$\{T_1, T_2, \ldots, T_n\}$. Within one PC, one or more instances of task τ_i will execute. Each *task instance* is denoted by τ_i^j where j is the jth instance of task τ_i. The utilization of a task τ_i is defined as, U_i=C_i/T_i. The utilization of the whole task set is $U=\sum_{\tau i} C_i/T_i$.

Error Detection and Recovery: Our error detection and recovery technique works as follows: when a task is released, two copies, known as *primary copies,* of the same task instance are run first. If an error is detected by comparison of results, or by other error detection mechanism, F more *extra/recovery copies* of the same task instance are run to mask at most F errors. Figure 1(a) and Figure 1(b) demonstrate this for F=2 and for a single task τ_1 with period T_1=10 and execution time C_1=2.



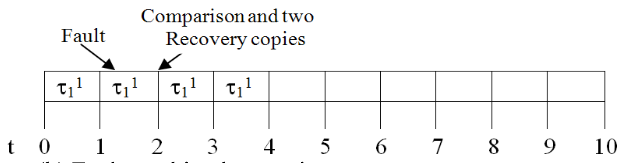


Figure 1. (a) Fault free execution of two *primary copies* (b) Fault masking by running two more *recovery copies*

Parallelism Exploitation by Multi-cores: The inherent parallelism that can be exploited by multi-cores is demonstrated here using an example. Consider a real-time system having two tasks τ_1 and τ_2 as in Figure 2(a), and F=1. Also consider that, one of the two primary copies of τ_1 is in error. The Rate-Monotonic schedule is in Figure 2(b) with recovery copy running from t=2 to t=3. The second instance of the first task, τ_1^2 (the error free instance of τ_1), finishes at time t=5. The first instance of task τ_2^1 does not have two time units within PC for execution of its two primary copies. So, the task set is ***not*** schedulable. Now, observe that the two primary copies of each task can run in parallel in two processing cores. Considering this inherent task level parallelism, the periodic task model we consider has higher schedulability for multi-cores. Figure 3 shows the rate-monotonic schedules in two processing cores for the task set in Figure 2(a). In the schedule in Figure 3, the task τ_2^1 is schedulable and other tasks have low response time and high slack is available in the schedule that can be used to tolerate more transient faults. Next question is how to generate such parallel tasks set for execution and the algorithm to schedule the tasks in multiple cores?

τ_i	T_i	C_i
τ_1	3	1
τ_2	6	1

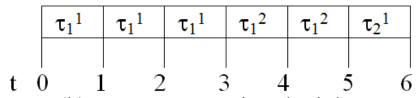


Figure 2. (a) Example task set (b) Rate-Monotonic schedule

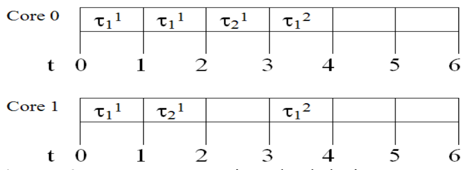


Figure 3. Rate monotonic schedule in two cores

Generating parallel tasks: There are three steps to generate the parallel task set; (i) **Step 1:** identify the set of primary copies of each task, (ii) **step2:** find the worst case distribution of F faults within PC and determine the set of recovery copies of tasks to mask F errors, (iii) **step3:** combine the task sets from step 1 and 2 as a set (called ***EXE***) of triplets ***EXE***={(Task Release time, Task Index, Task Execution Time)}. For the task set in Table 1, the set ***EXE*** with F=2 is, ***EXE*** ={(0,τ_1^1, 4), (0,τ_2^1, 2), (4,τ_1^1, 4), (6,τ_2^1, 2), (7,τ_1^1, 4), (11,τ_1^1, 4)}. Please see [3] about the details to generate task set ***EXE***.

Table 1: Task Set

τ_i	T_i	C_i
τ_1	7	2
τ_2	14	1

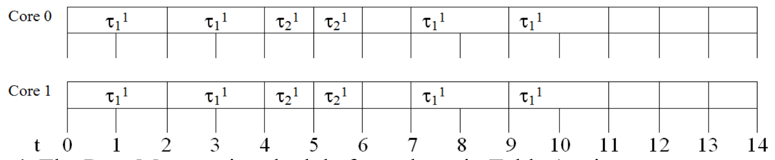


Figure 4. The Rate Monotonic schedule for task set in Table 1 using two cores

Rate-Monotonic Scheduling: Once the set ***EXE*** is found, next task is to schedule the tasks. In [3], a Rate-Monotonic algorithm to schedule task set ***EXE*** is outlined to find the minimum number of cores to tolerate F errors. At each time instant the highest priority task in set ***EXE*** is identified, and the task is executed before the deadline in a free processor as shown in Figure 4. The algorithm iteratively searches for required number processing cores for a feasible schedule.

Schedulability Test: Generating the task set ***EXE*** and then iteratively search for suitable number of cores is computationally expensive if there are a large number of tasks. A simple *utilization based test* can determine schedulability of a task set in linear time. A utilization based test guarantee schedulability of *any* task set if the total utilization of the task set is less than or equal to a particular fraction (known as *achievable utilization bound*) of the total processing capacity. Until year 2001, it was known for multiprocessor that the achievable utilization bound is 0% for hard real-time systems due to the well known *Dhall's effect* [6]. The essence of Dhall's effect is that even if we have infinitely many processors available, there are hard real-time task sets that can have almost 0% utilization and still unschedulable. In 2001, Andersson, Baruah and Jonsson in [7] have proved that if total utilization of the task is less than or equal to 33% of the capacity of all processors, the rate-monotonic schedule is feasible. They have also proved that more than 50% utilization is never possible no matter how powerful processors are used for scheduling of hard real-time tasks. We are currently working on achieving more than 33% utilization bound. We have observed that, if the ratio of any two task period is greater than 2, the achievable utilization is 40%. Moreover, if the maximum utilization of any single task is small, higher utilization is also possible to achieve. As periods or task utilization have strong impact on schedulability, we are trying to find a relationship among different task parameters to find a good utilization bound for multi-cores.

Conclusion: The inherent parallelism within real-time periodic tasks can be exploited by multiple cores to achieve high reliability of safety critical real-time systems. Redundant execution of tasks can be used for error detection and recovery in today's powerful multi-cores to tolerate transient faults. In addition more slack would become available in the schedule and real-time task set not schedulable in uniprocessor becomes schedulable using multi-cores even if when faults occur. We are investigating issues like, given a number of *m* cores and a task set, what is the maximum number of faults that can be tolerated, and what utilization bound we can have for *m* cores with *F* transient faults.

REFERENCES

[1] A. Campbell, P. McDonald, and K. Ray, *Single Event Upset Rates in Space*, IEEE Trans. On Nuclear Science, 39(6): 1828-1835, Dec, 1992.
[2] J. Aidemark, *Node-Level Fault Tolerance for Embedded Real-Time Systems*, Ph. D. Thesis, Chalmers University of Technology, 2004.
[3] R. M. Pathan, *Fault-Tolerant Real-Time Scheduling using Chip Multiprocessor*, in proceeding of supplemental volume, student forum paper, EDCC-7, pp. 97-102, Lithuania, 7-9 May, 2008. (*please email the author for a copy*)
[4] L. Hammond, B. A. Navfeh, K. Olukotun, *A Single-Chip Multiprocessor*, IEEE Computer Special Issue on "Billion-Transistor Processors", September 1997.
[5] P. Kongetira, K. Aingaran, K. Olukotun, *Niagara: A 32-Way Multithreaded Sparc Processor*, IEEE Micro, Vol 25, No 2, Mar.Apr. 2005, pp 21-19
[6] S. K. Dhall and C. L. Liu, *On a Real-Time Scheduling Problem*, Operation Research, vol. 26, no. 1, pp. 127-140, 1978.
[7] B. Andersson , S. Baruah , J. Jonsson, *Static-Priority Scheduling on Multiprocessors*, Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01), p.93, December 03-06, 2001.