Reducing the Cost of Redundant Execution in Safety-Critical Systems using Relaxed Dedication

Brett H. Meyer*, Nishant George†, Benton Calhoun†, John Lach†, and Kevin Skadron*
Computer Science Department* and Electrical and Computer Engineering Department†
University of Virginia
Charlottesville, VA 22904 USA
{bhm, skadron}@cs.virginia.edu, {niche, bcalhoun, jlach}@virginia.edu

***Abstract*—We introduce *on-demand redundancy*, a set of architectural techniques that leverage the tightly-coupled nature of components in systems-on-chip to reduce the cost of safety-critical systems. *On-demand redundancy* eases the assumptions that traditionally segregate the execution of critical and non-critical tasks (NCTs), making resources available for critical tasks at potentially arbitrary points in both space and time, and otherwise freeing resources to execute non-critical tasks when critical tasks are not executing. *Relaxed dedication* is one such technique that allows non-critical tasks to execute on critical task resources. Our results demonstrate that for a wide variety of applications and architectures, *relaxed dedication* is more cost-effective than a traditional approach that employs dedicated resources executing in lockstep. Applied to dual-modular redundancy (DMR), relaxed dedication exposes 73% more NCT cycles than traditional DMR on average, across a wide variety of usage scenarios.**

I. Introduction

The use of safety-critical computer systems is expanding rapidly in a variety of market segments, such as medical equipment and automobiles. To ensure that no harm comes to human users when transient errors occur, faults must be quickly detected. However, fast error detection is expensive. For example, dual-modular redundancy (DMR) [1], a special case of n-modular redundancy (nMR), duplicates resources so that divergent architectural state (resulting from a fault) is quickly detected. While DMR significantly improves reliability, it does so at great cost.

However, when safety-critical systems execute a mix of safety-critical and non-critical tasks, there is ample opportunity for cost reduction without sacrificing reliability. In practice, such systems are typically logically divided into two subsystems. The critical task (CT) subsystem is implemented using dual-modular redundancy (DMR) and executes CTs in *lockstep*. Under *lockstep execution*, processors perform the same calculations on the same inputs in the same clock cycle, the results of which are compared regularly to detect when a failure, whether transient, intermittent, or permanent, has occurred [2]. The CT subsystem is functionally isolated from the non-critical task (NCT) subsystem (though in practice they may share a communication medium) to prevent NCTs from disrupting the execution of CTs. However, when CTs do not execute continuously—as when retry reservations are scheduled so CTs can be re-executed in the event of a fault—such an architecture is wasteful.

To reduce the cost of such systems, we propose a set of techniques that provide what we call *on-demand redundancy*. Cost concerns already provide significant incentives for designers to integrate critical and non-critical functionality on the same chip. Rather than replace explicit hardware redundancy, which remains in use because of its superior fault coverage and detection latency, the goal of *on-demand redundancy* is to take advantage of increased integration and expose redundant resources for the execution of non-critical tasks in novel ways.

We present one form of on-demand redundancy, *relaxed dedication*, in which NCTs in a statically scheduled system are scheduled to opportunistically execute on critical task resources (CTRs) when CTs are not scheduled. Since NCTs need not execute in lockstep, each processor in a CTR pair can be used to execute independent NCTs during idle time and retry reservations, effectively multiplying the resources available for NCTs. Performing static scheduling using worst-case execution times (WCETs) prevents NCTs from interfering with CTs also executing on CTRs. This creates significant opportunity relative to traditional DMR to either (a) increase the NCT workload or (b) reduce the cost of non-critical task resources (NCTRs), all without compromising the reliability of the system.

In this paper, we estimate the potential performance and cost advantages of *relaxed dedication*. To quantify the performance advantage of relaxed dedication we first develop analytical models to evaluate the opportunity to increase the NCT workload under a variety of usage scenarios. We subsequently validate our models with extensive experimentation using a novel static scheduling technique that iteratively determines the largest set of NCTs that can be scheduled. When comparing cost-equivalent architectures (where area is equated with cost), that which successfully schedules the most NCTs in the same hyperperiod presents the best performance-cost trade-off.

We observed in our analysis and experiments that *relaxed dedication* significantly increases the cycles available to execute NCTs. DMR with relaxed dedication exposes 73% more NCT cycles on average than DMR with dedicated resources, across a wide variety of system sizes and applications. Relaxed dedication can also be applied to another redundancy technique in the literature and industry, *mission-monitor* (MM) pairs; we observe that MM with relaxed dedication increases NCT utilization by 54% on average, and in some

978-3-9810801-7-9/DATE11/©2011 EDAA

circumstances outperforms DMR with relaxed dedication. By enabling large increases in NCT utilization, relaxed dedication enables designers to significantly reduce costs or increase the functionality of safety-critical systems.

II. Related Work

The reader is referred to the literature for surveys of the fundamental structures of fault-tolerant computing [2]–[4] and transient errors and architectures to mitigate them [5].

A multiplicity of techniques have been developed to address the cost of hardware redundancy. Baleani, *et al.* have investigated the trade-offs of lockstep and more loosely coupled redundant execution [1]. Sloan and Kumar have developed a framework which distributes voting logic to support efficient nMR in chip multiprocessors (CMPs) [6]. Subramanyan, *et al.* reduce throughput losses in a CMP when a redundant thread lags the leading thread by forwarding loaded values and branch outcomes [7]. When cost limits redundancy to duplicated hardware or less, reliability can be categorized by their recovery mechanism: limited hardware or software replication [8], [9], re-execution [10], checkpointing [11], or some combination [12]. These efforts all reclaim or reduce the cost of explicit hardware redundancy, either working with it or replacing it; unlike any of these efforts, our research specifically focuses on the interaction of critical and non-critical tasks, reducing the overhead of redundancy by increasing NCT execution. To our knowledge, ours is the first research to explore the performance implications of allowing CTs and NCTs on the same resources in a fault-tolerant system.

Several reliability techniques use sets of static schedules to dynamically respond to failure [10], [13], [14]. Another approach adjusts a static schedule at runtime to enhance reliability, allowing tasks to re-execute by moving the execution of other tasks [15]. Alternatively, our approach performs static scheduling to opportunistically schedule non-critical tasks in the retry slots of CTs. The NCTs are pre-empted when a CT must retry.

Izosimov, *et al.* presents a fault-tolerant scheduling technique that enforces hard deadlines and selectively enforces soft deadlines in the presence of failures on a single processing node [16]. To our knowledge, ours is also the first research to explore how scheduling influences the extent to which underutilized resources in a multi-core fault-tolerant system may be leveraged to increase non-critical task workload.

III. On-demand Redundancy

Traditional DMR architectures achieve reliability by (1) dedicating resources to the execution of critical tasks, therefore limiting interference with non-critical tasks, and (2) executing critical tasks in lockstep, therefore ensuring that failures are detected immediately. However, this reliability is achieved only at great cost: critical task resources must be duplicated and are underutilized when critical tasks aren't executing.

If we weaken requirements (1) and (2) above, a design space emerges with a number of opportunities to reduce the cost or increase the workload of such a system. *On-demand redundancy* captures a set of techniques which, opposed to the baseline (dedicated resources in lockstep), makes resources available for CTs at potentially arbitrary points in both space and time, and frees resources in the system to execute NCTs when CTs are not executing.

We define two classes of techniques for *on-demand redundancy*. The first relaxes *dedication*, the assumption that only CTs may execute on critical-task resources. Examples of relaxed dedication include allowing NCTs to execute on CTRs, or allowing arbitrary processing resources (rather than just CTRs) to perform redundant execution. This has clear implications for system cost, as it enables a system to make more efficient use of its available resources.

The second relaxes *lockstep*, the assumption that the reliability of CT can only be ensured when they execute concurrently on homogeneous pairs. Examples of how lockstep can be relaxed include executing redundant tasks at different times, using heterogeneous software, or on heterogeneous resources. This too increases the flexibility with which the system is utilized, creating opportunity to reduce costs or improve reliability.

A. Relaxed Dedication

Relaxing the assumption that CTs execute on resources dedicated to CTs principally reduces cost (or makes workload increases possible) by allowing NCTs to execute on CTRs. Dedication has been historically employed to prevent NCTs from interfering with CTs. Today, however, cost considerations are driving designers to find new approaches to reduce the hardware required for reliable execution. By exposing idle time and retry reservations on CTRs, relaxed dedication is one promising technique for reducing the cost of redundancy. The focus of this paper is on evaluating the extent to which relaxed dedication supports increases in NCT workload.

When applied to a system that executes CTs in lockstep, relaxed dedication requires very few design changes. We assume our systems are synchronous and statically scheduled; static scheduling ensures that NCTs don't interfere with CTs, and aperiodic tasks (not included in this paper), can be executed in a periodically recurring slot reserved for that purpose. In this case, the only additional architectural state that is required is a single status bit that indicates whether or not the (existing) comparison logic should be used. When a processor performs a context switch to begin a CT, this status register is set, and the processor begins executing in lockstep with its redundant pair; synchronization is naturally enforced by the scheduler.

Relaxed dedication has implications for instruction and data caches: divergent cache behavior. For example, if a NCT evicts data used by a CT on one processor, but the data remains resident in the pair's cache, future accesses of that data will produce a hit in one cache and a miss in another. A number of potential solutions exist, but evaluating their performance impact and cost is beyond the scope of this paper. One brute force technique would be to invalidate caches prior to executing CTs. Another approach would be to use mission-monitor pairs, detailed in Section IV. Other techniques could be employed to resolve, if not eliminate divergence: considering divergence when evaluating CT WCET, required by static scheduling;

using write-through to push dirty data out of caches; or, extending hardware support for delayed lockstep (where one processor is n cycles ahead of its pair) to adaptively stall the lockstep pair when divergent cache behavior occurs.

B. Analytical Model

We have developed an analytical model of the potential performance gains supported by relaxed dedication. When CTRs execute only CTs, the number of cycles available for executing NCTs is limited to the capacity of the NCTRs. Given a hyperperiod t, the number of cycles available for non-critical tasks on a dual-modular redundancy (DMR) system with N NCTRs is

$$W_{DMR} = \sum_{i=1}^{N} f_i t \tag{1}$$

where f_i is the clock frequency of $NCTR_i$.

In DMR systems, critical tasks are often scheduled with reserved retry slots [10], [14], [17] such that when a transient failure occurs, there is sufficient time before the task's deadline to re-execute the task.Because transient failures are generally rare events, these retry slots are often unutilized. However, if dedication is relaxed and NCTs are allowed to execute on CTRs, either during idle time or in retry slots (where they will be safely preempted in the event of a transient failure), significant opportunities to reduce system cost or increase system workload emerge. In general, *relaxed dedication* (RD) increases the cycles available for NCTs on DMR systems with M DMR pairs to

$$W_{DMR+RD} = \sum_{i=1}^{M} 2 f_i t (1 - c_i) + W_{DMR} \tag{2}$$

where c_i is the fraction of time DMR pair CTR_i executes CTs (not including retry slots). When retry slots are scheduled for each CT, $0 \leq c_i \leq 0.5$.

The advantage of RD is obvious when considering a simple example. Suppose a homogeneous system consists of M CTR pairs and N NCTRs. Each CTR pair executes CTs for the same fraction of time c. In this case, DMR exposes $W_{DMR} = Nft$ cycles to NCTs. DMR+RD, however, has the potential to expose substantially more:

$$\frac{W_{DMR+RD}}{W_{DMR}} = \frac{2Mft(1-c) + Nft}{Nft} = 1 + \frac{2M(1-c)}{N}. \tag{3}$$

As expected, we observe in Eq. (3) that relaxed dedication always exposes more cycles to NCTs than DMR alone. For example, when $c = 0.5$, $M = 1$ and $N = 1$, the cycles exposed to NCTs doubles compared to DMR. As c shrinks or M grows, this already large advantage continues to grow.

IV. Mission-monitor Pairs

An alternative to dual-modular redundancy is to employ a *mission-monitor* (MM) pair to achieve reliability. The mission core executes the critical task, while a tightly-coupled monitor core replicates just enough of the execution of the mission core to ensure that important failures are detected. This approach has been previously proposed in the literature [18], and has recently appeared in industry [19].

The principal advantage of MM pairs is that system cost is reduced by replicating in the monitor core only that functionality which is needed to ensure the safe operation of the mission core. Toshiba reports that in its implementation, the monitor core is 58% smaller than the mission core. It should also be noted that MM systems are unlikely to experience any difficulty related to divergent cache behavior, as monitor cores are unlikely to internally cache instructions or data.

The disadvantage of MM is that the tightly coupled monitor core cannot be used to execute non-critical tasks. At first glance, the number cycles available for NCTs on a MM system is the same as a DMR system (see Eq. (1)). The cost (area) advantage of MM systems however, suggests that any NCTs in a MM system will be larger or more numerous than the corresponding NCTs in a cost-equivalent DMR system.

Consider a MM system of equivalent cost with the homogeneous DMR systems discussed above. To account for the decreased area of the monitor core, we assume that for each MM pair a MM system has a half-area, half-performance NCTR[1]. In this case,

$$W_{MM} = \frac{Mft}{2} + Nft. \tag{4}$$

When resource dedication is relaxed, the mission core is also available to execute NCTs. This increases the number of available cycles for NCTs on MM systems to

$$W_{MM+RD} = \frac{Mft}{2} + Mft(1-c) + Nft. \tag{5}$$

MM systems clearly expose more NCT cycles than DMR systems, and relaxed dedication further extends their advantage. Once again comparing cost-equivalent systems, whether DMR+RD or MM+RD prevails is a function of c:

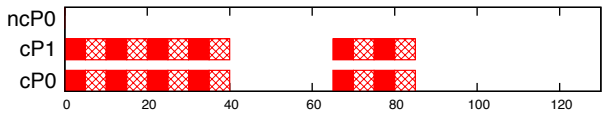
$$\frac{W_{DMR+RD}}{W_{MM+RD}} = \frac{M(2-2c) + N}{M(\frac{3}{2} - c) + N}. \tag{6}$$

When $c \leq 0.5$, $2 - 2c > \frac{3}{2} - c$, and DMR+RD exposes more cycles to NCTs than MM+RD. For example, when $M = 1, N = 1$, and $c = 0.25$, DMR+RD exposes 11% more cycles than MM+RD. When systems schedule retry slots or are not fully utilized by critical tasks, it is *theoretically* more cost-effective to apply relaxed dedication to DMR than MM pairs.

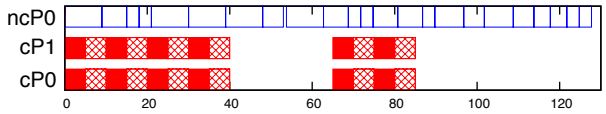
V. Scheduling for Performance Estimation

In order to compare the different safety-critical architectures described above, we have developed a static scheduling technique to determine the extent to which each may take advantage of opportunity to execute non-critical tasks. While the analytical models presented in Sections III and IV establish the upper bound on the number of cycles available for non-critical task execution, whether a sets of tasks can be executed will be determined by the distribution of sizes and periods of tasks, leaving unutilized gaps in the schedule.

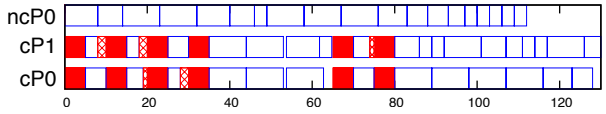
[1]This is a conservative assumption. For example, the ARM946E-S runs at 70% the rate of the ARM1156T2(F)-S while requiring 35% of the area.



(a) Critical tasks (solid red) and retry reservations (hatched red).



(b) Dedication limits NCT (blue outline) scheduling opportunity.



(c) Relaxed dedication significantly increases NCT utilization.

Fig. 1. NCT execution is increased by utilizing idle time and retry slots.

Our approach first schedules all critical tasks, ensuring that CT pairs are able to execute in lockstep. Next, our approach schedules as many NCTs as possible, taking advantage of idle time and retry slots on critical task resources in the case of relaxed dedication. Each system under comparison executes the same schedule of CTs; however, as the different systems present different opportunities to execute NCTs, they may be distinguished by the resulting NCT utilization.

A. Scheduling Safety-Critical Tasks

In this work, we consider only periodic singleton tasks; this is consistent with the model of many safety-critical embedded control systems, which read a sensor, perform a simple calculation, and possibly write an actuator. In this case, the only precedence relationships are between different instances of the same periodic task (earlier instances must complete before later instances arrive). We also assume that communication costs are negligible.

Since safety-critical tasks must execute in lockstep, to simplify the process of finding two perfectly aligned scheduling slots on two resources, we schedule CTs first. We have adopted a straightforward list scheduling algorithm [20]. All instances of each CT are sorted in a single list by their schedule slack—the difference between their latest and earliest finish time—and arrival time. Tasks with little slack are scheduled first. Ties are decided by arrival time; earlier arrivals are scheduled first. CTs are scheduled in idle slots as early as possible after their arrival, and retry slots are reserved immediately following each critical task [14], [17]. We assume that transient faults are rare enough that additional retry slots need not be allocated [14]; supporting additional retry slots would be trivial.

A simple example of a schedule of CTs and retry slots is illustrated in Figure 1(a). Solid red blocks indicate critical task reservations; the associated retry slots are indicated by the hatched red blocks. Note that the schedule is the same for the CTRs cP0 and cP1.

B. Scheduling Non-Critical Tasks

Once all CTs have been scheduled, NCTs are scheduled. As with CTs, NCTs are sorted in a list according to their slack and arrival time. When CTRs are dedicated, NCTs are only scheduled in the available idle time on NCTRs, as indicated by the empty blue blocks assigned to ncP0 in Figure 1(b). When dedication is relaxed, NCTs may be scheduled in idle and retry slots, and adjacent combinations thereof, as illustrated in Figure 1(c). In this simple example, relaxing dedication increases the number of NCT cycles that can be executed by $2.35\times$.

C. Performance Comparison Via Iterative Scheduling

We have employed the above critical and non-critical task scheduler to compare the performance of various safety-critical architectures. To do so, we have modified typical scheduling assumptions. Instead of attempting to select the best among feasible schedules given a fixed set of tasks, our objective is to schedule as many tasks from the same (effectively infinite) pool as possible. Whichever architecture is able to schedule the most tasks is the architecture that achieves the best performance-cost trade-off.

To achieve this, we perform iterative scheduling. After the CTs have been scheduled, our scheduler performs a binary search in an ordered set of task for the maximal set of schedulable NCTs; set of task graphs determines the performance point for the given architecture.

VI. Experimental Setup

We use the static scheduling approach in Section V to measure how effectively relaxed dedication exposes cycles for NCT execution in a variety of usage scenarios, as compared with the analytical models presented in Sections III and IV. In our experiments, we select a mission-monitor implementation as the baseline (MM), and compare traditional dual-modular redundancy (DMR), DMR with relaxed dedication (DMR+RD), and MM with relaxed dedication (MM+RD). We assumed a fixed system cost of eight NCT cores, and varied (a) the number of CTR pairs $M \in \{1, 2, 3, 4\}$, (b) the ratio of CT sizes to NCT sizes $\alpha \in \{0.5, 1, 2\}$, and the fraction of CTR execution dedicated to critical tasks $c \in [0.1, 0.48]$. As M increases, the number of NCTRs N decreases accordingly, $N \in \{6, 4, 2, 0\}$. A fair comparison between DMR and MM systems is ensured by giving additional NCTRs to MM systems to account for the cost advantage of this approach over strict DMR (see Section IV).

A. Critical Task Sets

We randomly generated a number of benchmarks using Task Graphs for Free (TGFF) [21] in order to determine how two factors influence of c and α on the amount of NCTs that can be scheduled on the different architectures in our evaluation.

NCT utilization is naturally expected to increase under lower values of c, not only because more idle time is available, but because statistically speaking, blocks of idle time are expected to be larger, facilitating the scheduling of larger NCTs. To explore the effect of variable c, we generated 10,000

TABLE I
NORMALIZED CYCLES EXPOSED FOR NCT EXECUTION

CTRs (M)	NCTRs (N)	MM	DMR	MM+RD	DMR+RD
1	6	1.00	0.92	1.11	1.14
2	4	1.00	0.80	1.28	1.37
3	2	1.00	0.57	1.61	1.79
4	0	1.00	0.00	2.42	2.84

sets of critical tasks, from which we selected 20 such that $c \in \{0.1, 0.12, ..., 0.48\}$. These different sets of tasks capture a range of reasonable applications, from those dominated by NCTs where safety is periodically monitored (*e.g.*, a tire-pressure monitoring system), to those dominated by safety-critical tasks (*e.g.*, an anti-lock braking system). Each set of CTs is composed of 20 tasks, each with an average task length of 20 ± 10, period multipliers in $\{1, 2, 5, 10, 20, 50, 100\}$, and task deadlines equal to task period. A retry slot is conservatively scheduled immediately following each task; studying the effect of less aggressive retry allocation on relaxed dedication is the subject of future work.

B. Non-critical Task Sets

To explore the effect of the relationship between CT size and NCT size, we created three pools of NCTs, each resulting in a different ratio α of CT size to NCT size. When NCTs are scheduled after CTs, the relative size of NCTs determines which idle and retry reservations they may be scheduled in, influencing the performance of systems employing relaxed dedication. Each NCT set has the same set of possible period multipliers as the CT sets. Average task lengths have been selected so that the three pools of NCTs have $\alpha = \{0.5, 1, 2\}$. $\alpha = 0.5$ is expected to result in lower NCT utilization, since finding slots for large NCTs among smaller, and potentially more tightly scheduled critical tasks may prove difficult.

C. CT-NCT Benchmarks

In order to account for the interaction of individual sets of CTs and NCTs (where one ill-sized NCT can prevent further NCTs from being scheduled during our iterative approach), we match each CT set with 33 randomly generated sets of NCTs from each pool. Each NCT set from the same pool has the same parameters, but different sets can result in significantly different scheduling outcomes. We derive the performance of an architecture for a given value of c by averaging the number of cycles utilized by NCTs across each of the 33 samples. In total, we performed static scheduling for over 30,000 combinations of architectures and task sets.

VII. RESULTS

The results of our scheduling experiments are summarized in Tables I and II. For each architecture with M CTR pairs and N additional NCTRs, Table I reports the average (across all c, and independent of α), normalized maximum number of cycles available for executing NCTs, and derived from the analytical models in Sections III and IV. For the sake of easy comparison with each other and Table II, all values are normalized to the

TABLE II
NORMALIZED CYCLES UTILIZED FOR NCT EXECUTION

α	CTRs (M)	NCTRs (N)	MM	DMR	MM+RD	DMR+RD
2	1	6	1.00	0.92	1.10	1.13
	2	4	0.99	0.80	1.27	1.35
	3	2	0.99	0.57	1.58	1.66
	4	0	0.99	0.00	2.30	2.32
1	1	6	0.99	0.91	1.10	1.13
	2	4	0.99	0.79	1.26	1.32
	3	2	0.98	0.56	1.55	1.53
	4	0	0.97	0.00	2.17	1.22
0.5	1	6	0.98	0.90	1.07	1.07
	2	4	0.97	0.78	1.19	1.08
	3	2	0.96	0.55	1.32	0.86
	4	0	0.95	0.00	1.43	0.03

NCT cycles exposed by MM; some usage scenarios result in 0 potential NCT cycles for DMR. Recall that when $N = 0$, MM retains some NCTRs due to its cost advantage over DMR.

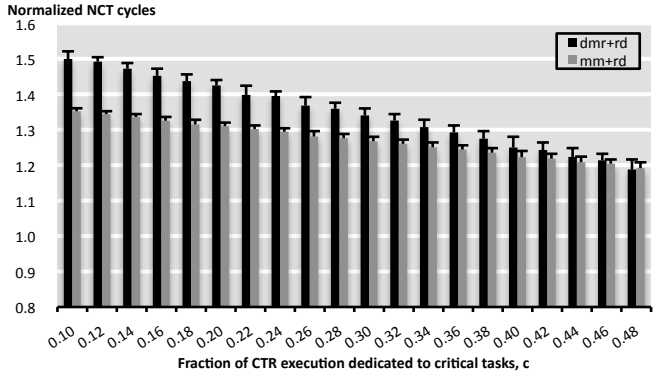
For each architecture and α, Table II reports the average (across all c), normalized number of NCT cycles successfully scheduled. We first observe that relaxed dedication, as expected, always exposes more utilizable cycles for NCTs than traditional approaches; for each benchmark, DMR+RD and MM+RD outperform DMR and MM respectively, and at times by sizable margins. For $\alpha = 2$, MM+RD increases NCT execution by 58% on average over MM; DMR+RD increases NCT execution by 95% on average over DMR when $M < 4$ (when $M = 4$ DMR schedules no NCTs).

We also observe that when NCTs are relatively smaller than CTs (*e.g.*, when $\alpha = 2$), and especially when systems have more NCTRs than CTRs, that the observed NCT cycles utilized approach the potential NCT cycles exposed by relaxed dedication. For $\alpha = 2$, DMR+RD and MM+RD execute NCTs in 93% and 98% of the available cycles, respectively; this figure increases to 97% for DMR+RD if we don't consider the architecture with no NCTRs ($M = 4, N = 0$).

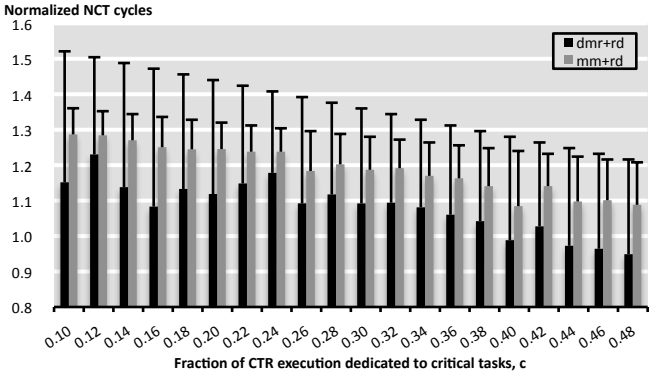
However, we also observe that when NCTs are relatively larger than CTs (*e.g.*, when $\alpha = 0.5$), and especially when systems have more CTRs than NCTRs, that it is more difficult for systems with relaxed dedication to take advantage of the exposed cycles to schedule NCTs. For $\alpha = 0.5$, DMR+RD and MM+RD execute NCTs in 82% and 56% of the available cycles, respectively; this figure decreases to 1% for DMR+RD if we consider only the architecture with no NCTRs.

The effectiveness of DMR+RD and MM+RD when $M = 2, N = 4$ for $\alpha = \{0.5, 2\}$ is illustrated in Figure 2. Figure 2 plots how the normalized cycles scheduled for NCTs decreases are c increases. The error bars indicate the maximum NCT cycles available as predicted by our analytical models.

Figure 2(a) plots the scheduled NCT cycles when $\alpha = 2$. In this case, both DMR+RD and MM+RD achieve very close to the optimal, taking advantage of 98% and 99% of the available cycles on average. As a result, DMR+RD consistently outperforms MM+RD, scheduling 6% more NCT cycles on average. DMR+RD's advantage is in the extra CTRs with retry and idle slots exposed for NCT scheduling; when NCTs are relatively small, the gaps in CTR schedules are easily utilized.



(a) Two CT pairs, four NCTRs; CTs twice as large as NCTs ($\alpha = 2$).



(b) Two CT pairs, four NCTRs; CTs half as large as NCTs ($\alpha = 0.5$).

Fig. 2. When NCTs are small relative to CTs ((a) $\alpha > 1$), retry and idle slots are more easily utilized by relaxed dedication.

Figure 2(b) plots the scheduled NCT cycles when $\alpha = 0.5$. In this case, MM+RD schedules NCTs in 93% of the available cycles; DMR+RD, on the other hand, schedules NCTs in 79% of the available cycles, and is consistently outperformed by MM+RD, scheduling 9% fewer NCT cycles on average. MM+RD's advantage is MM's area advantage over DMR. MM's extra NCTRs, though half-performance, make it easier to schedule the large NCTs that DMR+RD fails to schedule in retry and idle slots.

VIII. Conclusions

Cost concerns are driving designers of safety-critical systems to integrate more and more functionality on a single chip. We have proposed a novel approach, *relaxed dedication*, to reduce the cost of redundant execution by utilizing the retry reservations and idle on critical task resource to execute non-critical tasks. Our analytical models predict that when relaxed dedication is applied to dual-modular redundancy, these systems expose more cycles for NCT scheduling than equivalent-cost mission-monitor systems.

We developed an iterative static scheduling technique to validate our analytical models and investigate the effects of scheduling (*e.g.*, task length and period distributions) on relaxed dedication. By finding the maximum set of schedulable non-critical tasks, our technique is able to compare safety-critical architectures by comparing the number of cycles scheduled by each for use by NCTs.

We observed that DMR with relaxed dedication and MM with relaxed dedication expose 73% and 54% more NCT cycles on average, respectively, than DMR and MM with dedicated resources, respectively, across a wide variety of system configurations and benchmarks. By making all processing elements available for executing NCTs, relaxed dedication consistently exposes substantial opportunity to increase the workload or reduce system cost. Furthermore, we observed that the realities of scheduling may limit the benefit of relaxed dedication on some systems. Depending on the ratio of the size of NCTs relative to CTs (α), either DMR or MM make the best use of relaxed dedication: when $\alpha > 1$, DMR+RD has the advantage; when $\alpha < 1$, MM+RD has the advantage.

Acknowledgements

This work is supported by the Semiconductor Research Corporation through contract 2009-HJ-2042.

References

[1] M. Baleani, *et al.*, "Fault-tolerant platforms for automotive safety-critical applications," in *CASES'03*, 2003.
[2] V. Nelson, "Fault-tolerant computing: fundamental concepts," *Computer*, vol. 23, July 1990.
[3] V. Prasad, "Fault tolerant digital systems," *IEEE Potentials*, vol. 8, February 1989.
[4] F. Cristian, "Understanding fault-tolerant distributed systems," *Commun. ACM*, vol. 34, no. 2, 1991.
[5] S. Mukherjee, *Architecture Design for Soft Errors*. Morgan-Kaufmann, 2008.
[6] J. Sloan and R. Kumar, "Towards scalable reliability frameworks for error prone cmps," in *CASES'09*, October 2009.
[7] P. Subramanyan, *et al.*, "Multiplexed redundant execution: A technique for efficient fault tolerance in chip multiprocessors," in *DATE'10*, 2010.
[8] B. P. Dave and N. K. Jha, "COFTA: Hardware-software co-synthesis of heterogeneous distributed embedded systems for low overhead fault tolerance," *IEEE Transactions on Computer*, vol. 48, April 1999.
[9] Y. Xie, *et al.*, "Reliability-aware co-synthesis for embedded systems," in *ASAP'04*, 2004.
[10] N. Kandasamy, *et al.*, "Transparent recovery from intermittent faults in time-triggered distributed systems," *IEEE Trans. Comput.*, vol. 52, Feb. 2003.
[11] S. Punnekkat, *et al.*, "Analysis of checkpointing for real-time systems," *Real-Time Systems*, vol. 20, no. 1, 2001.
[12] P. Eles, *et al.*, "Synthesis of fault-tolerant embedded systems," in *DATE'08*, 2008.
[13] C. Dima, A. Girault, C. Lavarenne, and Y. Sorel, "Off-line real-time fault-tolerant scheduling," in *PDP'01*, 2001.
[14] V. Izosimov, *et al.*, "Synthesis of fault-tolerant schedules with transparency/performance trade-offs for distributed embedded systems," in *DATE'06*, 2006.
[15] G. Fohler, "Adaptive fault-tolerance with statically scheduled real-time systems," in *Euromicro Real-Time Systems Workshop*, 1997.
[16] V. Izosimov, *et al.*, "Scheduling of fault-tolerant embedded systems with soft and hard timing constraints," in *DATE'08*, 2006.
[17] Y. Liu, H. Liang, and K. Wu, "Scheduling for energy efficiency and fault tolerance in hard real-time systems," in *DATE'10*, 2010.
[18] T. M. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *MICRO-32*, November 1999.
[19] C. Holland, "Toshiba MCU gains SIL3 and ASILD safety approval." http://www.embedded.com/products/integratedcircuits/222400364, January 2010.
[20] R. P. Dick and N. K. Jha, "Mocsyn: multiobjective core-based single-chip system synthesis," in *DATE'99*, 1999.
[21] R. P. Dick *et al.*, "TGFF: Task graphs for free," in *CODES*, 1998.