Task Scheduling for Reconfigurable Systems in Dynamic Fault-Rate Environments

Adam Jacobs, Nicholas Wulf, and Alan D. George
NSF Center for High-Performance Reconfigurable Computing (CHREC)
University of Florida, Gainesville, Florida 32611
Email: {jacobs, wulf, george}@chrec.org

***Abstract*—Commercial SRAM-based, field-programmable gate arrays (FPGAs) have the capability to provide space applications with the necessary performance, energy-efficiency, and adaptability to meet next-generation mission requirements. However, mitigating an FPGA's susceptibility to radiation-induced faults is challenging. Triple-modular redundancy (TMR) techniques are traditionally used to mitigate radiation effects, but TMR incurs substantial overheads such as increased area and power requirements. Using partial reconfiguration (PR), FPGAs could be used to dynamically adjust the fault-tolerance scheme as the radiation environment changes over time. In order to manage these dynamic adjustments, a fault-tolerant task scheduler is necessary.**

We improve scheduling in the presence of time-varying fault rates by developing a fault-tolerant scheduling heuristic. Our heuristic combines task execution time and system fault rate to determine the optimal fault-tolerance mode for the task. The heuristic is evaluated using software simulations of a system in periodic and burst fault environments. Results show our scheduling technique is capable of reducing the task rejection ratio in periodic environments by 94% and in burst environments by 48% over static TMR, and the adaptive heuristic approaches the performance of an optimal predetermined heuristic. Integration of our fault-tolerant scheduling heuristic with other pre-existing PR architectures can enable their use in dynamic fault environments.

I. Introduction

The need for high-performance embedded space systems is constantly growing as new, high-fidelity sensors increase the amount of data collected by orbiting satellites. The capabilities of these sensors outpaces the ability to transmit their data to ground stations. Increasing future systems' onboard data-processing capabilities can alleviate this downlink bottleneck while enabling future space systems to keep up with stringent real-time constraints. However, increasing the onboard data-processing capabilities requires high-performance computing, which has largely been absent from space systems.

One approach for adaptive high-performance space system design leverages hardware-adaptive devices such as field-programmable gate arrays (FPGAs), which provide parallel computations at a high level of performance per unit size, mass, and power [1]. Fortunately, many space applications, such as synthetic aperture radar (SAR) [2], hyperspectral imaging (HSI) [3], image compression [4], and other image processing applications [5], where onboard data processing can significantly reduce data transmission requirements, are amenable to an FPGA's highly parallel architecture. FPGA reconfiguration enables multiple functions to be time-multiplexed onto the FPGA's hardware resources, reducing the number or size of processors required to meet the application's computational demand. Thus, FPGAs are capable of creating small, lightweight, yet powerful systems that can be optimized for a space application's time-varying hardware requirements.

In order to leverage FPGAs in space systems, the FPGA must operate correctly and reliably in high-radiation environments, such as those found in near-Earth orbits. Radiation-induced single-event upsets (SEUs) can cause errors within the FPGA user logic and routing resources, which can manifest as functional changes or incorrect data. Fault-tolerant techniques, such as triple-modular redundancy (TMR) and memory scrubbing, can protect the system from most SEUs and significantly decrease the SEU-induced errors, but designing an FPGA-based space system using TMR introduces at least 200% area overhead for each protected module. Depending on the expected upset rates for a given space system, other lower-overhead fault-tolerance methods could be used to provide sufficient reliability while maximizing the resources available for performance.

For traditional space systems, architectural requirements are determined by estimating the expected worst-case upset rates and including an additional safety margin. However, since SEU rates vary based on orbital position and the majority of orbital positions experience relatively low upset rates, a system designed for the worst-case upset-rate scenario contains processing resources that are wasted during the frequent low-upset-rate periods. In order to provide the necessary reliability during high-upset-rate periods and reduce the processing overhead incurred during low-upset-rate periods, the fault-tolerance method must change based on the current upset rate. Reconfigurable Fault Tolerance (RFT) is a framework that enables this run-time modification of fault tolerance [6]. During high-upset-rate periods, the system can be reconfigured to provide high reliability with TMR at the expense of reduced processing capabilities, while during low-upset-rate periods the system can be reconfigured to provide higher performance by re-provisioning the redundant hardware for additional application functionality. This upset-rate-based adaptability enables high performance while maintaining reliability.

In order to optimize a system for performance and reliability, the fault-tolerant mode must be selected carefully based upon the current fault conditions experienced by the system. In this paper, we propose a fault-tolerant scheduler that can schedule real-time tasks as well as select a heuristic to select a fault-tolerant RFT mode for each task. We then evaluate the

978-1-4799-1365-7/13/$31.00 ©2013 IEEE

effectiveness of the proposed heuristic using two case-study simulations.

The remaining sections of this paper are organized as follows. Section 2 provides background and surveys previous work related on fault-tolerant task scheduling for real-time embedded systems. Section 3 describes a novel FT-scheduling criteria for determining the optimal FT-mode. Section 4 describes a task-scheduling simulation framework that enables fault-injection in order to demonstrate the effectiveness of our environmentally-aware scheduling heuristics. Section 5 analyzes the simulation results and discusses the effectiveness of the tested heuristics. Finally, Section 6 presents conclusions and outlines directions for possible future research.

II. Related Work

Task scheduling algorithms can be categorized as either *online* or *offline*. Offline scheduling algorithms have complete knowledge of all tasks that must be scheduled. The general scheduling optimization problem is NP-hard, but efficient heuristics exist, and offline schedules can be pre-determined at compile-time. With online scheduling, tasks arrive at the scheduler periodically over time, and must be placed around previously scheduled tasks. Task arrival rates and patterns can greatly affect the quality of the scheduler's results. Arndt et al. [7] examined many online scheduling algorithms for distributed parallel computers, and used simulation to evaluate their performance. Of the several algorithms studied, the *First-Fit* algorithm, which prioritizes scheduling by the arrival time of each task, provided good schedule lengths while minimizing the average wait times.

Real-time systems introduce additional constraints for task scheduling. Each task must be completed by a *deadline*, otherwise the results will no longer be relevant or needed. A *hard* deadline must be met, otherwise the system is considered failed. A *firm* deadline can be missed, but the usefulness of the result after the deadline is zero. A *soft* deadline can be missed, but the value of the result decreases after the deadline has passed. For a hard real-time system all deadlines must be met, but the goal of a soft real-time system is to meet as many deadlines as possible while optimizing for other critera. Traditionally, schedulers attempt to minimize criteria such as makespan (total schedule length) or average task latency.

Han et al. [8] created a fault-tolerant scheduling algorithm for periodic real-time software tasks. For each primary task, an alternate less-precise task is also used to generate a sufficient result before the deadline. These alternate tasks are scheduled as close to the task deadline as possible. In the case of a primary task failure, the alternate task will be executed. If the primary task succeeded, the alternate tasks are discarded. Their algorithm is intended for offline use and was intended to protect systems against software faults. In [9], Pathan extends the rate-monotonic (RM) scheduling algorithm, used for scheduling periodic real-time tasks, to support temporal error masking (TEM). TEM schedules multiple copies of tasks to detect faults in any one copy, with additional copies scheduled to perform voting when faults are detected. These real-time scheduling algorithms require periodic tasks to perform a scheduling analysis. Scheduling aperiodic real-time tasks is a more difficult problem and is currently being studied.

Scheduling tasks for reconfigurable computing creates an additional level of complexity. Instead of scheduling a task for a one-dimensional array of processors, scheduling on a two-dimensional FPGA fabric becomes a constrained placement problem. Additionally, tasks may have multiple hardware or software implementations, increasing the overall search space. Banerjee et al. [10] present an offline KLFM heuristic [11] which incorporates detailed placement information in order to provide high-quality schedules. Mei et al. [12] combine a genetic algorithm to determine HW/SW placement with a traditional list scheduling algorithm to enable online scheduling of real-time reconfigurable embedded systems. Steiger et al. [13] developed two heuristic scheduling algorithms, Horizon and Stuffing, which provide good results while limiting the computational requirements.

III. Selection Criteria for Fault-Tolerant Mode

RFT mode switching can be triggered by *a priori* knowledge of the operating environment, application-triggered events, or external events. In an RFT system, the expected fault rate can be estimated either directly or indirectly. An external radiation sensor can be directly interfaced with the FPGA, allowing the system to track the current fault rate and predict future fault rates. Alternatively, the RFT system can indirectly determine fault rates using models of the expected fault environment [6]. By correlating the space system's current position to an existing model, a fault-rate estimate can be used to make scheduling decisions.

In the following sections we assume that tasks can be scheduled with no fault tolerance (Simplex), duplication with compare (DWC), or triple-modular redundancy (TMR). Data errors in tasks are detected by comparing or voting on the output of each task replica. We also assume that the system can estimate the current fault environment using a pre-existing model of the system's orbit.

A. FT-Mode Selection using Thresholds

One of the most straightforward methods for selecting an appropriate fault-tolerance mode is the use of thresholds. At very high fault rates, TMR is required to maintain reliability. At low fault rates, DWC or Simplex modes may provide sufficient reliability while increasing performance. At each time step, the current fault rate is measured, and new tasks are assigned based on the pre-selected rules. The optimal threshold occurs at the fault rate where the reliable performance of TMR and DWC are equal. The ideal scheduling heuristic will select TMR when the current fault rate is above the fault-rate threshold, f_{thresh}, and will select DWC otherwise. Selecting the appropriate values for the threshold is dependent on the application and environment. Determining f_{thresh} requires information about fault rates, task frequency, task load, and other factors which may not be static throughout a system's operation.

B. Time-Resource Metric for FT-Mode Selection

Instead of depending upon a user-defined threshold value to determine a fault-tolerance strategy, we explore a possible metric which can estimate the optimal threshold. The metric combines computation time (τ) and fault probability of a single

task (f) in order to select between FT modes. By developing a scheduling metric that incorporates a dynamic fault rate, we intend to improve overall system performance without the need for expert user input. Given the current fault rate, f, the reliability of a task at the end of its computation time, R, is given by the following equations (depending on mode):

$$\begin{array}{lcl} R_{Simplex} & = & (1-f)^{\tau} \\ R_{DWC} & = & (1-f)^{2\tau} \\ R_{TMR} & = & 3(1-f)^{2\tau} - 2(1-f)^{3\tau} \end{array} \quad (1)$$

If tasks are running in DWC or TMR mode, faults are discovered at the end of the task's computation time. Faulty tasks are then rescheduled until they complete successfully. The average number of times a task must be executed in order to successfully complete is then given by the following geometric series:

$$\tau_{eff} = \sum_{n=0}^{\infty} \tau(1-R)^n = \frac{\tau}{R} \quad (2)$$

We define a time-resource coefficient, α, which combines the effective execution time with the required resources of a given task ($\alpha = N \times \tau_{eff}$). Then, by comparing the α of a DWC or TMR task, we can determine which mode is optimal for reliability (lower is better). In Equation 3, we solve for conditions where DWC will provide lower α than TMR.

$$\frac{2\tau}{(1-f)^{2\tau}} \leq \frac{3\tau}{3(1-f)^{2\tau} - 2(1-f)^{3\tau}} \quad (3)$$

Simplifying Equation 3 provides the following simple relation:

$$(1-f)^{\tau} \geq \frac{3}{4} \quad (4)$$

Based on the definition of α, DWC provides more reliable performance than TMR when $R_{Simplex}$ is greater than 0.75. For low fault rates, DWC provides higher overall performance. For very high fault rates, or very long execution times, the reliability of TMR scheduling is preferred. A similar analysis can be performed for simplex tasks, however simplex scheduling has no method for detecting faults. We use this conclusion as the basis for an adaptive fault-tolerance threshold.

IV. Scheduler for RFT

Traditional fault-tolerant scheduling algorithms assume that the fault rate experienced by the system will be constant, and that the fault-tolerance strategy will also be constant. For an RFT-based system, a scheduler which uses the current fault rate is necessary to maximize system utilization while maintaining system availability. The fault-tolerant scheduler presented in this section can schedule tasks in any FT mode based on user-defined thresholds or the α-metric.

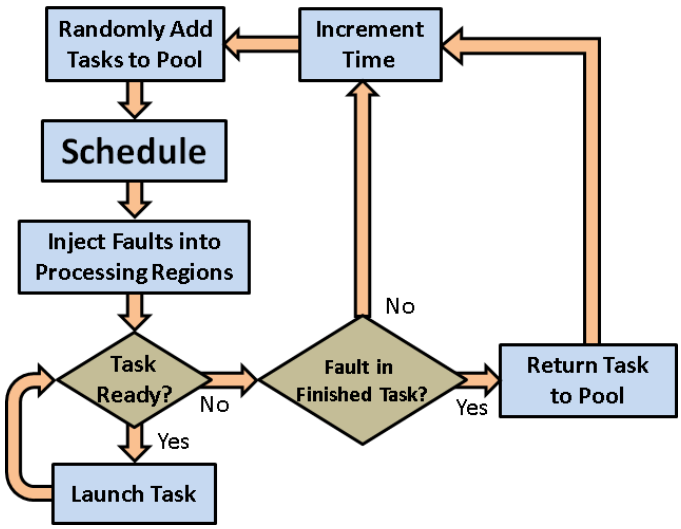

Fig. 1. Flowchart of Scheduling Simulator

A. RFT Architecture Description

The RFT system described in [6] contains a microprocessor connected to several large partially-reconfigurable regions (PRRs) through a shared system bus. During normal system operation, unique tasks can be scheduled to any of the PRRs. Depending on system configuration, the outputs of three contiguous PRRs can be voted on to provide coarse-grained TMR functionality, or two PRRs can provide DWC functionality. Each of these PRRs are large and identical in size, and can be represented with a 1D area model, reducing many of the scheduling problems presented in Section II.

B. Software Simulation

In order to evaluate our scheduling technique and possible heuristics, a software-based discrete-time simulator was developed in C++. The simulator enables us to specify task arrival rates, task deadlines, dynamic fault rates, and scheduling algorithms. In addition to scheduling tasks, the simulator can also inject faults into tasks and force re-scheduling of failed tasks.

Figure 1 shows the basic overview of how the simulator is used. At each time step, tasks are randomly added to a task pool. This process is modeled as a Poisson process with mean $\lambda_{arrival}$. All tasks in the task pool are scheduled, if possible, and then moved to the reservation list. When multiple tasks arrive simultaneously, tasks are scheduled using an earliest-deadline-first (EDF) heuristic. The scheduler does not employ preemption; tasks are scheduled on arrival only, and new tasks must be placed around the existing schedule. Tasks which cannot be scheduled before their deadlines are rejected. If a task is scheduled to begin at the current time step, the simulator moves the task from the reserved list to the execution list.

After tasks have been scheduled, faults are injected into each PRR with probability f in order to simulate the dynamic fault environment. At the end of the task's execution, the outcome of the task is determined based upon the number of faults encountered (i.e., Simplex and DWC tasks fail with 1 fault, TMR tasks fail with faults in 2 or more PRRs). Multiple

faults within a single PRR have no additional effect on the system. If the task fails, the scheduler then returns the task to the task queue to be re-scheduled. All other reserved tasks (scheduled, but not yet executing) are also returned to the task queue to be rescheduled. Fault-tolerant tasks are rescheduled until they successfully complete or can no longer meet their deadline.

When tasks must be re-scheduled due to faults, they are treated as a new task for scheduling purposes, although their original deadline is maintained. The fault-tolerant mode for rescheduled tasks will based on the fault rate at the time of rescheduling.

V. Analysis & Results

In the following analysis, task execution times (t_{exec}) are uniformly distributed in [10, 100] time steps with deadlines ($t_{deadline}$) of [100, 200] time units. For simplicity, we assume that a time step is 1 second. Simplex tasks use one processing region, DWC tasks use two processing regions, and TMR tasks use three processing regions. The simulated system uses 12 processing regions (N_{PRRs}) in order to enable flexibility in placing TMR and DWC tasks. For each experiment, we measure the performance of each metric with the scheduler's guarantee ratio (percentage of total tasks scheduled successfully) while attempting to schedule 100,000 tasks.

A. Constant Fault Rates

Initially, the simulator is used to get a fault-free baseline for comparison purposes. Figure 2 shows the effect of arrival rate on the performance of the system. At low arrival rates, Simplex, DWC, and TMR scheduling can all meet the system demand. However, arrival rates higher than 0.06 tasks per second begin to impact the schedulability of the TMR system because there are not enough resources to handle all incoming tasks. Using DWC for fault tolerance will result in higher guarantee ratios since more DWC tasks can be scheduled at any one time. The lack of a fault-tolerance mechanism excludes the use of Simplex scheduling in the presence of faults.

In order to investigate the effect of fault rates on our system, we chose a constant arrival rate of 0.075 tasks per second. With this arrival rate, the DWC system can successfully schedule all incoming tasks, while the TMR system cannot. Using the arrival rate in this way, we attempt to define a system which requires DWC to meet performance demands but can temporarily use TMR to meet reliability constraints. The effect on the guarantee ratio is shown in Figure 3. At low fault rates the DWC mode provides higher throughput, while TMR outperforms DWC at high fault rates. Our adaptive metric produces high throughput at low fault rates, closely tracking the performance of DWC, but performs between TMR and DWC at intermediate fault rates. At higher fault rates, the guarantee ratio of the adaptive heuristic produces results close to TMR. From these constant-fault-rate results, an ideal-threshold heuristic can be determined. The crossover fault rate for TMR and DWC occurs at 0.0025 faults/sec.

B. Dynamic Fault-Rate Case Studies

In order to get a benefit from the adaptive scheduling methods, the fault rates experienced by the system must vary.

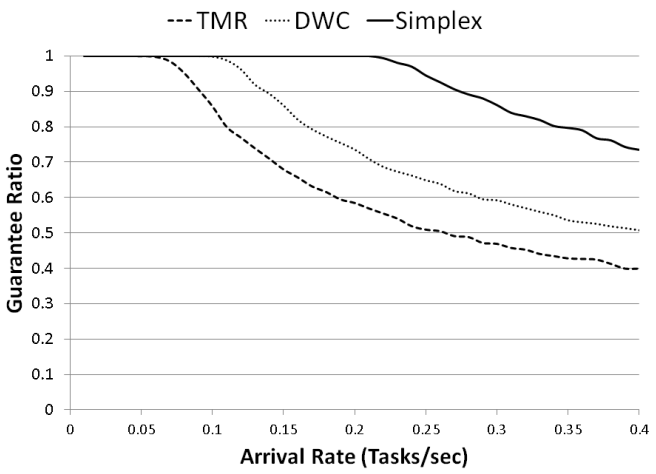

Fig. 2. Effect of Arrival Rate on Fault-Free Operation ($N_{PRRs} = 12$, $t_{exec}\epsilon[10, 100]$, $t_{deadline}\epsilon[100, 200]$)

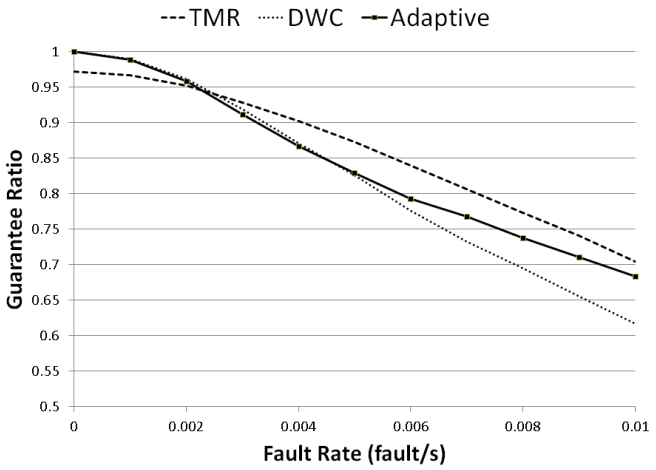

Fig. 3. Effect of Fault Rate on Task Rejection ($N_{PRRs} = 12$, $t_{exec}\epsilon[10, 100]$, $t_{deadline}\epsilon[100, 200]$, $\lambda_{arrival} = 0.075$)

We present two fault profiles which represent patterns commonly seen in space missions. Figure 4 shows the fault profiles used for the following analysis, based on the fault model in [6]. The first profile is a sinusoidal pattern with a 90-minute period which is characteristic of fault rates in Low-Earth Orbit (LEO). The second pattern (Burst) represents Highly-Elliptical Orbits (HEO), where the system experiences low fault rates for most of the orbit, with a large burst when making the closest approach to Earth, once every 12 hours. For the following case studies, four different scheduling heuristics are examined. The TMR-only and DWC-only heuristics will schedule every task in their respective mode. The ideal-threshold heuristic will use the fault-rate threshold measured in the previous section (0.0025 faults/sec) to choose between the DWC and TMR modes. The adaptive heuristic uses Equation 4 to determine the FT mode for each task. Each heuristic will be evaluated using an arrival rate of 0.075 tasks/sec and the same parameters used in Section V-A.

For the Sinusoidal case study, fault rates are low compared to the average task execution time. For this fault profile,

TABLE I. DYNAMIC SCHEDULING RESULTS

Case Study	FT Metric	Guarantee Ratio	Reject Ratio	Avg. Latency (s)
Sinusoidal	TMR-Only	0.970	0.030	51.2
Sinusoidal	DWC-Only	0.998	0.002	8.0
Sinusoidal	Ideal	0.998	0.002	7.9
Sinusoidal	Adaptive	0.998	0.002	8.2
Burst	TMR-Only	0.935	0.065	53.6
Burst	DWC-Only	0.962	0.038	9.9
Burst	Ideal	0.967	0.033	11.7
Burst	Adaptive	0.966	0.034	11.2

scheduling tasks with the DWC-only, ideal-threshold, or adaptive heuristics provide an equivalent rejection ratio, 0.2%. There are enough system resources to make re-computation of failed DWC tasks better than simply using TMR to protect against all failures. The fault rate rarely gets high enough for the threshold or adaptive heuristics to schedule tasks in TMR mode. The adaptive heuristic performs well, reducing the number of rejected tasks over the TMR-only strategy by 94%, while maintaining a low average task latency of 8 seconds per task.

In the Burst case study, fault rates are low except during a short window of time with extremely high fault rates. Unlike in the previous case study, the adaptive heuristics will benefit from the large range of fault rates and each heuristic has different performance characteristics. For this fault-rate profile, the adaptive heuristics perform the best, with 11% fewer rejected tasks than the DWC-only strategy and 48% fewer than the TMR-only strategy. Additionally, the adaptive heuristic has only 3% more rejected tasks than the ideal-threshold heuristic. TMR is only optimal when the high-fault-rate burst occurs. Otherwise, DWC will better utilize the system resources. The Burst fault profile is ideal for all dynamic metrics, since the two phases are highly separated.

C. Scheduling Improvements

At extreme fault rates (high or low), the adaptive heuristic will schedule all incoming tasks in the same mode. However, for moderate fault rates, both DWC and TMR tasks will be scheduled depending on task computation time. One drawback to the dynamic fault-tolerant selection metrics is the resource fragmentation that occurs when different sized objects are

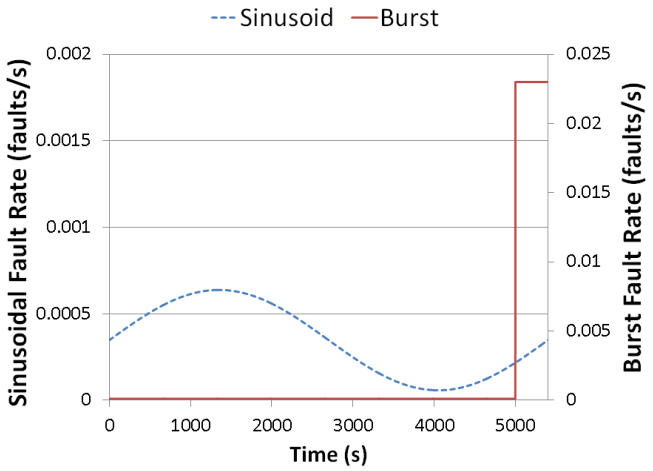

Fig. 4. Fault-Rate Profile for Case Studies

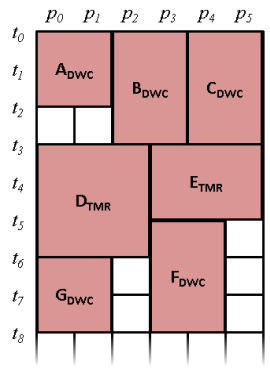

Fig. 5. Resource Fragmentation from Adaptive Placement

placed on the FPGA fabric. This effect can produce schedules similar to Figure 5, where fragmentation causes poor utilization of the available FPGA resources. As tasks arrive to the system, they are placed in PRRs p_0 through p_5, in either DWC or TMR mode. At time t_6 there are enough unused PRRs in the system for a DWC task, but because the resources are not contiguous the task cannot be placed. Unfortunately, the simplistic EDF scheduling heuristic currently in use does not account for FPGA fragmentation. The low effectiveness of the adaptive metric in Figure 3 can be explained by this FPGA fragmentation. By using a placement-aware scheduler, the adaptive heuristic should perform closer to optimal for all fault rates. For example, delaying the placement of task F_{DWC} for until t_6 would enable a more compact placement in PRRs p_2 and p_3, enabling space for the placement of an additional DWC task. Alternatively, task F_{DWC} could be placed in PRRs p_4 and p_5 at time t_5, leaving room for future DWC tasks in PRRs p_2 and p_3. An FPGA placement-aware scheduling algorithm such as the Horizon or Stuffing scheduler [13] should be incorporated in order to improve the performance of the adaptive heuristic.

Fault-rate lag is another possible problem. If a task is scheduled using a specific mode but does not execute for a long period of time, a different FT mode may become more appropriate. Limiting the scheduling window to only schedule a few tasks at a time may prevent this lag. Alternatively, using a prediction of the future fault rate during scheduling may reduce this effect. Finally, preemption enables the scheduler to return a currently running task to the task queue in order to start a higher priority task. By adding preemption capabilities to the scheduler, the guarantee ratio of all the tested metrics can be improved.

VI. CONCLUSIONS

In this work, we present a novel heuristic for determining optimal fault-tolerance settings for reconfigurable fault-tolerant systems. The heuristic determines the fault-tolerance mode for each task that will minimize system resources and task computation time in the presence of faults. An RFT scheduler and simulator were developed in order to test the effectiveness of the adaptive scheduling heuristic and to compare its performance to traditional static fault-tolerance strategies. When using our adaptive FT strategy in Burst-like fault environments, we maintain system reliability while

reducing the number of rejected tasks by 48% compared to a static TMR fault-tolerance strategy and 11% compared to static DWC. In the Sinusoidal case study, the static DWC, Ideal, adaptive heuristics reduce the number of rejected tasks by 94% compare to static TMR strategy. We have demonstrated that the adaptive heuristic performs similarly to an optimal user-defined threshold, without the need for detailed system simulation and measurement. Future work will combine the adaptive heuristic with a placement-aware task scheduler to further improve performance.

ACKNOWLEDGMENT

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422 and IIP-1161022.

REFERENCES

[1] J. Williams, C. Massie, A. D. George, J. Richardson, K. Gosrani, and H. Lam, "Characterization of fixed and reconfigurable multi-core devices for application acceleration," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 3, pp. 19:1–19:29, November 2010. [Online]. Available: http://doi.acm.org/10.1145/1862648.1862649

[2] C. Le, S. Chan, F. Cheng, W. Fang, M. Fischman, S. Hensley, R. Johnson, M. Jourdan, M. Marina, B. Parham, F. Rogez, P. Rosen, B. Shah, and S. Taft, "Onboard FPGA-based SAR processing for future spaceborne systems," in *Proceedings of the IEEE Radar Conference, 2004.*, april 2004, pp. 15 – 20.

[3] M. Hsueh and C.-I. Chang, "Field programmable gate arrays (FPGA) for pixel purity index using blocks of skewers for endmember extraction in hyperspectral imagery," *Int. J. High Perform. Comput. Appl.*, vol. 22, pp. 408–423, November 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1453081.1453083

[4] A. Gupta, S. Nooshabadi, D. Taubman, and M. Dyer, "Realizing low-cost high-throughput general-purpose block encoder for JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 7, pp. 843 –858, july 2006.

[5] A. Dawood, S. Visser, and J. Williams, "Reconfigurable FPGAs for real time image processing in space," in *14th International Conference on Digital Signal Processing, 2002. DSP 2002.*, vol. 2, 2002, pp. 845 – 848 vol.2.

[6] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ross, and H. Lam, "Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive fpga-based space computing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 4, pp. 21:1–21:30, Dec. 2012. [Online]. Available: http://doi.acm.org/10.1145/2392616.2392619

[7] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo, "A comparative study of online scheduling algorithms for networks of workstations," *Cluster Computing*, vol. 3, pp. 95–112, 2000. [Online]. Available: http://dx.doi.org/10.1023/A%3A1019024019093

[8] C.-C. Han, K. Shin, and J. Wu, "A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults," *Computers, IEEE Transactions on*, vol. 52, no. 3, pp. 362 – 372, march 2003.

[9] R. Pathan, "Fault-tolerant real-time scheduling algorithm for tolerating multiple transient faults," in *Electrical and Computer Engineering, 2006. ICECE '06. International Conference on*, dec. 2006, pp. 577 – 580.

[10] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-aware hw-sw partitioning for reconfigurable architectures with partial dynamic reconfiguration," in *Design Automation Conference, 2005. Proceedings. 42nd*, june 2005, pp. 335 – 340.

[11] B. Kernighan and S. Lin, "An eflicient heuristic procedure for partitioning graphs," *Bell system technical journal*, 1970.

[12] B. Mei, P. Schaumont, and S. Vernalde, "A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems," in *Proceedings of ProRISC*. Citeseer, 2000, pp. 405–411.

[13] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks," *Computers, IEEE Transactions on*, vol. 53, no. 11, pp. 1393 – 1407, nov. 2004.