Task Migration for Fault-Tolerance in Mixed-Criticality Embedded Systems

Prabhat Kumar Saraswat, Paul Pop, Jan Madsen
Technical Univ. of Denmark, DK-2800 Kgs. Lyngby
{pksa|pop|jan}@imm.dtu.dk

ABSTRACT

In this paper we are interested in mixed-criticality embedded applications implemented on distributed architectures. Depending on their time-criticality, tasks can be hard or soft real-time and regarding safety-criticality, tasks can be fault-tolerant to transient faults, permanent faults, or have no dependability requirements. We use Earliest Deadline First (EDF) scheduling for the hard tasks and the Constant Bandwidth Server (CBS) for the soft tasks. The CBS parameters determine the quality of service (QoS) of soft tasks. Transient faults are tolerated using checkpointing with rollback recovery. For tolerating permanent faults in processors, we use task migration, i.e., restarting the safety-critical tasks on other processors. We propose a Greedy-based online heuristic for the migration of safety-critical tasks, in response to permanent faults, and the adjustment of CBS parameters on the target processors, such that the faults are tolerated, the deadlines for the hard real-time tasks are satisfied and the QoS for soft tasks is maximized. The proposed online adaptive approach has been evaluated using several synthetic benchmarks and a real-life case study.

1. INTRODUCTION

Traditionally, hard and soft real-time systems have been implemented using very different techniques [10]. However, many applications have both hard and soft constraints [9], hence a unified approach is required. Moreover, economic pressures and multi-core architectures are driving the integration of several levels of safety-criticality onto the same platform. Such applications have been traditionally designed using static approaches [10].

However, in many application areas, static approaches are no longer feasible due to increasing complexity and tight cost constraints, and more flexible solutions are required. Hence, researchers are advocating *adaptive* approaches, which are able to provide "operational flexibility" [3], by changing at runtime the system configuration in response to changes in the requirements, environment, fault occurrences etc. Adaptive embedded systems are able to use resources more efficiently, provide graceful degradation, leading to reduced costs and increased dependability [3].

Researchers have proposed several hardware architecture solutions, such as TTA [10], that rely on hardware redundancy to tolerate permanent faults. Such approaches can also be used for tolerating transient faults but they incur a very large hardware cost, since transient faults are more numerous [9]. Alternatives to such purely hardware-based solutions are approaches such as software replication, re-execution and checkpointing. Fault-tolerance has been addressed separately for hard [7] and soft [4] systems. Recently, Izosimov et al. [9] have considered mixed hard/soft real-time applications, and shown how quasi-static schedules can adapt at runtime to the actual execution times of tasks and to transient faults.

Abeni and Buttazzo [1] have proposed the Constant Bandwidth Server (CBS) for integrating hard and soft tasks on the same processor. CBS is used in conjunction with a scheduling technique such as EDF or Rate Monotonic (RM), which guarantees the deadlines of hard tasks and schedules the servers. The soft tasks are scheduled by the servers, and the server parameters, i.e., the period T_i and bandwidth Q_i, determine the QoS for the particular soft task. Abeni et al. have later shown [2] how the server parameters can be adaptively adjusted at runtime using a PID controller in order to maximize the QoS of soft tasks. Offline and online techniques for the derivation of CBS parameters have been proposed in [13], aiming at increasing the "benefit"İ associated to soft tasks.

In this paper we use EDF for the hard tasks and CBS for the soft tasks. Transient faults are tolerated using checkpointing with rollback recovery and we use task migration for tolerating permanent faults. Researchers have addressed task migration for load [6] and thermal [12] balancing, for improving performance [14] and for tolerating faults[5]. However, none of the existing approaches can handle mixed-criticality hard/soft applications. Hence, we propose a Greedy-based online heuristic for the migration of safety-critical tasks, in response to permanent faults, and the adjustment of CBS parameters on the target processors, such that the faults are tolerated, the deadlines for the hard real-time tasks are satisfied and the QoS for soft real-time tasks is maximized (see Fig. 1 for an illustration of the problem)

2. APPLICATION MODEL

We model an application as a set $\mathcal{A}$ of interacting tasks $\tau_i \in \mathcal{A}$. The tasks are mapped on a distributed architecture. The mapping is determined by a function $M : \mathcal{A} \rightarrow \mathcal{N}$ where $\mathcal{N}$ is the set of processing elements in the architecture. The mixed-criticality requirements of each task are captured by two functions: $F : \mathcal{A} \rightarrow \{Permanent,\ Transient\ \&\ Permanent,\ \phi\}$ determines if the task has to tolerate permanent and/or transient faults, or does not have safety-critical requirements. $R : \mathcal{A} \rightarrow \{Hard, Soft\}$ determines if the task is hard or soft real-time, respectively. Hard real-time tasks will always have safety-critical requirements i.e., $\forall \tau_i : R(\tau_i) = Hard\, F(\tau_i) \neq \phi$.

Tasks communicate asynchronously through buffers. The buffer sizes have been determined such that there is no overflow or underflow [8, 11]. Tasks are periodic: each $\tau_i \in \mathcal{A}$ has a period T_i. If needed, traffic shapers[16] can be used to ensure periodic behavior.

Hard real-time tasks are characterized by their worst case execution times (WCETs) C_i and a deadline D_i. Soft tasks are characterized by the probability distribution functions (PDFs) U_i of their execution times and a soft deadline δ_i. $U_i(c)$ is the probability that the job $J_{i,k}$ of τ_i has an execution time of c. The QoS of a soft task τ_i is defined as the probability of meeting the deadline δ_i, i.e., $QoS(\delta_i) = P\{f_{i,k} \leq r_{i,k} + \delta_i\}$, where $f_{i,k}$ and $r_{i,k}$ are the finishing time and the arrival time of the k^{th} job of soft task τ_i, respectively. Missing this deadline would not cause a catastrophic failure in the system, but only lead to certain performance degradation. The WCETs and PDFs are different for different task–processor mappings.

3. PLATFORM MODEL

We consider hardware architectures consisting of a set $\mathcal{N}$ of heterogeneous processing elements (PEs), interconnected by a communication channel. PEs have access to a shared memory where the code of tasks and the *checkpoints* (last non-faulty state of a task) are stored. The software architecture is composed of an EDF scheduler and a middleware implementing the proposed online task migration mechanism. The soft tasks are scheduled using CBS. The hard tasks and CB servers are scheduled using EDF. CBS enforces temporal isolation between hard and soft real-time tasks, thus guaranteeing the schedulability of hard tasks.

Thus, each soft real-time task τ_i is assigned a CBS, characterized by the tuple (Q_i, T_i), where Q_i (also called as bandwidth) is the time that the soft task τ_i is allowed to use the CPU every period T_i[1]. Each time τ_i demands more than its allocated Q_i, the CB server postpones the deadline δ_i with T_i. The EDF scheduler will schedule τ_i using this new deadline thus ensuring that a soft task will never demand more than its assigned bandwidth.

We assume that there can be at most k transient faults within an execution cycle of the application $\mathcal{A}$. There can be any number of permanent faults in the PEs of the system, but beyond a certain number, the safety and timeliness requirements of $\mathcal{A}$ can no longer be satisfied. Transient faults are tolerated using equidistant checkpointing with rollback recovery [15], which uses time redundancy to tolerate transient faults. Permanent faults are tolerated using task migration, i.e., starting the safety-critical tasks on other processors.

[1]The period of the server is equal to the period of the soft task.

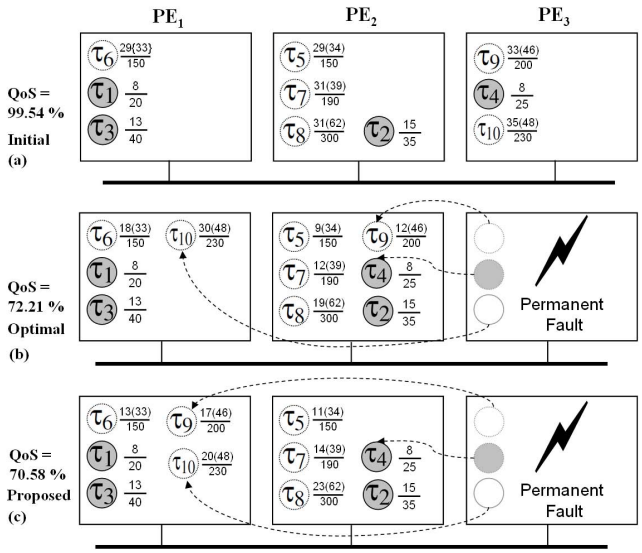

Figure 1: Optimal vs Greedy mapping

The principle of checkpointing is to restore the last non-faulty state (checkpoint) of the failing task, i.e., to *recover* from fault. The checkpoint has to be saved in advance into a stable storage, and will be restored if the task fails. The part of the task between two checkpoints is called *execution segment*. In our approach, we use the shared memory to store the checkpoints. If a permanent error is detected, the task will be recovered on other processors from its last checkpoint stored (if exists, otherwise from its start). Our proposed online task migration technique will decide on which of the healthy PEs to recover the tasks. The error detection and fault-tolerance mechanisms are part of the software architecture. Our approach takes into account the error detection and recovery overheads (for both transient and permanent faults). The use of shared memory checkpoints significantly reduces the task migration overhead. Our technique is also applicable on architectures that do not use shared memory. In that case, the overheads related to checkpointing and task migration will be different.

4. PROBLEM FORMULATION

As an input to our problem, we have a set of applications, modeled as presented in Section 2, mapped on a platform as described in Section 3. The mapping of tasks and CB server parameters (Q_i, T_i) are known. Given a set $\mathcal{N}_\mathcal{F}$ of PEs that have permanent faults, we are interested to determine for each failed safety-critical task τ_i (i.e, $M(\tau_i) \in \mathcal{N}_\mathcal{F}$ and $F(\tau_i) \neq \phi$), the target PE $X(\tau_i)$ where to migrate τ_i. We are also interested to calculate the bandwidth Q_i of all the servers on the target PEs. The migration and bandwidth calculation should be done such that the deadlines for the hard real-time tasks are satisfied (even in the case of tran-

sient faults) and the QoS of soft tasks is maximized (i.e., the system degrades gracefully) considering the remaining available resources ($\mathcal{N} \setminus \mathcal{N}_{\mathcal{F}}$).

Let us consider Fig. 1 where we have a system with 3 PEs, 4 hard real-time tasks (τ_1–τ_4), depicted in grey, and 6 soft real-time tasks (τ_5–τ_{10}). All tasks have to tolerate permanent faults (for simplicity, let's ignore transient faults in this example). The initial mapping is presented in the Fig. 1(a). The WCETs and periods for hard tasks are denoted as a fraction $\frac{C_i}{T_i}$ next to each hard task. The deadlines are equal to the periods. The bandwidth Q_i, soft deadline δ_i and the period T_i of the servers associated to each soft task are depicted as $\frac{Q_i(\delta_i)}{T_i}$. Each soft task has a different expectation value $E(c_i)$ for each PE. $E(c_i)$ of the PDF U_i is the probability-weighted sum of possible values of execution times c_i. We could not present the PDFs here due to lack of space, but they are available in the technical report. The expectations of the tasks τ_5–τ_{10} for PEs $\{PE_1, PE_2, PE_3\}$ are $\{25, 30, 32\}$, $\{27, 29, 31\}$, $\{29, 31, 32\}$, $\{28, 31, 33\}$, $\{26, 29, 33\}$, $\{28, 31, 34\}$, respectively.

Further, let us assume that PE_3 has a permanent fault. We are interested to decide where to migrate tasks τ_4, τ_9 and τ_{10} from PE_3 and how to calculate the new bandwidth of the servers on the target PEs such that all hard tasks meet their deadlines and the total QoS of all soft tasks is maximized. Fig. 1(b) shows the optimal mapping and bandwidth allocation solution, which guarantees all the deadlines for the hard tasks and results in a maximum QoS value of 72.21% (see next section on how the total QoS is calculated).

The decision on task migration and CBS bandwidth allocation cannot be taken offline. We don't know the pattern of permanent faults in advance (and would have to store too many possible combinations) and we also assume that the system is adaptive and can evolve beyond the initial implementation (both in terms of mapping and bandwidth). Determining online the optimal solution is infeasible due to the large computation times required. Hence, we are interested in an online heuristic that can quickly find a good solution. The proposed greedy task migration and bandwidth allocation heuristic will produce online the solution depicted in Fig. 1(c), which meets all the hard deadlines and results in a total QoS of 70.50%, very close to the optimal solution.

5. ONLINE TASK MIGRATION

5.1 *Schedulability Analysis*

We assume that the deadlines for hard real-time tasks are equal to the periods[2], and we use the utilization-based test [10]:

$$\sum_{\forall \tau_i : R(\tau_i) = Hard} \frac{C_i}{T_i} + \sum_{\forall \tau_i : R(\tau_i) = Soft} \frac{Q_i}{T_i} \leq 1.$$

For hard tasks, the WCET C_i contains the checkpointing and recovery overhead considering m_i equidistant checkpoints used to tolerate k transient faults [7]. For soft tasks, we assume that the PDFs take into account the checkpointing overhead. The number of checkpoints are specified by the designer [15] for each safety-critical task. The schedulability of the soft task τ_i (i.e., the probability of meeting its deadline δ_i) depends on the bandwidth Q_i. However, it should be noted that the relationship between the QoS and Q is not linear. For example, in Fig. 1, increasing $Q_8 = 20$ with 2 will lead to a QoS of 88.56% from 73.16% (an increase of 15.4%). However, further increasing Q_8 by 2 will only increase the QoS by 0.68% to 89.24%. The relation between Q and QoS is different for each process depending on its PDF shape.

[2]Arbitrary deadlines can be handled using the Processor Demand Criterion.

Algorithm 1 $TMBA$
Task Migration and Bandwidth Allocation

```
1: input: set of failed processors {N_F}
2: for all τ_i such that M(τ_i) ∈ N_F and F(τ_i) ≠ φ do
3:   QoS_best = 0
4:   for all PE_j ∈ N \ N_F do
5:     U_i = GetUtil(τ_i);
       U_PEj = Σ_{∀τ_k : M(τ_k) ∈ PE_j} GetUtil(τ_k)
6:     if 1 − U_PEj < U_i then
7:       AdjustQsProportionally(τ_i, PE_j)
8:     end if
9:     QoS_curr = CalculateQoS()
10:    if QoS_curr > QoS_best then
11:      PE_best = PE_j; QoS_best = QoS_curr
12:    end if
13:    UndoAdjustQs(PE_j)
14:  end for
15:  MigrateTask(τ_i, PE_best)
16: end for
```

Due to the temporal isolation property of CBS, each soft task can be analyzed individually. This property is particularly useful for an online approach: instead of performing the (time consuming) analysis online, the calculation can be done offline for all Q_i values and stored for online use. The number of Q_i values to be stored is limited. If $Q_i \geq WCET$ all deadlines are satisfied, whereas if $Q_i < AET$, the response time of the soft task would be infinite.

For a given soft real-time task τ_i mapped on a processor PE_j, $QoS(\tau_i, PE_j)$ is the probability that its job $J_{i,k}$ finishes before a deadline $d_{i,k} = r_{i,k} + \delta_i$ where $r_{i,k}$ is the arrival time of the job. This probability is calculated by modeling the CBS (serving the soft tasks τ_i) as a queuing system [2]. The arriving jobs $J_{i,k}$ are seen as tokens to be served by the server having the capacity Q_i. The length of queue when each job $J_{i,k}$ arrives is defined by a random process v_k. A state probability vector $\pi_m^{(k)} = P\{v_k = m\}$ for this random process is calculated, which captures the probability of having a certain length of the queue. Our heuristic will decided on the appropriate Q values that maximize the total QoS after task migration. By solving for the stationary solution of this state probability vector, the probability of meeting a certain deadline can be calculated. The detailed derivation can be seen in [2]. The total QoS of the system is calculated as a normalized sum over the QoS of each soft task. The designer can give weights to individual soft tasks to differ-

No.	*Avg. Util.*	*Init. QoS*	*PEs. Tot.(Failed)*	*Tasks Tot.(Migrated)*	*TMBA QoS(Time)*	*TS QoS*
1	92.97	99.53	3 (1)	10 (3)	70.85 (23)	71.84
2	92.82	99.26	4 (1)	16 (5)	71.89 (62)	73.41
3	93.26	99.79	5 (1)	21 (6)	74.82 (74)	75.23
4	93.16	99.51	7 (2)	29 (10)	77.01 (131)	77.60
5	92.30	99.78	8 (2)	33 (11)	77.90 (179)	78.75
6	92.86	99.79	9 (2)	37 (11)	80.03 (213)	80.81
7	92.28	99.78	10 (2)	49 (12)	81.70 (268)	81.97
8	92.22	99.84	16 (3)	67 (14)	88.58 (306)	88.88
9	93.29	99.82	18 (3)	78 (15)	88.99 (704)	89.28

Table 1: Experimental Results 1

No.	*Avg. Util.*	*Init. QoS*	*TMBA QoS*	*TS QoS*
1	95.38	99.38	67.91	68.95
2	90.10	99.35	70.22	73.02
3	85.01	99.39	75.36	77.04
4	80.07	99.38	79.80	82.04
5	75.02	99.37	82.45	85.84
6	70.11	99.35	86.33	89.92
7	65.08	99.45	88.62	91.36
8	60.08	99.40	93.60	94.23
9	55.09	99.41	93.62	96.32
10	50.07	99.42	96.28	96.32

Table 2: Experimental Results 2

entiate their importance.

5.2 *Greedy-Based Online Task Migration*

We have proposed a greedy-based online heuristic for Task Migration and Bandwidth Allocation (TMBA). A greedy algorithm works by making a locally optimal choice at each stage. The algorithm is presented in Alg. 1 and takes as input the set of failed PEs $\mathcal{N}_{\mathcal{F}}$ (more than one PE can fail). For each failed task τ_i the heuristic selects each healthy PE_j as a candidate for migration (lines 4–16). The hard tasks are considered before the soft tasks, and the tasks are ordered on the decreasing order of their utilization, i.e., $\frac{C_i}{T_i}$ for hard tasks and $\frac{E(c_i)}{T_i}$ for soft tasks. The utilization needed by τ_i is calculated in line 5. The utilization available on PE_j is found out by summing the utilizations of all the tasks mapped on PE_j (line 5).

If the available utilization on the processor is less than the utilization desired by the migrating task τ_i (line 6), the bandwidth Q associated to the soft tasks on that processor is decreased in proportion to their expectations, and then the task τ_i can be mapped on the processor. If the available utilization is enough, τ_i can be simply migrated without any adjustments. The QoS of this migration and bandwidth allocation decision is calculated by summing over the QoS of all soft tasks on the system (line 9). The PE which gives the best QoS is selected as the target for migration (lines 10–11).

The complexity of our greedy algorithm is polynomial. Note that for each soft task τ_i we compute offline the QoS value $QoS(\tau_i, PE_j)$ for all integer values of Q_i, with $E(c_i) \leq Q_i < WCET\,of\,\tau_i$. Thus, the QoS calculation for each bandwidth adjustment is a simple table lookup.

Considering the example in Fig. 1, the algorithm will first select τ_4 as it is the only hard task on the faulty processor, and consider PE_1 as the candidate for migration. The utilization needed by this task is 0.32, which is not available on PE_1. Even by reducing the bandwidth of soft task τ_6 to zero, τ_4 would not meet its deadline (the resulted task set on PE_1 is not schedulable). Hence, the algorithm selects next PE_2. If τ_4 is migrated to PE_2, the total utilization available for the soft tasks (τ_5, τ_7 and τ_8) is $1 - (\frac{15}{35} + \frac{8}{25})$ = 0.26. The utilization desired by τ_4 can be made available by decreasing the Q values of soft tasks. The adjustment is done in proportion to their expectations. τ_5, τ_7 and τ_8 have expectation values of 28, 29 and 31, respectively. Thus, the utilization available for τ_5 would be $\frac{28}{28+29+31} \times 0.26 = 0.09$ and the new Q value would be $0.09 \times 150 = 12$. Similarly, the Q values for other tasks are calculated. The QoS of this solution is 55%. Next, the soft task τ_{10} is selected for migration (before τ_9, since it has a greater utilization). The algorithm will produce the solution as depicted in Fig. 1(c).

6. EXPERIMENTAL EVALUATION

In the first set of experiments we were interested to determine the quality of the proposed heuristic. For this, we have used nine synthetic benchmarks of 10 to 78 tasks mapped on architectures with 3 to 18 PEs. We have used WCETs in the interval 3 to 18 ms and have generated PDFs to match the shape of those measured on real life benchmarks such as an MPEG decoder, with expectations between 15 to 60 ms. The applications were implemented such that the QoS is close to 100 % and there is on average 7 % spare utilization on each PE. All tasks are safety-critical and have to tolerate permanent faults and one transient fault. We have varied the number of permanent faults from 1 to 3, depending on the number of nodes in the architecture.

Table 1 presents the experimental setup details and the value of QoS (column 6, considering equal weights for the soft tasks) obtained after applying our TMBA online greedy heuristic. To determine the quality of TMBA we have compared these results with the results obtained by a Tabu-Search (TS) metaheuristic presented in the last column, which uses mapping and bandwidth allocation design transformations to explore the solution space. We have determined the parameters of TS such that the results are as close as possible to the optimal solution (i.e., no improvements were seen for large number of iterations). TS runs on average for 352 min., which makes it unsuitable to be used online.

The TMBA heuristic runs in polynomial time (see previous section) and the computation times (in milliseconds) on a 2.5 GHz Intel machine running Ubuntu Linux 8.04 are also presented in column 6. It can be seen from the results that the QoS values obtained by TMBA differ from those of TS by less than 1%. The obtained results are very close to those obtained by TS (QoS difference of only 0.66% on average; the hard deadlines were satisfied in both cases). Note that TMBA only performs mapping decisions for the failed tasks

(3 to 15 tasks), and the proportional bandwidth allocation used (see previous section) is often quite close to the optimal allocation.

For the second set of experiments as shown in Table 2, we have investigated how TMBA handles different levels of utilization. We have used a synthetic benchmark with 33 tasks (19 soft, 14 hard) mapped on an architecture with 8 nodes. By varying the execution times, we have considered 10 cases, where we have increased the utilization from 50% to 95%. In all the cases, *QoS* is close to 100%. We have applied TMBA considering two permanent faults, obtaining a schedulable system with *QoS* between 67.91–96.28%. Compared to TS, we have noticed that quality of solution given by TMBA degrades slightly from a difference of 0.05% for 50% utilization to a difference of 1.5% for 95% utilization.

We have also tested our approach on a real-life case study of a *portable media player*. The media player application consists of a total of 20 soft real-time tasks and 10 hard real-time tasks running on 4 processors. The frame rate of displayed video should be 25 FPS and the audio should be stereo with a bit rate of 300 Kbps for a good perceived *QoS*. We simulated that one of the processors failed and the tasks had to be migrated. The optimal mapping found out by TS gave a *QoS* of 73.42%, while *QoS* reported by TMBA (within 214 ms) was 74.19%, a difference of only 0.77%.

7. CONCLUSIONS

In this paper, we have addressed the issue of task migration in mixed-criticality embedded systems to tolerate permanent faults. We have proposed a Greedy-based online heuristic which performs the migration of safety-critical tasks, in response to permanent faults, and adjusts the CBS parameters on the target processors, such that the faults are tolerated, the deadlines for the hard real-time tasks are satisfied and the QoS for soft real-time tasks is maximized.

8. REFERENCES

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proc. of 19th IEEE Real-Time Systems Symp.*, pages 4–13, 1998.

[2] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *Proc. 23rd IEEE Real-Time Systems Symp.*, pages 71–80, 3–5 Dec. 2002.

[3] L. Almeida, S. Fischmeister, M. Anand, and I. Lee. A dynamic scheduling approach to designing flexible safety-critical systems. *Proc. of 7th ACM and IEEE Int. Conf. on Embedded Soft.*, pages 67–74, 2007.

[4] H. Aydin, R. Melhem, and D. Mosse. Tolerating faults while maximizing reward. In *Proc. of 12th Euromicro Conf. on Real-Time Systems*, pages 219–226, 2000.

[5] A. A. Bertossi and L. V. Mancini. Scheduling algorithms for fault-tolerance in hard-real-time systems. *Real-Time Syst.*, 7(3):229–245, 1994.

[6] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali. Supporting task migration in multi-processor systems-on-chip: A feasibility study. *Proc. of Design, Autom. & Test in Europe Conf.*, pages 1–6, 2006.

[7] A. Burns, R. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. *Euromicro Conf. on Real-Time Systems*, pages 29–33, 1996.

[8] J. Hu and R. Marculescu. Application-specific buffer space allocation for networks-on-chip router design. pages 354–361, 2004.

[9] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Scheduling of fault-tolerant embedded systems with soft and hard timing constraints. In *Proc. of Design, Autom. & Test in Europe Conf.*, pages 915–920, 2008.

[10] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[11] S. Manolache, P. Eles, and Z. Peng. Buffer space optimisation with communication synthesis and traffic shaping for nocs. pages 718–723, 2006.

[12] F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, D. Atienza, and G. De Micheli. Thermal balancing policy for streaming computing on multiprocessor architectures. In *Proc. of Design, Autom. & Test in Europe Conf.*, pages 734–739, 2008.

[13] A. Oliveira, E. Camponogara, and G. Lima. Dynamic reconfiguration in reservation-based scheduling: An optimization approach. In *Proc. of 15th IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 173–182, 2009.

[14] M. Pittau, A. Alimonda, S. Carta, and A. Acquaviva. Impact of task migration on streaming multimedia for embedded multiprocessors: A quantitative evaluation. In *Proc. of IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia*, pages 59–64, 2007.

[15] P. Pop, V. Izosimov, P. Eles, and Z. Peng. Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication. *IEEE Transactions on VLSI Systems*, 17:389–402, 2009.

[16] E. Wandeler, A. Maxiaguine, and L. Thiele. Performance analysis of greedy shapers in real-time systems. In *Proc. Design, Automation and Test in Europe DATE '06*, volume 1, page 6pp., 6–10 March 2006.