Reliability Improvements for Multiprocessor Systems by Health-Aware Task Scheduling

Robert Schmidt*, Rehab Massoud*, Jaan Raik†, Alberto García-Ortiz*, Rolf Drechsler*
*University of Bremen, agarcia@item.uni-bremen.de, {rschmidt,massoud,drechsler}@uni-bremen.de
†Tallin University of Technology, jaan.raik@ati.ttu.ee

***Abstract*—Multiprocessor systems are increasingly susceptible to faults due to shrinking feature sizes and denser integration of many cores. Fault activation correlates with reliability degradations, which are unacceptable for multiprocessor systems in critical applications. The reliability of multiprocessor systems can be improved exploiting its intrinsic redundancy, but current coarse-grained solutions wastefully require spare cores, which are not always available. This work proposes a health-aware task scheduling that leverages fine-grained intrinsic redundancy, without demanding spare cores, to obtain graceful degrading multiprocessor systems. Thorough simulation results are reported to quantify the advantages of our approach. With our graceful degradation approach the lifetime of a multiprocessor system improves by a factor of up to 2.32 for equal reliability levels.**

I. Introduction

The probability that a particular device is operational for a given duration, or reliability, is a dependability attribute and key metric for systems in critical applications. For example, systems for long-term autonomous exploration missions have to be operational during their complete mission. Other critical applications like banking, medical, automotive or aerospace face similar reliability requirements that are only met by dependable systems. Traditional dependable systems, compared to their non-dependable counterparts, have three key issues: They are more expensive, consume more power, and provide less performance.

A promising approach to overcome the previous limits is hardening commercial off-the-shelf (COTS) components, which are cheaper, faster, and consume less power, to build systems that meet the mission's dependability requirements [1]. The dependability challenge of COTS components for long term autonomous missions is to guarantee a certain reliability without affecting their affordability, power consumption, and performance benefits. The COTS challenge's relevance rises with the increasing demand for dependable systems by emerging artificial intelligence applications in autonomous systems. Most such systems are cyber-physical systems, interacting with their environment, which require high computational capabilities. These high computational capability demands are satisfied by multiprocessor systems.

Multiprocessor systems have some intrinsic redundancy, which can be leveraged by dependability-aware task scheduling. Task scheduling solutions are classified as either static or dynamic. Static scheduling assigns tasks ahead of actual execution to processing elements and requires a well characterized workload and system topology. Dynamic scheduling assigns tasks to processing elements at run time, overcoming worst-case assumptions for performance or failure related metrics.

Current dependability-aware global static scheduling approaches waste slack available at run time [2], and depend on a specific fault scenario, with exponentially growing solution storage and access overhead [3]. Dynamic scheduling overcomes worst-case assumptions, but current dependability-aware approaches either assume spare homogeneous cores, or are limited to slowly accumulating faults [4, 5].

We present a global dynamic scheduling solution, suitable for multiprocessor systems without spare cores, to improve reliability by graceful degradation. Our contributions are:

- a fine-grain CPU graceful degradation model to expose the intrinsic redundancy and modularity in a microarchitecture (Section III-A)
- results from simulation experiments showing improvements up to a factor of 2.32 in operational time with a reliability of $R = 0.99$ by using our approach (Section IV-A)

With our fine-grain dynamic scheduling approach the reliability of multiprocessor systems is asserted, without compromising their affordability, power consumption, and performance compared to traditional dependable systems.

II. Related work

Task allocation and scheduling problems are alternative formulations of the same general scheduling problem.For a single processor, scheduling is local and concerned with assigning tasks to processing time. For multiprocessor systems, additional global scheduling is needed to assign tasks to processing elements. Scheduling solutions are further classified as static or dynamic.

Related to our approach, system lifetime maximization by global dynamic scheduling for heterogeneous multiprocessor systems [6] uses a heuristic to solve the scheduling problem at run time, but requires special wear sensors which are not widely available. Comparable to us, the envisioned DeSyRe system can detect submodule failures and reschedule tasks to cores that provide the required functionality [7]. Although, their approach requires code annotations and duplicate code for applications that shall run on cores with failed submodules, exposing the redundancy scheme and burden to the application developer. Similar, proposed run-time heuristics for managed heterogeneous and degradable cores requires hardware reconfiguration capabilities [8], which is not always available. Complementary to our approach, health management based on IEEE 1687 infrastructure [9] provides an interconnect solution for health

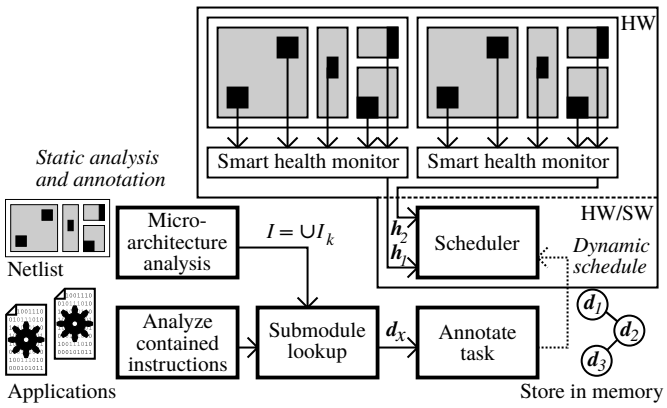

Figure 1. Combined static analysis and dynamic scheduling approach.

information and implementation details for a fault management architecture, which corresponds to the health monitors in our approach. Exploitation of the microarchitecture's existing redundancy for graceful degradation is expected to increase the baseline reliability by a factor of 1.42 for a single core [10], but neglects the potential for even further single core degradation in a multi-core system. Orthogonal to our approach, bespoke processors remove unused gates by symbolic simulation [11], to improve power instead of reliability.

III. Implementation

Our approach, as shown in Figure 1, is a global dynamic task scheduling approach enhanced with static knowledge $\boldsymbol{d}_x$ about each task, and dynamic knowledge about the current health status $\boldsymbol{h}_i$ of each processor. The processor's microarchitecture and all applications are analyzed at design time. At runtime, the scheduler is provided with the health status of each processor, which allows to schedule the annotated tasks in a health-aware manner. The fine-grain health status of each processor allows to schedule tasks on partially defect processors. This way they can still contribute to system performance and reliability.

A. Graceful degradation model

Modern CPUs are hierarchical and modular: Their microarchitecture, which implements the instruction set architecture, consists of several submodules. Submodules are either macro units, intellectual property cores, or arise by functional decomposition. Not all submodules are required to successfully execute every instruction. Such instructions add to the microarchitecture's graceful degradation potential, which we leverage by our fine-grain health-aware task scheduling. For this, the scheduler requires two sources of information: *1)* health status of each submodule in each core; and *2)* which submodules are required for each instruction. The later is derived by our design-time approach described in this section.

We divide the CPU's microarchitecture by connectivity into submodules. Submodules needed for successful execution of an instruction are added to the instruction's submodule dependency set I_k. These sets allow to derive the submodule dependencies of each task: they inherit the submodule dependencies of all their instructions, further described in Section III-B. Such annotated tasks enable the scheduler to schedule them on cores that have at least the required submodules error-free.

Our approach is formalized using set theory as follows: We divide the CPU's microarchitecture into N submodules m_j. Together all submodules form the whole microarchitecture submodule set $M = \{ m_j \mid j \in \mathbb{N} \wedge j < N \}$. Two subsets of M define each instruction's submodule dependencies: *1)* the functional submodule dependence set F_k, including all submodules needed for execution of instruction i_k; and *2)* the set of submodules that are required to be error-free E_k, taking into consideration the fault containment of the microarchitecture

The functional submodule dependence set F_k is acquired by semi-automatic microarchitecture analysis: $F_k = \{ m_j \mid \forall m_j \in M$ if m_j required by $i_k \}$. The error-free set E_k is derived by analysis of the remaining submodules $R_k = M_k \backslash F_k$, which are not required from a functional point of view: $E_k = \{ m_j \mid \forall m_j \in R_k \forall m_i \in F_k$ if error in m_j can propagate to $m_i \}$. Together F_k and E_k form the submodule dependency set for instruction i_k: $I_k = F_k \cup E_k$. If I_k is a strict subset of M, instruction i_k adds to the fine-grain graceful degradation potential of the microarchitecture, because a core with errors in the redundant submodules $M \setminus I_k$ is still able to execute i_k. This implicitly assumes that failures in the redundant modules do not stop the overall operation of the processor. The graceful degradation potential G of a microarchitecture which implements J instructions is $G = \sum_k^J |M \setminus I_k| / (J|M|)$, where $|M|$ denotes the cardinality of set M.

As an example we choose, due to public-domain availability, the Plasma MIPS I microarchitecture. Plasma is a conservative choice to demonstrate our approach, because the minimal microarchitecture provides only few submodules, limiting the graceful degradation potential. Despite these limits, the microarchitecture has three potential submodules for graceful degradation, which we evaluate for stuck-at faults: The arithmetic and logical unit, the shifter, and the multiplier. The register file, program counter, control path, and all remaining logic and memory are considered by annotating each instruction with a dependence on a further virtual submodule containing them. We analyzed the data path semi-automatically using Yosys to obtain the submodule connectivity.

B. Task model

The graceful degradation model, described in the last section, is made accessible to the scheduler by our task model. Our task model builds upon task graphs, extending them by our submodule dependence information: programs are modeled as directed acyclic graphs, where a node corresponds to an instruction sequence, or task, and each edge corresponds to a data dependence between tasks. All tasks are element of the task set V, and all edges are element of the edge set $E = \{ (v_i, v_j) \mid$ if execution of v_i needs to precede $v_j \}$. Together both sets form a tuple $P = (V, E)$ representing a program as a task graph.

Each task is annotated with the submodules it requires for execution, formalized as a submodule dependence vector

Address	Mnemonic	Submod. dependency I_k
`0040FA2C`	mult a3, t6	$I_{\text{mult}} = \{m_{\text{MULT}}\}$
`0040FA30`	andi v1, v1, 0xffff	$I_{\text{andi}} = \{m_{\text{ALU}}\}$
`0040FA34`	addiu a3, t7, 4	$I_{\text{addiu}} = \{m_{\text{ALU}}\}$
`0040FA38`	sll v1, v1, 2	$I_{\text{sll}} = \{m_{\text{SHIFT}}\}$

$$D_j = I_{\text{mult}} \cup I_{\text{andi}} \cup I_{\text{addiu}} \cup I_{\text{sll}} = \{m_{\text{MULT}}, m_{\text{ALU}}, m_{\text{SHIFT}}\}$$

Figure 2. Example annotation of task j. Instructions introduce submodule dependencies, resulting in a larger submodule dependence set D_j.

$\boldsymbol{d} \in \mathbb{B}^N, \mathbb{B} = \{0, 1\}$. The submodule dependence vectors are constructed from the submodule dependence sets of each instruction contained within the task: Tasks are sequences of n instructions i_k, which are elements of the instruction sequence set Q_j, with the instructions corresponding submodule dependence sets I_k. Their union, as shown in Figure 2, is the task's submodule dependence set $D_j = \bigcup_{k \in Q_j} I_k$. Such sets are encoded by $\varepsilon : D_j \to \mathbb{B}^N$ into submodule dependency vectors $\boldsymbol{d}_j$:

$$\boldsymbol{d}_j = \begin{bmatrix} m_1 & m_2 & \dots & m_N \end{bmatrix} \in \mathbb{B}^N : m_i = \begin{cases} 1 & m_i \in D_j \\ 0 & m_i \notin D_j \end{cases}$$

The vector $\boldsymbol{d}_j$ represents all submodule dependencies for task j, and is used by the scheduler described in Section III-C to calculate all scheduling possibilities for task j.

C. Task scheduler

The submodule dependencies $\boldsymbol{d}_j$ of task j can be evaluated as a scheduling location constraint irrespective of the implemented scheduling algorithm. For each processor p_i in our multiprocessor system the scheduler receives health status $\boldsymbol{h}_i = [h_{i1}, h_{i2}, \dots, h_{iN}]$ for all N submodules of p_i. Given a task x's submodule dependence vector $\boldsymbol{d}_x = [d_{x0}, d_{x1}, \dots, d_{xN}]$, and binary health status we can calculate the schedulability $s_{i,x} = \bigwedge_j h_{ij} \vee \neg d_{xj}$ of task x on processor p_i. If at least one schedulability evaluates to true, task x is executable by the multiprocessor system. Calculating the schedulability is computationally inexpensive, which allows even real-time constrained schedulers to support our additional scheduling location constraint.

D. Multiprocessor system

Our system contains multiple degradable processors and a hardened runtime manager. The runtime manager is either dedicated hardware, or another processor, protected either by process, design, or software techniques, executing the scheduler as part of the operating system. The operating system's scheduler supports tasks with deadlines, and failure to schedule a task in time is regarded as a system failure.

Each processor provides for all its submodules a binary status flag, or health information, which is generated by a function which maps available error information to module errors. This mapping function is implemented inside a health monitor, which collects error information during system runtime from logical checkers, built-in self-test routines, voltage- and temperature sensors, or dedicated delay/ageing detection circuits [12]. The health monitor, as part of the runtime manager beyond the scope of this paper, is either a software or hardware module, capable to transform various error information into submodule health information, and is treated as a black box module by our scheduling approach. Our approach is agnostic to the error's origin, because no matter if the error is due to radiation, timing, or manufacturing defects, the error information is condensed to a binary flag.

The system is designed for an average task arrival rate per processor μ given a workload of reoccurring tasks. Under error-free operating conditions the system has enough spare capacity to schedule and finish all tasks before their deadline. But with increasing errors the scheduling possibilities decrease, and the delay until a suitable processor is found increases. This delay reduces the time available for execution before the deadline, increasing the risk of missing the deadline. Once a deadline is missed, the system has by definition failed. The probability to fail during the system's mission time is used to derive the system's reliability $R = 1 - \Pr(\text{deadline missed})$.

IV. Experimental setup & results

The multiprocessor system used for the evaluation of our approach consists of five bus-connected processors: One runtime manager and four workers. We consider a synthetic Plasma MIPS I microarchitecture with minimal submodule dependencies and homogeneous execution times for all instructions.

To model a workload for embedded real-time systems we choose applications from MiBench. All applications are compiled for MIPS I, and disassembled using Capstone, for submodule dependency annotation using instruction submodule dependency sets from our microarchitecture analysis. Applications are given a recurrence frequency and deadline to form a workload for our multiprocessor system.

The scheduler uses earliest deadline first scheduling to assign tasks to the first available processor. Each processor is capable to execute one task at a time. Over time, faults manifest and their activation results in errors which degrade the processor's functionality. Our graceful degradation approach with health monitors allows us to confine faults with submodule granularity. If a submodule is in error, the task currently running on the processor is interrupted and returns to the task queue to be restarted from the beginning on the next available processor which provides the required submodules for execution. The erroneous, degraded submodule either stays in error in case of permanent faults, or returns operational with a mean time to recovery in case of transient faults. The baseline system without our graceful degradation approach degrades and restores with processor granularity.

Our simulator allows us to evaluate systems for different workloads and fault environments with single instruction temporal granularity. Due to the random nature of faults we designed our experiment for $r = 100$ repetitions, with the multiprocessor system's operational time as response variable. We considered the degradation model, mean time to failure (MTTF), and task arrival rate as factors in our experiment and report the results in Section IV-A.

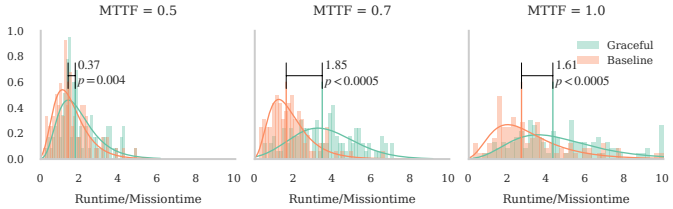

Figure 3. Density of system lifetime results with fitted log-normal approximation for the baseline system and our graceful approach.

A. Results & discussion

In this section we compare our graceful degradation approach with health monitors to a baseline systems with identical parameterization and fault environment. The fault environment is represented by the MTTF normalized to a predefined mission duration T_M. The interesting corner to quantify our approach are low workloads as they are common in automotive and aerospace applications. The chosen workload is reflected by the average task arrival rate per processor μ.

We report results for the median of the multiprocessor system's operational time, with and without our graceful degradation approach, normalized to T_M in Table I. As shown in Figure 3, the distribution of the operational time is approximately log-normal distributed. We use Welch's unequal variances t-test to test for our null hypothesis of equal medians, and report the smallest level of significance p that would lead to a rejection of the null hypothesis. We conclude that there is no evidence to suggest that the medians are equal for a significance level of $\alpha = 0.05$ across all interesting corners. By using our graceful degradation approach, we are able to extend the system's operational time by $1.85T_M$ for a average task arrival rate of $\mu = 0.11$ and MTTF of $0.7T_M$.

Furthermore the results in Table I confirm the intuition about the deteriorating effect of increasing the workload on operational lifetime. Two corner cases for an average task arrival rate of $\mu = 0.11$ are notable: very harsh fault environments with MTTF of $0.1T_M$ and very easy fault environments with MTTF of $9T_M$. Both cases are extreme up to the point that the differences between both systems are not relevant anymore. The harsh case, which is unlikely and not of practical interest for critical applications, exhausts the graceful degradation capabilities of our approach, as indicated by the lifetime degradation for $R = 0.99$ in Table I. In the easy case, the graceful degradation capabilities are only partly needed, and the differences between a system with- and without our approach are less severe.

Contrary, in the MTTF range of $0.7T_M$ to $3.0T_M$, our graceful degradation approach improves reliability up to $R = 0.99997$, which enables critical applications without demanding spare cores.

V. Conclusion

Our fine-grain CPU graceful degradation model successfully exposes the intrinsic redundancy of microarchitectures, eliminating the need for spare cores to assert reliability. Thorough simulation experiments confirm the effectiveness and significance of our graceful degradation approach to leverage fine-grained intrinsic redundancy of multiprocessor systems by global dynamic task scheduling, which extends the operational lifetime up to a factor of 2.32 while asserting reliability levels of $R = 0.99$. By extending the operational lifetime of multiprocessor systems we render long-term autonomous mission with COTS multiprocessor systems feasible, and provide a solution for the increasing demand of affordable dependable systems.

Table I

Comparison of baseline systems to graceful system in terms of reliability and extended system lifetime over different average arrival rates per processor μ and MTTF corners. System lifetime and MTTF are normalized to the planned mission duration.

		Lifetime median				Lifetime T for $R(t \le T) = 0.99$			Reliability $R(t > 1)$			Probability of failure $1 - R(t > 1)$	
μ	MTTF	Base	Grace	Factor	p	Base	Grace	Factor	Base	Grace	Factor	Base	Grace
0.11	0.1	0.26	0.45	1.71	<0.001	0.015 555	0.011 897	0.76	0.002 615	0.051 409	19.66	0.997 385	0.948 591
0.11	0.3	0.78	1.17	1.50	<0.001	0.011 206	0.019 646	1.75	0.325 049	0.598 037	1.84	0.674 951	0.401 964
0.11	0.5	1.41	1.77	1.26	0.004	0.013 393	0.014 751	1.10	0.714 986	0.841 337	1.18	0.285 014	0.158 663
0.11	0.7	1.62	3.47	2.15	<0.001	0.011 239	0.015 633	1.39	0.778 827	0.951 292	1.22	0.221 173	0.048 708
0.11	1.0	2.71	4.32	1.59	<0.001	0.011 085	0.025 740	2.32	0.920 987	0.982 772	1.07	0.079 014	0.017 228
0.11	3.0	7.21	8.79	1.22	<0.001	0.010 087	0.011 388	1.13	0.988 798	0.999 970	1.01	0.011 202	0.000 030
0.11	9.0	9.60	9.91	1.03	0.036	0.010 411	0.010 088	0.97	1.000 000	1.000 000	1.00	0.000 000	0.000 000
0.25	0.1	0.14	0.16	1.15	0.019	0.011 406	0.016 288	1.43	0.000 013	0.001 285	96.36	0.999 987	0.998 715
0.25	0.3	0.33	0.47	1.44	<0.001	0.011 539	0.014 696	1.27	0.025 254	0.144 605	5.73	0.974 746	0.855 395
0.25	0.5	0.47	0.89	1.89	<0.001	0.021 375	0.014 434	0.68	0.160 129	0.429 642	2.68	0.839 871	0.570 359
0.25	0.7	0.85	1.06	1.25	0.002	0.015 125	0.027 880	1.84	0.395 166	0.530 570	1.34	0.604 834	0.469 430
0.25	1.0	1.06	1.39	1.31	0.012	0.011 760	0.023 021	1.96	0.532 895	0.679 602	1.28	0.467 106	0.320 398
0.25	3.0	3.04	4.91	1.62	<0.001	0.010 722	0.010 809	1.01	0.912 674	0.975 746	1.07	0.087 326	0.024 254
0.25	9.0	8.27	8.63	1.04	0.290	0.012 107	0.011 579	0.96	0.997 963	0.999 145	1.00	0.002 037	0.000 855

References

[1] S. Esposito et al. (2015). COTS-Based High-Performance Computing for Space Applications. *IEEE Trans. Nucl. Sci.* 62(6):2687–2694.

[2] C. Bolchini et al. Run-Time Mapping for Reliable Many-Cores Based on Energy/Performance Trade-offs. *IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotech. Syst.* (2013), p. 58–64.

[3] A. Das et al. Fault-Aware Task Re-Mapping for Throughput Constrained Multimedia Applications on NoC-based MPSoCs. *23rd IEEE Int. Symp. Rapid Syst. Prototyping*. (2012), p. 149–155.

[4] C.-L. Chou et al. FARM: Fault-Aware Resource Management in NoC-based Multiprocessor Platforms. *Proc. Conf. Des., Autom. & Test Europe*. (2011).

[5] T. Chantem et al. Enhancing Multicore Reliability through Wear Compensation in Online Assignment and Scheduling. *Proc. Conf. Des., Autom. & Test Europe*. (2013).

[6] A. S. Hartman et al. Lifetime Improvement through Runtime Wear-based Task Mapping. *Proc. 8th IEEE/ACM Int. Conf. Hardware/Software Codesign Syst. Synthesis*. (2012), p. 13–22.

[7] D. Theodoropoulos et al. The DeSyRe Runtime support for Fault-tolerant Embedded MPSoCs. *Int. Symp. Parallel Distributed Process. Appl.* (2014), p. 197–204.

[8] S. Tzilis et al. Runtime Management of Adaptive MPSoCs for Graceful Degradation. *Proc. Int. Conf. Compilers, Architectures & Synthesis Embedded Syst.* (2016).

[9] K. Shibin et al. (2017). Health Management for Self-Aware SoCs Based on IEEE 1687 Infrastructure. *IEEE Des. Test* 34(6):27–35.

[10] J. Srinivasan et al. Exploiting Structural Duplication for Lifetime Reliability Enhancement. *Proc. 32nd Int. Symp. Comput. Architecture*. (2005), p. 1–12.

[11] H. Cherupalli et al. Bespoke Processors for Applications with Ultra-low Area and Power Constraints. *Proc. 44th Int. Symp. Comput. Architecture*. (2017).

[12] M. Agarwal et al. Circuit Failure Prediction and Its Application to Transistor Aging. *25th IEEE VLSI Test Symp.* (2007).