Compiler-Driven Dynamic Reliability Management for On-Chip Systems under Variabilities

Semeen Rehman, Florian Kriebel, Muhammad Shafique, Jörg Henkel
Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany
rehman@ira.uka.de, {florian.kriebel, muhammad.shafique, henkel}@kit.edu

***Abstract*—This paper presents a novel Dynamic Reliability Management System (DyReMS) for on-chip systems that performs resilience-driven resource allocation and mapping. It accounts for both the tasks' resilience properties and heterogeneous error recovery features of different cores. DyReMS also chooses a reliable task version (out of multiple reliability-aware transformed options) depending upon the reliability level of the allocated core. In case of error detection, rollbacks are performed. Our system provides 70%-87% improved task reliability compared to a timing reliability-optimizing core assignment, i.e. minimizing the probability of deadline misses (with EDF scheduling).**

Keywords—Reliability, Dependability, Soft Errors, Aging, Process Variations, Compiler, Run-Time Management.

I. Introduction and Motivation

Continuous transistor scaling in advanced technology nodes has enabled high core integration where on-chip systems feature 100s-1000s of cores [1][2]. However, this resulted in various reliability concerns like process variations, aging, soft errors, etc. The *process variations* and *aging* effects are typically manifested as design-time and run-time performance/power variations in different cores, respectively [1][3]. The process variations in the 45nm technology node may lead to threshold voltage variations of up to 42% [5]. These variation effects are foreseen to aggravate in 32nm and beyond [3]. Fig.1 illustrates the performance variations of different cores on a chip due to (a) process variations [6], and (b) aging [7]. While the above two effects are of permanent nature, another important reliability concern is *soft errors* that is transient in nature and manifests as temporary bit flips in the system due to, e.g., high energy particle strikes [8][9]. *Since a real-world system is subjected to all of the above reliability issues, joint consideration of these effects is inevitable for designing reliable on-chip systems*.

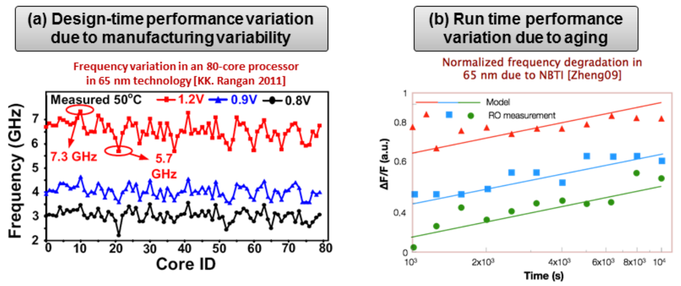

Fig.1: Performance variations among different cores on a chip due to (a) process variations [6], and (b) aging [7]

Several techniques have been proposed to combat these reliability threats where most of them typically mitigate *individual* reliability threats [3][23]. A compiler assisted approach is used in [10] where the workload characteristic is exploited to set different frequencies and voltages of different cores. A compiler-based approach is proposed in [12] where the stress is uniformly distributed to minimize the aging effects in the GPGPU architecture. The run-time approach in [11] addresses the non-uniform aging (due to imbalance workload) of different cores through regulating their idle time. Towards soft error issues, several mitigation techniques have been proposed at the hardware level [3][4] and at the software/compiler level [8][13][14], most of which rely on redundancy at different granularities (i.e. hardware component, pipeline, code, data, etc.). The works like SRT [15], CRT [16], and [17][18] have focused on multi-core based soft error mitigation where free cores in the multi-core systems are exploited to provide redundancy either at process-level or at thread-level. In CRT, two processor cores execute redundant threads. In presence of performance variations, one core may produce the output later than the other core. Eventually, both outputs will not synchronize for the comparison and an error may be detected due to delayed output from one of the cores. Alternatively, if a synchronization mechanism is designed for that, it may lead to severe performance degradation. Recent trends explore flexible approaches like providing instruction set architecture compliant cores with different error recovery functionalities [19]. These techniques assume that excessive area is available in multi-/many-core systems. However, *in case of area-constrained embedded systems, there may be scenarios where not all applications may be supported with full DMR or TMR due to resource competition*.

Problem Statement: First, given a many-core architecture under performance variation effects with heterogonous error recovery functionalities (i.e. fully protected, partially protected or un-protected cores), the problem is to allocate a set of cores with heterogeneous error recovery functionalities to the tasks of concurrently executing applications. The problem is intricate in resource competing scenarios where the total applications' demands may exceed the available resources to ensure full system reliability. Afterwards, given multiple task versions each exhibiting a unique resilience and performance characteristic, the problem is to choose an appropriate task version to map on the allocated core. The goal is to maximize the overall system reliability while meeting tasks' deadlines.

A. *Our Novel Contributions*

We propose a novel Dynamic Reliability Management System (DyReMS) for on-chip systems that performs resilience-driven resource allocation to the tasks of concurrently executing and resource competing applications. Depending upon the allocated core and its hardware-level error recovery features, DyReMS chooses an appropriate task version generated through a reliability-driven compiler while exploiting their variable resilience properties. It also performs execution and recovery control in case errors are detected.

II. System Models

Hardware Architecture Model: We consider a many-core processor with homogenous RISC cores that are instruction set architecture compliant but exhibit performance variations. The cores have heterogeneous error recovery functionality, i.e. some cores are *fully protected cores* (FPC) with hardware-level protection techniques like TMR or pipeline protection etc. [4], some are *partially protected cores* (PPC), e.g., offering information redundancy by ECC based memories [20], and some are un-protected cores (UPC), i.e. with no error recovery features. The unprotected cores can be utilized in pair to achieve modular redundancy, i.e. DMR. Partially protected cores with, e.g., ECC can only recover from memory faults.

Application Model: The application software is composed of n tasks given in a list T. Each task t has multiple task versions composed of a different set of function versions fa_k that are generated using reliability-aware transformations [8]. A transformed task version t_i has certain performance (in terms of execution time of a version on a particular core type) P_i, reliability (in terms of resilience [21]) R_i, and reliability-timing penalty [22] RTP_i characteristics. A task dependence graph (TDP) is also given as an input to our system to find the best possible cores to map a task while taking its dependent tasks into account.

Fault and Reliability Estimation Models: We consider transient faults/soft errors, single and multiple bit upsets in both the combinatorial and sequential logic. The process variations and aging effects are considered as performance variations following the data and model of [6][7]. We employ the resilience model of [19][21] that quantifies an application's resilience as the probabilistic measure of functional correctness in terms of the output quality in the presence of faults. In order to jointly account for the functional and timing reliability we employ the Reliability-Timing (RT) penalty model of [22]. It is given as the linear combination of the *functional reliability* (i.e., resilience R) and the *timing reliability* (i.e. *D_missRate*, the percentage of deadline violations). For a user-defined parameter $0 \le \alpha \le 1$, the RT penalty is given as:

$$RT_{pen} = \alpha R + (1-\alpha) D\_missRate$$

III. System Overview and Dynamic Reliability Management

Fig.2 presents our overall system overview showing the design-time and run-time steps. The following two design-time steps provide inputs to our novel Dynamic Reliability Management System (DyReMS, shown in orange-filled box).

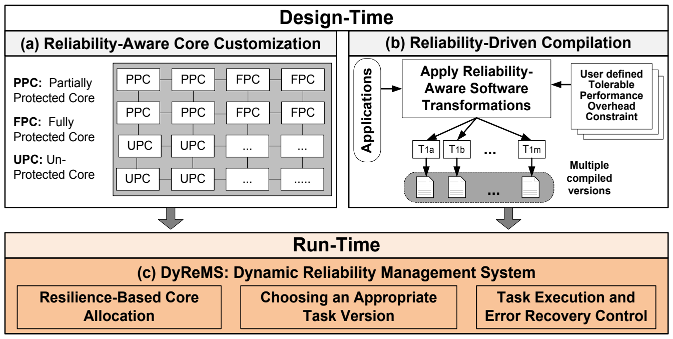
Design-Time
(a) Reliability-Aware Core Customization
PPC: Partially Protected Core
FPC: Fully Protected Core
UPC: Un-Protected Core
PPC
FPC
UPC
(b) Reliability-Driven Compilation
Applications
Apply Reliability-Aware Software Transformations
User defined Tolerable Performance Overhead Constraint
T1a
T1b
T1m
Multiple compiled versions
Run-Time
(c) DyReMS: Dynamic Reliability Management System
Resilience-Based Core Allocation
Choosing an Appropriate Task Version
Task Execution and Error Recovery Control

Fig.2 System Overview showing Design-Time and Run-Time Steps

a) *Reliability-Aware Core Customization:* In a many-core processor, different cores can be enhanced with specialized reliability features under a given area constraint. As discussed in the system model, the processor under consideration has some cores that are fully-protected (FPC), some cores that are partially protected (PPC), and some cores that are un-protected (FPC). Our selection of a certain reliability-aware customization follows the methodology of the DHASER approach presented in [19]. Note, in such architecture, all the cores have the same instruction set architecture (ISA).

b) *Reliability-Driven Compilation:* We leverage the concept of multiple compiled versions of a task (similar to the one proposed in [8]), such that different compiled versions have diverse resilience/reliability and performance (executing time) properties.

Given such an architecture with heterogeneous reliability features and multiple compiled versions of tasks with reliability vs. performance tradeoff, the challenge is to manage the system resources to jointly maximize the reliability of the executing tasks. Such a technique needs to be run-time adaptive in order to react to the run-time performance variations due to aging effects or workload variations and unpredictable application scenarios (i.e. which set of tasks executes concurrently is unknown at design time).

To address the above-discussed challenges, we propose a Dynamic Reliability Management System (DyReMS) that performs the following three key operations at run time:

1) Resilience-Based Reliable Core Assignment: Depending upon the resilience of a task, DyReMS assigns an appropriate core with certain reliability features. For instance, a task with very low resilience may get a fully protected core (FPC) as it has a high susceptibility towards program errors. However, a task with a very high resilience will get unprotected core (UPC) because a fault during the execution of this task will most probably be masked and will not result in a program error. Such a *resilience-based core allocator* needs to account for the deadlines and execution time of tasks along with the performance variations in the underlying hardware, especially when assigning cores with temporal redundancy.

In case FPCs and PPCs are all allocated to some low-resilience tasks, and there are still more low-resilience tasks that need further protection, our DyReMS uses multiple UPCs to realize spatial redundancy. Depending upon the number of remaining tasks that are yet to be executed, the number of cores available for spatial redundancy might be limited. In case of insufficient resources, the priority is to assign at least one core to all the remaining tasks to ensure their execution rather than engaging the cores in spatial redundancy for only a few tasks and not facilitating others.

Finally, after choosing a version (here with the highest performance) for each task, the tasks are sorted by their RT penalty values in the descending order. The task with the highest RT penalty value (the first entry in the list) will be assigned to the fastest core of the core type with the highest protection-level, i.e. FPC, followed by DMR using UPCs, PPC, and then UPC without any redundancy support.

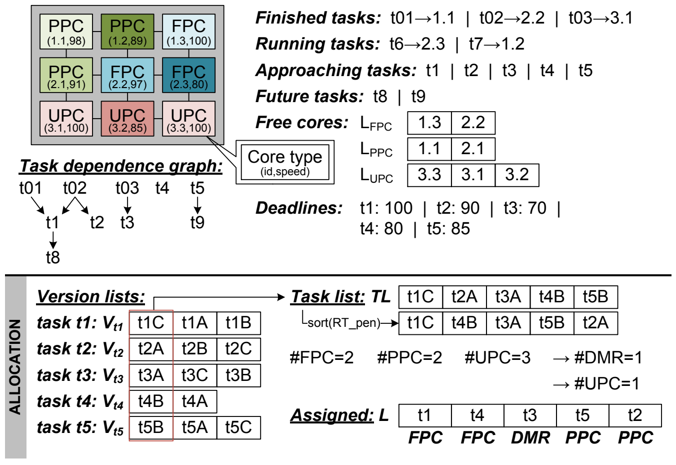
PPC (1.1,98)
PPC (1.2,89)
FPC (1.3,100)
PPC (2.1,91)
FPC (2.2,97)
FPC (2.3,80)
UPC (3.1,100)
UPC (3.2,85)
UPC (3.3,100)
Core type (id,speed)
Finished tasks: t01→1.1 | t02→2.2 | t03→3.1
Running tasks: t6→2.3 | t7→1.2
Approaching tasks: t1 | t2 | t3 | t4 | t5
Future tasks: t8 | t9
Free cores: L_FPC 1.3 2.2
L_PPC 1.1 2.1
L_UPC 3.3 3.1 3.2
Task dependence graph:
t01 t02 t03 t4 t5
t1 t2 t3 t9
t8
Deadlines: t1: 100 | t2: 90 | t3: 70 | t4: 80 | t5: 85
ALLOCATION
Version lists:
task t1: V_t1 t1C t1A t1B
task t2: V_t2 t2A t2B t2C
task t3: V_t3 t3A t3C t3B
task t4: V_t4 t4B t4A
task t5: V_t5 t5B t5A t5C
Task list: TL t1C t2A t3A t4B t5B
sort(RT_pen) t1C t4B t3A t5B t2A
#FPC=2 #PPC=2 #UPC=3 → #DMR=1 → #UPC=1
Assigned: L t1 t4 t3 t5 t2
FPC FPC DMR PPC PPC

Fig.3 An example scenario illustrating the flow of our DyReMS.

An Example: Fig.3 illustrates the flow of our *resilience-based core allocator* in DyReMS. A processor with 9 cores in 3x3 organization with 3 FPCs, 3PPCs, and 3 UPCs is given along with a task dependence graph. Three tasks (t01-t03) have just finished execution while two tasks (t6-t7) are still executing. Five tasks (t1-t5) are going to start execution and future tasks are t8-t9. Core properties are: type={FPC;PPC;UPC}, id/position in "x.y" format, and speed. The task version list with different tradeoffs (reliability/performance) per task sorted by performance is shown. The task list (initially unsorted) is sorted by *RT-penalty*. The number of sets per protection level {FPC, DMR, PPC, UPC} is computed. The final protection level assignment is shown in L and the allocation is given as: 2xFPC, 1xDMR is possible, 2x PPC → 1xUPC free (no task available for that).

2) Choosing Reliable Task Version: Depending upon the allocated core and task deadline, an appropriate task version is chosen, while taking into account the application's deadline. It is important to match up the task version's execution characteristic with the core performance properties. For the cores offering full protection (i.e. FPC), the performance-wise best version can be selected because the task resilience plays a minor role due to the hardware protection mechanisms provided by FPCs. In case of the other protection-levels (i.e. PPCs and DMR using multiple UPCs), we select the version that meets the deadline and offers the highest possible reliability in terms of resilience.

3) Reliable Execution and Rollback Control: For input replication and output comparison, we deploy the method presented in [15]. Fig.4(a) illustrates the procedure; the inputs are replicated using Load Value Queue (LVQ) such that the data is fetched by the thread executing on the faster core C1 and placed in the buffer for the slower thread running on C2. Output comparison is done using the store buffer. Both outputs and the addresses for storing the output data are kept for checking prior to the actual store operation. Finally, the store instruction is executed using the address (computed two times through redundant execution). Fig.4(b) presents a scenario where outputs from two UPCs (used for DMR) are compared. It shows different reaction scenarios in case an error is detected. Let us assume two potential cases of output mismatch:

(i) *Delay Faults due to Performance Variations:* two UPCs have performance variation, e.g. C2 is a slower core and C1 is a faster core such that core C2 produces a delayed output. When comparing C2's output with C1's output, a potential mismatch may happen due to timing delay and insufficient synchronization margin, thus leading to a detected error.

(ii) *Soft Errors:* core C1 experiences a soft error producing an incorrect output, while C2 produces a correct output within time. Both outputs are synchronized but an error is detected due to a soft error in C1.

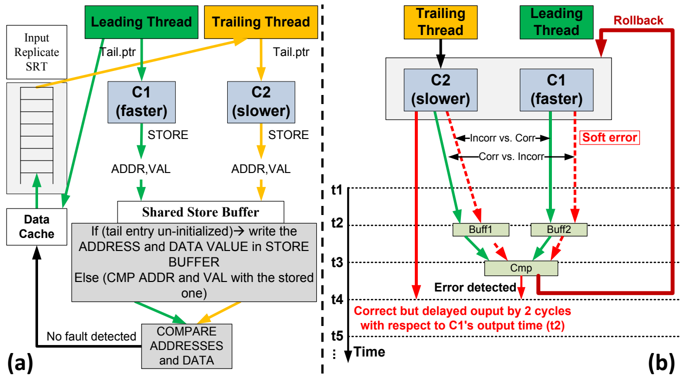

Fig.4 (a) Input replication & output comparison mechanism, (b) Output comparison issues under variability

In case an error is detected, our DyReMS selects one of the following options for recovery or continues execution.

Option 1: Ignore the error and "blindly" select the output of C1 executing a soft error-resilient task version. The confidence level of selecting C1's output as the correct result depends upon the resilience level of this task version. An error in a highly resilient task may not affect the quality of service for the end-user.

Option 2: "Only" rollback/re-execute the task on core C2 that produced the delayed output and re-compare with previous output of C1, i.e. generated in the previous execution. This will curtail the power overhead of computing the recovery.

Option 3: In case an error is again detected due to aging of C2, re-execute on an un-aged core C1 which has better reliability and performance. C2 is completely disabled, since repeated execution on C2 may further deteriorate the core and cause problems for future executions.

IV. Results and Discussion

A. Experimental Setup

An in-house reliability-aware many-core simulator is employed (see Fig.5). It simulates the SPARC V-8 instruction set architecture. The simulator is equipped with a fault generation engine that uses various parameters for generating different fault scenarios. An integrated fault injection engine then injects different transient faults depending upon the generated fault scenario during the application execution at the ISS level. Important fault generation parameters are: fault models (i.e. single or multiple bit flips), fault distribution properties (random, correlated, evenly distributed), process area results obtained from an ASIC synthesis flow. The fault rates for experiments are obtained using the neutron flux calculator [24] and city coordinates/altitudes. In our experiments multiple fault rates are considered (10^{-6}-10^{-8}), to obtain coverage from terrestrial to aerial operational scenarios. From the reliability-driven compiler [8], multiple transformed task versions are generated which are forwarded to our reliability-aware processor simulator for evaluation. We use the state-of-the-art H.264 video encoder and multiple non-multimedia applications from MiBench application suite [25] like "SHA", "CRC", "AES", "ADPCM", and "SUSAN". For evaluation we generate multiple application mixes following a random combination of tasks or representative application scenarios (like secure image processing and secure video conferencing).

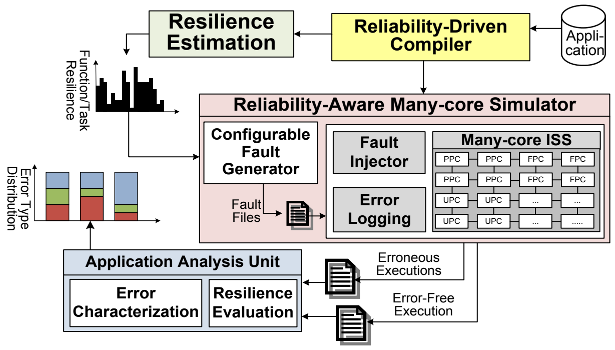

Fig.5: Our experimental setup

The resilience values are also estimated at task level using the methods of [21][19] and are forwarded to the simulator. After the fault injection experiments are performed, the erroneous and error free data is forwarded to the application analysis unit, to evaluate task reliability.

B. Comparison to State-of-the-Art

Fig. 6 illustrates a comparison between our DyReMS and the timing reliability-optimizing core assignment, i.e. minimizes the probability of deadline misses (with EDF scheduling). The task reliability improvements of our system are presented when running a different number of application scenarios on the many-core processor (where an application scenario is a sequence of applications, e.g., ADPCM giving input to CRC). The reliability improvements for a 4x4, 5x5 and 6x6 core configuration are shown for two different ranges of variation. For all configurations it is visible that for a higher number of tasks the reliability improvement is lower as more tasks have to be assigned to

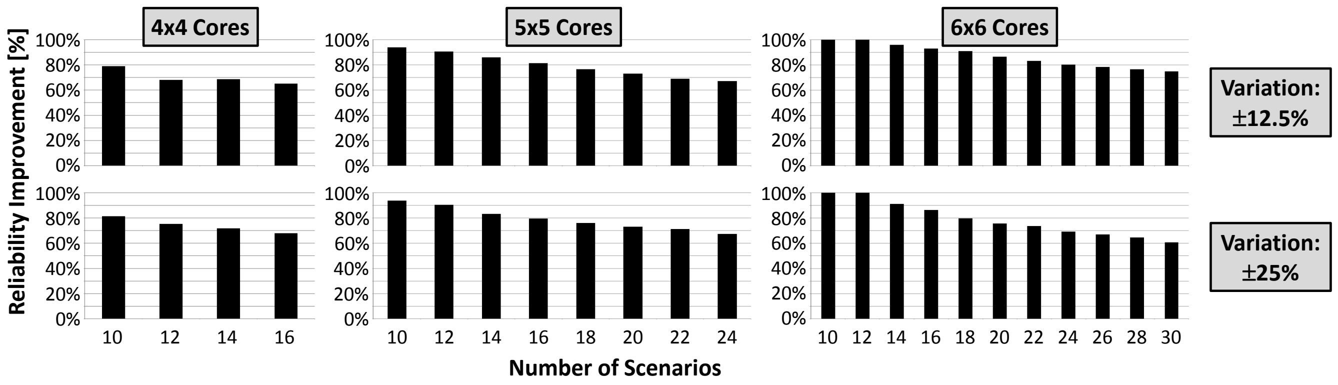

Fig. 6 Task reliability improvements of our DyReMS compared to the timing reliability-optimizing core assignment.

the PPCs or UPCs. However, the tasks still benefit significantly from the joint consideration of functional reliability and timing reliability for core allocation and the different compiled versions provided. For the 4x4 case the reliability improvement is around 70% on average for the lower frequency variation. When moving to a 5x5 configuration, more FPCs are available and are selected by DyReMS, while the timing reliability-optimizing core assignment also selects PPCs and UPCs in case they offer a better performance than the remaining FPCs. Consequently, the reliability improvement increases when comparing the "10" scenarios cases for the different core configurations. In the 6x6 configuration the 100% improvement can be explained by the fact that all tasks can be assigned to FPCs, while an average improvement of 87% is achieved.

V. CONCLUSIONS

We propose a novel Dynamic Reliability Management System (DyReMS) for on-chip systems. Our DyReMS performs resilience-driven resource allocation and mapping considering both the distinct resilience properties of tasks and the heterogeneous error recovery functionality of different cores. Afterwards, it chooses an appropriate soft-error tolerant compiled version based on the selected core. In case of error detection, rollbacks are performed. Our system provides a 70%-87% improved task reliability compared to a timing reliability-optimizing core assignment that minimizes the probability of deadline misses.

ACKNOWLEDGMENT

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 - spp1500.itec.kit.edu).

REFERENCES

[1] Int'l technology roadmap for semiconductors, 2009.

[2] A. Singh, M. Shafique, A. Kumar, J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends", ACM/IEEE DAC, 2013.

[3] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends", ACM/IEEE DAC, 2013.

[4] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, K. Flautner, "Razor: circuit-level correction of timing errors for low-power operation," IEEE MICRO, vol. 24, no. 3, pp. 10-20, 2004.

[5] L. Lin, W. Burleson, "Analysis and Mitigation of Process Variation Impacts on Power-Attack Tolerance," IEEE DAC, pp. 238-243, 2009.

[6] K.K. Rangan, M.D. Powell, G.-Y. Wei; D. Brooks, "Achieving Uniform Performance and Maximizing Throughput in the Presence of Heterogeneity," IEEE HPCA, pp. 3-14, 2011.

[7] R. Zheng, J. Velamala, V. Reddy, V. Balakrishnan, E. Mintarno, S. Mitra, S. Krishnan, Y. Cao, "Circuit Aging Prediction for Low Power Operation," IEEE CICC, pp. 427- 430, 2009.

[8] S. Rehman, M. Shafique, F. Kriebel, J. Henkel, "Reliable software for unreliable hardware: embedded code generation aiming at reliability", CODES+ISSS, pp. 237–246, 2011.

[9] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," IEEE TDMR, vol. 5, no. 3, pp. 305-316, 2005.

[10] I. Kadayif, M. Kandemir, I. Kolcu, "Exploiting processor workload heterogeneity for reducing energy consumption in chip multiprocessors," IEEE DATE, vol. 2, pp. 1158 – 1163, 2004.

[11] F. Paterna, L. Benini, A. Acquaviva, F. Papariello, A. Acquaviva, M. Olivieri, "Adaptive Idleness Distribution for Non-Uniform Aging Tolerance in Multiprocessor Systems-on-Chip," IEEE DATE, 2009.

[12] A. Rahimi, L. Benini, R.K. Gupta, "Aging-Aware Compiler-Directed VLIW Assignment for GPGPU Architectures," IEEE DAC, pp. 1-6, 2013.

[13] G.A. Reis, J. Chang, D.I. August, "Automatic instruction-level software only recovery", IEEE MICRO, pp. 36–47, 2007.

[14] N. Oh, P.P. Shirvani, E.J. McCluskey, "Error detection by duplicated instructions in super-scalar processors", IEEE Transaction on Reliability, 51-1, pp. 63-75, 2002.

[15] S.K. Reinhardt, S.S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," ISCA, pp. 25-34, 2000.

[16] S.S. Mukherjee, M. Kontz, S.K. Reinhardt, "Detailed design and evaluation of redundant multithreading alternatives," IEEE ISCA, pp. 99–110, 2002.

[17] A. Shye, T. Moseley, V.J. Reddi, J. Blomstedt, D.A. Connors, "Using Process-Level Redundancy to Exploit Multiple Cores for Transient Fault Tolerance," DSN, pp. 297 – 306, 2007.

[18] C. Bolchini, M. Carminati, A. Miele, "Self-Adaptive Fault Tolerance in Multi-/Many-Core Systems," IEEE Journal of ET, vol. 29, pp. 159-175, 2013.

[19] T. Li, M. Shafique, S. Rehman, J. A. Ambrose, J. Henkel, S. Parameswaran, "DHASER: Dynamic Heterogeneous Adaptation for Soft-Error Resiliency in ASIP-based Multi-core Systems," IEEE ICCAD, 2013.

[20] R. Teodorescu, J. Nakano, J. Torrellas, "SWICH: A prototype for efficient cache-level check pointing and rollback," IEEE MICRO, vol. 26, no. 5, pp. 28-40, 2006.

[21] S. Rehman, M. Shafique, P.V. Aceituno, F. Kriebel, J.J. Chen, J. Henkel, "Leveraging Variable Function Resilience for Selective Software Reliability on Unreliable Hardware", DATE, pp. 1759-1764, 2013.

[22] S. Rehman, A. Toma, F. Kriebel, M. Shafique, J.-J. Chen, J. Henkel, "Reliable Code Generation and Execution on Unreliable Hardware under Joint Functional and Timing Reliability Considerations," IEEE RTAS, pp. 273-282, 2013.

[23] A. Rajendiran, S. Ananthanarayanan, H. D. Patel, M. V. Tripunitara, S. Garg, "Reliable computing with ultra-reduced instruction set co-processors", IEEE DAC, pp. 697-702, 2012

[24] Flux calculator: www.seutest.com/cgi-bin/FluxCalculator.cgi.

[25] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", IEEE Workload characterization, pp.3-4, 2001.