Design and Evaluation of Reliability-oriented Task Re-Mapping in MPSoCs using Time-Series Analysis of Intermittent faults

Siva Satyendra Sahoo
National University of Singapore
Department of Electrical and Computer Engineering
Email: satyendra@u.nus.edu

Akash Kumar
Technische Universität Dresden
Center for Advancing Electronics Dresden(cfaed)
Email: akash.kumar@tu-dresden.de

Bharadwaj Veeravalli
National University of Singapore
Department of Electrical and Computer Engineering
Email: elebv@nus.edu.sg

***Abstract*—A large number of hardware faults are being caused by an increasing number of manufacturing defects and physical interactions during operation. This poses major challenges for the design and testing of modern Multiprocessor System-on-Chips (MPSoCs). Intermittent faults constitute a major part of hardware faults and their fault rates can be used as an indicator of the wear-out in a Processing Element (PE). We propose a run-time task re-mapping method that uses this information to improve the useful lifetime of MPSoCs. We also propose a scenario-aware system-level fault injection technique for intermittent faults to evaluate system-level design techniques in MPSoCs. Our performance results conclusively show that our strategy significantly scales on reliability metrics with respect to number of PEs. Specifically, we show that our method can achieve an increase in lifetime of up to 16% and tolerate up to 30% more faults than state-of-the-art techniques.**

I. Introduction

The ubiquitous nature of computers in today's workplace and home environment shows the growing needs of computer systems. As computer systems find increasing usage in systems of growing complexity, the performance demands on the systems keeps increasing as well. As a result, increasing processing power needs to be built into a system. Technology scaling and architecture innovations have led to the design of denser and more complex systems with very high performance. Reduced transistor size in deep sub-micron CMOS technology has led to a dramatic increase in processor performance and power density. Multiprocessor/Multicore systems take advantage of this high density of transistors to implement many processor cores of varying complexity and processing capability on the same chip. This enables applications with a large level of parallelism to execute very efficiently. However, the increased power density and reduced transistor size have introduced reliability issues [1]. With reduced transistor sizes, the number of faults due to manufacturing defects such as imperfect lithographic patterning, has increased. The rate of manufacturing defects is expected to reach approximately 1000 defects/m^2 in the next few years [2]. Moreover, the increased temperature due to higher power density increases the frequency of occurrence of these faults. These circumstances show the need for reliable design and testing of Multiprocessor System-on-Chips (MPSoCs).

Reliability-aware task mapping has been the subject of much research for improving the lifetime and availability of the system. However, majority of this research has focused towards using task mapping as a system-level fault tolerance technique for permanent faults. While some of these techniques can be also used for intermittent faults, the repairable nature of the fault mechanisms for intermittent faults provides unique design challenges and opportunities for improving the lifetime of the system. In Section II we give a brief background about intermittent faults, review some state-of-the-art techniques for fault tolerance of intermittent faults and provide their distinction from our current work.

Intermittent faults pose unique challenges to reliability testing as well. Insufficient data on fault distributions and the dependency of the fault mechanisms on random variations in the operating conditions are major challenges for fault injection based reliability testing. Lastly, published literature does not consider repairable nature of the intermittent faults and this sets as a primary motivation for our work reported in this paper. In our current work, we explore both aspects – *design* and *reliability testing* – of fault tolerance for intermittent faults. Our contributions can be listed as:

- We propose and implement a ***scenario-aware system-level fault injection*** technique for intermittent faults in MPSoCs. In Section III, we give an overview of the fault-injection based simulation and describe our proposed technique to simulate intermittent faults.
- We propose and implement a ***runtime task mapping*** method that uses the intermittent fault rate as an indicator of wear-out in the units to improve the useful lifetime of MPSoCs. The system modeling and algorithm for this technique is described in detail in Section IV.

We performed fault-injection based tests to quantify the effectiveness of our proposed task mapping method and compare the results with state-of-the-art methods. In Section V we describe the simulation setup, settings and metrics used for this evaluation and the results observed. Finally, in Section VI, we conclude the paper by summarizing the results and specifying the scope for future work.

II. Literature Survey

A. Intermittent Faults

A fault is an erroneous state of software or hardware resulting from failures of its components. Faults can occur due to design/manufacturing defects (*developmental faults*), wear/fatigue (*physical faults*), or external disturbances (*interaction faults*). Based on their frequency and persistence, hardware faults can be classified into – *Transient faults,* *Intermittent faults* and *Permanent faults* [1]. Intermittent faults are those that occur non-deterministically at the same location. Recent studies in [3], [4] show that intermittent faults are prevalent in commodity processors.

- **Causes:** Device wear-out is the major cause of intermittent faults. Solid-state devices tend to degrade with time and stress. Transistor scaling and higher temperatures make the devices more susceptible and also causes the acceleration in the occurrence of these faults [5]. Recent surveys [3] also suggest that the fault rates in processors correlate with the number of cycles executed by the processor. Fault mechanisms that are activated by wear-out, resulting in the faults are – *Gate-oxide breakdown, Negative Bias Temperature Instability* (NBTI), *Hot Carrier Injection* (HCI) and *Electromigration (EM)*.
  In HCI, carriers, as they are accelerated along the channel, can become energetic enough that, through scattering and/or impact ionization, they can be injected into the gate-oxide causing interface-state generation. NBTI is the leading reliability concern for current technology nodes. NBTI causes a shift in the p-channel threshold voltage and a decrease in the mobility of the inversion channel. More than 40% of faults due to NBTI and HCI are reversible and are manifested as intermittent faults [6]. Gate-oxide breakdown is defined as the time when a cluster of connected bonds beginning from a seed at one interface of the gate oxide reaches the opposite interface. These Time Dependent Dielectric Breakdowns (TDDB) first manifest as intermittent errors (as Soft Dielectric Breakdown (SDB)) and later progress to Hard Dielectric Breakdowns (HDB) under continued stress conditions. EM is the gradual displacement of metal atoms in a semiconductor. Two EM failure mechanisms occur due to the asymmetry in the ion flow – *open circuits* and *short circuits*.
- **Effects:** Intermittent faults may manifest as oxide failures, which starts with tunnel injection of electrons. This increases the leakage current and may eventually lead to permanent breakdown over time. Similarly, faults in the interconnects (like EM voids) may cause higher resistance and propagation delays, resulting in timing failures intermittently [7]. So, we can conclude that intermittent faults can eventually lead to permanent faults under continued stress and can be used as an indicator of the wear-out in the unit.

B. Fault Tolerant Task Mapping

Task mapping of application tasks on MPSoCs involves assignment and ordering of the tasks and their communications on the platform resources in view of some optimization criteria, such as *Execution time, Mapping time, Throughput, Resource utilization, Energy consumption* and *Reliability*. A survey of various task mapping methodologies, both *Design-time* and *Run-time*, based on different optimization goals is presented in [8]. Majority of the research for reliability-aware task mapping in MPSoCs has been towards adapting to permanent faults. Wells et al. explored techniques such as pausing execution on a faulty core, remapping tasks on spare cores and virtualization to adapt to intermittent faults [9]. Rashid et al. evaluate the impact of different intermittent error recovery scenarios on the MPSoC performance in [5]. Both [5], [9] treat the Processor Elements (PEs) of an MPSoC as either live or dead units at a particular instant. While this approach is suitable for permanent faults, the unique characteristics of intermittent faults show that it fails to model the effects of intermittent faults accurately.

Das et al. proposed a technique to model the availability of MPSoCs with intermittent and repairable device defects [10]. They provide a design time optimization technique to improve task communication energy. However, intermittent faults are highly susceptible to variations in temperature, operating environment and aging effects. Hence, a runtime approach to assess the health of the PEs can provide a better estimate of the state of the system. Moreover, design-time optimization fails to account for the varying number and type of applications in the system during its operation. In [11], Duque et al. proposed a fault tolerant approach to model core reliability at runtime and allocate resources accordingly. However the following shortcomings in [11] are noteworthy:

- The core reliability modeling suffers from *short-term memory*. A PE with a history of a large number of faults can get an improved ranking with a relatively small number of fault-free operations. This problem is more evident in the presence of spare cores in the MPSoC, where a faulty processor unit can get similar rank to a relatively lesser used spare unit.
- The simulations for the evaluation of the proposed method were done with a decreasing fault rate. The wear-out region of operating lifetime shows an increasing fault rate and evaluating the system performance in the wear-out region will give a better estimate of the lifetime of the system.

In our current work, we try to overcome the problem of short-term memory by performing a trend analysis of the faults over the lifetime of the system. We compared our method with [11] by performing simulations with increasing fault rate and observe up to 16% improvement in the lifetime of the MPSoC.

III. System Level Fault Injection

Fault injection is used to evaluate the dependability of the system. It involves inserting faults into the system and monitoring the system response in order to assess the fault tolerance of the system. In simulation-based fault injection both the target system and the possible hardware faults are modeled and simulated by a software program. One of the drawbacks of this technique is the possible lack of accuracy in the fault model. Usually, a random fault-injection model is used for reliability evaluation of MPSoC based systems. This approach is sufficient for permanent faults, where the processing units can attain only one of the two states–*dead* or *alive*, and for transient faults, where there is no underlying pattern to the fault mechanisms. However, a more appropriate model for intermittent faults should consider the following conclusions drawn from Section II:

- A PE with one or more intermittent faults has higher chances of having another.

$$
\begin{aligned}
Eff\_exec\_t_i &= Total\_exec_i - \alpha \times Total\_rest_i \\
Estimated\_FIT_i &= func(Initial\_FIT_i, Eff\_exec\_t_i, Fault\ model) \\
Norm\_FIT_i &= Estimated\_FIT_i \div max(Estimated\_FIT_i s) \\
PE\_fi\_idx_i &= Norm\_FIT_i \times PE\_fault\_cnt_i
\end{aligned}
$$

Fig. 1: Steps to calculate $PE\_fi\_idx_i$

- The chances of a PE facing an intermittent fault are proportional to the number of faults that have already occurred in it [3].
- Continued stress conditions on a PE increase its chances of facing another intermittent fault.
- Some fault mechanisms are repairable and might show some amount of recovery with a lesser workload [6].

To this end, we propose a scenario-aware fault injection technique for system-level, simulation-based evaluation of MPSoCs. This involves steps as presented in Algorithm 1. First, we compute the expected time to the next fault ($Next\_ttf$). This is done based on the input *Fault distribution model* and its parameters. Frequently used models for fault injection include – *Weibull, Exponential* and *Log-Normal*. The appropriate model to use is selected based on the fault-mechanism being evaluated. Next, we model the fault likelihood of each PE in the MPSoC based on the current mapping data and the evaluated $Next\_ttf$. This information is used to compute the total execution time ($Total\_exec_i$), and the total resting time ($Total\_rest_i$) for each of the PEs in use. The estimated fault rate ($Estimated\_FIT_i$) is calculated as a function of the effective execution time ($Eff\_exec\_t_i$), the initial fault rate ($Initial\_FIT_i$) of the PE and the fault model as shown in Fig. 1. $Initial\_FIT$ specifies the built-in reliability of the PE. Value of α can be varied to account for the repairable nature of the fault mechanism under consideration. In [6], Ershov et al. have reported that NBTI degradation contains both permanent and reversible components. Also, the relative magnitude of the reversible part with respect to the permanent part is very less. Similar behavior is also observed for hot-carrier degradation. So, the value of α can be used to reflect the fault mechanism under consideration. Only positive values should be allowed for $Eff\_exec\_t_i$. These estimated values are used along with the PE's intermittent fault count ($PE\_fault\_cnt_i$) to compute the $PE\_fi\_idx_i$. The computed $PE\_fi\_idx_i$ is used to assign fault probabilities to each PE. Thus, we obtain a non-uniform probability distribution for the fault-injection. The faulty node, PE_Faulty is then selected randomly based on the assigned probability values. The randomness is used to account for different operating conditions and PVT variations.

Algorithm 1 Scenario-aware Fault Injection

```
Require: current mapping data, fault distribution model
Ensure: time to next fault, PE to inject fault
 1: compute time to next fault: $Next\_ttf$
 2: for each PE in mapping $i \in 1 : PE\_count$ do
 3:     if $PE_i$ in use then
 4:         compute $PE\_fi\_idx_i$ based on current mapping
 5:     else
 6:         set $PE\_fi\_idx_i$ to 0
 7:     end if
 8: end for
 9: for each PE in mapping $i \in 1 : PE\_count$ do
10:     compute $PE\_fault\_prob_i$ based on $PE\_fi\_idx_i$
11: end for
12: select $PE\_Faulty$ randomly based on $PE\_fault\_prob_i$s
```

Thus it is clear that the proposed fault injection technique enables the user to fine-tune the effect of rest cycles and execution cycles. By considering the initial fault rate of each core, this technique enables us to evaluate MPSoCs with PEs of varying built-in reliability.

IV. Reliability-Aware Runtime Task Mapping

Reliability-aware task mapping is a widely used fault-tolerance technique for improving the lifetime of MPSoCs. It involves reconfiguring the task allocation in the MPSoC in the event of a fault occurring in one or more PEs. This re-mapping strategy can be formulated either during design time or during the runtime of the system. Design time re-mapping is usually used for static workloads, while runtime re-mapping is more suitable for dynamic workloads. The mapping strategy also varies with the kind of fault occurring in the system. For a permanent faults, the PEs in the MPSoC can be viewed as either functional or dead, and hence a suitable methodology can be adopted. However, intermittent faults exhibit behavior, as stated in Section II. The PEs can still be seen as either functional or non-functional at that particular instant. The following properties of intermittent faults needs to be considered during design of any fault tolerance technique:

1) The fault mechanisms of intermittent faults are partially reversible.
2) Intermittent faults lead to permanent faults under continued stress conditions.

Thus, it is clear that we need a different approach with respect to intermittent faults. To this end, the intermittent fault rate of each PE in the MPSoC can be used as an indicator of the level of wear in the PE at that instant. This information can be used to re-map the task allocation in the MPSoC in a way that reduces the likelihood of further faults occurring in the PE. This can lead to extended lifetime of each PE and hence result in increased lifetime of the system. An implementation of this approach was proposed in [11]. However, as discussed in Section II, this method and the evaluation methodology has some shortcomings. We propose a method of task re-mapping that offers improved reliability and scales very well with respect to the number of available PEs.

A. Estimating PE health:

The MPSoC with a number of PEs can be seen as a system with repairable components. The component repair can be by – *Self-repair*, stopping execution on the component, or *External repair*, by some dedicated hardware module in the system. The faults occurring in the components can be viewed as discrete events occurring randomly in time. Such phenomenon cannot be truly represented by a single continuous distribution function. These situations are *stochastic point processes* and can be analyzed using event series statistics [12]. *Non-Homogeneous Poisson Process* (NHPP) represent point processes in which the rate of occurrence changes with time.

A NHPP describes the sequence of random variables that are neither independently nor identically distributed. The occurrence of intermittent faults, with characteristics such as varying rates of occurrence with workload and wear and the susceptibility to random PVT variations, can be analyzed as NHPP. We use the *centroid test* to analyze the trend of faults occurring in the PEs, and use the resulting value (U_val) as the indicator of the health of the system. The computation of the U_val in terms of the arrival times of the faults $x_1, x_2, ...x_n$ is shown in (1) and Fig. 2 [12].

$$U\_val = \frac{\sum x_i/n - x_0/2}{x_0\sqrt{1/(12n)}} \quad (1)$$

A higher value of U_val indicates a faster increasing fault rate in the PE. The arrival time of the current fault is denoted by x_0.

B. Application modeling:

An application and its constituent tasks can be represented by *Synchronous Data Flow Graphs* (SDFG). The dependence of the tasks of the application is represented in the SDFG by the number of tokens along the edges of the graph. We use the method proposed in [13] to estimate the execution time of each task in one cycle of execution. This information is used to estimate the total execution time on the PE to which each task is allocated. Fig. 3 presents the estimation for one such application with the execution trace shown in Fig. 4. A sorted array of tasks, $Task\_list$, based on their estimated execution time, is used for the mapping.

C. System Description:

The system under test is assumed to be a NoC based MPSoC with a mesh architecture, similar to the one shown in Fig. 5. We assume uninterrupted communication between the PEs.Our work does not concern fault detection, diagnosis and repair. Also,we consider situations in which only one fault occurs in the system at any instant.

The block diagram to implement the proposed task mapping technique is presented in Fig. 5. The $Task\_Allocation\_unit$ keeps track of the PE and running application's state. The values that define the states of the application, PE and the current mapping is shown in Table I. The $Fault\_Management\_unit$

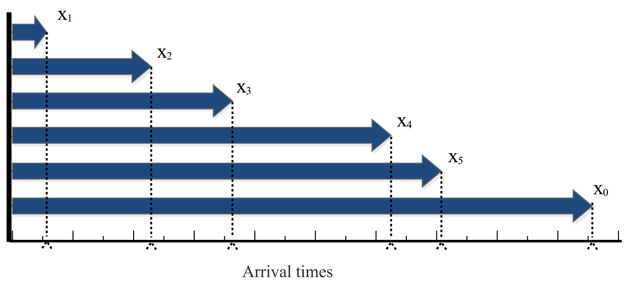
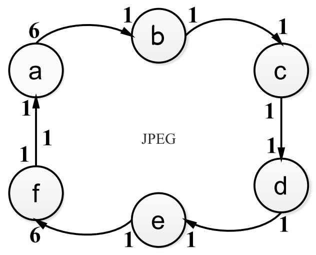


Fig. 2: Arrival time of faults (x_i). The faults happen at non-periodic time intervals. The most recent fault is indicated by x_0.

TABLE I: System State Variables

PE	$PE\_id, flag\_perm\_fault, flag\_current\_fault$
Task	$Task\_id, execution\_time$
Mapping	$PE\_id_i, Task\_id_i, U\_val_i, cumulative\_ttf_i$



Task	Execution time	
	from SDFG	from execution trace
a	28800	28800
b	2400	14400
c	2400	14400
d	7200	43200
e	2400	14400
f	36000	36000

Fig. 3: SDFG of JPEG application. The table shows the execution time of each node and the estimated execution time of each node for one execution cycle. This is estimated form the execution trace as shown in Fig. 4

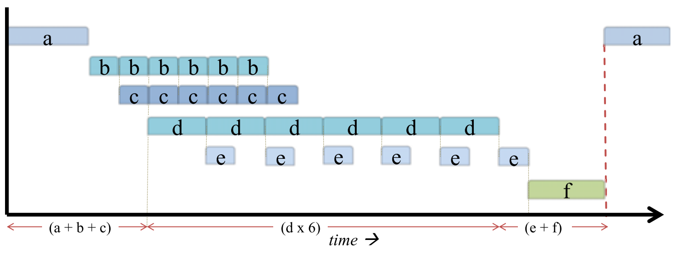


Fig. 4: Execution trace for one execution cycle of SDFG shown in Fig. 3

is a hardware module that concerns the detection, diagnosis and repair of hardware faults in the MPSoC. The events that can activate the $Task\_Allocation\_unit$ are: *Detection of a permanent or intermittent fault, addition or removal of an application* and *repair of a faulty core*. The response of the $Task\_Allocation\_unit$ in the event of the occurrence of any of these is presented in Algorithm 2. When the sorted and updated lists – $Available\_PEs$ and $Task\_list$ are available, the tasks from $Task\_list$ are mapped sequentially onto the PEs in the $Available\_PEs$ list.

V. Experiments and Results

A. Experiments

We conducted performance evaluation studies to estimate the extended lifetime of the system. The setup and metrics

Algorithm 2 Runtime Task-remapping

```
if intermittent fault detected on $PE_i$ then
    remove $PE_i$ from $Available\_PEs$ list
    add earlier intermittently faulted PE to $Available\_PEs$
    sort $Available\_PEs$ list based on $U\_val$
    map tasks from sorted $Task\_list$
    set $flag\_current\_fault$ for $PE_i$
    compute $U\_val$ for $PE_fault$
end if
if faulty PE $PE_i$ is repaired then
    add $PE_i$ to $Available\_PEs$ list
    set $U\_val_i$ and $cumulative\_ttf_i$
end if
if permanent fault is detected on $PE_i$ then
    remove $PE_i$ from $Available\_PEs$ list
    re-map tasks from sorted $Task\_list$
    set $flag\_perm\_fault_i$
end if
if change in application then
    create sorted $Task\_list$ with changed tasks
    map tasks from $Task\_list$ on $Available\_PEs$
end if
```

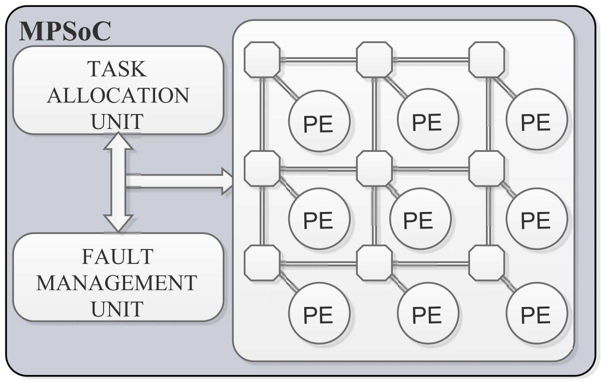


Fig. 5: System Block Diagram

used for the comparison are discussed below.

- **Experiment Setup:** Our evaluation involves– *Fault Injection, Task Re-mapping* and *Data collection*. The fault-injection was done as described in Section III. However, random fault injection was also used to verify the results. Current microprocessors are expected to have a mean-time-to-failure (MTTF) of 5 years. This translates to a fault rate of $22,831 FIT$. We use a starting fault rate of 2×10^7 FIT, an acceleration factor of about 10^3, to limit the simulation time while still being able to generate a large number of fault scenarios. We use the Weibull model, with fault rate as shown in (2), for the fault distribution.

$$\lambda(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} \tag{2}$$

The shape parameter β was set to 2 to simulate the wear-out region of the model. For each experiment, results data was collected from $10,000$ simulations and the mean values were used for comparison.
- **Metrics for Comparison:** To compute the reliability of the MPSoC, we consider the relevant metric mean-time-to-crash (MTTC) [14]. We compute the MTTC as the mean time till there are insufficient PEs left in the system. With increasing fault rate, the PEs go into a state of permanent fault. The PEs are assumed to go into a permanent fault state when the intermittent fault rate exceeds some threshold value,as highlighted in [4]. We use a value of 4 faults in 24 $hours$ as the threshold. We used the number of faults tolerated (NFT) by the system as another measure of the efficiency of the re-mapping technique.
- **Comparison points:** We implemented our proposed method as well as the one proposed in [11] (DDY) and another method, referred to as BASE, that does not consider the wearing of the PEs. Experiments were done based on all three methods and were evaluated on three benchmark applications and 2 random SDFGs. These experiments were carried out for MPSoCs with $6, 9, 12$ and 16 PEs.

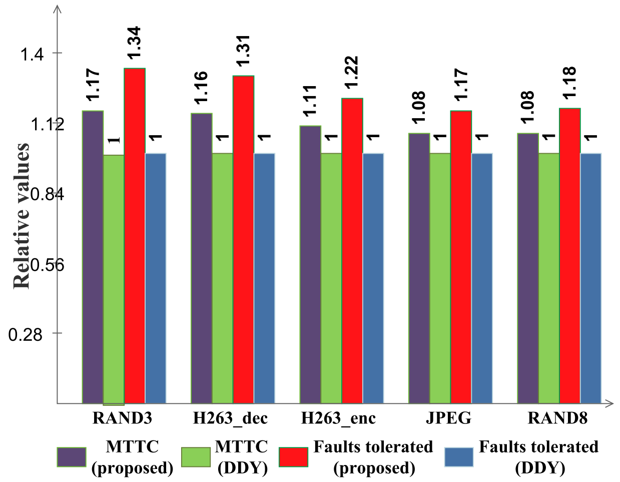


Fig. 6: Reliability comparison with [11]

B. Results

- **Reliability:** Table II shows the percentage increase in MTTC and number of faults tolerated by the system over the BASE method. The numbers of tasks in each application is shown in parenthesis. Fig. 6 presents the relative values of the reliability metrics, when compared with [11] for an MPSoC with 16 PEs. It can be observed from the results, that our proposed technique and that by Duque et al. show improvement in the estimated lifetime over the BASE method. However, with increasing number of spare PEs, our proposed method shows a better improvement in all experiments. Fig. 7 shows the percentage improvement in reliability over BASE method of JPEG with increasing number of PEs. The BASE method does not consider the history of faults of the PEs while re-mapping. This might result in PEs with intermittent faults getting allocated more tasks continuously. This results in PEs getting into permanent faults early. The DDY method proposed in [11] suffers from *short-term memory* as discussed in Section II. Any PE, with some-level of wear, can get an equal ranking with a PE that has not shown any signs wear, by a small number of fault-free operations. This may result in higher workloads being allocated to defective PEs and can lead to re-activation of the fault mechanism. Our approach will require a relatively large number of decreasing fault rates to improve its ranking and will therefore be more immune to random faults.
- **Fault Injection:** Table III presents the increase in reliability by using our proposed mapping technique over that proposed in [11] for 16 PEs. We present the results for both-random fault-injection and scenario-based fault-injection method. Random fault injection assumes a uniform distribution of fault probability for all PEs. The scenario-based fault-injection technique considers the execution time and number of faults that have already occurred in the PE, while predicting faults. This provides a better model for simulating intermittent faults. Our proposed method performs better than DDY for both the

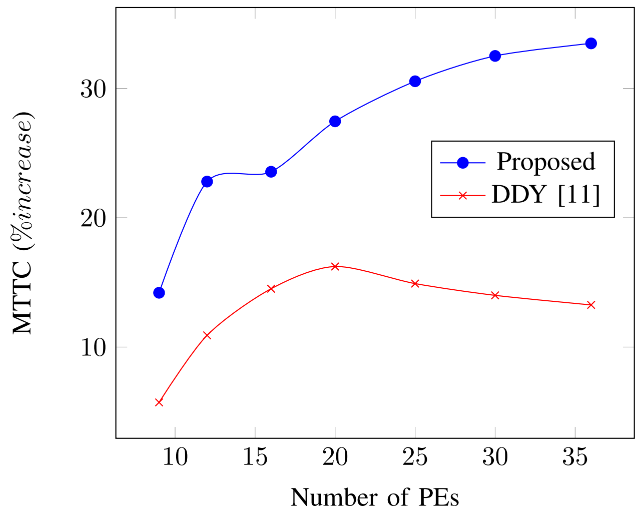


Fig. 7: Percentage increase in MTTC over BASE method for JPEG

fault-injection models, and we get a higher increase in MTTC when we try to simulate intermittent faults.

- **Computation:** We have evaluated the reliability achieved by three different methods of task re-mapping. The BASE method does not involve any computations for re-mapping. In [11], a method for PE's health estimation is proposed which is easy to realize but does not scale well as the number of available PEs increases. In contrast, our proposed method offers this scaling as the number of spare PEs increases. It may be noted that the computations do not affect the re-mapping time, as the faulty PE's reliability index (U_val) is updated after re-mapping. The sorting of PEs based on their U_val is also done after the re-mapping is completed for the current intermittent fault. However, depending on the platform in use, relevant approximations can be used in the computation of U_val.

VI. Conclusions

In our current work, we explored the application of time-series analysis to estimate the health of the PEs in a MPSoC.

TABLE II: Percentage Increase in Reliabilty data over BASE Method

Application	Number of PEs	MTTC (% increase)		NFT (% increase)	
		Proposed	DDY [11]	Proposed	DDY [11]
RAND3(3)	6	2.68	0.73	5.69	1.54
	9	8.47	1.65	17.84	3.48
	12	14.44	2.40	30.26	5.03
	16	20.84	3.52	43.46	7.33
H263_dec(4)	6	3.27	1.50	6.90	3.17
	9	12.94	3.26	27.12	6.82
	12	20.97	5.22	43.61	10.81
	16	24.76	7.89	51.94	16.15
H263_enc(5)	6	3.85	5.14	8.10	10.83
	9	16.59	5.85	34.94	12.31
	12	23.68	9.35	49.41	19.21
	16	26.63	14.20	57.07	28.91
JPEG(6)	9	14.06	5.59	28.63	11.33
	12	22.78	10.81	47.21	21.86
	16	23.49	14.44	49.87	28.48
RAND8(8)	12	18.56	10.57	37.60	20.67
	16	25.44	15.88	55.24	31.56

TABLE III: Increase in Reliability Over [11] under different Fault Injection(FI) Method

Application	MTTC (% increase)		NFT (% increase)	
	Random FI	Proposed FI	Random FI	Proposed FI
RAND3	7.07	16.73	14.54	33.67
H263_dec	9.50	15.64	18.32	30.81
H263_enc	5.05	10.89	9.43	21.84
JPEG	2.04	7.90	4.41	16.65
RAND8	2.49	8.25	5.40	18.00

This information was used to reduce the likelihood of wear-out based faults, thereby increasing the lifetime of the system. We compared our method with one recent work and observed an increase in lifetime of the MPSoC. All the results were obtained from fault-injection based simulations. The accuracy of such experiments is highly dependent on the accuracy of the fault model and the system model. To this end, we proposed a system-level fault-injection technique for intermittent faults that takes the characteristics of such faults into consideration. An interesting extension following this work is in using the proposed technique along with ones that use thermal and layout information to provide better fault-injection. Another plausible extension is in considering faults occurring in the MPSoC due to wearing of the communication links.

References

[1] C. Constantinescu, "Trends and challenges in vlsi circuit reliability," *IEEE micro*, 2003.

[2] ITRS, "International technology roadmap for semiconductors." [Online]. Available: http://www.itrs.net

[3] E. B. Nightingale, J. R. Douceur, and V. Orgovan, "Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer pcs," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011.

[4] C. Constantinescu, "Intermittent faults and effects on reliability of integrated circuits," in *Reliability and Maintainability Symposium, 2008. RAMS 2008. Annual*. IEEE, 2008.

[5] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan, "Intermittent hardware errors recovery: Modeling and evaluation," in *Quantitative Evaluation of Systems (QEST), 2012 Ninth International Conference on*. IEEE, 2012.

[6] M. Ershov, S. Saxena, H. Karbasi, S. Winters, S. Minehane, J. Babcock, R. Lindley, P. Clifton, M. Redford, and A. Shibkov, "Dynamic recovery of negative bias temperature instability in p-type metal–oxide–semiconductor field-effect transistors," *Applied physics letters*, 2003.

[7] C. Constantinescu, "Impact of intermittent faults on nanocomputing devices," in *DSN 2007 Workshop on Dependable and Secure Nanocomputing*, 2007.

[8] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013.

[9] P. M. Wells, K. Chakraborty, and G. S. Sohi, "Adapting to intermittent faults in multicore systems," *ACM SIGOPS Operating Systems Review*, 2008.

[10] A. Das, A. Kumar, and B. Veeravalli, "Communication and migration energy aware design space exploration for multicore systems with intermittent faults," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013.

[11] L. A. R. Duque, J. M. M. Diaz, and C. Yang, "Improving mpsoc reliability through adapting runtime task schedule based on time-correlated fault behavior," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015.

[12] P. P. O'Connor and A. Kleyner, *Practical Reliability Engineering*, 5th ed. Wiley Publishing, 2012.

[13] A. Das, A. K. Singh, and A. Kumar, "Execution trace–driven energy-reliability optimization for multimedia mpsocs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2015.

[14] W. Dweik, M. Annavaram, and M. Dubois, "Reliability-aware exceptions: Tolerating intermittent faults in microprocessor array structures," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014.