CL(R)Early: An *Early*-stage DSE Methodology for *C*ross-*L*ayer *R*eliability-aware Heterogeneous Embedded Systems

Siva Satyendra Sahoo
Institute of Computer Engineering
Technische Universität Dresden
Dresden, Germany
siva_satyendra.sahoo@tu-dresden.de

Bharadwaj Veeravalli
Department of ECE
National University of Singapore
Singapore, Singapore
elebv@nus.edu.sg

Akash Kumar
Institute of Computer Engineering
Technische Universität Dresden
Dresden, Germany
akash.kumar@tu-dresden.de

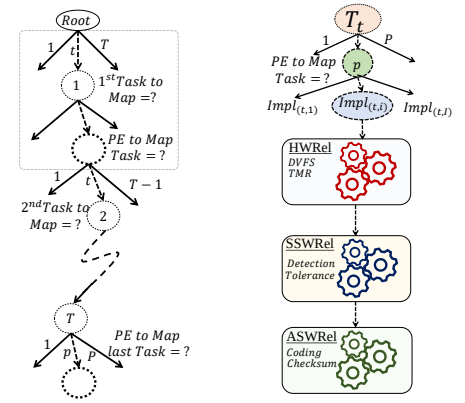
***Abstract*—Cross-layer reliability (CLR) presents a cost-effective alternative to traditional single-layer design in resource-constrained embedded systems. CLR provides the scope for leveraging the inherent fault-masking of multiple layers and exploiting application-specific tolerances to degradation in some Quality of Service (QoS) metrics. However, it can also lead to an explosion in the design complexity. State-of-the art approaches to such joint optimization across multiple degrees of freedom can lead to degradation in the system-level Design Space Exploration (DSE) results. To this end, we propose a DSE methodology for enabling CLR-aware task-mapping in heterogeneous embedded systems. Specifically, we present novel approaches to both task and system-level analysis for performing an early-stage exploration of various design decisions. The proposed methodology results in considerable improvements over other state-of-the-art approaches and shows significant scaling with application size.**

***Index Terms*—Embedded Systems, Cross-layer Reliability, Design Space Exploration, MOEA, Reliability Modeling**

I. Introduction

The proliferation in the variety of application areas that use embedded systems has led to an ever increasing and varying **Q**uality of **S**ervice (QoS) requirements for electronic systems. Consequently, **H**eterogeneous **M**ulti-**P**rocessor **S**ystem-on-**C**hip (HMPSoC), with improved parallelism and customized computation, are being increasingly used in embedded systems to meet application-specific QoS requirements. However, the primary enablers of such heterogeneous systems—*technology scaling* and *architectural innovations*—have also resulted in increasing physical fault-rates [1]. Traditional single-layer[1] reliability approaches (such as **T**riple **M**odular **R**edundancy (TMR)) for mitigating such increased fault-rates can be infeasible for resource-constrained embedded systems due to the large area and power overheads. Moreover, a single-layer approach can prove insufficient for stricter QoS requirements in varying operating conditions. For instance, while operating at higher altitudes with very high fault-rates, using only hardware-based fault-mitigation can lead to inadequate functional correctness [2]. Therefore, reliability methods need to be implemented at multiple layers [3]. However, an *other-layer-agnostic* approach to such multi-layer fault-mitigation may lead to costly over-designing.

In contrast, the **C**ross-**l**ayer **R**eliability (CLR) approach provides a more application-specific and cost-efficient method for reliability-aware system design by implementing an appropriate combination of methods across multiple layers [4]. In addition to leveraging the effect of implicit fault-masking at different layers [5], it allows the designer to exploit application-specific tolerance to degradation in one or more QoS metrics [2]. However, implementing CLR introduces additional design complexity of *selecting* and *configuring* the reliability methods for each layer. For instance, the reliability of matrix multiplication can be improved by using different implementations that use various levels of *spatial* (TMR-based hardware), *temporal* (Checkpointing) and *information* (Checksum) redundancies. An application usually contains a number of such tasks that can be executed on multiple **P**rocessing **E**lement (PE)s. Fig. 1 shows the increased complexity of the search tree of **D**esign **S**pace **E**xploration (DSE) of the corresponding task-mapping problem. The joint optimization across multiple design aspects—CLR, **D**ynamic **V**oltage and **F**requncy **S**caling (DVFS), application-specific QoS requirements and PE heterogeneity in the hardware platform—can result in an explosion in the design space for task-mapping. The search for the optimal Pareto-front in such high dimensional design spaces have led to increasing popularity of **M**ulti-**O**bjective **E**volutionary **A**lgorithms (MOEA)-based optimization [6], [7]. However, such approaches can suffer from scalability issues with increasing application size. A naive problem-agnostic implementation of such methods can result in poor quality of solutions. To this end, we propose an MOEA-based DSE methodology for enabling an early-stage exploration of system-level design decisions in CLR-aware task-mapping The related contributions are listed below.



(a) Task-mapping only (b) Task-mapping with cross-layer reliability

Fig. 1: Search tree for system-level DSE of CLR-aware design. (a) Regular task-mapping involves *task-to-PE* binding and *task-scheduling* decisions for each task. (b) Additional design decisions have to be made regarding the optimal implementation, $Impl_{(t,i)}$, and CLR configuration for each task $Task_t$.

Contributions:

- We propose a framework for task-level analysis and estimation of functional and timing reliability incorporating the impact of implicit fault masking and imperfect fault mitigation on functional reliability. Specifically, novel Markov chain-based models for estimating the effect of any arbitrary CLR configuration are proposed.
- We propose a multi-stage DSE methodology for implementing CLR-integrated task-mapping using MOEA-based optimization. Specifically, we use **G**enetic **A**lgorithms (GA) as a representative method to demonstrate the impact of the proposed design space pruning and modified stochastic search methods. With the proposed modifications we report up to 231% improvement in the solution quality of multi-objective optimization.
- The proposed framework for both task-level and system-level analysis will be available as an open source tools at *https://cfaed.tu-dresden.de/pd-downloads*.

The rest of the paper is organized as follows. In Section II,

[1]For the purpose of the current article, *layers* refers to the system stack—*Hardware*, *System Software* and *Application Software*

TABLE I: Comparing related Works for design-time CLR-integrated task-mapping

Related Works	CLR	DVFS	HMPSoC	Multi-Objective	Imperfect Mitigation
Das, et al. [10]	✗	✓	✓	✓	✗
Cheng, et al. [11]	✓	✗	✗	✓	✗
Izosimov, et al. [12]	✓	✗	✓	✗	✗
Rehman, et al. [13]	✓	✗	✗	✓	✗
Savino, et al. [14]	✓	✗	✗	✓	✗
proposed	✓	✓	✓	✓	✓

we provide the relevant background and brief overview of related work. The system model used for the evaluation of the proposed methods is presented in Section III. The proposed task-level analysis and system-level DSE methodology are presented in Section IV and Section V respectively. In Section VI, we discuss the results from the experimental evaluation of the proposed methods and conclude the article in Section VII with a discussion on the scope for related future research.

II. Background and Related Works

For our current work, we express the QoS in terms of: (1) *functional reliability* – the probability of correctness in computation results; (2) *timing reliability* in terms of *average makespan* – the average time taken for execution of an application; (3) *lifetime reliability* in terms of **M**ean **T**ime **T**o **F**ailure (MTTF) – the expected duration of system operation that does not result in permanent faults.

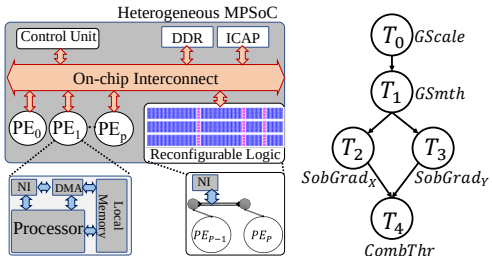
A. Cross-layer Reliability

In the CLR approach, the fault-mitigation activities are not limited to the hardware layer and an appropriate combination of methods that meets the design goals and constraints can be implemented. As discussed in [8] and [2], implementing separate fault tolerance stages at different layers can result in reduced power and area overheads. Further, distributing fault tolerance tasks to higher layers enable the designer to take advantage of the masking effects of more layers [5]. In [9], the authors proposed new techniques – *Error-aware placement* and *Failure prediction* – for globally-optimized cross-layer resilience. Similarly, in [3], the authors propose various cross-layer techniques – from *micro-architecture* to *application* level – for both general purpose processors and reconfigurable processor-based embedded systems. In [8], the authors present a cross-layer approach providing resilience in multimedia applications. However, all these works [2], [3], [5], [8], [9] do not specify a DSE methodology or framework for finding the optimal CLR configuration.

B. System-level CLR Optimization

Task-mapping of application tasks on HMPSoCs involves assignment and ordering the execution of tasks on the platform's PEs for some optimization criteria. CLR-aware task-remapping introduces the additional complexity due to the joint consideration of the PE heterogeneity, the implicit masking across multiple layers, the application-specific constraints, and designing appropriate CLR configurations for varying operating conditions [15].

For instance, in [11], a methodology is presented for optimizing the CLR configuration of a task by maximizing the fault-mitigation in software layers. Usually, software mitigation of hardware faults involves some form of *temporal redundancy* resulting in lesser area/power overheads. However, the increased execution time can lead to faster aging. In [16], the authors show the adverse effects of increasing checkpoints on permanent fault tolerance. In [14], the authors present significant improvements over [11] by using Bayesian-based methods for finding the optimal CLR configuration on different processor architectures. Similarly, in [13], the authors use layer-wise Pareto-filtering for joint optimization of timing and functional reliability. However, proposed approaches lack a methodology for implementing CLR in multi-processor systems.



(a) Architecture (b) Application

Fig. 2: System Model

In [10], the authors presented a GA-based multi-objective optimization methodology for reliability-aware task-mapping in homogeneous multi-processor systems. The proposed methods aim to provide trade-offs between functional reliability and system lifetime by varying the repetitions in task execution and the DVFS mode of operation. However, a naive extension of this approach can lead to scalability issues for larger applications. Similarly, the proposed methodology of tabu-search with list scheduling in [12] involves rejecting schedules that do not meet makespan criteria and during the search focus on changes in the critical path only. Such an approach is not scalable for multi-objective optimization since every initial schedule may lead to a feasible solution w.r.t. functional reliability and needs separate exploration. A summary of the comparisons of our current work against those discussed is shown in TABLE I.

III. System Model

The proposed DSE framework is based on a high-level representations for the system. Such an abstraction enables efficient early-stage evaluation of various design choices in terms of their feasibility and aids in design space pruning for later stages of the design.

A. Architecture Model

For the architecture model, we assume an HMPSoC containing P PEs with a distributed shared memory architecture, similar to the one shown in Fig. 2(a), with centralized control of task-remapping and CLR implementation. Each of the PEs, PE_p is characterized by the tuple: $(ID_p,\ PEType_p)$: the PE's index and type. $PEType_p$ denotes the heterogeneity among PEs and may represent the combinations of – (1) the type of processor, such as general purpose embedded processors or accelerator on reconfigurable logic (2) aging-related fault profile of the PE (characterized by β_p of the Weibull aging model) and (3) soft-error masking factor for the PE such as the Architectural Vulnerability Factor (AVF) [17].

B. Application Model

We model the application as a task-graph G_{app}, represented by a tuple $(T_{app}, E_{app}, P_{app})$, the set of task nodes, the directed connectivity of the nodes representing task dependencies, and the periodicity of the application respectively. Fig. 2(b) shows a sample task-graph for *Sobel Edge Detection* with 5 tasks (of four types) and 5 edges. Each task, $T_t \in T_{app}$, in the task-graph is characterized by the tuple $(ID_t, Type_t, Impl_t)$: the task index, task type (functionality) and the set of implementations for the task. Each i^{th} implementation of T_t, $Impl_{(t,i)} \in Impl_t$, is characterized by the following: (1) the type of PE, (2) system software – *bare-metal* system or some operating system and (3) application software – algorithms and programming languages.

C. Reliability Model

TABLE II shows the CLR model adopted for the analysis. For our current work, we consider reliability methods across three layers – Hardware (**HWRel**), System Software (**SSWRel**) and Application Software (**ASWRel**). Varying the selection and configuration of reliability methods for each layer leads to varying performance of the tasks' implementations. The corresponding task-level performance

TABLE II: CLR Model and Task-level performance metrics

Abstraction Layer	Redundancy Type	Sample Methods	Task-level Performance Metrics of $Impl_{(t,i)}$
Hardware	Spatial	**HWRel** Partial TMR, DVFS, Circuit Hardening	Scale parameter (stress indicator): $\eta_{(\mathbf{t},\mathbf{i})}$ Minimum execution time: $\mathbf{MinExT}_{(\mathbf{t},\mathbf{i})}$ Average execution time: $\mathbf{AvgExT}_{(\mathbf{t},\mathbf{i})}$ Probability of error during execution: $\mathbf{ErrProb}_{(\mathbf{t},\mathbf{i})}$ Mean time to failure: $\mathbf{MTTF}_{(\mathbf{t},\mathbf{i})}$ Average power: $\mathbf{W}_{(\mathbf{t},\mathbf{i})}$
System Software	Temporal	**SSWRel** Retry, Task-mapping Checkpointing	
Application Software	Information	**ASWRel** Code Tripling, Hamming Correction Checksum [18]	

TABLE III: System-level QoS metrics Estimation

Metric	Estimation Method	
Average Makespan ($\mathcal{S}_{app}$)	$\mathcal{S}_{app} = \max_{T_t \in T_{app}} \{\mathcal{S}_{ET_t}\}$	(1)
Lifetime Reliability ($\mathcal{L}_{app}$)	$MTTF_{(t,i,p)} = \eta_{(t,i)} \times \Gamma(1 + 1/\beta_p)$ $MTTF_p = \frac{P_{app}}{\sum_{T_p} \frac{AvgExT_t}{MTTF_{(t,i,p)}}}$ $\mathcal{L}_{app} = MTTF_{sys} = \min_{all\ PEs}(MTTF_p)$	(2)
Functional Reliability ($\mathcal{F}_{app}$)	$\mathcal{F}_t = 1 - ErrProb_{(t,i)}$, *where* $Impl_{(t,i)}$ *is used for* T_t $\mathcal{F}_{app} = \sum_{T_t \in T} \mathcal{F}_t \times \zeta_t$ *where* ζ_t = *Normalized criticality of* T_t	(3)
Power ($\mathcal{W}_{app}$), **Energy** ($\mathcal{J}_{app}$)	$\mathcal{W}_{app} = \max_{x \in (0, \mathcal{S}_{app}]} \sum_{T_t \in T_{app}} \mathcal{I}(x) \times W_t$ $where,\ \mathcal{I}(x) = \begin{cases} 1, & if\ x \in (\mathcal{S}_{ST_t}, \mathcal{S}_{ET_t}] \\ 0, & else \end{cases}$ $J_{app} = \sum_{T_t \in T_{app}} AvgExT_t \times W_t$	(4)

metrics, as described in TABLE II, are used for system-level analysis and design. The scale parameter $\eta_{(t,i)}$ is a function of the thermal profile of executing $Impl_{(t,i)}$, and is used in estimating the system's lifetime. The impact of DVFS on reliability is modeled similar to that presented in [10].

D. System-level QoS Estimation

Task-mapping with CLR involves executing an implementation, $Impl_{(t,i)}$, with a CLR configuration, say $\mathcal{C}_t$, for every task, $T_t \in T_{app}$, on any of the available PEs of the hardware platform in some ordering. With the resulting execution schedule, the relevant system-level QoS and performance metrics are estimated as shown in TABLE III.

1) Average makespan: $\mathcal{S}_{ST_t}$ and $\mathcal{S}_{ET_t}$ refer to the average start and end execution time respectively for task T_t.

2) Functional Reliability: We use a task criticality-based method for functional reliability estimation.

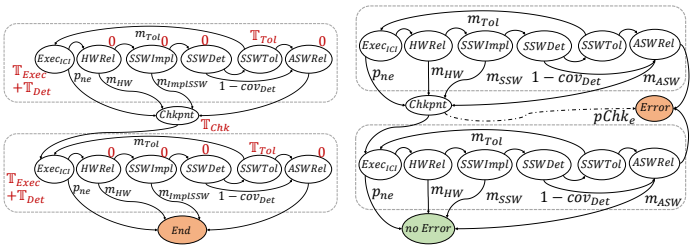
3) Lifetime Reliability: We represent the system's lifetime reliability by the MTTF of the system, $MTTF_{sys}$ with the the reliability model similar to that presented in [19].

4) Power and Energy: The system's peak power dissipation is obtained from the sum of average power dissipation of all tasks executing at any time instant.

IV. Task-level Analysis

A. Markov Chain-based Reliability Modeling

In [20], a Markov chain-based modelling of the execution of a task implementing Checkpointing with roll-back recovery was presented. The properties of the corresponding transition matrix was then used to estimate the average execution time of the task. In our current work, we introduce additional states in the Markov chain to



(a) Timing Reliability (b) Functional Reliability

Fig. 3: Markov Chain-based reliability modeling for a task implementing a single checkpoint along with *HWRel* and *ASWRel*

model cross-layer reliability. Fig. 3(a) shows the resulting Markov chain for a task with two inter-checkpoint intervals. State $Exec_{ICI}$ with residence time $\mathbb{T}_{exec} + \mathbb{T}_{Det}$ (the sum of useful execution time and error-detection time) represents useful computation state. Similarly, the $HWrel$ state, with zero residence time, represents the net effect of any spatial redundancy based fault-mitigation. The states SSW_{Impl}, SSW_{Det}, and SSW_{Tol} represent the implicit masking and the overheads due to detection and tolerance respectively at the SSW layer. The effect of $HWRel$ masking, implicit SSW Masking and the coverage of the detection and tolerance methods are denoted by m_{HW}, $m_{implSSW}$, cov_{Det} and m_{Tol} respectively. The $\mathbb{T}_{Det}$ is added to $Exec_{ICI}$ state as detection is always executed irrespective of any masking from the HW layer. However the overhead $\mathbb{T}_{Tol}$ occurs only when any error is detected and hence assigned to state SSW_{Tol}. The state $Chkpnt$ represents the Checkpoint creation state and the End state denotes the end of execution.

We used a similar approach to model the functional reliability, as shown in Fig. 3(b) for the same CLR configuration as Fig. 3(a). However, in this case, we used two absorbing states to denote the occurrence of an error despite the CLR implementation, ($Error$) and the absence of any errors during execution ($noError$). The masking probabilities are similar to those describe in the earlier paragraph. Additionally, any errors during Checkpointing itself can be modelled in the Markov chain, as shown by the dotted line. This Markov-chain based approach enables modelling of various factors such as unequal checkpoint intervals, imperfect fault detection or tolerance and implicit masking. With this approach, any arbitrary CLR configuration can be modeled into a Markov chain and the same estimation methods can be used to estimate task-level reliability metrics. With a given absorbing Markov chain the estimation of the average execution time follows that proposed in [20]. The term p_{ne} denotes the probability of no errors during useful execution time and is obtained as $p_{ne} = e^{-\lambda \mathbb{T}_{exec}}$, where λ is the **S**ingle **E**vent **U**pset (SEU) rate. The functional reliability is estimated from the absorbing probability of the $noError$ state [21].

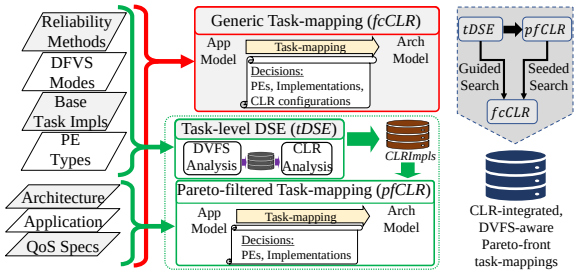
V. System-level Analysis

A. Problem Statement

The set of all possible cross-layer reliability configurations for each task is represented by:

$\mathcal{C}_t = \mathcal{HWR}el_t \times \mathcal{SSWR}el_t \times \mathcal{ASWR}el_t \ , \quad \forall T_t \in T_{app}$

It denotes the Cartesian product of the combinations of fault (error) masking, detection, and tolerance methods for a task T_t across different layers. Similarly, $\mathcal{M}_{app}$ is used to denote the set of task-to-PE binding and task-scheduling choices of the application. $\mathcal{M}_{app}$ depends on the number of available PEs – P, in the architecture and the number of tasks, T, in the application. For a generic CLR-aware problem, the constraints of maximum $\mathcal{S}_{app}$, minimum $\mathcal{F}_{app}$, minimum $\mathcal{L}_{app}$, maximum $\mathcal{J}_{app}$ and maximum $\mathcal{W}_{app}$ (TABLE III) are represented by the terms $\mathcal{S}_{SPEC}$, $\mathcal{F}_{SPEC}$, $\mathcal{L}_{SPEC}$, $\mathcal{J}_{SPEC}$ and $\mathcal{W}_{SPEC}$ respectively. Any arbitrary CLR-integrated task-mapping configuration, $\mathcal{X}_i$, represents the task-to-PE binding, task-scheduling and CLR-related design choices for the application. The notations



(a) DSE approaches (b) Proposed

Fig. 4: System-level DSE Methodology

for the resulting system-level performance metrics are denoted by $\mathcal{S}_{app}(x_i)$, $\mathcal{F}_{app}(x_i)$, $\mathcal{L}_{app}(x_i)$, $\mathcal{J}_{app}(x_i)$ and $\mathcal{W}_{app}(x_i)$. With this terminology, a generic CLR-integrated task mapping optimization problem is shown in Equation (5). The terms $w_{\langle m \rangle}$ determine the application-specific priority of the system-level metrics ($\langle m \rangle$). Similarly, the terms $c_{\langle m \rangle}$ are used to denote the existence of constraints due to application-specific QoS requirements. $Impl_t$ refers to the set of implementations (not considering CLR choices) for each task T_t.

$$\begin{aligned} \underset{x_i \in x_{app}}{\text{minimize}} \quad & \left\{ \begin{matrix} w_{\mathcal{S}}\mathcal{S}_{app}(x_i), w_{\mathcal{F}}\mathcal{F}_{app}(x_i), \\ w_{\mathcal{L}}\mathcal{L}_{app}(x_i), w_{\mathcal{W}}\mathcal{W}_{app}(x_i), w_{\mathcal{J}}\mathcal{J}_{app}(x_i) \end{matrix} \right\} \\ \text{s.t.,} \quad & \mathcal{S}_{app}(x_i) \leq c_{\mathcal{S}}\ \mathcal{S}_{SPEC}, \mathcal{F}_{app}(x_i) \geq c_{\mathcal{F}}\ \mathcal{F}_{SPEC}, \\ & \mathcal{L}_{app}(x_i) \leq c_{\mathcal{L}}\ \mathcal{L}_{SPEC}, \mathcal{J}_{app}(x_i) \leq c_{\mathcal{J}}\ \mathcal{J}_{SPEC} \\ & \mathcal{W}_{app}(x_i) \leq c_{\mathcal{W}}\ \mathcal{W}_{SPEC} \end{aligned} \tag{5}$$

where,

$$X_{app} = \begin{cases} \mathcal{M}_{app} \times \prod\limits_{T_t \in T_{app}} Impl_t, & \text{for task-mapping only} \\ \prod\limits_{T_t \in T_{app}} \mathcal{C}_t, & \text{for cross-layer-reliability only} \\ \mathcal{M}_{app} \times \prod\limits_{T_t \in T_{app}} (Impl_t \times \mathcal{C}_t), & \text{for CLR task-mapping} \end{cases}$$

B. DSE Methodology

DSE in vast design spaces using MOEA-based methods usually involves multiple optimization stages that implement some form of decomposition and directed seeding to inject problem-specific knowledge [22]. We use a similar approach in our proposed multi-stage optimization methodology, shown in Fig. 4(a), with successive design space pruning and directed search for the optimal Pareto-front. The constituent techniques are described below.

1) Full configuration CLR (fcCLR): This method refers to a problem-agnostic approach to task-mapping with CLR [10]. All the CLR configuration decisions along with those related to task-to-PE binding and scheduling are treated as separate degrees of freedom. The total number of possible design points can be expressed as $P^T \times T! \times \prod_{t=1}^{t=T}(I_t\ FM_{CL})$, where I_t is the number of possible $Impl_t$ choices for task $Task_t$ and $FM_{CL} = |\mathcal{C}_t|$ is the product of the number CLR configurations for task T_t.

2) Task-level Pareto-filtered CLR (pfCLR): This method refers to the system-level DSE with the Pareto-filtered implementations only. In this case the exhaustive number of possible design points is reduced to $P^T \times T! \times \prod_{t=1}^{t=T}(Ipf_t)$, where Ipf_t is the number of CLR-integrated Pareto-filtered implementations of task T_t.

3) Proposed Methodology: In *fcCLR*, the joint consideration of all CLR-based design decisions can lead to a design space explosion and the random selection of choices for each decision can lead to wasted search time in the infeasible region. Further, the fitness evaluation of each individual involves calculating the task-level metrics for the selected CLR-configuration for each task, in addition to system-level QoS metrics estimation. This approach can result in higher computation time for estimating the population's fitness and does not scale with increasing problem complexity. In *pfCLR*, the design space is pruned due to the layer-wise Pareto-filtering and the fitness evaluation of each design point involves *just* system-level QoS metrics estimation, resulting in reduced computation time.

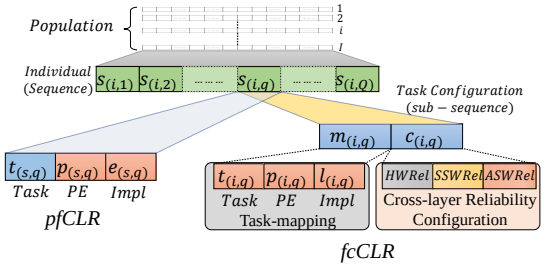


Fig. 5: Encoding for GA-based DSE

However, the number of implementations choices for each task may be increased considerably, since $Ipf_t \geq I_t$, and can lead to degraded search results. We adopt a two-stage approach to take advantage of each approach. As shown in Fig. 4(b), the system-level Pareto-fronts obtained from *pfCLR* are used as seeds for the *fcCLR* stage. This initial seeding enables a directed search and better quality results are obtained.

C. Encoding for Genetic Algorithms

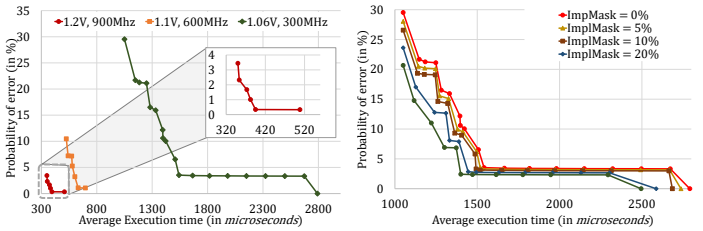
Genetic Algorithms involve using stochastic search techniques to tackle NP-hard problems based on the principles of evolution and natural genetics. The problem-specific adaptations of our proposed GA-based optimization framework are described below.

- **Individual**: As shown in Fig. 5, each ordered sequence of configuration for all tasks forms an individual in the population. The configuration decisions for *pfCLR* include the task index, the PE index to execute the task and the index of Pareto-filtered implementation to be used. Additional decisions for CLR configuration are used in *fcCLR*. The schedule of task execution is implicitly encoded in the ordering of the tasks in the individual .
- **Crossover**: We use two operations for implementing crossover: (1) A *two-point* crossover for exchanging the configuration data of some tasks; (2) A *single-point* crossover to exchange the scheduling information for some tasks.
- **Mutation**: Two mutation methods are implemented. (1) A *single-point* mutation for randomly altering the configuration of a randomly selected task. (2) A *two-point* mutation for altering scheduling data by swapping the position of two randomly selected subsequences.
- **Selection**: We use a tournament selection method for choosing individuals for the next generation. This selection method involves randomly choosing 5 (in our case) individuals from the current population and selecting the one with *best* fitness for the next generation.

VI. Experiments and Results

A. Experiment Setup

The experiments were performed on a computer with two CPUs – Intel^{TM} Xeon^{TM} E5-2609 v2 @ 2.50GHz (each CPU is quad-core) and 32 GB of memory. Experiments were carried out for a real-life application—Sobel Edge detection (shown in Fig. 2(b))–and synthetic task-graphs. The synthetic task-graphs and tasks' execution times for the synthetic applications were generated using Task Graphs For Free (TGFF) tool. All the applications were mapped to an HMPSoC with 6 PEs of 3 different types– four embedded processors with two different masking factors and two partially reconfigurable regions. The optimization methods were implemented in Python using the DEAP and PYGMO packages for GA. Probability parameters of 0.8 and 0.05 were used for crossover and mutation respectively. The experimental evaluation of the proposed DSE approaches involved estimating the effects of using different reliability methods at multiple layers. Three methods – $GenM$, $GenD$, and $GenT$ – with tunable performance parameters to model generic fault/error masking, detection, and tolerance methods respectively. Further, models for the methods mentioned in TABLE II were used for the different layers.



(a) Pareto-fronts with DVFS Modes (b) Pareto-fronts with varying implicit masking

Fig. 6: Task-level DSE

TABLE IV: Number of Pareto-front design points of each task-type in *Sobel Edge Detection* (Fig. 2(b)

Sl.	Optimization Objectives	$GScale$	$GSmth$	$SobGrad$	$CombThr$
I	Average Execution time	2	2	2	2
II	I + Error Probability	12	18	15	12
III	II + MTTF	40	89	63	38
IV	III + Energy	40	89	63	38
V	IV + Power Dissipation	40	89	63	38
VI	V + Peak Temperature	40	89	63	38

B. Task-level Analysis

The proposed framework enables estimating the effect of varying the different degrees of freedom on task-level performance metrics. Fig. 6(a) shows the resulting Pareto-fronts obtained from task-level DSE (*tDSE*) for three different DVFS configurations for a single task. While each DVFS mode would map to a single design point, the inclusion of multiple reliability methods results in multiple Pareto implementations of the same task.

TABLE IV shows the number of Pareto-front design points of each task-type in *Sobel Edge Detection* (Fig. 2(b)). The number of execution cycles and power dissipation for each task type was estimated using Gem5 [23] and McPAT [24]. As shown in the table, the number of Pareto-front design points increases when multiple performance metrics are considered. The two points for each task in first row (I) is due to one implementation for each of the two $PETypes$ in the architecture model. Further, the number of implementations for each task-type remains constant after row III. The determination of MTTF, energy, power dissipation and peak temperature depends on similar factors and hence do not result in additional dominant points on the Pareto-front. The proposed framework allows the designer to consider optimization problems with all the metrics shown in the table.

Fig. 6(b) shows the variation in the Pareto-front with increasing implicit masking. The proposed Markov chain-based functional reliability model was used for the estimation. While the current experiments for *tDSE* use a brute-force search, other stochastic search methods can also be used along with the proposed framework.

C. System-level Analysis

1) Cross-layer Optimization: The CLR approach involves joint consideration of all degrees of freedom during optimization of task-mapping. A more traditional approach would entail separately optimizing with each degree of freedom and then combining those results. Fig. 7 shows the DSE results using such an *other-layer-agnostic* approach compared to that using CLR for a synthetic application with 20 tasks. The results from optimization runs using DVFS *only*, HWRel *only*, SSWRel *only* and ASWRel *only* are combined and the dominant points are used to obtain the *Agnostic* Pareto-front. As shown in the figure, the CLR approach results in a considerably improved Pareto-front. The improvement in the hypervolume of CLR-based Pareto-fronts over that of *Agnostic* are shown in TABLE V for different application sizes. The extreme improvements observed in the application with 10 tasks is an outlier. In this case, the *Agnostic* approach resulted in a single Pareto implementation compared to 15 points with CLR.

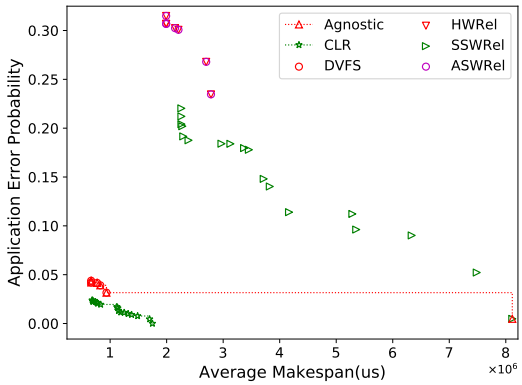


Fig. 7: Comparison of Pareto-front obtained from cross-layer optimization and the combination of points obtained from single-layer optimizations for an application with 20 tasks.

TABLE V: Improvement in DSE results (Pareto-front hypervolume) with cross-layer optimization over an *other-layer-agnostic* approach

#Tasks	10	20	30	40	50	60	70	80	90	100
% increase in hypervolume	24664	251	190	198	139	175	196	196	135	182

2) System-level DSE: The proposed approach for system-level DSE takes advantage of task-level Pareto-based design space pruning and uses multiple runs to improve the quality of the solution. Fig. 8 shows the resulting Pareto-fronts with the proposed approach and that with an extension of that proposed in [10]. The result is shown for minimization of $\mathcal{S}_{app}$ and $\mathcal{F}_{app}$ in an application with 50 tasks. The improvements in the hypervolume of the same optimization problem for different application sizes is shown in TABLE VI. Improvements of up to 231% and an average of 129% are observed.

The proposed DSE method improves upon the *pfCLR* method by executing an additional *fcCLR* optimization with starting seed solutions from *pfCLR* results. To demonstrate the shortcoming of the standalone *pfCLR* method, we executed three optimizations—with increasing number of task-level implementations for each task-type. Fig. 9 shows the number of implementations obtained from task-level DSE for each of the 10 task-types used in the synthetic applications.

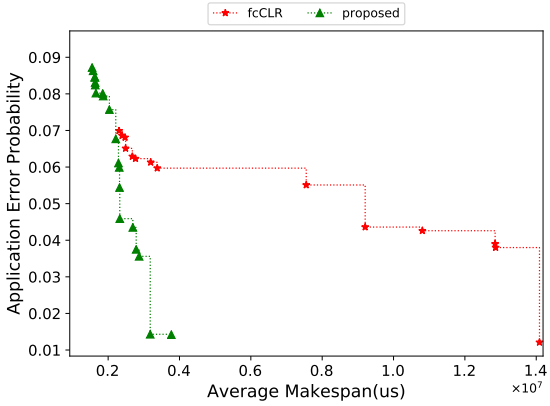


Fig. 8: Comparison of Pareto-front obtained from proposed and *fcCLR* [10] methods for an application with 50 tasks.

TABLE VI: Percentage increase in the Pareto-front hypervolume with proposed approach over *fcCLR* optimization for applications with varying number of tasks.

#Tasks	10	20	30	40	50	60	70	80	90	100
% increase in hypervolume	92	84	207	231	103	167	111	88	132	73

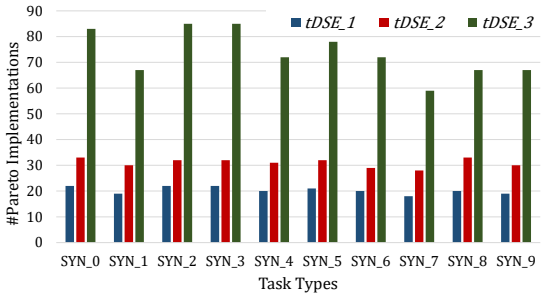


Fig. 9: Number of task-level Pareto-implementations for different task-types for three different *tDSE* executions resulting in increasing number of implementations.

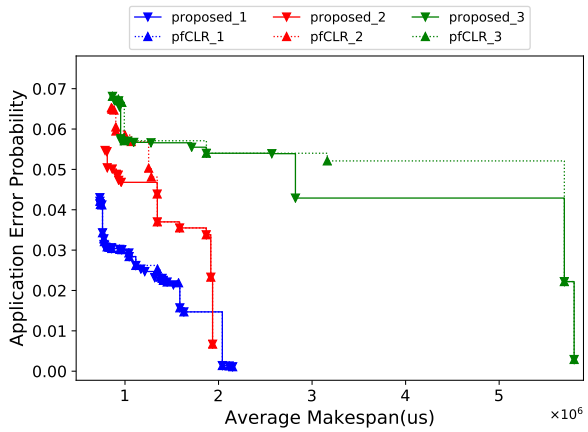


Fig. 10: Pareto-fronts for three optimization runs with proposed and *pfclr* methods with varying number of task-level implementations for an application with 30 tasks.

The resulting Pareto-fronts for an application with 50 tasks using both *pfCLR* and the proposed method for each of the three runs is shown in Fig. 10. With an increase in the number of task-level implementations, the result quality degrades with both approaches. However, as highlighted in TABLE VII, with the proposed approach, the result quality is either equal or marginally improved in most cases.

The higher number of task-level implementations were obtained by using additional task-level objectives. So, the first approach (*tDSE_1*)uses the task's average execution time and error probability as optimization objectives. Additional optimization objectives were used for *tDSE_2* and *tDSE_3*. This experiment demonstrates the importance of task-level DSE for effective system-level exploration. The proposed framework allows the designer to select task and system-level objectives independently to enable the evaluation of task-level metrics on various system-level multi-objective optimization problems.

VII. Conclusion

With the rise in the physical fault-rates due to technology scaling and architectural innovations, cross-layer reliability is becoming increasingly necessary for resource-constrained embedded systems. However, with the resulting increase in the degrees of freedom in the associated optimization problem, an early-stage exploration is necessary for determining the feasibility of different methods and hardware platforms. To this end, a framework for early-stage DSE for CLR is presented in this article. This enables the designer to perform different optimizations—both at task and system-level, evaluate the effectiveness of different reliability methods and integrate the effect of implicit masking across multiple layers. Further, the proposed DSE method shows considerable improvement over related state-of-the-art approaches. Related future research would involve further improvements to the framework by integrating the effect of the communication and storage constraints of the hardware platform.

TABLE VII: Percentage increase in Pareto-front hypervolume over *pfCLR_3* for different application sizes

#Tasks	proposed_1	pfCLR_1	proposed_2	pfCLR_2	proposed_3	pfCLR_3
10	**108**	99	71	71	**4**	0
20	**62**	59	**53**	39	**28**	0
30	**325**	324	**265**	257	**38**	0
40	92	92	**57**	56	**3**	0
50	85	85	40	40	0	0
60	**52**	51	**21**	20	**4**	0
70	80	80	**37**	36	**2**	0
80	102	102	**46**	44	**4**	0
90	**70**	69	**24**	23	**3**	0
100	**95**	94	**38**	36	**2**	0

Acknowledgment

This work is supported in part by the German Research Foundation (DFG) within the Cluster of Excellence "Center for Advancing Electronics Dresden" (cfaed) at the Technische Universität Dresden.

References

[1] P. Shivakumar, et al. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings International Conference on Dependable Systems and Networks*, pages 389–398, 2002.
[2] S. S. Sahoo, et al. Cross-layer fault-tolerant design of real-time systems. In *DFTS*, pages 63–68, 2016.
[3] J. Henkel, et al. Multi-layer dependability: From microarchitecture to application level. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.
[4] N. P. Carter, et al. Design techniques for cross-layer resilience. In *DATE*, pages 1023–1028, March 2010.
[5] T. Santini, et al. Evaluation of failures masking across the software stack. *MEDIAN*, 2015.
[6] J. Huang, et al. Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems. In *CODES+ISSS*, pages 247–256, Oct 2011.
[7] S. Kang, et al. Multi-objective mapping optimization via problem decomposition for many-core systems. In *2012 IEEE 10th Symposium on Embedded Systems for Real-time Multimedia*, pages 28–37, Oct 2012.
[8] K. Lee, et al. Mitigating the impact of hardware defects on multimedia applications: A cross-layer approach. In *Proceedings of the 16th ACM International Conference on Multimedia*, MM '08, pages 319–328, New York, NY, USA, 2008. ACM.
[9] L. Leem, et al. Cross-layer error resilience for robust systems. In *ICCAD*, pages 177–180, Nov 2010.
[10] A. Das, et al. Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs. In *DATE*, pages 1–6, March 2014.
[11] E. Cheng, et al. CLEAR: Cross-Layer Exploration for Architecting Resilience - Combining Hardware and Software Techniques to Tolerate Soft Errors in Processor Cores. In *DAC*, pages 68:1–68:6, New York, NY, USA, 2016. ACM.
[12] V. Izosimov, et al. Analysis and optimization of fault-tolerant embedded systems with hardened processors. In *DATE*, pages 682–687, April 2009.
[13] S. Rehman, et al. Cross-layer software dependability on unreliable hardware. *IEEE Transactions on Computers*, 65(1):80–94, Jan 2016.
[14] A. Savino, et al. Redo: Cross-layer multi-objective design-exploration framework for efficient soft error resilient systems. *IEEE Transactions on Computers*, 67(10):1462–1477, Oct 2018.
[15] S. S. Sahoo, et al. A Hybrid Agent-based Design Methodology for Dynamic Cross-layer Reliability in Heterogeneous Embedded Systems. In *DAC*, 2019.
[16] A. Das, et al. Aging-aware hardware-software task partitioning for reliable reconfigurable multiprocessor systems. In *CASES*, pages 1–10, Sept 2013.
[17] S. S. Mukherjee, et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *MICRO-36.*, pages 29–40, Dec 2003.
[18] M. Nicolaidis. *Soft Errors in Modern Electronic Systems*. Springer Publishing Company, Incorporated, 1st edition, 2010.
[19] Y. Xiang, et al. System-level reliability modeling for MPSoCs. In *CODES+ISSS*, pages 297–306, Oct 2010.
[20] S. S. Sahoo, et al. CLRFrame: An analysis framework for designing cross-layer reliability in embedded systems. In *VLSID*, pages 307–312, 2018.
[21] J. G. Kemeny, et al. *Introduction to Finite Mathematics*. Prentice Hall Inc, 1974.
[22] T. Chen, et al. On the effects of seeding strategies: A case for search-based multi-objective service composition. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 1419–1426, New York, NY, USA, 2018. ACM.
[23] N. Binkert, et al. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, pages 1–7.
[24] S. Li, et al. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. MICRO 42, New York, NY, USA, 2009. ACM.