Reliability-Driven Task Mapping for Lifetime Extension of Networks-on-Chip Based Multiprocessor Systems

Anup Das, Akash Kumar and Bharadwaj Veeravalli
Department of Electrical and Computer Engineering
National University of Singapore
Email: {akdas, akash, elebv}@nus.edu.sg

***Abstract*—Shrinking transistor geometries, aggressive voltage scaling and higher operating frequencies have negatively impacted the lifetime reliability of embedded multi-core systems. In this paper, a convex optimization-based task-mapping technique is proposed to extend the lifetime of a multiprocessor systems-on-chip (MPSoCs). The proposed technique generates mappings for every application enabled on the platform with variable number of cores. Based on these results, a novel 3D-optimization technique is developed to distribute the cores of an MPSoC among multiple applications enabled simultaneously. Additionally, reliability of the underlying network-on-chip links is also addressed by incorporating aging of links in the objective function. Our formulations are developed for directed acyclic graphs (DAGs) and synchronous dataflow graphs (SDFGs), making our approach applicable for streaming as well as non-streaming applications. Experiments conducted with synthetic and real-life application graphs demonstrate that the proposed approach extends the lifetime of an MPSoC by more than 30% when applications are enabled individually as well as in tandem.**

I. Introduction

To accommodate the ever increasing demand of applications and the ease of scalability, multiple cores are integrated on a single chip to form multiprocessor systems-on-chip (MPSoCs) with networks-on-chip (NoCs) as the communication backbone. However, shrinking feature-size and growing transistor density have negatively impacted their dependability by increasing the chances of failures (both permanent and transient) [1]. Permanent faults reduce MPSoC lifetime [2] and therefore, techniques are proposed to improve a system's lifetime (measured in terms of mean time to failure (MTTF)) using application task mapping and scheduling [3]–[6].

The existing lifetime reliability-aware task mapping techniques suffer from the following limitations. First, all these techniques analyze the starting mapping for a single application. However, MPSoCs are often designed for multiple applications, many of them enabled simultaneously (use-case). Formally, an use-case is defined as a collection of multiple applications that are active simultaneously on an MPSoC [7]. An important problem to address in this respect is to distribute the available cores among simultaneous applications such that the overall lifetime is improved.

Second, when applications are mapped on cores, data is communicated on the NoC links among dependent tasks which are mapped on two different cores. If the lifetime reliability of NoC links is not considered in the application mapping phase, a link can age faster potentially isolating a core from the rest of the system. This can impact system performance and indirectly the lifetime of other cores due to increased load.

Third, all existing techniques generate a starting application-core mapping but do not address recovery from faults. In modern embedded devices, tasks on faulty core(s) are distributed among functional core(s) after system restart following fault-detection. However, simple re-mapping cannot guarantee application performance requirement (throughput for example) and can potentially enhance aging of other core(s).

Finally, prior works on lifetime reliability-aware task mapping are heuristic-driven. These techniques generate a fixed number of mappings and select one that satisfies the MTTF requirement. Clearly, these mappings can be far from MTTF optimality. As a result, variations in the factors affecting aging (many of which are beyond user control) can result in a different MTTF than what is estimated. An optimal solution guarantees to generate a mapping which satisfies the MTTF requirement, provided such a solution exists in reality. Moreover, these heuristic techniques are based on task graphs and cannot be applied to cyclic graphs limiting their usage to non-streaming applications.

Contributions: Following are the key contributions.

- MTTF optimization of MPSoCs with multiple applications enabled simultaneously
- Consideration of the aging of NoC links for the MTTF computation of a core
- Generation of maximum MTTF mapping for multiple fault-scenarios
- Disciplined convex optimization based task mapping for maximization of MTTF
- Use of cyclic and acyclic graphs for lifetime reliability optimization

The convex problem formulated is solved at compile-time (design-time) for every application enabled on the platform. The solutions are task-core mappings satisfying the application deadline (or throughput) and resulting in maximizing the MTTF. Experiments conducted with synthetic and real application graphs demonstrate that our approach improves the MTTF by 30% as compared to the existing techniques when applications are considered individually. Moreover, for use-cases, our approach outperforms all the standard techniques by achieving 60% more MTTF.

To the best of our knowledge, none of the contributions of this paper have been addressed in any prior research on lifetime-reliability aware task-mapping.

The rest of the paper is organized as follows. Overview of the prior art is provided in Section II followed by preliminaries on MTTF analysis in Section III. Our methodology is discussed in Section IV followed by the problem formulation for single application and use-cases in Sections V and VI respectively. The extension of our approach for cyclic graphs (SDFG) is provided in Section VII. Finally, Section VIII

978-3-9815370-0-0/DATE13/©2013 EDAA

provides simulation results and execution time of our approach and Section IX concludes the paper with future directions.

II. Related Works

Permanent device defects have gained a lot of research focus over the past decades due to their adverse effects in the deep-submicron technologies. Quite a few research works were directed towards evaluating a single core's MTTF [2]. This was later extended for MTTF evaluation of an MPSoC employing multiple cores [8]. A parallel research direction is to optimize system MTTF at the application mapping phase assuming constant failure rate for cores [9]. Wear-out related effects are not incorporated in the analysis leading to inaccuracies in lifetime computation and optimization.

Another related research is to map tasks on an MPSoC platform with the objective of balancing the temperature of the cores [10]–[12]. Although, lifetime reliability of a core is closely related to temperature, other aging factors such as operating frequency, voltage and current-density are not captured. Recently, there are some studies incorporating lifetime reliability explicitly in the task mapping decision. Processor speed and reliability trade-off for a mapping is studied in [3]. However, generation of the ideal mapping for lifetime optimization is not addressed. A run-time fault-aware resource management technique is proposed in [4]. Although, data-communication is optimized jointly with reliability, this doesn't guarantee minimization of core failure due to aging of NoC links. Failure of cores is the minimum of the effects due to the aging of cores and the aging of links.

A simulated annealing-based task-mapping technique is proposed in [5]. MTTF is maximized assuming a series failure system i.e. a single fault breaks the complete system. This is not always true for modern MPSoCs with task migration capabilities. Ant colony heuristic is proposed in [6] to optimize MTTF. Although task migration is presumed, it also suffers from the four limitations highlighted in Section I.

III. Preliminaries on Lifetime Reliability

Extrinsic failures or wear-out related faults are well-studied phenomena for Integrated Circuits [13]. These are results of transistor feature reduction and increase in transistor count. There are four dominant wear-out effects studied for ICs: electro migration (EM), time-dependent dielectric breakdown (TDDB), stress migration (SM) and thermal cycling (TC).

For this research, EM related wear-out failures are assumed, however, any other wear-out related failure can be easily incorporated either standalone (by changing the scale parameter [2]) or using Sum-of-Failure Rate (SOFR) model for any combination of the above failure effects.

Assuming a Weibull distribution with slope parameter β, the scale parameter due to EM is calculated using Equation 1 (Black's equation ref. [13]), where Γ is the gamma function, A_0 and n are material-related constant, $J(J_{crit})$ is the (critical) current density, E_a is the activation energy, K is the Boltzman's constant and T is the temperature.

$$\alpha(T) = \frac{MTTF(EM)}{\Gamma\left(1+\frac{1}{\beta}\right)} = \frac{A_0(J-J_{crit})^{-n}e^{\frac{E_a}{KT}}}{\Gamma\left(1+\frac{1}{\beta}\right)} \quad (1)$$

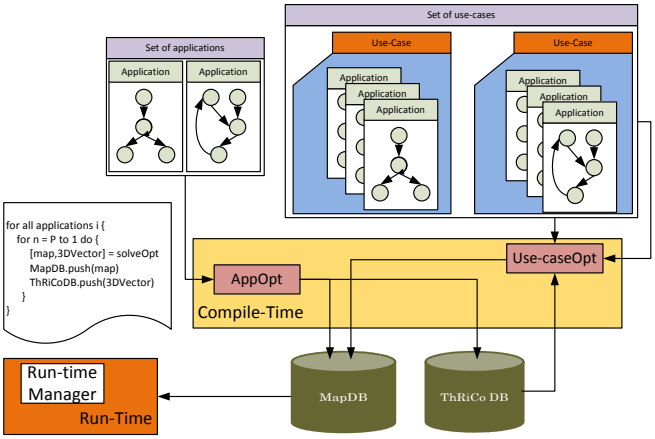

Fig. 1. Reliability-aware mapping methodology

IV. Reliability-Aware Task Mapping Methodology

The reliability-aware task mapping methodology consists of two phases – analysis (application and use-case level) at compile-time and execution at run-time. The focus of this research is on the compile-time analysis; however, for the sake of completeness, a brief overview is provided on how to use the compile-time analysis result at run-time.

A. Compile-Time Analysis

The compile-time design methodology is highlighted in Figure 1. There are two databases for the target MPSoC – the set of applications (N) and the set of use-cases. Assuming the target architecture consists of P cores, the task-mapping problem (to maximize MTTF) is solved for $n = 1$ to P cores for each application. Thus, $N * P$ optimization are solved at compile-time to generate $N * P$ mappings. This is performed in the *AppOpt* block. The result of each optimization is a task-core mapping (stored in *MapDB*) and a three-dimensional (3D) vector – throughput, reliability (MTTF) and core count. The 3D vector is stored in a database in memory called *ThRiCoDB*.

In the final phase, optimization is performed with the given set of use-cases (*UseCaseOpt* block). The result of this analysis is the core distribution among simultaneous applications. This information is also stored in the *MapDB*.

B. Run-Time Resource Management

Run-time resource management requires addressing two scenarios – run-time resource variability due to core failures and resource availability due to simultaneous applications.

1) Dealing with core failures: When an application is enabled individually, the entire set of cores is dedicated to the application. The optimum task-mapping for the application (corresponding to the set of cores) is fetched from the *MapDB* and applied. When one or more cores fail, the system will restart and the maximum MTTF mapping with the reduced set of resources will be fetched. Thus, multiple core failures are addressed and for every fault-scenario, a mapping is applied to maximize the operational lifetime of the MPSoC.

2) Managing use-cases: When a use-case is enabled, corresponding mapping is fetched from the *MapDB* and cores are distributed accordingly. If the MPSoC contains same number of functional cores as was available initially (no fault occurred), core distribution is according to the mapping

fetched for the use-case. However, if there are failures, one or more cores need to share tasks of multiple applications.

V. Problem Formulation For Single Application

The application and the architecture are modeled as follows.
Application: An application is a directed acyclic graph $G_{app} = (V, E)$, where V is the set of nodes (representing tasks of the application) and E is the set of directed edges (representing the data dependency among various tasks). Let L define the set of leaf tasks of the application.
Architecture: Architecture platform consists of cores interconnected using a regular mesh-based NoC. (Figure 2). This is represented as a directed graph $G_{arch} = (S, C)$, where S is the set of switches of the MPSoC platform and C represents the connection among the switches. Each switch $s \in S$ can be attached to one or more cores. Let P represent the set of cores for the MPSoC and P_s denote the set of cores attached to the switch s. Then, $|P| = \sum_{\forall s \in S} |P_s|$.

A. Variables for Problem Formulation

$$
\begin{aligned}
x_{ik} &= \begin{cases} 1 & \text{if task } i \text{ is mapped on core } k \\ 0 & \text{otherwise} \end{cases} \\
d_{ij,k} &= \begin{cases} 1 & \text{task } i \text{ and } j \text{ are mapped on core } k \\ & \text{and } i \text{ starts execution before } j \\ 0 & \text{otherwise} \end{cases} \\
s_{ik} &= \text{start time of task } i \text{ on core } k
\end{aligned}
$$

B. Constraints used in the optimization formulation

- Every task must be assigned to a single core

$$\forall i \in V : \sum_{k=1}^{|P|} x_{ik} = 1 \tag{2}$$

- Finish time of every leaf task is less than the deadline

$$\forall i \in L, k \in P : s_{ik} + et(i) \leq D + (1 - x_{ik})M \tag{3}$$

where $et(i)$ is the execution time of task i, D is the deadline and M is a very large number

- Data dependency constraint

$$\forall (i,j) \in E \text{ and } k,l \in P : s_{ik} + et(i) \leq s_{jl} \tag{4}$$

where task j is dependent on the data produced from task i and k,l are two cores (can be both same).

- Independent tasks mapped on the same core must not be executed simultaneously

$$
\begin{aligned}
&\forall (i,j) \notin E \text{ and } k \in P \\
&s_{ik} + et(i) \leq s_{jk} + (3 - x_{ik} - x_{jk} - d_{ij,k})M \\
&s_{jk} + et(j) \leq s_{ik} + (2 - x_{ik} - x_{jk} + d_{ij,k})M
\end{aligned} \tag{5}
$$

where the first equation constraints the starting time of i before j and the second with j before i.

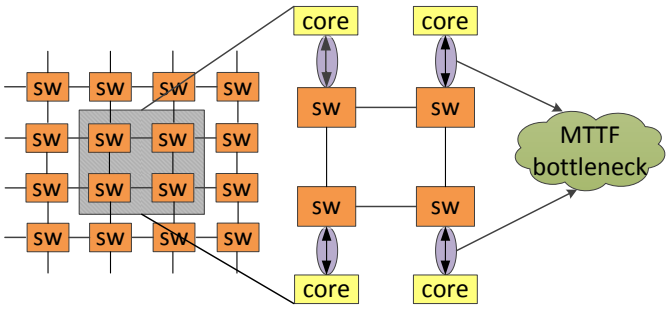

Fig. 2. Regular Mesh-based NoC

C. Objective Function

The objective function includes two quantities – MTTF of cores due to internal wear-outs (eg. EM within core) and MTTF of cores due to aging of NoC links.

1) MTTF due to internal effects ($MTTF_I$): The lifetime reliability of core k at the end of the first period of the graph is calculated according to the following equations (ref. [5]).

$$
\begin{aligned}
R_I(k, t_p) &= e^{-(A_I(k))^\beta} \text{ where} \\
t_p &= \text{period of the application} \\
A_I(k) &= \text{(Internal) Aging effect of core} k = \sum \frac{\Delta t_i}{\alpha(T_i)} \\
\Delta t_i &= \text{time intervals within period } t_p \\
\alpha(T_i) &= \frac{A_0 (J - J_{crit})^{-n} e^{\frac{E_a}{KT_i}}}{\Gamma\left(1 + \frac{1}{\beta}\right)} \text{ (refer Eq. 1)}
\end{aligned}
$$

The reliability of core k after m periods of the application graph and the closed form expression for MTTF are given by the following equations.

$$R_I(k, t_{mp}) = e^{-(m \times A_I(k))^\beta} \tag{6}$$

$$MTTF_I(k) = \sum_{i=0}^{\infty} e^{-(i \times A_I(k))^\beta} \times t_p \tag{7}$$

2) MTTF due to NoC ($MTTF_E$): The reliability of a core due to underlying NoC is derived based on the assumption that a functional core becomes un-usable (equivalent to faulty) when the link connecting it to the switch breaks (refer Figure 2). Assuming electro migration effect for links (interconnect), the closed-form expression for reliability of core k due to NoC can be derived in a similar manner with the exception of the aging effect which is computed as follows:

$$A_E(k) = \frac{C(k)}{BW \times \alpha(T)} \tag{8}$$

where $C(k)$ is the volume of data communicated to and from core k and BW is the bandwidth of the link.

3) Overall MTTF of a single core: A core is treated as un-usable in two conditions – wear-out fault in the core or fault in the link connecting it to the switch of the NoC. The overall MTTF of core k is therefore the minimum of the two MTTF and is given by

$$MTTF(k) = min\{MTTF_I(k), MTTF_E(k)\} \tag{9}$$

Algorithm 1 Core distribution for use-case

Input: $ThRiCoDB$
Output: distribution of cores among applications
1: Initialize : $x_{a_i} := 0,\ 1 \leq i \leq n$
2: Initialize : $RiList.push(a_i, x_{a_i}, 0),\ 1 \leq i \leq n$
3: **for** $j = 1$ to $|P|$ **do**
4: $\quad RiList.sort()$
5: $\quad a_k :=$ Task with least MTTF
6: $\quad x_{a_k} := x_{a_k} + 1$
7: $\quad M_{a_k} := ThRiCoDB.getMTTF(a_k, x_{a_k})$
8: $\quad RiList.update(a_k, x_{a_k}, M_{a_k})$
9: **end for**

4) MTTF of MPSoC: The MTTF of the overall system is governed by the minimum MTTF of all cores. The optimization objective can be written as

$$\text{maximize } f(x) = \min_k \{MTTF_I(k), MTTF_E(k)\} \tag{10}$$

5) Simplification: The MTTF is an exponential function which is hard to solve. However, maximization of MTTF is equivalent to minimization of the aging effect $A_I(A_E)$. Equation 10 can therefore be re-written as

$$\text{minimize } f(x) = \max_k \{A_I(k), A_E(k)\} \tag{11}$$

D. Solution Approach

The objective function in Equation 11 is convex (proof is omitted for space limitations) and is solved using CVX, a package for specifying and solving convex programs [14], [15]. However, to use this tool, the objective and the constraint functions needs to be modified into *disciplined convex programming*. This is shown in Equation 12.

$$\begin{array}{ll} \text{minimize} & f(x) = \lambda \\ \text{subject to} & \forall k \in P : A_I(k) \leq \lambda \\ & \forall k \in P : A_E(k) \leq \lambda \\ & \text{constraints [2-5]} \end{array} \tag{12}$$

VI. Problem Formulation For Use-Cases

In this section use-case level optimization problem is formulated based on the results obtained in Section V. As established previously, the *ThRiCoDB* contains 3D databases with throughput and MTTF number for every core count of every application. The problem addressed here is to merge these 3D databases for applications enabled simultaneously such that the distribution of cores among these applications maximizes the system MTTF. For the ease of problem formulation, the following notations are defined:

$$\begin{array}{rcl} a_1, \cdots, a_n & = & \text{n applications enabled simultaneously} \\ x_{a_i} & = & \text{number of cores for application } a_i \\ M_{a_i} & = & \text{MTTF of } a_i \text{ mapped on } x_{a_i} \text{ cores} \\ & = & ThRiCoDB.getMTTF(x_{a_i}) \\ T_{a_i} & = & \text{Throughput of } a_i \text{ mapped on } x_{a_i} \text{ cores} \\ & = & ThRiCoDB.getThr(x_{a_i}) \end{array}$$

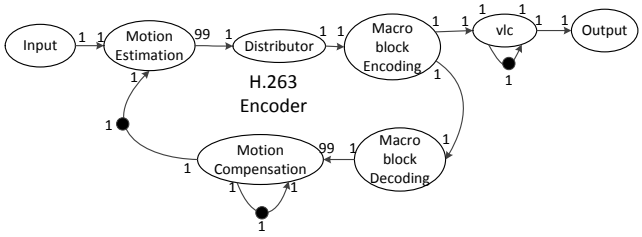

Fig. 3. SDFG of H.263 Encoder

A. Formulation

The optimization problem is formulated as below.

$$\begin{array}{ll} \text{maximize} & f(x) = \min_i \{M_{a_i}\} \\ \text{subject to} & \sum_{i=1}^{n} x_{a_i} = |P| \\ & \forall j, T_{a_i} \geq \text{throughput constraint of } a_i \end{array} \tag{13}$$

B. Solution Approach

Algorithm 1 provides the pseudo-code to solve Equation 13. A list is defined ($RiList$) to store the applications (its ID) of the use-case, the number of cores dedicated to it and the corresponding MTTF value. For every core in the target MPSoC (line 3), the $RiList$ is sorted to determine the application with the least MTTF (line 4-5). A core is dedicated to this application (line 6) and the corresponding MTTF is fetched from the *ThRiCoDB* (line 7) and the $RiList$ is updated.

VII. Extension to SDF Graphs

Synchronous Data Flow Graphs (SDFGs, see [16]) are often used for modeling modern DSP applications and for designing concurrent multimedia applications implemented on a multiprocessor system-on-chip. The nodes of an SDFG are called $actors$; they represent functions that are computed by reading $tokens$ (data items) from their input ports and writing the results of the computation as tokens on the output ports. The number of tokens produced or consumed in one execution of actor is called port $rate$, and remains constant. The rates are visualized as port annotations. Actor execution is also called $firing$, and requires a fixed amount of time, denoted with a number in the actors. The edges in the graph, called $channels$, represent data that is communicated from one actor to another.

Figure 3 shows the SDF Graph of H.263 encoder. There are eight actors in this graph. In the example, actor $motion\ estimation$ has an input rate of 1 and output rate of 99. An actor is called *ready* when it has sufficient input tokens on all its input edges and sufficient buffer space on all its output channels; an actor can only fire when it is ready. The edges may also contain $initial\ tokens$, indicated by bullets on the edges, as seen on the edge from actor $motion\ compensation$ to $motion\ estimation$ in Figure 3. A set $Ports$ of ports is assumed, and with each port $p \in Ports$ a finite rate $Rate(p) \in N \setminus \{0\}$ is associated.

TABLE I
EXECUTION TIME WITH VARYING TASKS AND CORES

Tasks	Execution Time (in sec.)			
	cores = 2	cores = 4	cores = 6	cores = 8
8	1.95	47.0	500	600
16	500	1,100	2000	2,500
24	1,050	2,500	3,850	5,500
32	1,800	4,000	8,000	13,500

A. Changed Variable Definitions and Additional Constraints

Let $G_{app} = (V, E)$ represent an application SDFG with V actors and E edges. The following are defined.

$$s_{ik,u} = \text{start time of } u^{th} \text{ iteration of actor } i \text{ on core } k$$

$$d^{uv}_{ij,k} = \begin{cases} 1 & \text{task } i \text{ and } j \text{ are mapped on core } k \\ & \text{and } u^{th} \text{ iteration of } i \text{ starts execution before} \\ & v^{th} \text{ iteration of } j \\ 0 & \text{otherwise} \end{cases}$$

- *Actor iteration assignment* (iterations of an actor must be assigned to the same core)

$$\forall i, j : \sum_{u=1}^{r(i)} x_{ik,u} = 0 \text{ or } r(i) \tag{14}$$

- *Auto-concurrency of actors* (multiple iterations of an actor are not enabled simultaneously)

$$\forall i, k \text{ and } 2 \leq u \leq r(i) : s_{ik,u} \geq s_{ik,u-1} + et(i) \tag{15}$$

- *Data-dependency of actors* (u^{th} iteration of task i can start only after its dependent task finishes)

$$\forall k, l \in P \text{ and } \forall (i, j) \in E : s_{ik,u} \geq e_{jl,m} \tag{16}$$

where m is defined as follows

$$m = \lceil \frac{kp}{q} \rceil + init(i, j)$$

p = tokens produced by actor i on edge (i, j)

q = tokens consumed by actor j from edge (i, j)

$init(i, j)$ = initial token on edge (i, j)

B. Objective Function

The objective function is same as that for the acyclic graph derived in Section V.

VIII. RESULTS

Experiments are conducted on a quad-core Intel Xeon 2.4GHz server running Linux with synthetic application task graphs generated using *TGFF* tool [17]. The number of tasks range from 4 to 32 and the targeted MPSoCs consist of 2 to 8 homogeneous cores. Additionally, a set of real-life applications are considered both from streaming and non-streaming domain. The following parameters are used for computing MTTF, [5]: current density $J = 1.5 \times 10^6 A/cm^2$, activation energy $E_a = 0.48eV$, slope parameter $\beta = 2$, temperature $T = 350K$ and $n = 1.1$. The scale parameter of each core is normalized so that its MTTF is 75 years for the characterization temperature of $350K$.

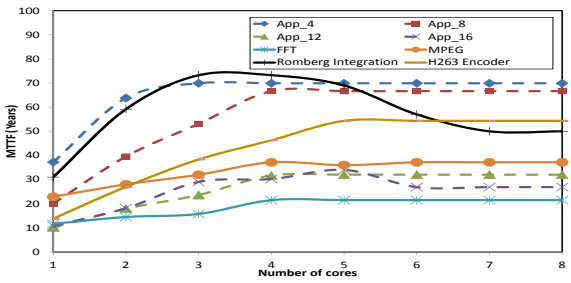

Fig. 4. MTTF of applications with varying cores

A. Single Application Results

Table I reports the execution time of our approach (using CVX solver) as the number of tasks and cores scales. In comparison, the simulated annealing-based heuristic proposed in [5] reported an execution-time of 50 to 200sec for the same range of tasks and cores on a similar CPU configuration. The convex optimization approach proposed here is slower than the heuristic approach and grows exponentially with the number of tasks and cores. However, as the problem is solved at compile-time, the reported time growth is feasible.

Figure 4 plots the variation in MTTF for different applications as the number of cores in the MPSoC increases. MTTF of most applications increases initially due to the distribution of tasks among the available cores (fewer computations per core imply less aging). As the number of cores increases further, the aging due to increased usage of NoC links (as communicating tasks are mapped on different cores) becomes significant. Thus, the two opposing effects balance out and the MTTF saturates either to its peak value (called the point of diminishing returns, *PODR*) or somewhat lower value. The *PODR* for an application depends on the task computation and data communication for the application.

In previous research works, NoC aging is ignored (no *PODR*). MTTF is assumed to grow with the number of cores and therefore the calculated MTTF value is an overestimate.

Figure 5 plots the MTTF (in years) of a reference MPSoC with 4 cores for 8 applications (synthetic and real-life). The number of tasks in each application is indicated in the corresponding name of the application. There are three results for each application as shown in the figure. The first bar corresponds to the MTTF obtained by optimizing the aging of cores assuming no communication medium in the platform. However, this is not realistic. The MTTF for the same platform with NoC included but optimizing only the aging of cores is indicated by the second bar. Finally, the third bar plots the results obtained using our technique, where MTTF is obtained by jointly optimizing the aging of cores and NoC-links.

As can be seen from the figure, ignoring NoC aging can over estimate MTTF of MPSoC by average 50% (first and second bar in the figure). The MTTF obtained using our technique is better than the MTTF obtained by optimizing the aging of cores alone (second and thrid bar in the figure). The percentage improvement is dependent on the ratio of the computation and communication in an application. For computation-dominated applications such as App_24 and App_32, the improvement is less than 0.5%. On the other end, for communication-dominated applications such as Romberg integration, more than 80% gain is obtained. On average, our proposed approach increases the MTTF of an MPSoC by more than 30% as

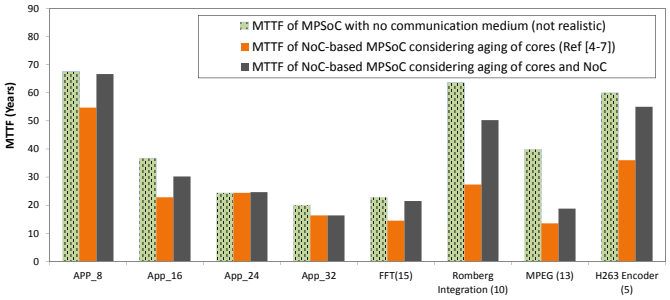

Fig. 5. Optimization results with synthetic and real applications

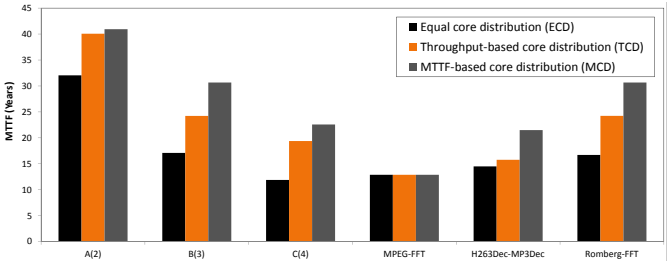

Fig. 6. MTTF improvements with synthetic and real use-cases

compared to the existing approaches.

B. Use-case Results

Since our work is the first work on use-case level MTTF optimization, there is no reference for comparison. However, two standard strategies are developed for core distribution among parallel applications – Throughput-based Core Distribution (TCD) and Equal Core Distribution (ECD). Our approach is referred as MTTF-based Core Distribution (MCD).

Figure 6 plots the MTTF for the three approaches for different use-cases with synthetic (denoted by alphabets) and real-life applications. The number in parenthesis for each use-case indicates the number of simultaneous applications.

As can be seen (and also expected), the MCD achieves highest MTTF among all the three approaches. On average, MCD increases the lifetime of an MPSoC by more than 15% as compared to TCD and more than 60% as compared to ECD. The important conclusion to make from these results is that if core distribution for use-case is not performed with MTTF as an objective, some cores can age faster than others, thereby reducing the operational life of an MPSoC.

Finally, the complexity of Algorithm 1 is calculated as follows. For every iteration of the outer loop (number of cores), sorting of MTTF is performed once followed by memory lookup. If the memory lookup time is assumed to be constant and there are n applications enabled simultaneously on $|P|$ cores, every loop is executed in $O(n \log n)$. The overall complexity of Algorithm 1 is therefore $O(|P| \times n \log n)$. On the same simulation platform, this algorithm takes between 80-100μsec for 2 to 6 simultaneous applications on MPSoC with 4 homogeneous cores.

Currently, one limitation of the use-case level analysis is that the set of use-cases needs to be pre-defined for the target MPSoC. However, the small execution time of use-case optimization clearly motivates to shift the use-case optimization to run-time. Thus, any combination of applications can be supported. This is left as future research.

The heuristic techniques proposed in [5] and [6] needs to be re-executed for every use-case to meet an MPSoC MTTF requirement. This is because the sub-optimal results for individual application cannot guarantee to satisfy the MTTF requirement when multiple of them (applications) are enabled simultaneously. This can have huge impact on the execution time (not feasible to move it to run-time phase) and a possible limitation on the number of use-cases to be supported.

IX. Conclusion

This paper presents for the first time a convex optimization-based mapping generation technique to maximize the MTTF of an MPSoC considering aging of NoC-links. The paper also proposes an algorithm to distribute the cores of an MPSoC when multiple applications are enabled simultaneously. Results with synthetic and real-life streaming and non-streaming applications demonstrate advantages of the proposed approach. Although, homogeneous architectures are considered in this paper, extension of the technique to heterogeneous MPSoC platforms and analysis with multiple applications sharing the same core are left as future work.

Acknowledgment

This work was supported by Singapore Ministry of Education Academic Research Fund Tier 1 with grant number R-263-000-655-133.

References

[1] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, 2003.
[2] J. Srinivasan *et al.*, "The case for lifetime reliability-aware microprocessors," in *IEEE International Symposium on Computer Architecture (ISCA)*, 2004.
[3] S. Wang *et al.*, "Thermal-aware lifetime reliability in multicore systems," in *International Symposium on Quality Electronic Design (ISQED)*, 2010.
[4] C.-L. Chou *et al.*, "FARM: Fault-aware resource management in NoC-based multiprocessor platforms," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2011.
[5] L. Huang *et al.*, "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2009.
[6] A. Hartman *et al.*, "A case for lifetime-aware task mapping in embedded chip multiprocessors," in *IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2010.
[7] A. Kumar *et al.*, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2008.
[8] L. Huang *et al.*, "AgeSim: a simulation framework for evaluating the lifetime reliability of processor-based SoCs," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2010.
[9] X. Zhang *et al.*, "A Dependability Solution for Homogeneous MPSoCs," in *IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2011.
[10] A. K. Coskun *et al.*, "Temperature aware task scheduling in MPSoCs," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2007.
[11] T. Chantem *et al.*, "Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2008.
[12] L. Thiele *et al.*, "Thermal-aware system analysis and software synthesis for embedded multi-processors," in *ACM Design Automation Conference (DAC)*, 2011.
[13] J. S. S. T. Association *et al.*, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122-B*, 2003.
[14] M. Grant and S. Boyd, "CVX: Matlab Software for Disciplined Convex Programming, version 1.21," 2011.
[15] ——, "Graph implementations for nonsmooth convex programs," *Recent Advances in Learning and Control (a tribute to M. Vidyasagar), Lecture Notes in Control and Information Sciences, Springer*, 2008.
[16] E. Lee *et al.*, "Synchronous data flow," *Proceedings of the IEEE*, 1987.
[17] R. P. Dick *et al.*, "TGFF: task graphs for free," in *IEEE Workshop on Hardware/software Codesign*, 1998.