Run-Time Mapping for Reliable Many-Cores Based on Energy/Performance Trade-offs

C. Bolchini, M. Carminati, A. Miele
Politecnico di Milano
Dip. Elettronica, Informazione e Bioingegneria
Milano - Italy
{bolchini|mcarminati|miele}@elet.polimi.it

A. Das, A. Kumar, B. Veeravalli
National University of Singapore
Dep. of Electrical and Computer Engineering
Singapore
{akdas|akash|elebv}@nus.edu.sg

***Abstract*—This paper presents a run-time resource manager for NoC-based many-core architectures that dynamically determines the most effective mapping of tasks on the processing nodes of the architecture to optimize system reliability while leveraging on performance and communication energy. An adaptive engine is exploited to pursue the given optimization goal taking into account various metrics. Experimental results for a set of benchmarks show that the adaptive engine (with MTTF optimization only) achieves 16% improvement in MTTF with respect to static pre-computed mapping at a reasonable communication energy overhead of 10%. Further, with MTTF and energy optimization engine turned-on, the adaptive engine is able to minimize communication energy consumption by 50% while exploiting the available MTTF slack.**

I. Introduction

To accommodate the applications' ever-increasing demands and to achieve and exploit easily-manageable scalability, complex many-core systems are built by integrating low-cost Commercial Off-The-Shelf (COTS) components. Most recent many-core systems consist of processing nodes interconnected via networks-on-chip (NoCs) in a mesh-based architecture. Data processing applications (e.g., audio/video processing) mapped onto these platforms are typically executed multiple times in a periodic fashion, with the average number of iterations per unit time determining the overall throughput; they are often characterized by large data exchange among the tasks. Therefore, task allocation is pivotal in determining energy consumption associated with communication among dependent tasks of the application. Data communication agnostic mapping of these applications on a many-core system can lead to a significant energy consumption on the NoC communication infrastructure, contributing to as much as $\approx 60\%$ of the overall application energy consumption [1].

Shrinking transistor geometries and aggressive voltage scaling are negatively impacting the dependability of the processing elements and the communication backbone of many-core systems [2]. Permanent device defects have gained a lot of research focus over the past decades due to their adverse effects in the deep sub-micron technologies. Quite a few research works were directed towards application mapping on many-core platforms with the objective of balancing the temperature of the cores [3]–[5]. 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. If aging is not incorporated explicitly in the application mapping, some cores can age faster than others, thereby reducing the operational life of a system.

Aging-aware task mapping techniques can be classified into design-time [6]–[10] and run-time based [11]–[13] ones. The former techniques suffer from the following limitations. First of all, most of these approaches use worst-case task execution time to determine the application task mapping, thus wasting the significant slack available at run-time due to the dynamism in the application execution time. Secondly, the same application can have multiple modes, characterized by different task execution time. Static techniques involve determining mapping for all these modes (treating them as individual applications) thereby exploding the design-space. Finally, the existing techniques are limited by the number of fault-scenarios (and hence core availability). As established in [14], the number of mappings grows exponentially with the number of fault-scenarios, which implies a significant overhead for storing and accessing them. The state-of-the-art dynamic approaches mainly suffer from two limitations; on one hand, the extra computation effort spent in defining mappings from scratch and discarding any design-time decision and, on the other hand, these approaches do not address *concurrently* the minimization of energy consumption, throughput degradation and aging.

This paper proposes a run-time technique that adapts a set of pre-computed, design-time decisions, based on run-time application dynamism. The objective of the mixed design-time/run-time approach is to mitigate aging in a many-core system and minimize application communication energy while satisfying throughput requirements to provide the desired quality-of-service to end users. The key contributions are:

- a design-time/run-time adaptive engine for application mapping on many-core systems;
- a dynamic mitigation of core aging and application communication energy;
- a fast heuristic to minimize the execution time of the run-time adaptive engine.

Experiments conducted on a set of real-life applications demonstrate that the proposed dynamic approach is able to in-

crease reliability of a 16%, with no energy optimization. With multi-application and multi-throughput scenarios, the proposed technique achieves on average 27% and 22% better results, respectively. In terms of joint optimization, the proposed dynamic engine is able to exploit the slack in the MTTF (with respect to the static scenario) to minimize the communication energy achieving 50% lower energy on average for all the applications.

The rest of the paper is organized as follows. A brief overview of the related works on reliability and energy is provided in Section II. This is followed by an introduction to application and architecture model along with problem statement in Section III. The proposed methodology is discussed next in Section IV. Experimental results are presented in Section V. Finally, the paper is concluded in Section VI with key future directions.

II. Related Work

The problem of scheduling dependent tasks with precedence constraints on a finite set of processing elements, with the aim of maximizing or minimizing an objective function, is NP-complete. Energy/reliability-aware task mapping and scheduling fall within this set of problems; it can be performed at design-time [15] or at run-time [16]. Design-time approaches can devote much more time in finding the best solution, since the computation is performed statically and off-line once in the entire system lifetime [6]–[10]. In [6] and [7], a simulated annealing-based technique is proposed to address the lifetime reliability-aware task mapping problem with the objective of maximizing system lifetime measured as mean time to failure, MTTF. Another reliability-driven task mapping approach is proposed in [8]: a cluster-based allocation technique to cope with fault-tolerant issues by smartly allocating the extra tasks needed for this purpose. In [9], two static energy-aware heuristics for task mapping are presented to optimize performance with respect to energy consumption. Finally, a convex optimization-based mapping generation technique to maximize system MTTF is proposed in [10]. Many mapping solutions for several use cases are computed at design-time and stored; at run-time, the best pre-computed solution is applied according to the current fault-scenario. This approach assumes to know all the possible run-time scenarios and requires substantial amount of memory to store a mapping database; moreover, it does not consider energy consumption.

The main limitation of all these static design-time policies is the fact that they assume a-priori knowledge of the workload (e.g., task-graph composition, execution times, and applications' schedules) and of the system (e.g., faulty nodes, and architecture aging trend). These assumptions are admissible when considering a static application scenario, but they do not hold in scenarios where the applications number and their characteristics are unknown in advance and can change in an unpredictable manner. This paper presents a novel approach to simultaneously optimize the three dimensions viz: components aging, reliability and energy consumption while adapting the system to run-time dynamism.

Dynamic approaches are more suitable to adapt task mapping at unknown run-time scenarios. An interesting work is the one presented in [12]: simple mapping configurations are statically computed and then enhanced through run-time heuristics, allowing to track actual components aging. In [13], a run-time task mapping technique is proposed to explicitly optimize system lifetime; application mapping is computed at run-time together with the frequency at which the re-mapping algorithm needs to be invoked. While these approaches share some common points with the one this paper introduces, they do not take into account energy consumption which is critical for modern embedded systems. The work that is more closely related to the one here proposed is [11]; although data-communication is optimized together with reliability, it does not guarantee maximization of the system lifetime.

Thus, to the best of the authors' knowledge, the aim of the proposed work of optimizing energy consumption and aging, while meeting a throughput constraint, represents an innovation in the many-core architecture scenario.

III. Background and Problem Statement

Before getting into the details of the proposal, we set the ground by defining the used models and the addressed problem.

A. Relevant Models

Application Model: In data-intensive computing environments, as the one we consider, it is common to execute parallel multi-threaded applications as sample applications. An application is a directed graph $G = (V, E)$, where V is the set of nodes representing tasks of the application and E is the set of edges $\{e_{i,j} \mid 1 \leq i, j \leq |V|\}$, representing data dependency among tasks v_i and v_j. Each task $v_i \in V$ is annotated with its execution time τ_i, and each edge $e_{i,j}$ with the size of the exchanged message $d_{i,j}$.

In the experimental scenarios presented later in this paper, we consider multiple applications, being executed in a mutually exclusive fashion on the reference architecture. This means that all applications are known at design-time, but not the workload in terms of the order of execution and the starting times, thus leading to a highly-variable workload scenario and the need for a run-time mapping policy.

Architecture Model: The architecture we refer to is a many-core computing fabric, highly modular and configurable, devoted to the acceleration of parallel multi-threaded data-intensive applications. The architecture is organized in basic processing nodes, interconnected to each other and to the I/O interface by means of a Network-on-Chip (NoC) infrastructure. It is not necessary, within the scope of this work, to further detail the internal structure of a single node, may it be a multi or a single processor platform, made up of general purpose processors or specific hardware accelerators; however, all the cores are considered to be homogeneous.

Being connected through a NoC infrastructure, each processing node has a specific position described by its (x, y) integer coordinates; a node can communicate with any other node in

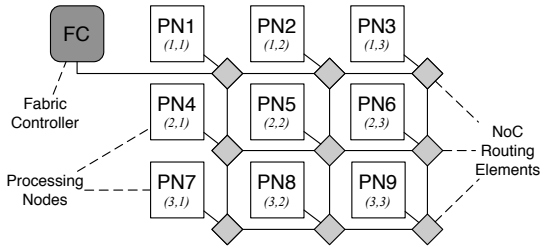

Figure 1. The reference architecture is composed of several processing elements and coordinated by a unique *fabric controller*.

the architecture by means of messages sent through the NoC routing elements. The location of the nodes is fundamental to properly model the communication among them, in terms of message latency and energy consumption, as explained in Section IV.

The platform activity is coordinated by a special node, which is named *fabric controller* (refer to Figure 1). This node is in charge of dispatching the applications to the other processing nodes, by computing the best mapping and scheduling configuration, according to specific ad-hoc designed metrics. This special node is connected to the NoC infrastructure through a special link, it is hardened by design so to achieve fault tolerant properties, and it is assumed not to influence the thermal profile of the system.

B. Problem Statement

Given i) a set of n applications $A = \{a_1, a_2, \ldots, a_n\}$, modeled as directed graphs $G = (V, E)$ and executed in a mutually exclusive fashion, ii) a computing architecture composed of m processing elements, each one described by its coordinates (x, y) and connected through a NoC infrastructure to the others, and iii) a user-defined throughput constraint, which may vary during the execution, the given applications must be mapped on the available processing nodes so as to meet the given constraint. Moreover, the mapping policies must be chosen so as to minimize the energy consumption and to maximize the nodes lifetime, by balancing the components aging to extend the system lifetime.

IV. The Proposed Approach

The proposed solution combines a static approach together with a dynamic one to solve the mapping problem in the presence of conflicting optimization goals (as performance, reliability and energy consumption) while retaining the best from both worlds. Static approaches have the flexibility to explore all possible mapping solutions to generate the optimum result, at the expense of longer design space exploration time. However, these approaches are not able to cope with changing environments of resource availability or scenarios where the exact execution time of an application is not known a-priori. Dynamic approaches are able to tackle these problems, but, on the other hand, they usually suffer from high execution time overhead or low quality solutions.

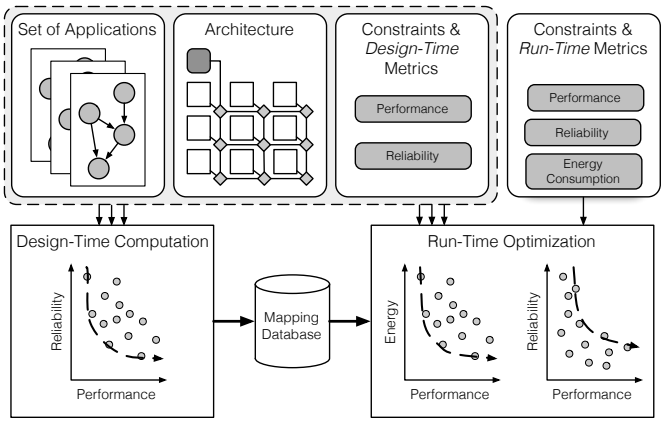

Figure 2. Overview of the proposed methodology.

An overall schema of the proposed approach is presented in Figure 2: it exploits design-time strategies to compute good mapping solutions to start from; these solutions are then dynamically optimized. In this proposal, we considered a state-of-the-art static strategy [10] aiming at optimizing reliability under a performance constraint; this approach is able to exploit all the needed information (the applications task-graphs, the architecture topology, the performance constraint and reliability/performance metrics – grouped by a dashed line in Figure 2) available at design-time and to perform an accurate design space exploration devoted to the identification of the best solution. The output of this first step is a database of solutions (*Mapping Database*), which contains the best mapping for each input application. Note that, although the static mapping of [10] is used as the starting mapping in the proposed dynamic approach, the strategy is orthogonal with respect to the static mapping generation and can be used in conjunction with other existing static techniques (e.g., [6]).

The dynamic approach selects the best mapping for an application from the database and uses it at run-time to optimize energy and reliability (measured as mean time to failure, MTTF) while fulfilling the throughput constraint.

A. Metrics Definition

When minimizing energy consumption and maximizing the components' reliability, to meet a performance constraint the proposed system must be able to capture the relevant aspects of all three considered dimensions. Performance can be directly measured in terms of the execution times, while reliability and power consumption, since direct sensors are not available, are to be modeled.

Performance: Within the proposed methodology, system performance is measured in terms of throughput. Throughput is defined as the amount of data processed by the system in a unit of time; in particular, throughput is computed, every time an application instance terminates, as the ratio between the amount of processed data during the current execution and the time elapsed between the execution start and end times, defined as *makespan*.

More formally:

$$throughput_i = \frac{data_i}{makespan_i} \tag{1}$$

where $makespan_i = (t_{end,i} - t_{start,i})$ and i is the index that is incremented by 1 every time the application (belonging to the set of n applications analyzed at design-time) is issued in the system.

Reliability: Extrinsic failures or wear-out related faults are well-studied phenomena for ICs. As established in [17], wear-out related defects are a result of transistor feature reduction and increasing transistor and power density. There are four dominant wear-out effects for ICs: electromigration (EM), time-dependent dielectric breakdown (TDDB), stress migration (SM) and thermal cycling (TC).

For the current research, since the focus of the work is on the mapping policies, EM related wear-out failures are assumed, however, any other effects can be easily incorporated either standalone or using Sum-of-Failure Rate (SOFR) model for any combination of the above failure effects. EM refers to the movement of metal atoms from the interconnect wires and vias due to the flow of current through it. The fault density due to EM is shown to have Weibull distribution with scale parameter given by the following equation (refer [17]).

$$\alpha_{EM}(T) = A_0(J - J_{crit})^{-n} e^{\frac{E_a}{KT}} \tag{2}$$

where 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.

The lifetime reliability of a processor at the end of the first period of an application graph is calculated according to the following equation (ref. [10]).

$$\begin{aligned} R(t_p) &= e^{-(A)^\beta} \text{ where} \\ A &= \sum \frac{\Delta t_i}{\alpha(T_i)} \\ \alpha(T_i) &= \frac{A_0(J - J_{crit})^{-n} e^{\frac{E_a}{KT_i}}}{\Gamma\left(1 + \frac{1}{\beta}\right)} \end{aligned} \tag{3}$$

where t_p is the period of the application, A the aging effect of processor, Δt_i the time intervals within period t_p, and β the slope parameter of the Weibull distribution. The reliability after m periods of the application graph and the closed form expression for the mean time to a *permanent* fault (MTTF) are given by the following equations.

$$R(t_{mp}) = e^{-(m \times A)^\beta} \tag{4}$$

$$MTTF = \sum_{i=0}^{\infty} e^{-(i \times A)^\beta} \times t_p \tag{5}$$

Energy Consumption: Two are the main contributions to energy consumption in NoC-based MPSoCs: *computation* and *communication* energy. The former is almost constant, regardless of the mapping, when considering homogeneous architectures [1]. For this reason it is neglected in the following analysis. The latter is mapping-dependent and represents $\approx 60\%$ of the overall application energy consumption. In [18], the authors defined bit energy (E_{bit}) as the energy consumed when one bit of data is communicated through the routers and links of a NoC.

$$E_{bit} = E_{S_{bit}} + E_{L_{bit}} \tag{6}$$

where $E_{S_{bit}}$ and $E_{L_{bit}}$ are the energy consumed by the switch and the link, respectively. The energy per bit consumed in transferring data between task v_i and v_j mapped on processor p and processor q, respectively, and positioned $n_{hops}(p,q)$ away is given by Equation 7 according to [1].

$$E_{bit}(p,q) = \begin{cases} n_{hops}(p,q)E_{S_{bit}} + (n_{hops}(p,q) - 1)E_{L_{bit}} & \text{if } p \neq q \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $n_{hops}(p,q)$ is the number of routers between processors p and q. The total communication energy is thus given by

$$CE = \sum_{(i,j) \in E} d_{ij} \times E_{bit}(p,q) \tag{8}$$

B. Run-Time Optimization

The dynamic approach has been defined in terms of an adaptive engine running on the architecture's fabric controller, able to monitor the system behavior and status (both architectural parameters and application ones), and consequently take decisions and act to modify the working conditions and parameters with the aim to improve the pursued goals. This engine has been implemented in terms of the so called Observe–Decide–Act (ODA) control loop [19]; it continues observing a set of sensed parameters and computing aggregated metrics and, based on the analysis of the metrics, the engine takes decisions about the distribution of the applications on the architecture cores, and remaps the various tasks.

The *observe* phase is devoted to the computation of the set of adopted metrics that describe the status of the system in a given instant of time. In particular, it computes the throughput according to the start and end time of each application iteration and estimates the energy consumption according to the current mapping. However, the engine cannot compute the actual MTTF because it would require to be able to predict changes in the architecture. Indeed, a good parameter representing the current aging status of each of the core is A (refer to Equation 3), and it can be computed by monitoring temperature variations in time by means of hardware sensors.

The engine enters the *decide* phase when a specified activation condition on the monitored metrics is triggered. As an example, it is possible to set an activation condition when a core is aging faster than the others (how much faster is defined by the designer). In this first proposal, we adopted an activation rule based on a fixed sleep period, i.e., the engine periodically adapts. In the future versions, activation rules that trigger adaptation on-demand will be investigated. The *decide* phase is devoted to the identification of the most convenient task, or set of tasks, to be remapped and the selection of the cores where to move them, in order to improve the current values of the optimization metrics while fulfilling the given performance constraint on the throughput. In this proposal, we considered only the relocation of a single task per move,

however, multiple relocations are under investigation to speed up the adaptation process.

The algorithm for the definition of the remapping moves is shown in Algorithm 1. The basic idea is to improve the architecture lifetime by periodically unloading the eldest core and distributing a part of the tasks mapped on it to other units, while trying at the same time to limit the energy consumption. Thus, according to the A metric, the engine selects the eldest node C_o and a set of the k youngest ones $\mathbb{C}_\mathbb{N} = \{C_{n_1}, ..., C_{n_k}\}$ (Lines 1-6). Then, the engine defines all the possible moves as a single relocation of a task t_j from C_o to C_{n_i} (Lines 7-11), and sorts them according to the following priority order: considering the age of the node C_{n_k} (youngest first, since it is the most unstressed one) and, in case of same value, considering task t_j duration (largest first, since it is the one mainly contributing to core aging – Line 12). Moreover, to pursue energy saving, in the sorting process, the age of the moves causing an increase in the communication energy will be weighted by the energy variation $\Delta CE_{t_j,C_{n_i}}$. It is worth noting that the energy variation contribution can be turned-off while optimizing reliability only.

The engine evaluates each move's costs/benefits ratio by applying it for one cycle to analyse the achievable make-span and, consequently, whether the throughput constraint is met or not. Indeed, differently from the energy variation that is estimated according to a defined model, the current make-span cannot be measured off-line, since it depends on the actual execution of the application on the architecture; in fact, an estimation of such a value would require the scheduling to be known in advance, and this computation would be too time-consuming to be performed on-the-fly on the fabric controller. Therefore, in the *act* phase the engine will attempt a move per cycle in the given order; if the move is accepted, the engine will sleep until the activation condition will be triggered another time, otherwise it will try with the subsequent move in the list (Lines 13-21).

Algorithm 1 *Task remapping strategy*

1: $T_C \leftarrow$ *specified throughput constraint*
2: $C_o \leftarrow$ *eldest core in the architecture*
3: $\mathbb{C}_n \leftarrow$ *k youngest cores in the architecture*
4: $\mathbb{T}_o \leftarrow$ *set of tasks mapped on* C_o
5: $CE_{ref} \leftarrow$ *Energy consumption of the current mapping*
6: $\mathbb{M} \leftarrow \oslash$ *– Set of candidate moves*
7: **for each** $t_j \in \mathbb{T}_o \wedge C_{n_i} \in \mathbb{C}_n$ **do**
8: *define move* $t_j : C_o \rightarrow C_{n_i}$
9: $\Delta CE_{t_j,C_{n_i}} \leftarrow CE_{ref} - CE_{move}$
10: $\mathbb{M} \leftarrow \mathbb{M} \cup \{t_j : C_o \rightarrow C_{n_i}\}$
11: **end for**
12: *Sort* $\mathbb{M}$ *according to priority function* $f(A_{C_{n_i}}, \tau_j, \Delta CE_{t_j,C_{n_i}})$
13: *apply first* $t_j : C_o \rightarrow C_{n_i} \in \mathbb{M}$
14: *run the application per 1 cycle*
15: $T \leftarrow$ *current throughput*
16: **while** $T < T_C$ **do**
17: *undo previous move*
18: *apply next* $t_j : C_o \rightarrow C_{n_i} \in \mathbb{M}$
19: *run the application per 1 cycle*
20: $T \leftarrow$ *current throughput*
21: **end while**
22: **return**

V. Experimental Results

We compare the effectiveness of the proposed dynamic approach with the *Static MaxMTTF* one, proposed in [10]. A functional simulator has been implemented using SystemC/TLM [20] to model the dynamic engine together with the described NoC-based many-core architecture running applications modeled as task-graphs. A set of six real-life applications is considered, namely FFT, MPEG Decoder, MWD, Picture-in-Picture (PiP), VOPD, and Romberg Integration, from [21]; two different architectural platforms are selected, composed of a 3×3 and 3×4 mesh NoC, respectively. The bit energy (Ebit) for modeling communication energy of an application is calculated using expressions provided in [18] for packet-based NoC using 65nm technology parameters from [22]. The following parameters are used for computing aging [10]: current density $J = 1.5 \times 10^6 A/cm^2$, activation energy $E_a = 0.48eV$, slope parameter $\beta = 2$, temperature $T = 295K$ and $n = 1.1$. HotSpot [23] has been used to characterize the temperature of the cores to account both the self-activity and the temperature of neighbour cores. The considered performance requirement is computed by taking the best performance possible on the given architecture (computed at design-time through a design space exploration) and by adding to it an extra 20%. This value has been chosen to be consistent with the application scenario and with the state of the art [10].

The introduction of an adaptive engine can cause both a performance and energy communication overhead. The former one is due to the actual time needed by the engine to compute the next move; in the considered system, it is completely hidden since such computation is performed during the applications' execution. However, changing the mapping of the architecture implies communicating information to the cores, thus introducing delays and increasing the used communication energy. Indeed, although this aspect has been neglected in this work, we expect the impact to be limited, if not negligible, with respect to the overall performance/energy consumption, being the amount of data very small.

A. Reliability Performance

Figure 3 plots the normalized aging of the proposed dynamic technique with respect to the maximum MTTF obtained using the static approach for the adopted experimental setup. In this experiment, energy optimization was disabled, and the proposed dynamic adaptation engine migrates tasks on cores to optimize MTTF. The reported MTTF values are normalized with respect to the MTTF of an unstressed architecture.

A few considerations can be drawn. First of all, for all applications considered (including those not shown explicitly in the figure), the MTTF obtained using the proposed dynamic approach is better than the one achieved in the *Static MaxMTTF* scenario, as indicated on the bars for the proposed technique. This is due to the adaptation of the architecture to balance the stress of different units thereby improving the overall MTTF. For the six considered applications, the proposed technique improves MTTF by 16% on average. Moreover, the MTTF improvement increases as the number of

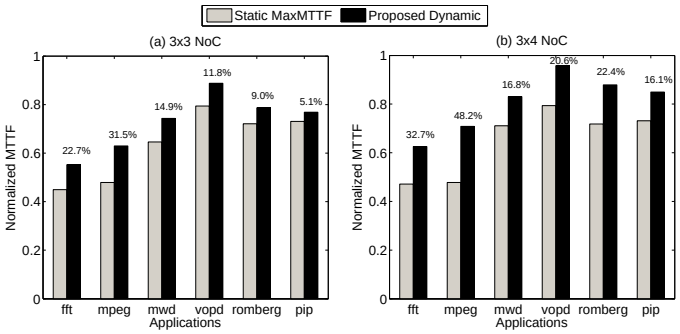

Figure 3. MTTF performance of the proposed approach.

Table I
ENERGY PERFORMANCE OF THE PROPOSED DYNAMIC APPROACH WITH MTTF OPTIMIZATION ONLY

Applications	Static MaxMTTF	Proposed Dynamic	
		3×3	3×4
FFT	1	1	1.162704082
MPEG	1	1.017674265	1
MWD	1	1.040284091	1
VOPD	1	1.138318841	1.286558603
Romberg	1	1	1.052919355
PiP	1	1.346435185	1.430726496
Average	1	1.090452064	1.155484756

cores in the architecture grows. This is because the adaptation engine is able to better balance the architecture load/aging by switching between the unused units.

Table I reports a comparative analysis of the communication energy between the static and the proposed dynamic technique (with energy optimization turned off) on the two different architectures. Energy consumption increases by an average of 9% and of 15%, on the two architectures, due to the adaptation engine that uses extra energy for task migration.

B. Multi-Application & Multi-Throughput Scenarios

Figure 4 plots the result for three multi-application and three multi-throughput test cases on the 3×3 architecture. The multi-application and multi-throughput scenarios are generated by randomly selecting the applications to increase the workload, in a not homogeneous way. The number of iterations for each application in the test cases are specified in Table II. As an example, one iteration of MultiApp01 consists of $2,000$ iterations of VOPD, $10,000$ of Romberg, and $4,000$ of FFT; this workload is repeated infinitely. The iteration of multi-throughput application MultiThr01 consists of $10,000$ iterations of MPEG with the same deadline used in the static scenario, and $10,000$ iterations with deadline relaxed by $2\times$.

The static approach generates application mappings considering an unused architecture, i.e., age of all the cores are 0. As a result, when applications are switched at run-time, the static pre-computed mappings are no-longer optimal in terms of MTTF because they ignore the current age of the different cores. Such information, on the other hand, is exploited by the dynamic approach that can adapt accordingly and produce

Table II
PARAMETERS FOR MULTI-APPLICATION AND MULTI-THROUGHPUT

MultiApp01	MultiApp02	MultiApp03
VOPD (2,000)	MPEG (5,000)	MPEG (10,000)
Romberg (10,000)	MPEG (1,000)	PiP (5,000)
FFT (4,000)	Romberg (4,000)	VOPD (4000)
	MPEG (1,000)	PiP (8,000)

MultiThr01	MultiThr02	MultiThr03
MPEG_1 (10,000)	VOPD_1 (10,000)	PiP_1 (10,000)
MPEG_2 (10,000)	VOPD_2 (10,000)	PiP_2 (10,000)

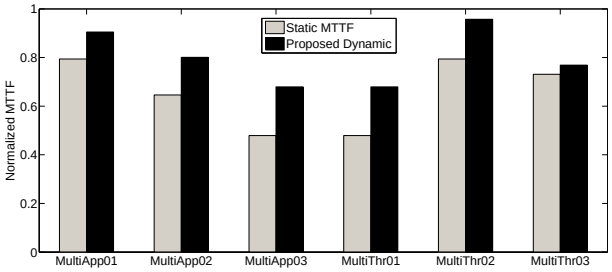

Figure 4. MTTF performance with multi-application and multi-throughput scenarios.

MTTF optimized solutions. For all the three multi-applications considered, the proposed technique improves the architecture MTTF by 27% on average.

Applications are often characterized by varying throughput requirements. Computing the application mapping at design-time for all applications and for all throughput requirements results in an explosion in the design space. With 20 applications having on average two different throughput requirements, the number of application mappings to be pre-computed at design-time is 2^{20}. This imposes a significant overhead in terms of storage for the mapping, and of time to retrieve the information at run-time. An alternative approach adopted for most design-time approaches is to generate one mapping for each of these applications by using the stricter throughput requirement. Therefore, such approach gives good results when the throughput requirement is strict, but is not able to optimize the MTTF when the requirement is relaxed. Indeed, the dynamic approach is able to adapt to this changing scenario. Although, the initial mapping for the dynamic approach is the one pre-computed at design-time with a stricter deadline, the adaptive engine is able to explore different other mappings fulfilling the relaxed throughput requirement, possibly achieving better results. Experimental results for the three multi-throughput applications indicate that the proposed technique improves system MTTF by 22%, on average, when compared to the one using static mapping without adaptation.

C. Communication Energy Performance

Figure 5 plots the normalized communication energy of the proposed technique compared to the static approach, when optimizing both communication energy and MTTF. Similarly to the previous experiments, the initial mapping for this

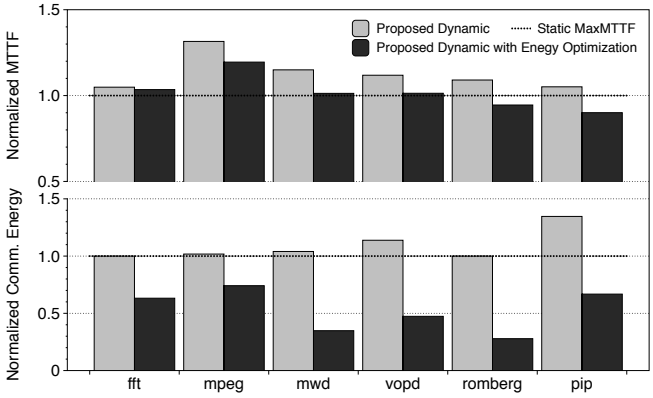

Figure 5. Communication energy performance for the proposed technique.

experiment is the best MTTF mapping coming from the static approach. However, the adaptive engine incorporates communication energy and system lifetime when selecting a local move. For a more comprehensive comparison, we report results with and without the communication energy optimization, to highlight how MTTF and energy consumption are affected. All values are normalized with respect to the data obtained using the static approach. As it can be seen, MTTF-communication energy joint optimization results in energy savings between 25% to 75% for all the applications considered with an average 5% MTTF improvement. Thus by trading-off MTTF, the communication energy can be minimized by 50%, if considering the average of the results obtained on the considered benchmark applications.

VI. Conclusions and Future Work

This paper proposes a run-time engine for NoC-based many-core architecture to adapt a pre-computed optimal application task mapping identified at design-time using a limited set of data to the current architecture/application status, aiming at optimising communication energy, system lifetime or both, depending on the requirements, under a performance constraint. Experiments conducted on a set of real-life applications demonstrate that the proposed technique improves lifetime by 16% with less than 10% communication energy overhead. When jointly optimizing reliability and energy, the proposed approach is able to exploit the lifetime slack to minimize communication energy consumption by 50%. There are however few areas of improvement. Although the proposed approach is configured to periodically adapt, a future extension will consider introducing thresholds on MTTF/energy to trigger the adaptation, thus moving to an on-demand scenario. Moreover, a detailed study will be carried out to investigate the impact of increasing network size on MTTF or energy, possibly introducing additional adaptation hooks.

References

[1] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proc. Conf. Design, Automation & Test in Europe*, 2004, pp. 234–239.

[2] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in *Proc. ACM/IEEE Design Automation Conference*, 2004.

[3] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. Conf. Design, Automation & Test in Europe*, 2007, pp. 1659–1664.

[4] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs," in *Proc. Conf. Design, Automation & Test in Europe*, 2008, pp. 288–293.

[5] L. Thiele, L. Schor, H. Yang, and I. Bacivarov, "Thermal-aware system analysis and software synthesis for embedded multi-processors," in *Proc. Design Automation Conference*, 2011, pp. 268–273.

[6] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for mpsoc platforms," in *Proc. Conf. Design, Automation & Test in Europe*, 2009, pp. 51–56.

[7] ——, "On task allocation and scheduling for lifetime extension of platform-based mpsoc designs," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2088–2099, 2011.

[8] S. Srinivasan and N. Jha, "Safety and reliability driven task allocation in distributed systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 3, pp. 238–251, 1999.

[9] A. Das, A. Kumar, and B. Veeravalli, "Communication and migration energy aware task mapping for reliable multiprocessor systems," *Future Generation Computer Systems*, 2013.

[10] ——, "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems," in *Proc. Conference on Design, Automation & Test in Europe*, 2013, pp. 1–6.

[11] C. Chou and R. Marculescu, "FARM: Fault-aware resource management in NoC-based multiprocessor platforms," in *Proc. Conf. Design, Automation & Test in Europe*, 2011, pp. 1–6.

[12] A. Hartman and D. Thomas, "Lifetime improvement through runtime wear-based task mapping," in *Proc. Int. Conf. Hardware/software codesign and system synthesis*, 2012, pp. 13–22.

[13] T. Chantem, Y. Xiang, X. S. Hu, and R. P. Dick, "Enhancing multicore reliability through wear compensation in online assignment and scheduling," in *Proc. Conf. Design, Automation & Test in Europe*, 2013, pp. 1–6.

[14] A. Das and A. Kumar, "Fault-aware task re-mapping for throughput constrained multimedia applications on noc-based mpsocs," in *Proc. IEEE Int. Symp. Rapid System Prototyping*, 2012, pp. 149–155.

[15] Y. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, 1999.

[16] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs," in *Proc. IEEE/IFIP Int. Workshop on Rapid System Prototyping*, 2007, pp. 34–40.

[17] JEDEC Solid State Technology Association and others, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122-B*, 2003.

[18] T. Ye, L. Benini, and G. De Micheli, "Packetized on-chip interconnect communication analysis for MPSoC," in *Proc. Conf. Design, Automation & Test in Europe*, 2003, pp. 344–349.

[19] J. Kephart and D. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, pp. 41–50, 2003.

[20] Accelera Systems Initiative, "http://www.accellera.org," accessed: 2013-05-19.

[21] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, 2005.

[22] W. Zhao and Y. Cao, "Predictive technology model for nano-cmos design exploration," *J. Emerg. Technol. Comput. Syst.*, vol. 3, no. 1, pp. 1–17, 2007.

[23] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 94–125, 2004.