Energy-Aware Dynamic Reconfiguration of Communication-Centric Applications for Reliable MPSoCs

Anup Das, Amit Kumar Singh and Akash Kumar
Department of Electrical and Computer Engineering
National University of Singapore
Singapore, 117583
Email: {akdas,eleaks,akash}@nus.edu.sg

***Abstract*—To accommodate the ever increasing demands of applications and for the ease of scalability, multiprocessor systems-on-chip (MPSoCs) are becoming a popular design choice in current and future technologies with streaming multimedia and other communication-centric applications constituting a large fraction of the application space. Mapping and scheduling of these applications on an MPSoC to minimize energy consumption while guaranteeing to satisfy the performance requirement is an NP-hard problem. This is coupled with the run-time variability associated with MPSoC resource availability due to the occurrence of faults. The existing studies on fault-tolerance and energy minimization are either based on static (offline) analysis which fails to capture application dynamism or do not consider throughput degradation. This paper proposes an execution trace-based run-time technique to reconfigure application mapping to minimize communication energy of an application, simultaneously dealing with the occurrence of transient, intermittent and permanent faults. Experiments conducted with synthetic and real-life applications modeled using Synchronous Data Flow Graphs (SDFGs) demonstrate that the proposed technique achieves significant improvement with respect to the state-of-the-art approaches in terms of throughput and storage overhead with less than 10% energy overhead.**

I. Introduction

Streaming multimedia applications such as H.264 decoder, MP3 encoder etc. constitute a large fraction of the embedded application space for multiprocessor systems [1]. These applications are characterized by large data exchange among different tasks. Data communication agnostic mapping of these applications on a multiprocessor system-on-chip (MPSoC) can lead to a significant energy consumption on the communication infrastructure such as networks-on-chip (NoCs), constituting as high as $\approx 60\%$ of the overall application energy consumption [2]. This has motivated researchers to investigate into communication-aware application mapping on MPSoCs [3].

Shrinking transistor geometries and aggressive voltage scaling are negatively impacting the dependability of the processing elements (cores, for example) and the communication backbone of an MPSoC [4]. One of the design objectives in deep sub-micron technologies is to provide support for tolerating multiple faults (transient, intermittent and permanent) without sacrificing solution quality (measured as throughput for streaming applications) and respecting the given energy budget. One of the traditional techniques for fault-tolerance is redundancy (hardware and/or software) [5]. This involves using spare processing elements to assume responsibility when faults occur (hardware redundancy) or executing the same task multiple times on same or different cores (software redundancy). However, stringent area and energy budgets are prohibiting the use of hardware redundancy in modern systems. Techniques such as task mapping and scheduling are gaining popularity among research community [6]. Task mapping-based fault-tolerant techniques involve offline generation of application mappings for all processor fault-scenarios [6], [7] or determining the mapping as and when faults occur [8]. The existing task mapping-based fault-tolerance techniques suffer from the following limitations.

First of all, the static techniques of [6], [7] fail to capture the dynamism observed at run-time due to unavailability of one or more component cores. Such unavailability can be attributed to resource blockage due to execution of multiple simultaneous applications or running maintenance jobs. The static mappings fail to adapt to such changing environment leading to sub-optimal results both in terms of throughput and energy consumption. Secondly, the existing static techniques determine task mapping for every application enabled on the multiprocessor platform, for every fault-scenario. These mappings are stored in a table to be looked-up as and when faults occur. The size of the look-up table grows exponentially with the number of applications and the level of fault-tolerance. This is crucial, especially for multimedia MPSoCs where storage space is limited. Third, the existing static techniques also suffer from run-time overhead of table look-up to fetch a mapping of an application for a fault-scenario. The size of the mapping table needs to be small in order to avoid the requirement of large memory space. In other words, the number of faults and/or applications that can be practically supported for an MPSoC is limited. Finally, the existing run-time fault-tolerance techniques [8] do not guarantee application throughput requirements making them unsuitable (if not completely in-applicable) for multimedia applications.

Contributions: This paper proposes a trace-based run-time reconfiguration of application mapping to minimize communication energy while satisfying throughput requirement for all processor fault-scenarios. Following are the key contributions.

- Execution trace-based dynamic reconfiguration of application mapping for different fault scenarios.
- Communication energy and storage overhead minimization on a given MPSoC.
- Consideration of throughput degradation for communication-centric multimedia applications.

The proposed technique analyzes the execution trace of an application modeled as Synchronous Data Fllow Graphs (SDFGs) [9]. Based on such analysis, a run-time manager determines the most suitable task remapping which minimizes communication energy while satisfying the application throughput requirement. Experiments are conducted with synthetic and real-life application SDFGs demonstrate that the proposed technique achieves significant improvement both in

terms of throughput and storage overhead with less than 10% energy overhead from a static mapping.

To the best of our knowledge, this is the first work on run-time resource management for SDFGs, simultaneously dealing with communication energy, fault-tolerance and throughput.

The rest of this paper is organized as follows. A brief overview of the related works is presented in Section II. This is followed by an introduction to SDFGs and the problem formulation in Section III. The execution-trace driven design methodology is presented in Section IV. Experimental results are presented in Section V and finally Section VI concludes the work with a discussion on future improvements.

II. Related works

Task mapping and scheduling has received significant attention among researchers starting from the classical optimization metric such as performance and power to the recent ones such as reliability. Details of the performance and power driven mapping techniques is beyond the scope of the current paper. Interested readers are urged to refer to [10]. Some of the key studies on communication energy and reliability-aware task mapping are presented here.

The existing communication energy aware fault-tolerant techniques can be classified into two categories – static and dynamic. Static task mapping techniques compute task mapping decisions at design-time for different fault-scenarios. As faults occur, these mappings are looked up at run-time to carry out task-migration. A fixed order Band and Band reconfiguration technique is studied in [11]. Cores of target architecture are partitioned into two bands. When one or more cores fail, tasks on these core(s) are migrated to other functional core(s) determined by the band in which these tasks belong. The core partitioning strategy is fixed at design-time and is independent of the application throughput requirement. Consequently, throughput is not guaranteed by this technique. A re-execution slot based reconfiguration mechanism is studied in [12]. Normal and re-execution slots of a task are scheduled at design-time using evolutionary algorithm to minimize certain parameters like throughput degradation. At run-time, tasks on a faulty core migrate to their re-execution slot on a different core. However, a limitation of this technique is that the schedule length can become unbounded for high fault-tolerant systems. Task remapping technique based on offline computation and virtual mapping is proposed in [7]. Here, task mapping is performed in two steps – determining the highest throughput mapping followed by generation of a virtual mapping to minimize the cost of task migration to achieve this highest throughput mapping. A limitation of this technique is that the migration overhead significantly increases as this is not considered in the initial optimization process. Moreover, throughput constrained streaming applications do not benefit from a throughput higher than required and can increase buffer requirements at output. In [13], the authors propose to minimize migration overhead while satisfying the throughput requirement for different fault-scenarios. However, energy is not considered in this technique. The technique in [6] jointly minimizes communication energy, migration overhead and throughput degradation for streaming multimedia applications modeled using Synchronous Data Flow Graphs (SDFGs). A limitation of this technique is that scheduling is not considered and therefore suffers from huge schedule construction overhead at run-time or schedule storage overhead from design-time. The limitation of these fault-tolerant techniques are discussed in Section I.

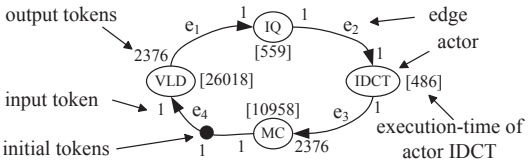

Fig. 1. SDFG model of an H.263 decoder.

Dynamic approaches monitor system status and decide to migrate tasks at run-time. A fault-aware resource management is proposed in [14] to deal with the occurrence of transient, intermittent and permanent faults. The remapping decisions are taken at run-time to minimize data communication while avoiding the faulty and other stressed processors. An integer linear programming based task remapping technique is proposed in [8] to remap tasks while minimizing communication energy. A common limitation of these approaches is that throughput is not guaranteed. Further these techniques are based on directed acyclic graphs and therefore require significant modification (if not completely in applicable) for multimedia applications represented as cyclic graphs.

III. Problem Formulation

This section provides a brief overview of the application & MPSoC platform model and challenges involved for dynamic fault-tolerance.

A. Application & MPSoC Platform Model

The streaming multimedia applications with timing constraints are considered as communication-centric applications and are modeled as Synchronous Dataflow Graphs (SDFGs) [9]. Fig. 1 shows an SDFG model of H.263 decoder. The nodes (*VLD, IQ, IDCT* & *MC*) and edges (e_1, e_2, e_3 & e_4) model tasks and dependencies, respectively. The nodes are referred to as *actors* that communicate with *tokens* sent from one actor to another through the edges. Each actor has its attributes execution time and memory requirement when mapped on a tile. Each edge has following attributes: size of a token, memory needed on the tile when connected actors are allocated to the same tile, memory needed in source and destination tiles when connected actors are allocated to different tiles and respective bandwidth requirements between the tiles. An actor *fires* (executes) when there are sufficient input tokens on all of its input edges and sufficient buffer space on all of its output connections. In each firing, the actor consumes a fixed amount of tokens from the input edges and produces a fixed amount of tokens on the output edges. These token amounts are referred to as *rates*. An edge may contain *initial tokens*.

Throughput of an application is determined as the inverse of the long term period that is calculated as the average time needed for one iteration of the application. An iteration is defined as the minimum non-zero execution such that the original state of the SDFG is obtained. Period for the example H.263 decoder is equal to the summation of ExecTime(*VLD*), 2376×ExecTime(*IQ*), 2376×ExecTime(*IDCT*) and ExecTime(*MC*), where ExecTime is the execution time of respective actors. This period does not include network and memory access delays. It should be noted that actors *iq* and *idct* have to execute 2376 times in one iteration and the number of executions is referred to as *repetition vector* of the actor. The rate 2376 is pertaining to the used video frames that have a resolution of 348 by 288 pixels.

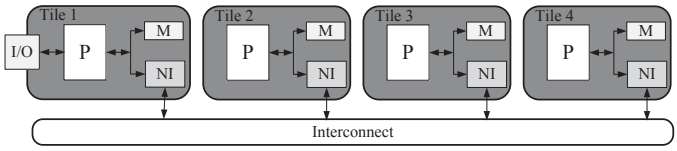

Fig. 2. Example MPSoC platform.

The MPSoC platform used in this work is a tile-based architecture as shown in Fig. 2. The example platform contains four tiles, which are connected by an interconnection network in order to facilitate communication amongst the tiles. Each tile consists of a processor (P), a local memory (M, size in bits) and a network interface (NI) to connect with the interconnect. The interconnection network provides end-to-end connections between the tiles. However, the latencies of connections can be modeled according to different network-on-chips (NoCs).

B. Dynamic Fault-tolerance

Achieving fault-tolerance at run-time due to failure of a tile executing some actors involves exploration of new actors to tiles allocations (mappings). Since several mapping options are feasible, designers need to perform exploration with some optimization objectives such as throughput and energy, in order to prune the possible vast design space (mapping options). A similar process needs to be adopted when a fault occurs in one or more additional tiles. The detection of faults and their cure is of paramount importance for many real-time systems, where a fault may lead to catastrophic consequences.

A majority of existing works perform fault-aware analysis at design-time to explore mapping solutions that can be used to cure faults incurred at run-time. The exploration is performed for all possible failure scenarios while aiming to optimize for some performance metrics. At run-time, actors are simply remapped using the compile-time decisions. The exploration involves finding different mappings and their provided performance figures, such as throughput and energy consumption. For each mapping, actors are bound to tiles and edges to memory inside tiles or to connections in the platform. The binding is considered valid if memory imposed, allocated input/output connections and allocated incoming/outgoing bandwidth are less than or equal to the maximum available on each tile. The exploration process considers only the valid bindings. These exploration methodologies require large memory space to store the mappings to cater for all the possible fault scenarios, such as 1-tile fault, 2-tiles fault, etc.

Further, the exploration process is time consuming when a large number of mappings needs to be evaluated by employing simulative evaluations. The number of mappings increases exponentially with the number of actors and used platform tiles. For example, while executing 14 actors on 14 tiles, a total of 190,899,321 mapping options need to be evaluated exhaustively to cater for fault-tolerance in case any number of tiles from 1 to 13 becomes faulty. The complete evaluation will take almost 220 days if we assume 100 milliseconds (ms) for simulating (evaluating) one mapping. Even pruning the exploration space with existing analysis strategies does not lead to acceptable evaluation time for complex applications.

To cater for a fault scenario at run-time, performing the exploration by employing simulative evaluations may lead to missed timing deadlines. Therefore, analytical estimations need to be employed to get fast results. However, the accuracy of the estimations with respect to the simulations needs to be validated. In contrast to existing approaches, our approach performs faster exploration for a run-time fault scenario by analyzing the execution traces of actors and edges of the application executing on a fixed number of tiles. Further, our approach jointly optimizes for the throughput and energy consumption unlike most of the existing approaches.

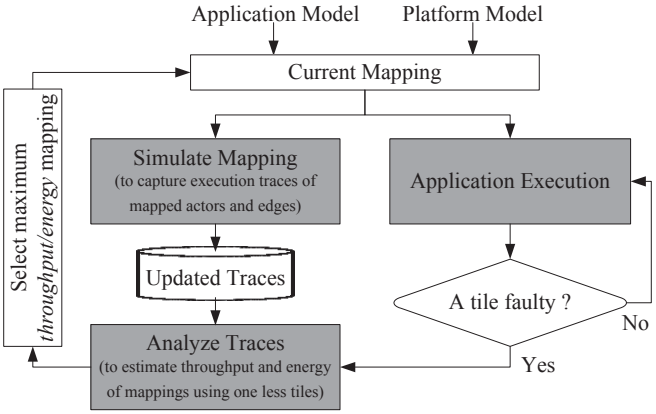

Fig. 3. Proposed dynamic fault-tolerant reconfiguration approach.

IV. Proposed Dynamic Fault-tolerant Reconfiguration Methodology

This section describes the proposed dynamic fault-tolerance approach. In contrast to conventional existing DSE methodologies, the proposed methodology differs in following aspects: *1)* performs run-time analytical analysis on execution traces to get faster results for different fault scenarios, *2)* allows executions and execution trace update (by employing simulations) in parallel to facilitate faster and accurate analysis for next possible fault scenarios, *3)* requires small storage, and *4)* jointly optimizes throughput and energy consumption.

An overview of the proposed dynamic fault-tolerant reconfiguration is presented in Fig. 3. The flow starts with a mapping (*current mapping*) to execute the application actors on the platform resources. The platform is configured based on the current mapping to execute the application. Initially, the current mapping allocates n actors of the application onto n processor tiles such that each tile contains exactly one actor and the edges are mapped onto connections. Such allocations enable to exploit all the parallelism present in the application. This kind of initial mapping assumes that at least n tiles are available in the platform. However, in case of lower number of tiles than the actors, the initial mapping can be computed by employing state-of-the-art run-time mapping approaches [3] [15]. Then, in parallel, the application is executed based on the current mapping (configuration) and the mapping is simulated (*Simulate Mapping*) to capture execution traces of actors and edges. The simulation process computes throughput of the mapping as well. The captured execution traces are stored in a database by deleting the earlier present traces (*Update Traces*) to avoid the needs for large storage. Updated traces are analyzed to estimate throughput and energy consumption of the mappings using one lower number of tiles in case a tile becomes faulty during the application execution. Out of all the evaluated mappings, the mapping having maximum throughput/energy ratio is selected as the current mapping to execute the application towards achieving the fault-tolerance. Selection of such a mapping optimizes jointly the throughput and energy consumption.

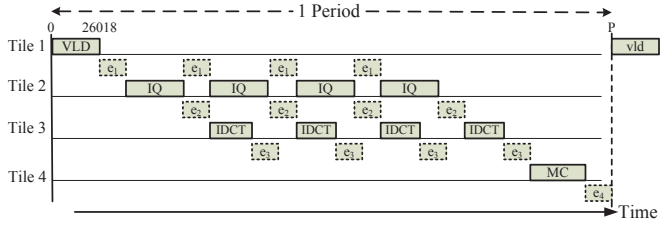

Fig. 4. Execution trace of actors/edges of H.263 decoder for one periodic execution.

The parallel simulation of the current mapping along with the application execution prepares the updated traces faster, which is always desired and might be required in case a fault occurs in the early stage of the application execution. The simulation process also guarantees for accurate execution traces, which facilitates for more accurate analytical estimations (*Analyze Traces*). In case one additional tile becomes faulty, the same process is repeated to achieve fault-tolerance. To handle the faults in more than one tiles at a time, the same process is repeated by considering faults in one tile until all the faulty tiles are covered. For executing multiple applications on the platform and achieving fault-tolerance for all of them, the same flow (Fig. 3) can be employed for all the applications.

Now, we discuss the simulation and analytical estimation strategy that is used for simulation and execution trace analysis of the current mapping, respectively.

A. Simulation Strategy

The simulation of the current mapping involves its throughput computation and execution trace capturing, which are briefed subsequently.

1) Throughput Computation: The throughput for a mapping is computed by taking the resource allocations of actors/edges on the platform into account. In order to compute the throughput, first, static-order schedule for each tile is constructed, which orders the execution of bound actors. Thereafter, all the binding and scheduling decisions are modeled in a graph called binding-aware SDFG. Finally, the throughput is computed by self-timed state-space exploration of the binding-aware SDFG [16]. Towards this, states visited during self-timed execution are examined and stored until a recurrent state is found. The throughput is computed from the periodic part of the state-space.

2) Execution Trace Capturing: The execution traces of actors and edges are captured based on their execution pattern for a given mapping during one periodic execution. For example, Fig. 4 shows execution pattern of actors and edges of H.263 decoder when each actor and edge is mapped on a different tile and connection between tiles, respectively. First, actor *VLD* fires (executes) as it has sufficient input tokens on its incoming edge e_4. Then, it generates 2376 tokens to be transferred through e_1 to process them one by one by *IQ*. The transfer of tokens through edges and their processing by different actors follows the shown trace. For easier realization, the shown trace considers 4 tokens in places of 2376 and thus actors *VLD, IQ, IDCT* & *MC* fire 1, 4, 4 & 1 times respectively during one period. The execution traces for each actor and edge is captured as the start and end time of their active executions (firings) in the whole period. For example, 4 active executions of actor *IQ* will get captured with different start and end times.

Algorithm 1: Analysis Strategy

Input: Execution trace for the current *mapping* μ using m tiles.
Output: Mappings & their throughput and energy consumption, using $(m-1)$ tiles.
Initialize the mapping set M, i.e., $M = \{\ \}$;
Select m tiles containing actor(s);
for *each unique pair of selected tiles* **do**
  Move actor(s) from one tile to another to generate a $new\ mapping\ \eta$ using $(m-1)$ tiles;
  Estimate $throughput$ & $energyConsumption$ of η;
  Add η with its $throughput$ & $energyConsumption$ to set M;
end

B. Analysis Strategy

The analysis strategy to perform the analytical estimation is presented in Algorithm 1. The strategy takes the updated execution traces of actors/edges for the current mapping μ using m tiles as input and estimates throughput and energy consumption of mappings using $(m-1)$ tiles when a tile becomes faulty. The strategy first selects m tiles containing actor(s). Then, for each unique pair of selected tiles, actors of one tile are moved to another to generate a new mapping η that uses $(m-1)$ tiles. For each new mapping, its throughput (1/period) and energy consumption are estimated and the mapping with its throughput and energy values is added to mapping set M.

Period (P) of the mapping using $(m-1)$ tiles (P_η) is estimated by utilizing period of the current mapping using m tiles (P_μ) by Equation 1, where $inc_{\mu,\eta}$ and $dec_{\mu,\eta}$ are the increase and decrease in the period of the mapping μ when the new mapping η is generated by moving actors from one tile to another in μ.

$$P_\eta = P_\mu + inc_{\mu,\eta} + dec_{\mu,\eta} \tag{1}$$

The period increases when parallel executing actors (e.g., *IQ* and *IDCT* in Fig. 4) mapped on selected pair of tiles in mapping μ are forced to execute sequentially by mapping the actors on the same tile in mapping η. The period decreases when execution of the edge(s) between the selected pair of tiles is not in parallel with other actors and edges (e.g., execution of edge e_1 in Fig. 4). The $inc_{\mu,\eta}$ is calculated by assuming sequential execution of the actors mapped on the selected pair of tiles. The non-parallel executions of the actors (with executions of other actors/edges) contribute to $inc_{\mu,\eta}$. The $dec_{\mu,\eta}$ is calculated by considering execution traces of edge(s) mapped between the selected pair of tiles. The non-parallel executions of the edge(s) (with executions of other actors/edges) contribute to $dec_{\mu,\eta}$.

Energy consumption (E) of a mapping ($energyConsumption$) is estimated as the sum of communication and computation energy for one iteration of the application. The *communication energy for each edge* (e) mapped to a connection (c) is estimated as product of the number of tokens (in bits) to be transferred through c, delay (D) and power consumption (P_{bit}) for transferring one bit through c. Total communication energy for all the edges is estimated from Equation 3. The number of tokens for an edge is computed as the product of repetition vector (*repV*) of source (or destination) actor and source (or destination) port rate (equation 2). The power required to transfer one bit is denoted as P_{bit} [2]. *Computation energy for each actor* (a) mapped to tile (t) is estimated as product of the number of executions of a ($repV[a]$), execution time ($ET[a]$) and power consumption (pow) on t. Total computation energy for

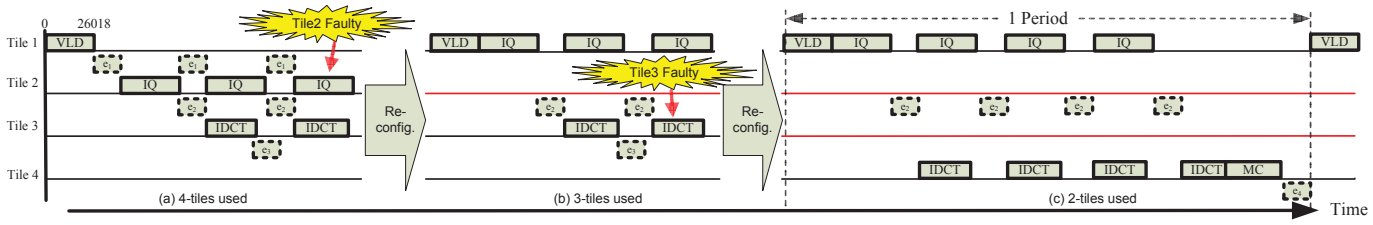

Fig. 5. Dynamic fault-tolerance demonstration.

all actors is estimated from equation 4. Power consumption on a tile is estimated as $C \times v^2 \times f$, where C, v and f denote average load capacitance, supply voltage and operating frequency, respectively.

$$nrTokens[e] = repV[e \rightarrow srcActor] \times (e \rightarrow srcPortRate) \quad (2)$$

$$E_{COMM} = \sum_{\forall e \rightarrow c} [\{nrTokens[e] \times tokenSize[e]\} \times D \rightarrow c \times P_{bit}] \quad (3)$$

$$E_{COMP} = \sum_{\forall a \rightarrow t} [repV[a] \times (ET[a] \rightarrow t) \times (pow \rightarrow t)] \quad (4)$$

In Algorithm 1, a total of m-choose-2 ($^mC_2$) unique pairs are found for the selected m tiles. Each unique pair provides a mapping that uses $(m-1)$ tiles. Out of all the mappings M using $(m-1)$ tiles, the proposed flow (Fig. 3) selects the mapping having maximum throughput/energy in order to jointly optimizing for through and energy consumption towards achieving the fault tolerance.

Example Demonstration

The proposed dynamic fault-tolerant flow has been applied onto the example H.263 decoder (Fig. 1) executing on 4-tiles periodically, as shown in Fig. 4. Fig. 5 (a) shows that a fault has occurred on tile 2 during a particular periodic execution. In order to achieve fault-tolerance, the proposed strategy finds the best mapping (having maximum throughput/energy) using 3 tiles. In the best mapping, the actor from the faulty tile (tile 2) is moved to some other tile. The platform is then reconfigured with the best mapping to start the application execution, as shown in Fig. 5 (b). Now, let a fault occur on tile 3, as shown in Fig. 5 (b). The proposed strategy moves the actor from faulty tile to a non-faulty tile to get the best mapping using 2 tiles. The best mapping contains actors *VLD* & *IQ* on tile 1 and *IDCT* & *MC* on tile 4. The platform is reconfigured with the current best mapping to start the periodic application execution, as shown in Fig. 5 (c).

V. Experimental Results

Experiments are conducted on a quad-core Intel Xeon 2.4GHz server running Linux with ten synthetic and ten real-life applications modeled as SDFGs. The synthetic applications are generated from the SDF^3 tool [17] with the number of actors ranging from 4 to 32. The real applications are derived from the benchmarks provided in the tool. These applications are executed on an MPSoC with six tiles arranged in 2×3 mesh architecture. All algorithms developed in the paper are coded in C++ and used in conjunction with the SDF^3 tool for throughput computation.

A. Throughput Comparison for Different Fault-Scenarios

The throughput obtained using the proposed trace-based dynamic reconfiguration technique is compared with the throughput obtained using the communication energy-aware static fault-tolerant technique of [6] (referred to as *TConCEMin*) and communication energy-aware dynamic technique of [2] (referred to as *DCEMin*). Further, to determine how far the proposed approach is from the highest throughput obtained for different fault-scenarios, the results are compared with the throughput maximization technique of [7] (referred to as *TMax*).

Figure 6 (a) & (b) plot the throughput of the *TConCEMin*, *DCEMin* and the proposed technique normalized with respect to the throughput obtained using the *TMax* technique for five real applications for single and double faults respectively. Single fault refers to fault in one PE and fault in 2 PEs is referred to as double fault. There are few trends to follow from these figures. First of all, the throughput of the *DCEMin* is the least amongst all the techniques. This is expected as *DCEMin* does not consider throughput degradation. Secondly, the throughput of the proposed fault-tolerant technique is better than the energy aware fault-tolerant technique, i.e. *TConCEMin* ([6]). This is expected because the proposed technique remaps the tasks from the faulty tile to other working tile(s) such that communication energy is minimized with the least degradation of throughput, facilitating for least actor migration. The static fault-tolerant technique of *TConCEMin* on the other hand determines one mapping which results in minimum communication energy satisfying the throughput requirement even at the expense of more actor migrations. On average for all single-fault scenarios, the proposed technique achieves 74% and 17% better throughput than *DCEMin* and *TConCEMin* technique respectively. For double-fault scenarios theses numbers are 94% and 20% respectively. Finally, the proposed technique is only within 2% and 3% of the highest throughput technique of *TMax* for single and double fault-scenarios respectively.

B. Energy Comparison for Different Fault-Scenarios

Figure 7 plots the energy consumption of the same four techniques for the same set of applications. The energy values are average for all single and double faults and are normalized with respect to the energy obtained using *TMax* technique. As can be seen from the figure, the energy of *DCEMin* is the least as communication energy is explicitly minimized in this technique without considering the throughput degradation. Secondly, the communication energy of the *TConCEMin* is better than the proposed technique. This is because *TConCEMin* uses integer linear programming to solve the minimum communication energy problem and is guaranteed to determine the optimum solution. The proposed technique on the other hand uses a heuristic to solve the same. On average for single and double faults, the proposed technique results in 15% and

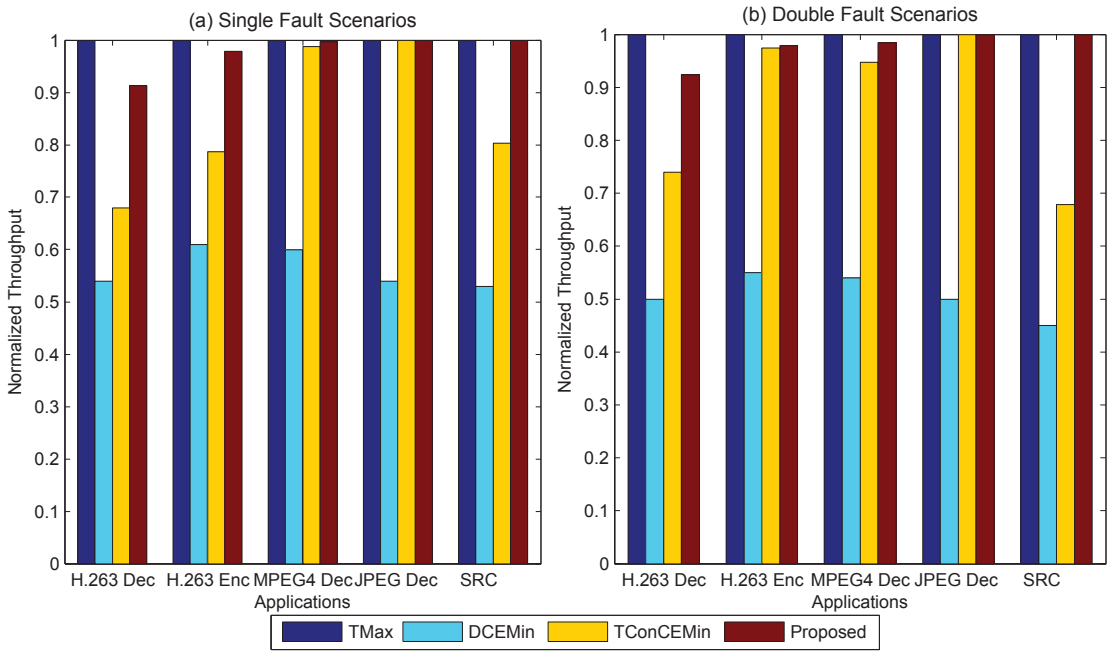

Fig. 6. Throughput Performance of the proposed technique

TABLE I. STORAGE REQUIREMENT WITH INCREASING TILES FOR A 3-FAULT-TOLERANT SYSTEM

Tiles	*TConCEMin* [6]	Proposed
4	4.6	1
8	32.8	1.2
12	92.4	1.3
16	187.5	1.4
20	320.7	1.6
24	494.3	1.8
28	709.8	1.9
32	968.7	2.0

30% lower energy than the static technique of *TMax*. The energy of the proposed heuristic is within 10% of the minimum energy of *TConCEMin*.

The important conclusion to make from these results is that, the proposed technique maximizes throughput (by average 74% and 80% for single and double faults respectively) in comparison with the existing dynamic techniques with less than 10% degradation of energy as compared to existing static throughput-aware fault-tolerant techniques.

C. Storage Overhead Performance

Table I shows the storage requirement of the proposed dynamic technique in comparison to the static technique of [6] for different number of tiles. The number of actors for the application is the same as the number of tiles. Synthetic applications with different number of actors are used for different number of tiles. As can be seen from the table and also expected, the static technique requires significant storage overhead. This is due to the fact that the static technique evaluates and stores the application mapping and scheduling for all processor fault-scenarios. On the other hand, the proposed dynamic approach derives the schedule for a fault-scenario from a master execution trace as and when faults occur. Thus, the storage overhead associated with the dynamic technique is only that required to store the master execution trace. This is shown in column 3 of the table.

Thus, the proposed technique achieves significant savings in storage which is crucial for multimedia applications where the storage space is limited.

TABLE II. ALGORITHM EXECUTION TIME (MS) FOR SINGLE-FAULT SCENARIO FOR DIFFERENT APPLICATIONS

Applications	*TConCEMin* [6]	Proposed
H.263 decoder	2055.20	4.81
H.263 encoder	2124.61	109.00
sample rate converter	17922.48	3.80

D. Algorithm Execution Time

Table II shows the execution time of different approaches to find mappings in the case of single fault scenario for different multimedia applications. The initial mapping for each application has been assumed to use the same number of tiles as the number of actors in the application. Therefore, for each application, the approaches need to evaluate mappings using one less number of tiles. Both the approaches evaluate the same number of mappings for the single fault scenario. The number of mappings is n-choose-2 (${}^{n}C_2$), where n is the number of actors in the application. However, the static technique of [6] employ simulative evaluations and the proposed approach employ analytical estimations. As can be seen from the table and also expected, the static technique needs large time compared to the proposed technique. The different in execution times is observed due to simulative and analytical evaluations. In case of other fault scenarios such as double-fault, similar results are obtained. Thus, the proposed technique reduces the execution time significantly, which is essential to take rapid decisions at run-time.

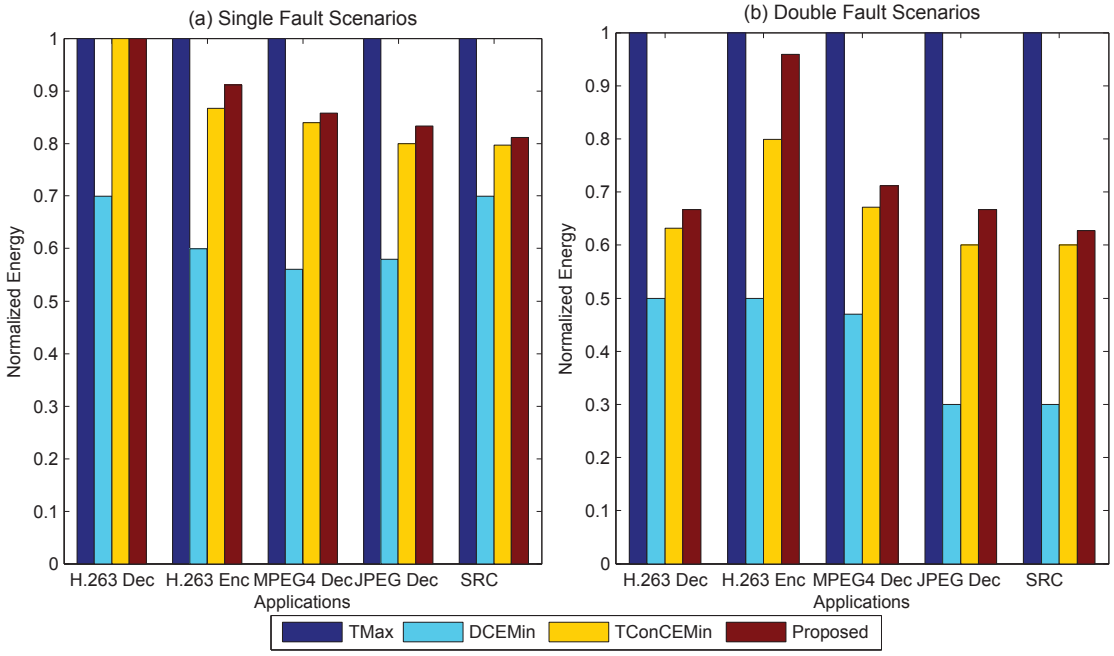

Fig. 7. Energy Performance of the proposed technique

VI. Conclusions and Future Work

This paper proposes an execution trace based run-time technique to minimize the communication energy and throughput degradation of applications for different processor fault-scenarios. Experiments conducted with applications modeled as Synchronous Data Flow Graphs clearly indicate that the proposed technique provides significant throughput improvement (average 74% and 80% for single and double faults respectively) with respect to the existing dynamic technique with less than 10% deviation in communication energy obtained with an ILP-based technique. Processor heterogeneity and task computation energy minimization are left as future works.

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] W. Wolf, "Multimedia applications of multiprocessor systems-on-chips," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2005.

[2] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2004.

[3] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms," *Elsevier Journal of Systems Architecture (JSA)*, vol. 56, no. 7, pp. 242–255, 2010.

[4] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in *ACM Design Automation Conference (DAC)*, 2004.

[5] I. Koren and C. Krishna, *Fault-tolerant systems*. Morgan Kaufmann, 2007.

[6] A. Das, A. Kumar, and B. Veeravalli, "Energy-Aware Communication and Remapping of Tasks for Reliable Multimedia Multiprocessor Systems," in *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2012.

[7] C. Lee, H. Kim, H. Park, S. Kim, H. Oh, and S. Ha, "A task remapping technique for reliable multi-core embedded systems," in *IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*, 2010.

[8] O. Derin, D. Kabakci, and L. Fiorin, "Online task remapping strategies for fault-tolerant Network-on-Chip multiprocessors," in *IEEE/ACM Symposium on Networks on Chip (NoCS)*, 2011.

[9] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[10] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on Multi/Many-Core Systems: Survey of Current and Emerging Trends," in *ACM Design Automation Conference (DAC)*, 2013.

[11] C. Yang and A. Orailoglu, "Predictable execution adaptivity through embedding dynamic reconfigurability into static MPSoC schedules," in *IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*, 2007.

[12] J. Huang, J. Blech, A. Raabe, C. Buckl, and A. Knoll, "Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems," in *IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*, 2011.

[13] A. Das and A. Kumar, "Fault-aware task re-mapping for throughput constrained multimedia applications on noc-based mpsocs," in *IEEE International Symposium on Rapid System Prototyping (RSP)*, 2012.

[14] C.-L. Chou and R. Marculescu, "FARM: Fault-aware resource management in NoC-based multiprocessor platforms," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, march 2011, pp. 1 –6.

[15] L. Ost, M. Mandelli, G. M. Almeida, L. Moller, L. S. Indrusiak, G. Sassatelli, P. Benoit, M. Glesner, M. Robert, and F. Moraes, "Power-aware dynamic mapping heuristics for noc-based mpsocs using a unified model-based approach," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 3, pp. 75:1–75:22, 2013.

[16] A. H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. R. Mousavi, "Throughput Analysis of Synchronous Data Flow Graphs," in *IEEE Conference on Application of Concurrency to System Design (ACSD)*, 2006, pp. 25–36.

[17] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *IEEE Conference on Application of Concurrency to System Design (ACSD)*, 2006. [Online]. Available: http://www.es.ele.tue.nl/sdf3.