Energy-Aware Communication and Remapping of Tasks for Reliable Multimedia Multiprocessor Systems

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

***Abstract*—Shrinking transistor geometries, aggressive voltage scaling and higher operating frequencies have negatively impacted the dependability of embedded multiprocessor systems-on-chip (MPSoCs). Fault-tolerance and energy efficiency are the two most desired features of modern-day MPSoCs. For most of the multimedia applications, task communication energy constitutes more than 40% of the overall application energy. In this paper, an integer linear programming (ILP) based approach is proposed to reduce the communication energy and fault-tolerant migration overhead of throughput-constrained multimedia applications modeled using synchronous data flow graphs (SDFGs). The ILP is solved at compile-time for all fault-scenarios to generate task-core mappings satisfying an application throughput requirement. These mappings are stored in a table which is looked up at run-time as and when faults occur. Experiments conducted with real and synthetic applications demonstrate that the proposed technique reduces communication energy by an average 40% and migration overhead by 33% as compared to the existing fault-tolerant techniques.**

***Keywords*-Fault-Tolerance; Task Mapping; Communication Energy; Synchronous Data Flow Graph; Linear Programming;**

I. Introduction

To accommodate the ever increasing demands of applications and for the ease of scalability, multiprocessor systems-on-chip (MPSoCs) are becoming the obvious design choice in current and future technologies with streaming multimedia applications constituting a large fraction of the application space [1]. With reducing feature size and increasing transistor count, MPSoCs are becoming susceptible to permanent and transient faults [2]. This research focuses on permanent fault-tolerant techniques.

Permanent faults are traditionally tackled using hardware redundancy [3]. However, stringent area and power budgets are prohibiting the use of hardware redundancy in modern systems. Software fault-tolerant techniques like task-migration are gaining popularity among research community [4]–[9]. Task migration involves remapping of tasks from a faulty core to other functional cores by moving the task's code and data memory (referred to hereafter as *state space*).

Most modern day MPSoCs consist of cores interconnected with networks-on-chip (NoCs) in a mesh-based architecture[1]. Multimedia applications mapped to these MPSoCs are typically executed multiple times in a periodic fashion with the average number of iterations per unit time determining the throughput. When one or more cores of an MPSoC become faulty, the MPSoC is unavailable during the task remapping time. The new location (core) for a task on a faulty core is pivotal in determining the energy consumption associated with communication among its dependent tasks. Tasks migrated further away from their dependent tasks will consume more energy per iteration of the application graph.

Multimedia applications are characterized by fixed throughput requirement, violation of which directly impacts user experience. The task migration objective for multimedia applications can therefore be summarized as reduction of migration overhead and communication energy while satisfying the application throughput requirement. Most of the fault-tolerant task-migration research works have focused on minimizing migration overhead [5] and load balancing [7] or maximizing the reliability of a system [4]. These techniques provide no guarantee on the application throughput making them unsuitable for multimedia systems. Recently, there is a study to maximize throughput under faulty scenarios [9]. Migration overhead and task communication cost are not accounted and therefore can increase significantly in this technique. On the power minimization research direction, there are works focusing on energy-aware scheduling techniques [10]–[14]. However, either they do not address task movement under faulty scenarios or the joint consideration of throughput, energy and migration overhead are lacking.

Contributions: The focus of this paper is on software technique for tolerating permanent faults in homogeneous multimedia multiprocessor systems with applications modeled using Synchronous Data Flow Graphs (SDFGs) [15]. Key contributions of this paper are the following.

- Communication energy aware fault-tolerant task mapping of throughput constrained multimedia applications
- Integer linear programming (ILP) based minimization of migration overhead, throughput degradation and communication energy
- Consideration of hop-count in the migration overhead computation

The ILP is solved at compile-time to determine task-core mappings for all fault scenarios. These mappings are stored in a table for lookup at run-time as and when faults occur. Experiments conducted with synthetic and real application graphs demonstrate that the proposed fault-tolerant technique minimizes communication energy by 40% with a reduction of 33% in migration overhead as compared to the existing techniques [6] [9].

[1]While a mesh-based topology is assumed for the target MPSoC, the research is orthogonal to any other topologies such as torus and tree

The rest of the paper is organized as follows. A brief overview of the prior art is provided in Section II followed by a motivating example in Section III to emphasize the importance of this work. SDFG is introduced next in Section IV. Task communication energy and migration overhead are modeled in Section V & VI respectively. ILP formulation is discussed in Section VII and the proposed methodology in Section VIII. Section IX provides simulation results and Section X concludes the paper with future directions.

II. Related Works

The need for dependable and energy-efficient designs for battery-powered MPSoCs have led to two research directions on task mapping – fault-tolerance and energy minimization. Recently, efforts are made towards joint optimization of energy and fault-tolerance. This section provides some key initiatives for each of these research directions.

A. Fault-tolerant task mapping and scheduling

The existing fault-tolerant research can broadly be classified into two categories – architecture level and application level. A popular architecture level fault-tolerant technique is to replicate critical design components [3]. Stringent area and power budget are increasingly prohibiting the use of redundancy based designs for MPSoCs.

One of the widely used application level fault-tolerant techniques is task migration. Task remapping decisions can be pre-computed at compile-time (analyzing all possible fault-scenarios) or can be decided at run-time as and when faults occur. Accordingly, task migration can be categorized as static and dynamic. Dynamic approaches monitor system-status and decide to migrate tasks at run-time to minimize migration overhead [5] [6] or balance processor load [7]. A limitation of these techniques is that throughput is not always guaranteed. Moreover, migration algorithms need to be simple to minimize computation overhead.

Static task migration techniques compute task mapping decisions at compile-time for different fault-scenarios [8], [9]. As faults occur, these mappings are looked up at run-time to carry out task-migration. An advantage of these techniques is that any sophisticated algorithm can be used at compile-time albeit the storage overhead.

B. Energy-aware task mapping and scheduling

To accommodate the ever increasing demand of performance and features in MPSoCs, energy budget is becoming more stringent. Researchers have focused on every aspect of energy reduction techniques. These techniques can be classified into circuit-level approaches (power gating, for example) and software approaches such as energy-aware scheduling where tasks are scheduled on cores to minimize the overall energy consumption.

Recently, software approaches are gaining a lot of interest among research community. A dynamic voltage scaling technique is proposed in [10] to minimize the energy consumption. The slack budgeting technique of [11] distributes execution time slack of a task among other tasks, to reduce their frequency of operation. A gradient-based energy minimization is proposed in [12]. However, none of these research works address the task movement from faulty cores.

C. Energy-reliability joint optimization

In recent years, quite some efforts have been made towards joint optimization of fault-tolerance and energy. An ILP based approach is presented in [13]. Energy optimization is performed under the constraint of task-execution time which incorporates fault-tolerance overhead using checkpointing based recovery model. This technique is not suitable for permanent failures as it does not address the actual task migration under different faulty-scenarios. Moreover, throughput, communication energy and migration overhead are not addressed in this paper.

A lifetime-reliability aware scheduling technique is developed in [14] to minimize energy consumption. Tasks are scheduled on processors equipped with dynamic voltage scaling (DVS) capabilities. However, throughput is not guaranteed in this technique either.

To summarize, none of the existing techniques address minimization of throughput degradation, communication energy and migration-overhead simultaneously.

III. Motivation

Most streaming multimedia applications, like H.263 decoder, demand fixed throughput which is manifested as the quality-of-service (QoS) requirement. The importance of migration cost for multimedia applications has already been established [16]. In this section, examples are provided to signify the importance of hop distance for migration overhead and task communication energy.

A. Importance of hop distance

Figure 1(a) shows a synthetic application with 9 tasks mapped on a target architecture with 6 cores. The no-fault task mapping is shown in Figure 1(b) with the number in parenthesis against each task indicating the size of its *state space*. Figure 1(c) and 1(d) show two different task mappings satisfying the application throughput requirement with core c_3 as faulty. The migration overhead for Figure 1(c) involves migrating 180 units of *state space* of task F from core c_3 to core c_0 through one hop and 120 units for task I from core c_3 to core c_1 through two hops. Thus, the total overhead is $180 + 2 \times 120 = 420$ units. The migration overhead for Figure 1(d) are 180 units and 120 units through one hop each for task F and I respectively. The total overhead is therefore $180 + 120 = 300$ units. If only *state space* is considered for migration, the two configurations 1(c) and 1(d) are equally good to be selected (*state space* are 300 units each). However, selecting 1(c) results in 40% extra migration overhead in reality than 1(d) due to extra hops.

B. Importance of task communication energy

A multimedia application (MP3 decoder for example) is characterized by periodic execution of its constituent tasks. The energy associated with data communication among these tasks contributes to $\approx 40\%$ of the total dynamic energy of the application. With reference to Figure 1(a), the directed arcs of the graph indicate producer-consumer relationship. As an example, the data produced by source task A is consumed by B; the data produced by B is consumed by both C and D and so on. When tasks are mapped to cores,

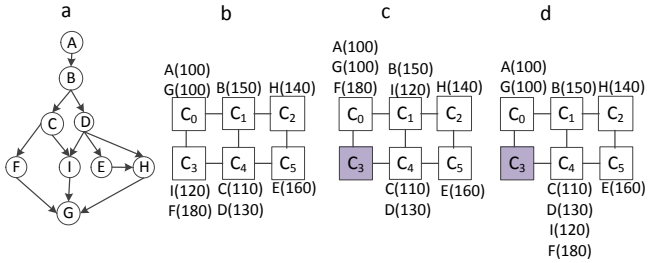

Figure 1. Importance of hop-count and communication energy

Table I
COMMUNICATION ENERGY ESTIMATE

Links	Hop Distance		Token size of	Comm. Energy	
	cfg 1(c)	cfg 1(d)	source task	cfg 1(c)	cfg 1(d)
$C-F$	2	0	32	$64E_{bit}$	0
$C-I$	1	0	32	$32E_{bit}$	0
$D-I$	1	0	64	$64E_{bit}$	0
$F-G$	0	2	64	0	$128E_{bit}$
$I-G$	1	2	64	$64E_{bit}$	$128E_{bit}$
Total				$224E_{bit}$	$256E_{bit}$

the data produced from a task on a source core need to be communicated to a task on the sink core. If the producer and the consumer tasks are mapped to the same core, there is no communication energy involved. However, if the cores of the source and the sink tasks are different, energy is required to transfer every bit of data between these cores [17].

Figure 1(c,d) represent two task mappings obtained from Figure 1(b) with core c_3 as faulty. There are five producer-consumer relations that are affected due to task remapping: $C-F$, $C-I$, $D-I$, $F-G$ and $I-G$. In Figure 1(c), the two tasks F and I of core c_3 are mapped to cores c_0 and c_1 respectively. In Figure 1(d) however, the two tasks F and I are both mapped to core c_4. Table I shows the hop count (distance), the data communicated and the energy associated for the two mappings. E_{bit} is energy required to communicate every bit of information. The energy of communication is directly proportional to the hop distance and the data communicated (referred to hereafter as $token$).

From the table it can be concluded that although, the computation energy remains unaltered due to homogeneous platform, extra communication energy is incurred in every iteration of the graph if 1(d) is selected as the mapping after core c_3 becomes faulty.

IV. SYNCHRONOUS DATA FLOW GRAPH (SDFG)

Synchronous Data Flow Graphs (SDFGs, see [15]) are often used for modeling modern DSP applications and for designing concurrent multimedia applications implemented on a multi-processor 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,

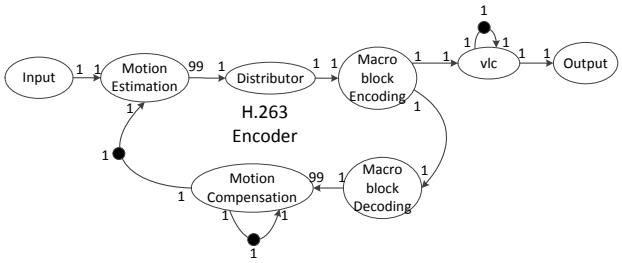

Figure 2. H.263 Encoder

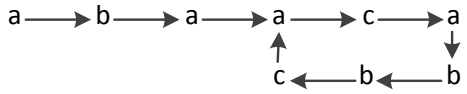

Figure 3. Self-timed execution

denoted with a number in the actors. The edges in the graph, called $channels$, represent data that is communicated.

Figure 2 shows the SDF Graph of H.263 encoder. There are eight actors in this graph. In the example, actor $motion\ estimation$ has rate of 1 on all input edges and a rate of 99 on its output edge. 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 2. 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.

When an actor $\boldsymbol{a}$ starts its firing, it removes $Rate(q)$ tokens from all $(p,q) \in InC(a)$. The execution continues for $\tau(\boldsymbol{a})$ time units and when it ends, it produces $Rate(p)$ tokens on every $(p,q) \in OutC(\boldsymbol{a})$.

Self-timed execution (ref [18]) is widely used for scheduling SDFG. Static-schedules are constructed using worst-case actor execution times at compile-time. The actor ordering on tiles are retained discarding the timing information. At run-time, actors are fired maintaining the same order as determined at compile-time. In this respect the following lemmas are stated. For proof, readers are urged to refer [18].

Lemma 1: For a consistent and strongly connected SDFG, the self-timed execution consists of a transient phase followed by a periodic phase.

Lemma 2: For a consistent and strongly connected SDFG, the throughput of an actor is given by the average firing of the actor per unit time in the periodic phase of the self-timed execution.

V. MODELING COMMUNICATION ENERGY

Energy modeling for NoC-based MPSoCs has received significant attention in recent times. In [17], bit energy (E_{bit}) is defined as the energy consumed when one bit of data is communicated through the routers and links of a NoC.

$$E_{bit} = E_{S_{bit}} + E_{L_{bit}} \quad (1)$$

where $E_{S_{bit}}$ and $E_{L_{bit}}$ are the energy consumed on the switch and the link respectively. The energy per bit con-

$G(A, C)$	= Given SDFG	t	= Number of actors
S_i	= Self timed scedule for core i	p	= Number of tiles
$trans(S_i)$	= Transient phase of S_i	F	= Level of fault-tolerance desired
$steady(S_i)$	= Periodic phase of S_i	τ	= Application throughput constraint
$actors(trans(S_i))$	= Set of actors in the transient phase of S_i	M_t^{p-f}	= Set of mappings of t actors on $p-f$ tiles
$actors(steady(S_i))$	= Set of actors in the periodic phase of S_i	M	= Complete set of mappings $= \{M_t^p, M_t^{p-1}, \cdots, M_t^{p-F}\}$
$rpt(trans(S_i), a)$	= Number of firing of actor a in the transient phase of S_i	$m_t^{p-f}(i)$	= a mapping $\in M_t^{p-f}$
$rpt(steady(S_i), a)$	= Number of firing of actor a in the periodic phase of S_i	m_t^p	= initial no-fault mapping
$connect(a)$	= Set of tasks dependent on task a	T	= Set of throughputs corresponding to mapping $m \in M$
$map(a, b)$	= Binary variable takes on the value 1 when tasks a and b are mapped to different tiles and is 0 otherwise	$(e_0, e_1, \cdots, e_f)$	= f-fault-scenario with first fault occurring at tile e_0 second at tile e_1 and so on

Table II
NOTATIONS AND LEGENDS USED IN THIS PAPER

sumed in transferring data between tile i and tile j, situated $n_{hops}(i, j)$ away is given by Equation 2 according to [11].

$$E_{bit}(i, j) = n_{hops}(i, j) \times E_{S_{bit}} + (n_{hops}(i, j) - 1) \times E_{L_{bit}} \quad (2)$$

As has been established in Section IV, the self-timed execution of SDFG consists of transient and periodic phase where the transient phase is executed once while the periodic phase is executed repeatedly. Figure 3 shows an example self-timed execution of 3 actors – ***a***, ***b*** and ***c*** on a core.

When actor ***a*** on tile t_a fires, the total number of tokens produced on channel p in one iteration is $Token(p) \times Rate(p)$. If channel p connects to another actor ***b*** on tile t_b, then the energy (E_a^b) of communicating tokens from actor ***a*** to actor ***b*** per iteration is given by Equation 3.

$$E_a^b = E_{bit}(t_a, t_b) \times (Token(p) \times Rate(p)) \times TokSize \quad (3)$$

where $TokSize$ is the token size (in bits) communicated. However, if actors ***a*** and ***b*** are mapped to the same tile, then $E_a^b = 0$. Using notations from Table II, the periodic energy per iteration of a schedule S_i of mapping m is given by

$$E_{steady}(S_i) = \sum \left(rpt(steady(S_i), a) \sum E_a^b \right) \quad (4)$$

where the outer sum is evaluated $\forall a \in actors(steady(S_i))$ and the inner sum $\forall b \in connect(a)$. In a similar way, the transient energy is formulated with $steady$ replaced by $trans$. The overall communication energy of mapping m is

$$E_{comm}(m) = \sum_{\forall i} (E_{trans}(S_i) + n * E_{steady}(S_i)) \quad (5)$$

where $i \in [1, 2, \cdots, p]$, p is the number of tiles and n is the number of iterations of the periodic phase of the application graph. Usually, the number of periodic iterations is a large number (can be regarded as periodic decoding of every frame for a video application) and hence for all practical purposes, the energy of the periodic phase dominates over the transient energy. Throughout the rest of this paper, communication energy will imply energy of the periodic phase (E_{steady}) per iteration.

VI. MODELING MIGRATION OVERHEAD

The migration overhead of an actor is measured by the size of its *state space* times the hop distance traveled during migration. In order to couple this with the actor communication energy, the migration overhead is converted to energy overhead. Specifically, if $src(a)$ is the tile for actor ***a*** before fault occurrence and $dst(a)$ is its new location after fault, the energy associated with this migration is given by Equation 6.

$$\begin{aligned} E_{mig}(a) = &\ StateSpace(a) * E_{bit}(src(a), dst(a)) \\ \approx &\ StateSpace(a) * n_{hops}(src(a), dst(a)) * E_{bit} \end{aligned} \quad (6)$$

From Equation 6, it is clear that minimizing the migration overhead is equivalent to minimizing the migration energy.

VII. PROBLEM FORMULATION

This section formulates the minimization of migration overhead considering one of the tiles to be faulty. Later in Section VIII, details are provided on the integration of this formulation with the actor communication cost to form the proposed fault-tolerant methodology.

Given:

- Application SDFG $G(A, C)$ ($|A| = t$)
- Mapping of t actors on k tiles m_t^k ($k \leq p$)
- New mapping of t actors on $k-1$ tiles m_t^{k-1}
- Faulty tile ID f

Objective:

Minimize migration overhead (equivalently migration energy) in moving from m_t^k to m_t^{k-1}.

Simplification:

In order to simplify the objective, we form an energy matrix (Table III) with tiles from mapping m_t^k forming the rows (indicated by o_i) and the tiles from m_t^{k-1} forming

Table III
ENERGY TABLE

$\mathbf{EM(o_i, n_j)}$	$\mathbf{n_1}$	$\mathbf{n_2}$	$\mathbf{n_3}$	$\cdots$	$\mathbf{n_f}$	$\cdots$	$\mathbf{n_{k-1}}$
o_1	50	205	180	$\cdots$	0	$\cdots$	175
o_2	200	100	180	$\cdots$	0	$\cdots$	200
o_3	200	175	130	$\cdots$	0	$\cdots$	125
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
o_f	0	0	0	$\cdots$	$\cdots$	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
o_k	165	110	120	$\cdots$	0	$\cdots$	135

the columns (indicated by n_j). The rows (and columns) corresponding to the unused tile(s) and the fault ID f are filled with zeroes. The non-zero entries (o_i, n_j) of the energy matrix (referred hereafter as EM) correspond to the migration energy associated with the extra actor(s) on tile n_j of m_t^{k-1} which is (are) not present on tile o_i of m_t^k. This is computed as follows.

$$actors(o_x, k) = \text{actors mapped on tile } o_x \text{ of } m_t^k$$
$$actors(n_x, k-1) = \text{actors mapped on tile } n_x \text{ of } m_t^{k-1}$$
$$EM(o_i, n_j) = \sum_{\substack{\forall a \in actors(n_j, k-1) \\ a \notin actors(o_i, k)}} E_{mig}(a)$$

ILP Formulation:
Base Variables: $X_{ij},\ i \in [1,k],\ j \in [1,k-1]$
Objective: Minimize z = $\sum_{ij} X_{ij} \times EM(o_i, n_j)$
Constraints:
One element from each row and column is to be selected

$$\sum_{j=1}^{k-1} X_{ij} := 1,\ \forall i \in [1,k],\ \sum_{i=1}^{k} X_{ij} := 1,\ \forall j \in [1,k-1] \quad (7)$$

VIII. FAULT-TOLERANT METHODOLOGY

Figure 4 gives an overview of the proposed technique. The methodology is split into two phases – compile-time analysis phase and run-time execution phase. At compile-time, analysis is performed for every fault-scenario to determine a set of mappings which give minimum communication energy. Migration overhead for these mappings is computed using the ILP technique of Section VII. The mapping with minimum migration overhead is selected. Thus, for every fault-scenario, a mapping is determined. These mappings are stored in a table for use at run-time. The proposed methodology prioritizes communication energy over migration overhead based on the fact that migration overhead is incurred only when faults occur, but communication energy is consumed in every iteration of the application.

The pseudo-code of the proposed methodology is presented in Algorithm 1. The first step is the generation of the mapping set M and the throughput set T (line 1-2)[2]. The mapping set is pruned to retain those mappings which satisfy the application throughput constraint (line 3). Following this, there are F iterations (line 4-24). At each iteration f, mappings are selected for every fault-scenario

[2]Throughput for a mapping is computed using the SDF^3 tool [19].

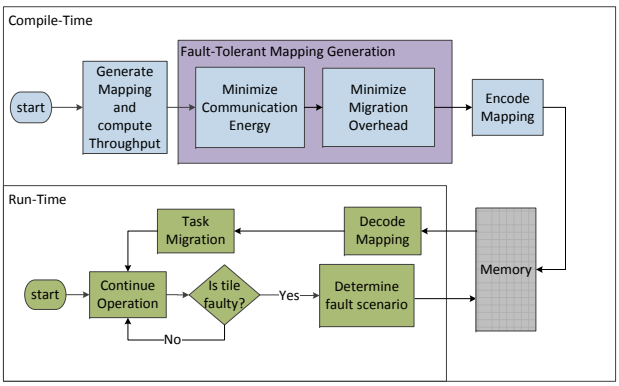

Figure 4. Design Methodology

with f tiles as faulty. These mappings (stored in $HashMap$ data-structure) give the lowest communication energy and migration overhead satisfying the throughput constraint τ.

In every iteration of the outer for loop (line 4-24), the mapping set M_t^{p-f} is further pruned to retain mappings with minimum communication energy (line 7). A set of fault-scenarios are then determined (line 8). As an example, the set of 3-fault scenarios for an architecture with 4 tiles are $\{(0,1,2),(0,1,3),(0,2,3),\cdots\}$. For every fault-scenario of this set, a reduced scenario is generated along with the current fault ID (line 11-12). A reduced scenario of $(e_0, e_1, \cdots, e_f)$ is $(e_0, e_1, \cdots, e_{f-1})$ with e_f as the current fault-ID. It is to be noted that the mapping set for the reduced scenario is already obtained in the previous iteration and is stored in the table. This mapping is fetched from the $HashMap$ (line 14) (referred to as m_t^{p-f+1}). For each mapping $m_t^{p-f}(i) \in M_t^{p-f}$, the ILP is solved (line 16) to obtain the least migration energy. The mapping which gives the minimum migration energy is stored in $HashMap$ for the particular fault-scenario. For the ease of representation, a linearization technique is applied where each mapping m_i is represented by a tuple $(t_i^0, t_i^1, .., t_i^{s-1})$ where, t_i^k is the tile to which actor a_k is mapped. A mapping ID (mID_i) is assigned to mapping m_i which is calculated as

$$mID_i = \sum_{j=0}^{t-1} t_i^j \times p^j \quad (8)$$

IX. EXPERIMENTAL RESULTS AND DISCUSSIONS

This section provides an overview of the computations performed and the complexity of the proposed algorithms. Simulation experiments are conducted with synthetic and real application graphs on a quad-core Intel Xeon 2.4GHz server running Linux. Algorithms are implemented in C++ integrated with Matlab and used in conjunction with the SDF^3 [19] tool for throughput computation. The ILP is solved using Matlab optimization toolbox.

As established in Section II, there are three categories of research relevant to this work – throughput maximization, migration overhead minimization and energy minimization. The proposed technique is compared with the representative from each of these categories i.e. with throughput maximization technique of [9] (referred to as TMax), migration overhead minimization technique of [6] (referred

Algorithm 1 Determine fault-tolerant mappings

Input: m_t^p, $SDFG(A, C)$, τ, F, HashMap (H)
Output: minimum energy mappings satisfying throughput constraint for $f = 1$ to F faults

Determine M
$T := SDF^3\_getThroughput(M)$
$\forall m \in M$, if $T(m) \geq \tau, \hat{M}.add(m)$
for $f \in [1, F]$ **do**
  $M_t^{p-f} :=$ mappings (of set $\hat{M}$) with f less tiles
  $\forall m \in M_t^{p-f}$, determine $E_{comm}(m)$
  $\Omega_t^{p-f} :=$ mappings (of set M_t^{p-f}) with min. energy
  $S^f := genFaultScenarios(f)$
  for all $s_f \in S^f$ **do**
    $mapEnergy := \infty$, $map = NULL$
    $(e_{i_1}, e_{i_2}, \cdots, e_{i_f}) := s_f$
    $s_{f-1} := (e_{i_1}, e_{i_2}, \cdots, e_{i_{f-1}})$
    $fID := e_{i_f}$
    $m_t^{p-f+1} := HashMap[s_{f-1}].getMap()$
    for all $m_t^{p-f}(i) \in \Omega_t^{p-f}$ **do**
      $E_{mig} := solveILP(m_t^{p-f+1}, m_t^{p-f}(i), fID)$
      if $E_{mig} \leq mapEnergy$ **then**
        $mapEnergy := E_{mig}$
        $map := m_t^{p-f}(i)$
      end if
    end for
    $HashMap[s_f].setMap(map)$
  end for
end for

to as OMin) and energy minimization technique of [11] (referred to as EMin). Although EMin does not address fault-tolerance, results are compared with it to determine how far our approach is from the minimum energy possible with the application on the given architecture (without considering throughput degradation and fault-tolerance). Our approach is referred as Throughput constraint COmmunication Energy and Migration overhead minimization (TCOEM).

A. Migration cost and communication energy performance

Figure 5 plots the average communication energy of TCOEM for single and double faults of synthetic applications with varying actors on an architecture with 8 tiles. An application is represented by *App_j*, where j is the number of actors. TMax is the only approach which considers throughput (suitable for multimedia applications) and therefore our approach is compared with TMax. Further, to signify the potential energy savings possible, TMax technique is split into two categories – one resulting in maximum communication energy (TMax_EMax) and one resulting in minimum communication energy (TMax_EMin). The communication energy of all three techniques are normalized with respect to the computation energy for an application. The percentage change of TMax_EMin and TCOEM with respect to TMax_EMax and TMax_EMin respectively are indicated on the bars. Although, not explicitly captured in the figure, the communication energy (of TMax_EMax) for the applications constitute on average 55% of the total energy.

As can be seen from Figure 5, communication energy

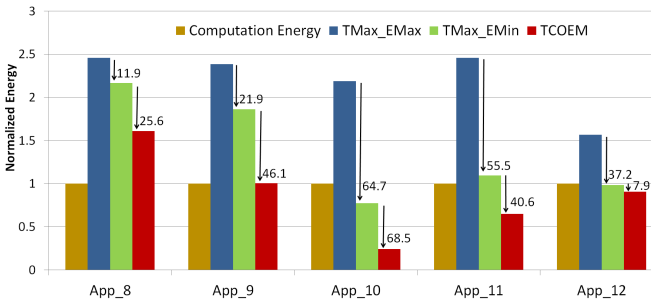

Figure 5. Energy savings for different applications

Table IV
MIGRATION ENERGY PERFORMANCE OF TCOEM

		Migration Energy (nJ)	Comm. Energy (nJ)
App_8	EMin_OMax	187,836	9,589
	TMax_OMin	2,406	13,876
	TCOEM	1,575	8,868
	OMin	1,402	18,153
App_10	EMin_OMax	192,305	1,257
	TMax_OMin	3,581	7,651
	TCOEM	2,418	1,221
	OMin	1,559	56,967

of TMax_EMin is 38% less on average, as compared to TMax_EMax. Clearly, any mapping which does not consider communication energy can have this extra energy overhead per iteration of the application graph. This justifies the consideration of communication energy for fault-tolerant task remapping of throughput-constrained multimedia applications. Additionally, our TCOEM approach reduces the communication energy further by exploring all mappings (including the highest throughput mappings) which satisfy application throughput requirement. The energy gain of TCOEM for T_10 is more than 60% with respect to TMax_EMin. This is because for this application, the highest throughput mapping and the least energy mapping points are significantly different in terms of energy. On the other hand, for T_12, these two mappings are closer and therefore the energy gain is $\approx 8\%$. On average, the proposed TCOEM approach achieves energy savings of 40% with respect to TMax_EMin. In terms of the total energy (communication + computation), this saving is around 20%.

Table IV reports the migration overhead of our approach for two different applications with 8 and 10 actors on an architecture with 8 tiles. The migration overhead is measured in terms of energy needed to transfer the state-space of the actors that are migrated. Our approach is compared with OMin to identify how far our approach is from the minimum migration overhead point. Like in the case of energy, TMax technique can be split based on migration overhead; however, TMax with minimum overhead (TMax_OMin) is only included for comparison. Additionally, to signify the potential migration overhead incurred with energy minimization alone, our approach is also compared with EMin, which results in highest migration overhead satisfying throughput requirement (EMin_OMax). An important point to note is that TCOEM is fundamentally EMin followed by overhead minimization (i.e. EMin_OMin) with throughput requirement. However, the naming convention for our approach is

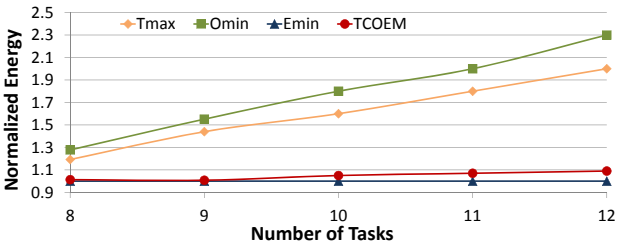

Figure 6. Communication energy with varying number of actors

retained for consistency.

As can be seen from the table, TCOEM achieves the minimum migration and communication energy among TMax_OMin and EMax_OMax. On average, TCOEM achieves 99% less migration energy with respect to EMin_OMax. Thus, minimizing communication energy alone can result in substantial migration overhead. Moreover, TCOEM reduces migration energy by 33% (on average) with respect to TMax_OMin signifying the importance of our approach for fault-tolerant task migration. In comparison with OMin (minimum migration technique without considering throughput), Table IV shows that our approach incurs an extra migration energy of $\approx 860nJ$ for App_10. However, the communication energy savings per iteration of the application graph is $\approx 55,000nJ$. Thus, the energy lost in migration due to fault occurrence is recovered within the next iteration of the application graph. Similar result is obtained for App_8. These results verify the assumption that communication energy savings is more critical than migration overhead for multimedia applications.

B. Energy gain with varying problem size

Figure 6 plots the average communication energy of the four techniques (TMax, OMin, EMin and TCOEM) for single and double fault scenarios with varying number of actors. The reference architecture consists of 8 homogeneous tiles and the energy numbers are normalized with respect to the EMin technique. As can be seen from the figure, the energy overhead of all the fault-tolerant techniques (TMax, OMin and TCOEM) increases as the number of actors increases. This is expected due to the increase in the data traffic with increase in the number of actors. An important trend to follow from this figure is that, although the energy consumption of TCOEM increases with the number of actors, the growth is slow and is close to the optimal energy savings obtained using EMin. Energy consumption of the other two techniques grows rapidly with the number of actors. With 12 actors mapped on 8 cores, TCOEM has an energy overhead of 10% whereas for TMax and EMin, overheads are more than 100%. These results clearly demonstrate the advantage of our approach for solution to energy minimization and permanent fault-tolerance.

C. Throughput-energy performance

The experiments in previous sections are conducted for throughput-constrained applications. Therefore, throughput is not integrated into the optimization objective. A mapping is selected as long as it satisfies the application throughput requirement. However, to enable the use of our algorithm for scalable throughput applications, experiments are conducted incorporating the throughput into the optimization objective. Two real applications are considered – H263 encoder with 7 actors and MP3 decoder with 14 actors. The applications are run on architectures with 6 and 12 tiles respectively. Figure 7 plots the throughput per unit energy of the three fault-tolerant techniques for some of the fault scenarios. The energy considered is the sum of communication and computation energy. The results are normalized with respect to TMax. As expected, the performance of TCOEM is superior among the existing fault-tolerant techniques due to the consideration of throughput, communication energy and migration overhead. On average, for all single and double fault scenarios, TCOEM delivers 130% and 200% better throughput per unit energy performance than TMax and OMin respectively. Another trend to follow from Figure 7 is that as the number of actors increases, the throughput/energy also increases. MP3 Decoder with 14 actors has higher throughput/energy for most of the fault-scenarios considered. A similar trend is observed for other fault-scenarios and with different application set. Due to space limitations, these results are omitted.

Table V
EXECUTION TIME AND MAPPINGS OF VARYING NUMBER OF ACTORS ON 8 TILES

Actors	Homogeneous Mappings	Min. Energy Mappings	Execution Time (sec) TMax	TCOEM
8	1	1	0.616	0.616
9	36	6	9.718	1.012
10	750	11	71.269	1.014
11	11880	133	259.171	2.011
12	123411	311	933.282	3.533

D. Complexity

The complexity of Algorithm 1 is measured in terms of number of computations performed. The total number of fault-scenarios with f-tiles as faulty is given by $permute(p, f)$ where the $permute$ function gives the number of permutations of p tiles taken f at a time. For every fault-scenario, the ILP is solved N_f times to obtain the migration energy ($N_f = |M_t^{p-f}|$). The number of computations for f faulty tiles $= permute(p, f) * N_f$. Hence, the overall complexity of Algorithm 1 is given by Equation 9 where O(ILP) is the complexity of the ILP solver and N_f approximated with p^t.

$$complexity = \sum_{f=1}^{F} (permute(p, f) * N_f * O(ILP)) \quad (9)$$
$$\approx O(F * p^F * p^t * O(ILP))$$

E. Execution Time

Table V reports the total number of homogeneous mapping (i.e. the set M) evaluated and the execution-time of TMax and TCOEM as the number of actors are scaled on an architecture with 8 tiles. The number of mappings in column 2 of Table V is reported considering homogeneity of tiles [20]. For example, if there are 4 actors (a_0, a_1, a_2, a_3) to be mapped on 4 homogeneous tiles (t_0, t_1, t_2, t_3), all mapping permutations: $(t_0 - a_0, t_1 - a_1, t_2 - a_2, t_3 - a_3)$,

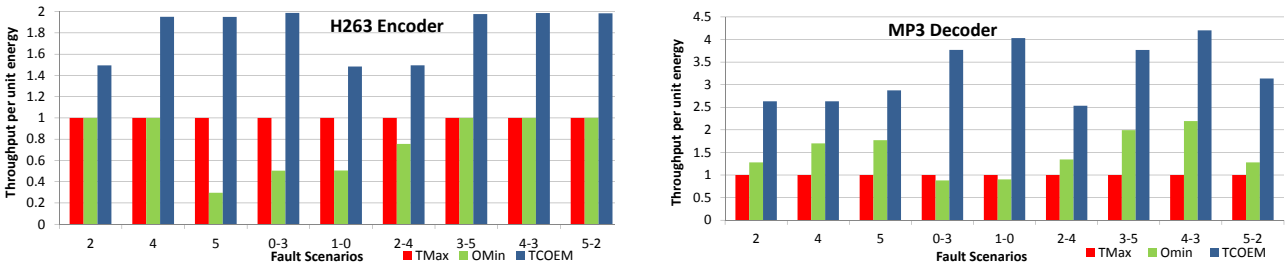

Figure 7. Normalized communication + computation energy for some fault scenarios with real audio and video applications

$(t_0 - a_1, t_1 - a_2, t_2 - a_0, t_3 - a_3)$, $\cdots$ are equivalent. Therefore, there is one mapping of 4 actors on 4 tiles. All entries of column 2 are filled in a likewise manner.

The table also reports the number of mappings (out of set M), which gives minimum energy. The execution time of TMax is dependent on the number of homogeneous mappings evaluated and therefore grows exponentially with the number of actors. The execution times for OMin and EMin are similar to the time taken by TMax. The TCOEM approach solves the ILP for every mapping of the minimum energy mapping set. Clearly, the execution time of TCOEM is less and is therefore scalable with larger problem size.

X. Conclusions & Future Direction

This paper minimizes the actor communication energy and migration overhead jointly for fault-tolerant remapping of tasks while satisfying the application throughput requirement. Experiments with real and synthetic applications demonstrate that the proposed approach achieves 40% reduction in communication energy and 33% reduction in migration overhead. Moreover, the proposed approach is within 10% of the minimum energy achievable on the platform. For scalable throughput applications, our approach outperforms existing fault-tolerant techniques by more than 100% in terms of throughput/energy. In future, minimization of actor computation energy can be investigated. Heuristic approaches can also be considered as an alternative to ILP for reduction of execution time. Finally, an approach to reduce storage associated with the mappings and the heterogeneity of the cores can be targeted in future.

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] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *Micro*, 2003.

[3] Y. Xie *et al.*, "Reliability-aware co-synthesis for embedded systems," *Journal of VLSI Signal Processing*, 2007.

[4] 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.

[5] V. Izosimov *et al.*, "Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2005.

[6] O. Derin *et al.*, "Online task remapping strategies for fault-tolerant Network-on-Chip multiprocessors," in *IEEE/ACM Symposium on Networks on Chip (NoCS)*, 2011.

[7] Y. Zhang *et al.*, "Workload-balancing schedule with adaptive architecture of MPSoCs for fault tolerance," in *IEEE Conference on Biomedical Engineering and Informatics*, 2010.

[8] J. Huang *et al.*, "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.

[9] C. Lee *et al.*, "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.

[10] N. Jha, "Low power system scheduling and synthesis," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2001.

[11] J. Hu *et al.*, "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.

[12] L. Goh *et al.*, "Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2009.

[13] T. Wei *et al.*, "Reliability-Driven Energy-Efficient Task Scheduling for Multiprocessor Real-Time Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2011.

[14] L. Huang *et al.*, "Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2010.

[15] E. Lee *et al.*, "Synchronous data flow," *Proceedings of the IEEE*, 1987.

[16] M. Pittau *et al.*, "Impact of task migration on streaming multimedia for embedded multiprocessors: A quantitative evaluation," in *IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, 2007.

[17] T. Ye *et al.*, "Packetized on-chip interconnect communication analysis for MPSoC," in *IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2003.

[18] A. Ghamarian *et al.*, "Throughput analysis of synchronous data flow graphs," in *IEEE Conference on Application of Concurrency to System Design (ACSD)*, 2006.

[19] S. Stuijk *et al.*, "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.

[20] A. K. Singh *et al.*, "A hybrid strategy for mapping multiple throughput-constrained applications on MPSoCs," in *ACM Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, 2011.