# FARM: Fault-Aware Resource Management in NoC-based Multiprocessor Platforms

Chen-Ling Chou and Radu Marculescu
*Department of Electrical and Computer Engineering*
*Carnegie Mellon University, USA*
*{chenlinc,radum}@ece.cmu.edu*

***Abstract*** - In this paper, we address the problem of run-time resource management in non-ideal multiprocessor platforms where communication happens via the Network-on-chip (NoCs) approach. More precisely, we propose a system-level fault-tolerant technique for application mapping which aims at optimizing the entire system performance and communication energy consumption, while considering the occurrence of permanent, transient, and intermittent faults in the system. As the main theoretical contribution, we address the problem of spare core placement and its impact on system fault-tolerance (FT) properties. Then, we investigate several metrics and provide insight into the fault-aware resource management process for such non-ideal multiprocessor platforms. Experimental results show that our proposed resource management technique is efficient and highly scalable and significant throughput improvements can be achieved compared to the existing solutions that do not consider failures in the system.

## I. Introduction

Resource utilization and system reliability are critical issues for the overall computing capability of multiprocessor systems-on-chip (MPSoCs) running a mix of small and large applications. This is particularly true for MPSoCs consisting of many cores that communicate via the network-on-chip (NoC) approach since any failures propagating through the computation or communication infrastructure can degrade the system performance, or even render the whole system useless. Such failures may result from imperfect manufacturing, crosstalk, electromigration, alpha particle hits, *etc.* and be permanent, transient, or intermittent in nature [1].

Existing fault-tolerant (FT) techniques for NoC resilience target the device, packet/data, or end-to-end transaction levels of abstraction [2]. However, there is a need to complement these approaches by handling failures at *system-level* and thus ensuring resiliency while maintaining the required levels of system performance. Some studies have shown that adding spare cores and wires can significantly improve the reliability, reduce the cost, and be a substitute for the burn-in process [1].

In this paper, our target NoC platform (shown in Fig. 1) is a 2-D tile-based architecture, which consists of various resources and network elements. More precisely, the resources consist of manager cores[1], computational tiles (*i.e.* operational processors/cores) and memory tiles, while the network elements consist of routers, links, and resource-network interfaces. The role of the manager cores

1. For large-scale systems, it is suggested to integrate several distributed manager cores in the platform to perform distributed management and to avoid a single point of failure [23].

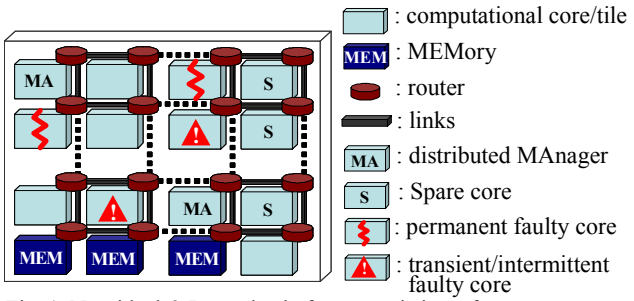



Fig. 1. Non-ideal 2-D mesh platform consisting of resources connected via a network. The resources include distributed managers, computational tiles (operational cores and spares) and memory titles with occurring permanent, transient, or intermittent faults.

('MA' tiles in Fig. 1) is to control the status of the entire system, *i.e.* decide on resource management and control the migration process via the platform OS [4]. For computational tiles, we assume a *j*-out-of-*i*-core model [1]; that is, the platform consists of *i* cores, where at least *j* of these *i* cores should be defect-free (*i.e.*, active and reachable) cores responsible for running the application tasks in order to satisfy the performance requirements. In other words, if there exist *k* (permanent) faulty cores in the system due to the imperfections in manufacturing (see the 'flash sign' in Fig. 1), then we assign *i-j-k* cores as spares for application computation. The role of a spare core ('S' in Fig. 1) is to replace the (temporarily or intermittently) faulty cores (see '!' in Fig. 1) or other unreachable cores (*e.g.*, due to the failure of the system interconnect). Of note, some design parameters given for the model, *i.e. i*, *j*, and *k,* are related to the chip yield or manufacturing process [1].

Doing effective resource management at *system-level* for such irregular MPSoCs with failures occurring dynamically, and minimizing the communication energy consumption while maximizing the entire system performance is a challenging task. It is obvious that the lack of regularity increases the distance among various cores; this may further incur a higher network contention on inter- or intra- application communication which may degrade the system throughput. Critical factors for causing the system degradation need to be quantified in order to handle the run-time (or dynamic) application mapping on such irregular platforms. In addition, when a transient, intermittent, or permanent failure occurs, the system must be able to *isolate* the failure from the offending source and mechanisms are needed in order to avoid failure propagation to the rest of the system.

Given the above considerations, our contributions in this paper are as follows:

978-3-9810801-7-9/DATE11/©2011 EDAA

- First, we explore the spare core placement problem and investigate the impact on failure propagation probability.
- Second, we investigate critical metrics for measuring the network contention and system fragmentation, as well as their impacts on system performance.
- Third, we propose and evaluate an efficient algorithm for fault-aware resource management with the goal of minimizing the communication energy consumption, while maximizing the overall system performance.

Taken together, these specific contributions improve the system-level resiliency, while optimizing the communication energy consumption and the overall system performance.

The remaining of this paper is organized as follows. In Section II, we review the relevant work. Section III discusses the modeling and spare core placement problems. In Section IV, we investigate several critical metrics and provide insight into the fault-aware resource management problem on irregular platforms. The problem formulation and details of the proposed FT algorithms are presented in Section V. Experimental results are presented in Section VI. We summarize our contribution in Section VII.

## II. Related Work

There is a considerable work on online failures/errors diagnosis and detection for multiprocessor systems at micro-architecture level with low power and area overhead [11][12]. Besides this work, operating system control on NoC-based multiprocessor platforms has been proposed to support system-level fault-tolerance [8]. Other techniques for failure/error as well as thermal monitoring for NoC platforms have been proposed in [22]. More recently, Huang *et al.* have taken the system lifetime reliability and system lifetime into consideration at design time while dealing with the application task mapping in NoC-based MPSoCs [7]. In addition, transient failures on NoC *links* have also been considered under stochastic and adaptive routing schemes [2][10].

In the area of packet-based communication for MPSoCs, there exists prior work on run-time application mapping on NoCs that aims at optimizing the packet latency [23] and power/energy consumption [11][14]. However, to the best of our knowledge, this is the first work that explores the spare core placement problem and investigates critical metrics for run-time fault-aware resource management on non-ideal NoC platforms while applications entering and leaving the system dynamically.

## III. Modeling Issues

### A. Application Modeling

We assume that the incoming applications have been already analyzed off-line, similarly to the approach discussed in [3]. Consequently, each application is characterized and represented by a *directed* Application Characterization Graph $ACG = (V, E)$ (see Fig. 2(a)), where the type of each vertex $v_i$, $type(v_i)$, can be either '*t*' (*i.e.*, representing a cluster of tasks) or '*b*' (*i.e.,* representing a buffer or memory module). Tasks belonging to the same cluster/vertex should run on their own defect-free computational core ('*CP*'), while the buffer module should be assigned to the memory tile '*MEM*' of the NoC platform. Each directed edge $e_{ij}$ in $E$ characterizes the *communication* from vertex $v_i$ to vertex $v_j$, while weights $r(e_{ij})$ stand for the communication rate (*i.e.,* bits per second) from vertex $v_i$ to vertex $v_j$.

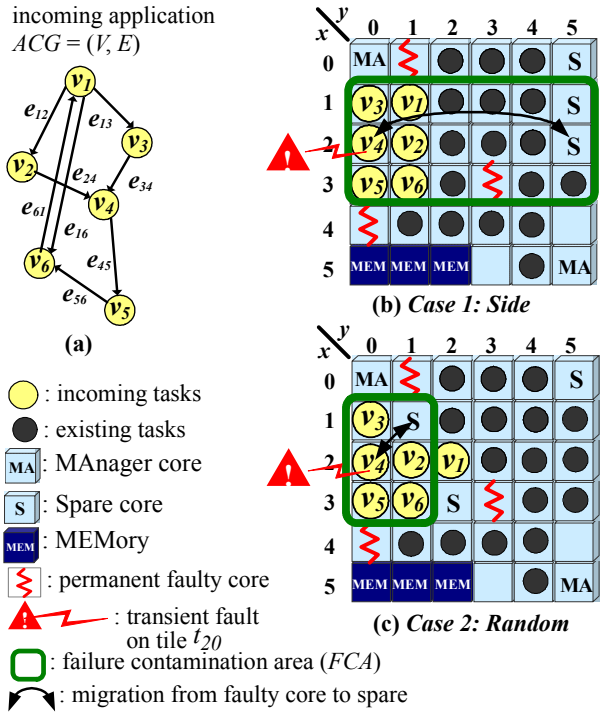



Fig. 2. (a) Application Characterization Graph ($ACG$) (b) Spare cores ('S') are assigned towards the sides of the system. (c) Spare cores 'S' are randomly distributed in the system.

### B. Communication Energy Modeling

The communication energy consumption of the entire system is modeled using the bit energy metric [13]; then, the communication energy consumption of each application $ACG_i$ can be calculated as:

$$E_{ACG_i} = \sum_{\forall e_{ij} \in E \text{ in } ACG_i} r(e_{ij}) \times E_{bit}(e_{ij}) \quad (1)$$

where $E_{bit}(e_{ij}) = (MD(e_{ij}) + 1) \times E_{Rbit} + MD(e_{ij}) \times E_{link}$. The parameter $MD(e_{ij})$ represents the Manhattan Distance between tiles where vertices $v_i$ and $v_j$ are allocated to, respectively. The parameters $E_{Rbit}$ and $E_{link}$ are constants representing the energy consumed in routers (including the crossbar switch and buffers) and one unit link, while transmitting one bit of data. Therefore, the total communication energy consumption of the entire system, $E_{entire}$, from time $t = 0$ to $t = T$ can be calculated by

$$E_{entire} = \sum_{t=1}^{T} \sum_{all\ ACGs} E_{ACG_i} \times \Delta(ACG_i, t) \quad (2)$$

where $\Delta(ACG_i, t) = 1$ if the application $ACG_i$ is running in the system from time $t$-1 to $t$, otherwise, $\Delta(ACG_i, t) = 0$.

In practice, applications can enter and leave the system dynamically. In addition, due to wear-out, soft errors, *etc.*, the faults existing in the system can be transient, intermittent, or permanent, and the locations of faulty cores may change at run-time. Therefore, our goal is to find a *mapping function*, *map*( ), for allocating the incoming application tasks to the reachable, available and fault-free cores such that *i*) the communication energy consumption is minimized *ii*) the network contention is minimized and *iii*)

the *entire* system performance is maximized. Of note, the applications execution time and their relative ordering are *not* known in advance, thus considering the *entire* system optimization during mapping is a critical challenge for solving this problem.

### C. Spare Core Placement

Any fault tolerant (FT) scheme needs to show *i*) No single point of failure *ii*) No single point of repair *iii*) Fault detection and recovery *iv*) Fault isolation to the failing core *v*) Fault containment to prevent propagation of the failure. For the first two requirements, it is clear that since the spare cores exist, if any of the cores in the system fails, it is unlikely to bring the entire system to a complete halt. In addition, we do not need to shut down the entire system in order to replace a failed core; instead, we can simply have the state recovery scheme in each core or replace the failed core with the spare one at run-time *via* task/process migration[2]. However, from a system-level point of view, the *spare core placement* problem needs to be addressed since it directly affects the last three properties of the FT scheme, especially for systems relying on NoC-based communication. Indeed, with a good spare core placement, not only the distances between the spare and faulty cores decrease, but also the failures propagation across the rest of the system is avoided.

Assume that an incoming application (see its *ACG* in Fig. 2(a)) needs to be mapped onto a 6 × 6 NoC platform interconnected via a mesh network under wormhole switching and deadlock-free and minimal-path routing with virtual channels supported (default scheme as *XY* routing if applicable). Several spare core placement schemes are first studied here: *Case 1*) *Side assignment*: Assign the spare cores to the side of the system (shown in Fig. 2 (b)), *Case 2*) *Random assignment*: Randomly distribute the spare cores in the system (shown in Fig. 2 (c)), and *Case 3*) *Uniform assignment*: Evenly distribute the spare cores in the system[3]. Of note, each tile $t_{xy}$ is considered to be located in the NoC at the intersection of $x^{th}$ row and $y^{th}$ column (see the *x*- and *y*- coordinates in Fig. 2(b) and (c)).

Intuitively, the distance among the active cores in *Case 2* and *Case 3* is higher than that in *Case 1* since the system size grows by considering the spare cores; this, in turn, results in higher communication energy consumption and lower system performance. For example, it can be seen that for the incoming application in Fig. 2(b) and (c), $MD(e_{12}) = MD(e_{13}) = 1$ in *Case 1*, while $MD(e_{13}) = 3$ in *Case 2*. However, when a transient fault occurs at tile $t_{20}$, the master will assign the closest spare to recover the fault. Therefore, tiles $t_{25}$ and $t_{11}$ are selected in these two cases which means that the distance between the faulty core and the closest spare is 5 and 2, respectively. Moreover, we define the *failure contamination area* (*FCA*) to reflect the failure propagation probability, namely the largest area resulting from the communication re-routing while replacing the faulty core with a spare. As seen in Fig. 2(a), since vertices $v_2$, $v_3$ and $v_5$ undergo communication with vertex $v_4$, the *FCA* in *Case 1* is much higher than that in *Case 2*; this may further degrade the performance of some other existing application shown with black dots in the system. As shown with thick frames in Fig. 2(b) and (c), the *FCA* value in *Cases 1* and *2* is 18 and 6, respectively. We can conclude that distributing spares at random or evenly within the system slightly increases the MD among the active cores; this may causes some communication energy consumption during the mapping process, but it maintains useful levels of fault isolation and containment compared to the scenario when spares are placed on the sides.

2. We note that task migration at task- and resource-level has been well studied for reduced response time [4] and proactive/reactive interrupt between processors [5] and so this is out of the scope for this paper.

3. Due to space limitations, we plot only *Cases* 1 and 2 in Fig. 2.

## IV. Investigations involving Critical Metrics

We introduce next three performance metrics that are needed in order to reach the three goals of finding the FT mapping function, *map*( ), as explained in Section III.*B*.

1. *Weighted Manhattan Distance* (*WMD*): Let vertex $v_i$ and $v_j$ be mapped onto tiles $t_{ab}$ and $t_{cd}$, respectively. The weighted MD between any two vertices is defined by

$$r(e_{ij}) \times MD(map(v_i), map(v_j)) = r(e_{ij}) \times (|a\text{-}c| + |b\text{-}d|) \quad (3)$$

Based on the bit energy metric [13], it is obvious that the weighted MD is positively correlated with the communication energy consumption.

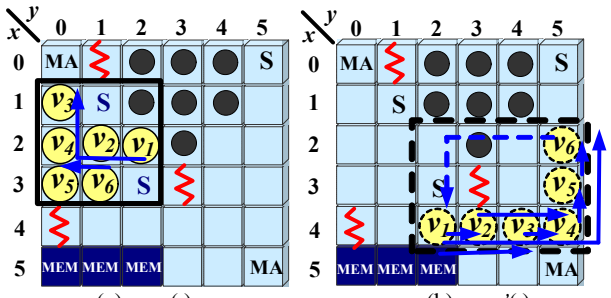
2. *Link Contention Count* (*LCC*): The link contention occurs when two communication flows $e_{ij}$ and $e_{kl}$ from the same or different *ACGs*, where $i \neq k$ and $j \neq l$, contend for the same link somewhere in the network. Such link contention can produce a significant degradation on system performance [9].

3. *System Fragmentation Factor* (*SFF*): This factor reflects the degree to which the non-contiguity of one application may affect other regions where vertices in different applications may be allocated to. The system fragmentation factor is defined as

$$SFF = \frac{w \times h - |V| - f - s}{w \times h} \quad (4)$$

where *w* and *h* are the width and the height of the minimal enclosing rectangle covering the mapping solution of *ACG* = (*V*, *E*), while *f* and *s* are the number of faulty cores and spares in that rectangle, respectively. Therefore, the smaller *SFF* value for each application on the system is a good indication of optimizing the entire system performance.

One example can be seen in Fig. 3 where two possible mapping results (*map*( ) and *map*'( )) are shown with solid and dotted circles, respectively. With the same *ACG* shown in Fig. 2(a), the *WMD* of vertex $v_1$ and vertex $v_6$ in *map*( ) is $r(e_{16}) \times 2$, while in *map*'( ) is $r(e_{16}) \times 5$. Under the *XY* routing used in Fig. 3, the *LCC* in *map*( ) for the *ACG* is 1 (*i.e.* $e_{13}$ and $e_{24}$ share the same link,); the *LCC* in *map*'( ) is 5. In addition, the *SFF* in *map*( ) for




(a) *map*( ) (b) *map'*( )

Fig. 3. Two mapping results for the *ACG* in Fig. 2(a) where the spare cores are randomly placed on the platform.

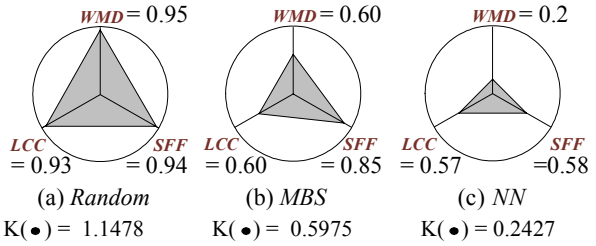



Fig. 4. 3D Kiviat plots with *WMD*, *LCC*, and *SFF* metrics for three difference mapping schemes (*i.e.*, *Random*, *MBS* [6], and *NN* [14]).

the *ACG* is 0.11 = 1/9 while that in *map*'( ) is 0.33 = 4/12. As seen, the larger the *SFF*, the more interference exists between the cores from different applications (see the dashed line in Fig. 3(b)).

Now, we evaluate the composite effect of these three metrics (*i.e.*, *WMD*, *LCC*, *SFF*) on the average packet latency and communication energy consumption. Several *ACG*s are generated using the TGFF package [18], where the number of vertices ranges from 5 to 15. Then, we implement three different scenarios (*i.e.*, Random mapping (*Random*), Multiple Buddy Strategy (*MBS*) [6] and Nearest Neighbor approach (*NN*) [14]) for allocating application tasks onto a 10 × 10 NoC with randomly selected faulty and spare cores. For each scenario, the average packet latencies, as well as the average values of the three metrics defined above are calculated for twenty different system configurations.

We employ 3D Kiviat graphs to provide a composite view of the impact of these three metrics [19]. A Kiviat graph consists of three dimensions, each representing one of the aforementioned metrics emanating from a central point (Fig. 4). Each metric varies from zero to the largest value observed in the experiments. As shown, the composite view of these metrics lies within the shaded area, *i.e.* Kiviat area (K( • )). Intuitively, the smaller the area, the better the system performance and the lower communication energy consumption are.

As shown, the average packet latency (measured in cycles) for *Random*, *MBS*, and *NN* approaches are 342.1, 148.27, and 94.22, while the normalized communication energy consumption to *Random* approach for *MBS* and *NN* approaches are 0.48 and 0.32. We can conclude that due to its smaller Kiviat area, the *NN* approach performs better and consumes less communication energy than the *MBS* and *Random* approaches; this also indicates the significant impact of these three metrics on system performance and communication energy consumption.

## V. Fault-AWARE Resource Management

The fault-aware resource management discussed in this paper covers *i*) migrating tasks from faulty cores to spares at system-level (discussed in Section V.*A*) and *ii*) allocating the incoming application tasks to available, reachable, and faulty-free cores, *i.e.* FT mapping (discussed in Section V.*B*). Before formulating the fault-aware scheme, some properties need to be defined first:

- There are two sets of cores/tiles $t_{mn}$ on the NoC platform, namely {*CP*, *MEM*}, where all computational processing cores for running the application tasks belong to the *CP* set and the on-chip buffer and memory titles belong to the *MEM* set.
- For $t_{mn} \in CP$, $s(t_{mn})$ stands for the status of core located at $t_{mn}$. $s(t_{mn}) = -3$ if the core is assigned to be a spare, $s(t_{mn}) = -2$ if the core is permanent faulty, $s(t_{mn}) = -1$ if the core is affected by transient or intermittent faults, $s(t_{mn}) = 0$ if the core has been already assigned to some application and $s(t_{mn}) = 1$ if the core is idle/available.
- *map*( ): $v_i \rightarrow map(v_i) = t_{mn}$ stands for a mapping function from one vertex to one core.

```
while(1){
01  if (faults on t_mn are detected ||
        t_mn has high probability to get failed)
02    RUN_MIGRATION(t_mn);
03  if (one application ACG_Q enters && resources are enough)
04    map(v_i ∈ V in ACG_Q) =RUN_FT_MAPPING(conf, ACG_Q);
05  if (one application ACG_P leaves) update t_mn status where
06      ∀ t_mn ∈ map^-1(v_i ∈ V in ACG_P), s(t_mn) → 1;}
```

Fig. 5. The fault-aware resource management framework.

The fault-aware resource management framework is described in Fig. 5. With the NoC monitoring scheme [8][13], the distributed managers track the core status (line 01 in Fig. 5) and do reactive or even proactive migration if necessary (see line 02 in Fig. 5, details of migration explained in Section V.*A*). As seen, our scheme supports multiple applications entering and leaving the system dynamically (see line 03-06 in Fig. 5, details explained in Section V.*B*).

### A. RUN_MIGRATION Process

Similarly to the control scheme in [8], our NoC platform includes the data and control network. The task migration procedure is given in Fig. 6.

**01**: Control messages are send through the control network for notifying the distributed managers reactively or proactively.
**02**: The corresponding distributed manager searches the closest available spare which results in a smaller *FCA* value.
**03**: Execute the code migration or related data transmission through the data network.

Fig. 6. Main steps of RUN_MIGRATION process.

We note the energy consumption of sending control messages in Step 1 and the *FCA* value in Step 02 of Fig. 6 are evaluated in Section VI.*B*. The run-time and energy overhead of Step 3 are discussed in [4][5] based on the process response time and code sizes at task- and resource-level and is out of the scope of this work. Indeed, we consider here task migration at *system-level* so we focus on spare core placement, spare selection for faulty core replacement, *etc.* instead of showing the details of the task migration at task- and resource-level (*i.e.*, setup the interrupts in the codes for doing the migration, *etc*).

### B. RUN_FT_MAPPING Process

#### B.1. Problem Formulation

***Given*** the current system configuration, *conf*, and the incoming application $ACG_Q$

***Find*** *map*( ): $v_i \rightarrow map(v_i) = t_{mn}$, $\forall v_i \in V$ in $ACG_Q$ which minimizes the Kiviat area, *i.e.* (K( • )) corresponding to the three metrics, *WMD*, *LCC*, and *SFF*

***Such that*:**

$$\forall v_i \neq v_j \in V,\ map(v_i) \neq map(v_j) \quad (5)$$

$$\forall v_i \in V \text{ and } type(v_i) = \text{'}t\text{'},\ map(v_i) = t_{mn} \in CP \text{ and } s(t_{mn}) = 1 \quad (6)$$

$$\forall\ v_i \in V \text{ and } type(v_i) = \text{'}b\text{'},\ map(v_i) = t_{mn} \in MEM \qquad (7)$$

Equation 5 means that each vertex should be mapped to exactly one tile and no tile can host more than one vertex. Equation 6 and Equation 7 imply that the vertices should be assigned to the correct type of resources in the system.

### *B.2. RUN_FT_MAPPING Algorithm*

Any run-time algorithm must be lightweight and have a low energy consumption. Hence, we define a few variables to achieve such a goal. For each tile *t*, the number of available (and non-faulty) neighboring cores is stored in the variable *neighbor*[*t*], while the center of the current resulting region *R* is stored in the variable *center*[*R*]. $ED(t_{ij}, t_{kl})$ stands for the Euclidean Distance between tiles $t_{ij}$ and $t_{kl}$, *i.e.* $ED(t_{ij}, t_{kl}) = (|i-k|^2 + |j-l|^2)^{1/2}$.

The steps of the RUN_FT_MAPPING algorithm for one incoming application are shown in Fig. 7. We assume the number of vertices of type '*b*' and '*t*' in the incoming *ACG* are S1 and S2, respectively.

**Input: (1)** current system configuration, *conf*
**(2)** one incoming application $ACG_Q = (V, E)$
**Output:** mapping solutions for all vertices in $ACG_Q$ to the fault-free and the corresponding types of tiles.
**01:** Set a region $R \leftarrow \varnothing$
**02:** If S1 > 0, *mode* = 1; otherwise, *mode* = 2.
**03:** If *mode* = 1, select a tile $t_{mn} \in MEM$ and $R \leftarrow R \cup \{t_{mn}\}$, then go to **04**.
If *mode* = 2, randomly select a tile $t_{mn} \in CP$ and $R \leftarrow R \cup \{t_{mn}\}$, then go to **06**.
**04:** If S1 > (# of tiles $\in$ *MEM* in *R*), then select $t_{mn} \in MEM$ with smallest $neighbor[t_{mn}] + ED(t_{mn}, center[R])$ and $R \leftarrow R \cup \{t_{mn}\}$
**05:** Repeat **04** until S1 = (# of tiles $\in$ *MEM* in *R*), then go to **06**.
**06:** If S2 > (# of tiles $\in$ *CP* in *R*) with lowest temperature, then select $t_{mn} \in CP$ with smallest $neighbor[t_{mn}] + ED(t_{mn}, center[R])$ and $R \leftarrow R \cup \{t_{mn}\}$
**07:** Repeat **06** until S2 = (# of tiles $\in$ *CP* in *R*), then go to **08**.
**08:** Start with vertex $v_k$ in *ACG* with the largest $\sum_{i=1 \sim |V|} r(e_{ki}) + r(e_{ik})$ value and map it onto $t_{mn} \in CP$ or *MEM* closed to $center[R]$ if $type(v_k) =$ '*t*' or '*b*'
**09:** Pick an unassigned vertex $v_i$ with the largest $r(e_{ki}) + r(e_{ik})$ value to all assigned vertices $v_k$ in $ACG_Q$, and map it onto one available tile $t_{mn}$ in *R* (*i.e.* $map(v_i) = t_{mn}$) such that the *WMD* value between $v_i$ and all other assigned $v_k$ is minimized. If more than one tile is satisfied, select one which results in the smallest *LCC* value.
**10:** Repeat **09** until all vertices get assigned to tiles selected in *R*.

Fig. 7. Main steps of RUN_FT_MAPPING process.

In Steps 01-07, we select a region from the current system configuration with the goal of minimizing the *SFF*; this helps reducing the *LCC* caused by different applications. More precisely, by selecting tiles with the smallest *neighbor* and *ED* values continually into the region (see Steps 4 and 6), we are able to decrease the probability of the discontinuity for the formed regions as multiple applications enter and leave the system dynamically, which minimizes the *SFF* effectively. Also the number of tiles belonging to the *MEM* and *CP* set should be equal to the number of vertices of types '*b*' and '*t*' in *ACG*. Then, in Steps 08-10, we assign vertices to the selected region with the goal of minimizing the *WMD* and *LCC* caused by this incoming application. With the minimization of these three metrics, we seek to get a smaller Kiviat area through the incremental mapping process.

## VI. EXPERIMENTAL RESULTS

### *A. Evaluation with Specific Patterns*

In this section, we evaluate our FT mapping algorithm using a set of widely-used workloads consisting of 1) communication-intensive applications with all vertices communicating in an *all-to-all* fashion and 2) applications where all vertices communicate with each other through a central memory only (denoted as *one-to-all* communication). Several sets of applications are generated using the TGFF package [18] with the number of vertices ranging from 5 to 35 in one *ACG*. The communication rates are randomly generated according to some specified distributions. The sequences of incoming applications are also generated randomly.

In terms of spare cores, we consider two spare core placement scenarios: 1) *Side placement*, where all spares are assigned towards the sides and 2) *Random placement*, where spares are randomly distributed across the platform. Also, 10% of the computational cores are assumed to be permanently faulty due to the manufacturing process and randomly distributed across the platform. The uniform spare core placement scenario (*Case 3* discussed in Section III.*C*) gives similar results to the random spare core placement. Due to space limitations, here we report only the comparison between side and random placements.

In terms of failure rate modeling for computational tiles, the fault probability of each core is calculated by enhancing the MIL-HDBK-217 model [16] (which contains failure prediction models for many types of electronic systems) with the Arrhenius model [17] which is needed to capture the relationship of failure rate to temperature, *i.e.*

$$\lambda_p = A \times e^{\frac{-E}{kT}} \qquad (8)$$

where *E* is the *activation energy* for the process (for semiconductor failure modes, *E* is set to 0.9), *k* is Boltzmann's constant, set to $8.6 \times 10^{-5}$, and *T* the absolute temperature. *A* is a constant, being calculated such that the failure rate per cycle for each core operating at useful life is $10^{-9}$ under normal core temperature, *i.e.* 55° C. Moreover, to make more reliable predictions for systems while running applications dynamically, we apply the temperature modeling tool, HotSpot [15], to measure the temperature of each computational core at runtime.

Of note, we assume that the probability of getting permanent failures in memory tile is zero during simulation since one reports that the memory mean-time-between-failure rates is around 700 years [20]. Also, since the on-chip memory tiles benefit from built-in-self diagnostics and repair schemes, the transient, or intermittent failure in memory tiles can also be ignored.

Table 1 shows the throughput and communication energy consumption comparison for two mapping approaches, namely 1) our proposed FT mapping (*FT*) and 2) Nearest Neighbor (*NN*) [14], an heuristic which maps vertices with higher communication as closely as possible, for NoCs of different sizes. All results are experimented from an NoC simulator using C++ language combining with the HotSpot thermal measurement.

As seen in Table 1 for the *all-to-all* communication, our proposed technique (*FT*) achieves a higher throughput and has a lower communication energy consumption compared to the *NN* approach, especially for larger NoC platforms. Of note, our *FT* approach works even better when the spares are located ran-

domly on the platform compared to the *NN* approach. For *one-to-all* communication, the system performance cannot improve much since the bottleneck is mainly due to accessing data to/from memory. Despite this, we still can achieve more communication energy savings compared to the *NN* approach.

TABLE 1: THROUGHPUT AND ENERGY CONSUMPTION BETWEEN PROPOSED *FT* AND NEAREST NEIGHBOR (*NN*) APPROACHES FOR ALL-TO-ALL AND ONE-TO-ALL COMMUNICATION PATTERNS.

| | *spare core placement-Side* | | *spare core placement-Random* | |
|---|---|---|---|---|
| **specific *ACGs in different NoC size*** | ***throughput improvement* (*FT vs. NN*)** | ***comm. energy consumption savings* (*FT vs. NN*)** | ***throughput improvement* (*FT vs. NN*)** | ***comm. energy consumption savings* (*FT vs. NN*)** |
| **5 × 5** *all-to-all* | 23.2% | 12.5% | 23.5% | 15.8% |
| **10 × 10** *all-to-all* | 98.1% | 32.1% | 102.1% | 36.2% |
| **5 × 5** *one-to-all* | 3.4% | 13.8% | 4.1% | 17.8% |
| **10 × 10** *one-to-all* | 5.7% | 17.5% | 6.9% | 23.6% |

### *B. Evaluation with Real Applications*

We evaluate the potential of our algorithm on several real applications, namely five benchmarks from the Embedded System Synthesis Benchmarks Suite [21], a video object plane decoder, the MPEG4 decoder, picture-in-picture, and multi-window display applications, where the last four applications include several memory modules. The *ACG*s of these nine applications are built through an off-line analysis; applications are randomly selected to enter and leave the system.

The Nearest Neighbor (*NN*) mapping approach in [14] is evaluated against our *FT* method. Also, the comparison of 1) the average packet latency (*i.e.*, the time elapsed between packet generation at the source core and packet arrival at the destination core, in cycles), 2) communication energy consumption, and 3) the Kiviat area, on these approaches are given in Table 2. In each run, 5%-15% of the computational cores are assumed to be permanently faulty and randomly distributed in the system. We report the average results of running 50 runs for each mapping approach under different NoC sizes (*e.g.* $10 \times 10$ *NN*). We note that the range of each metric in the Kiviat graph is normalized from zero to the largest value observed in random mapping implementation. The same fault model is applied as that in Section VI.*A*. In addition, the overhead of the energy consumption for running our FT mapping algorithm and sending out the control messages are included in the communication energy consumption measurement.

TABLE 2: COMPARISON BETWEEN THE NEAREST NEIGHBOR (*NN*) AND OUR *FT* MAPPING RESULTS ON THE OVERALL SYSTEM PERFORMANCE.

| ***NoC size*** | ***mapping approach*** | ***avg. latency*** | ***normalized comm. energy consumption*** | ***Kiviat area*** |
|---|---|---|---|---|
| $10 \times 10$ | ***NN*** [14] | 105.37 | 1 | 0.264 |
| | ***FT*** | 63.82 | 0.54 | 0.051 |
| $20 \times 20$ | ***NN*** [14] | 191.28 | 1 | 0.351 |
| | ***FT*** | 67.33 | 0.37 | 0.042 |

As shown in Table 2, our approach can obtain lower average packet latency and smaller communication energy consumption compared to the *NN* approach. The data of the Kiviat area also imply that by minimizing the *WMD*, *LCC*, *SFF* metrics, we are able to reduce the average packet latency quite significantly; this, in turn, increases the system performance and decreases the communication energy consumption. The run-time overhead for running the *FT* (see Section V.*B.2*) and *NN* approaches on a 100MHz MicroBlaze processor acting as a distributed manager is, on average, 68*m*s and 46*m*s, respectively; these values are small enough to be suited for this kind of on-line optimizations.

## VII. CONCLUSION

In this paper, we have proposed a run-time fault-aware technique for allocating the application tasks to the available, reachable, and fault-free cores of embedded NoC platforms. By exploring a few critical metrics and their impact on the overall system behavior, we have shown several ways to decrease the distance connecting various tasks, mitigate the link contention, and avoid the system fragmentation. As shown, all these parameters have a great impact on system performance and communication energy consumption. We have also explored the spare core placement problem for improving systems reliability. Last but not least, our proposed approach can be applied to other system topologies like torus, higher dimensional platforms, or even hybrid topologies; this is left for future work.

## ACKNOWLEDGEMENT

The authors acknowledge support from NSF via grant CCF-0916752 and SRC via grant 2008-HJ-1823.

## REFERENCES

[1] S. Shamshiri, *et al.*, "A cost analysis framework for multi-core systems with spares," in *Proc. Int. Test Conference*, 2008.
[2] T. Schonwald, *et al.*, "Fully adaptive fault-tolerant routing algorithm for Network-on-Chip architecture," *Proc. Digital System Design Architectures, Methods and Tools*, 2007, pp. 527-534.
[3] A. M. Pastrnak, *et al.*, "Parallel implementation of arbitrary-shaped MPEG-4 decoder for multiprocessor Systems," in *Proc. Visual Communication and Image Processing*, Jan. 2006.
[4] S. Bertozzi *et al.*, "Supporting task migration in multi-processor systems-on-chip: a feasible study," in *Proc. DATE*, 2006, pp. 1-6.
[5] T. T. Suen and J. S. Wong, "Efficient task migration algorithm for distributed systems," *IEEE Trans. Parallel Distrib. Syst.* vol. 3, 1992, pp. 488-499.
[6] V. Lo, *et al.*, "Non-contiguous processor allocation algorithms for mesh-connected multicomputers," *IEEE Trans. on Parallel and Distributed Computing*, 1997.
[7] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," in *Proc. DATE*, 2009.
[8] V. Nollet, T. Marescaux, and D. Verkest, "Operating-system controlled network on chip," in *Proc. DAC*, June 2004, pp. 256-259.
[9] C-L Chou; R. Marculescu, "Contention-aware application mapping for Network-on-Chip communication architectures," in *Proc. ICCD*, Oct. 2008, pp.164-169.
[10] S. Manolache, P. Eles, and Z. Peng, "Fault and energy-aware communication mapping with guaranteed latency for applications implemented on NoC," in *Proc. DAC*, 2005, pp. 266-269.
[11] M-L Li, *et al.*, "Accurate microarchitecture-level fault modeling for studying hardware faults," *Intl. Conf.* on High Performance Computer Architecture, 2009, pp. 105-116.
[12] S. Das, *et al*, "RazorII: In situ error detection and correction for PVT and SER tolerance," *IEEE J. of Solid-State Circuits*, pp.32-48, Jan. 2009.
[13] T. T. Ye, L. Benini, and G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," in *Proc. DAC*, June 2002, pp. 524-529.
[14] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs," in *Proc. IEEE/IFIP Workshop Rapid Syst. Prototyping*, 2007, pp. 34-40.
[15] http://lava.cs.virginia.edu/HotSpot/
[16] http://www.sre.org/pubs/Mil-Hdbk-217F.pdf
[17] http://www.sematech.org/docubase/document/3955axfr.pdf
[18] Task graphs for free (TGFF v3.0) Keith Vallerio, 2003. [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/tgff/
[19] M. F. Morris, "Kiviat graphs: conventions and figures of merit," *SIGMETRICS Perform. Eval. Rev.* 3, 3 (Oct. 1974), pp. 2-8.
[20] http://src.alionscience.com/
[21] http://ziyang.eecs.umich.edu/~dickrp/e3s/
[22] P. Rantala, *et al.*, "Agent-monitored fault-tolerant Network-on-Chips: Concept, hierarchy, and case study with FFT application," in *DAC* Workshop on Diagnostic Services in Network-on-Chips, April 2008.
[23] M. A. Al Faruque, *et al.*, "ADAM: run-time agent-based distributed application mapping for on-chip communication," in *Proc DAC*, 2008, pp. 760-765.