# HEFT: A Hybrid System-Level Framework for Enabling Energy-Efficient Fault-Tolerance in NoC based MPSoCs

Yong Zou
Dept. of Electrical and Computer Engineering
Colorado State University, Fort Collins, CO, USA
yong.zou@colostate.edu

Sudeep Pasricha
Dept. of Electrical and Computer Engineering
Colorado State University, Fort Collins, CO, USA
sudeep@colostate.edu

*Abstract* - In emerging CMOS process technologies, network-on-chip (NoC) fabrics are increasingly becoming susceptible to transient faults. Fault-tolerance mechanisms that are typically employed in NoCs usually entail significant energy overheads that are expected to become prohibitive as fault rates increase in future CMOS technologies. We propose a system-level framework called HEFT to trade-off energy consumption and fault-tolerance in the NoC fabric. Our hybrid framework tackles the challenge of enabling energy-efficient resilience in NoCs in two phases: at design time and at runtime. At design time, we implement an algorithm to guide the robust mapping of cores on to a die while satisfying application bandwidth and latency constraints. At runtime we devise a prediction algorithm to monitor and detect changes in fault susceptibility of NoC components, to intelligently balance energy consumption and reliability. Experimental results show that HEFT improves energy/reliability ratio of synthesized solutions by 8-20%, while meeting application performance goals, when compared to multiple prior works on reliable system-level NoC design.

**Categories and Subject Descriptors:** C.5.4 [**VLSI Systems**]
**General Terms:** Design, Experimentation, Reliability.
**Keywords:** System-level design, networks-on-chip, fault-tolerance

## 1. INTRODUCTION

Networks-on-Chip (NoCs) have emerged as a promising communication architecture for multi-processor systems-on-chip (MPSoCs). NoC fabrics have been shown to provide better scalability, predictability, and performance than bus-based communication architectures [1], [31]. Routers in NoC fabrics play a key role in ensuring successful delivery of packets from the source to the destination node. When data packets enter a router, they are first stored in the input buffers. Subsequently the route allocator (RA), virtual channel allocator (VA), and switch allocator (SA) reserve the necessary resources for the packet to be sent to the appropriately allocated output port buffer. Unfortunately, transient faults in routers can corrupt the data in packets and may even cause misrouting of the packets passing through a NoC router. As CMOS technology scales downwards, MPSoCs are becoming increasingly susceptible to such transient faults due to a variety of factors that are becoming more pronounced with scaling, such as higher capacitive and inductive crosstalk, elevated levels of electromagnetic noise, alpha particle strikes, etc. Corrupted or misrouted data packets due to such faults can lead to the failure of an entire application.

To protect NoC routers against transient faults, encoding techniques such as Hamming Error Correction Codes (HECC) [2], [3] and techniques that replicate sub-components such as Triple Modular Redundancy (TMR) [4], [5] have been proposed in literature. The focus of these techniques has been on protecting the buffers inside a NoC router, because packets spend a large portion of their in-flight time inside a router waiting either in the input or output buffers. However, naively equipping these protection techniques for all buffers in all NoC routers can result in high power/energy consumption overhead at the chip-level. As power and energy costs have become essential constraints during chip design, a systematic design methodology is required to balance reliability measures and their associated power/energy costs. This is a synthesis problem that impacts many phases of an MPSoC design flow, such as core-to-die mapping, NoC router configuration, NoC routing algorithm selection, etc. In the past, there have been many efforts to synthesize (customize) NoC fabrics at design time, at the system level, driven by application-specific design goals (e.g., [6], [7], [8], [9], [28]), where the synthesis objective has been to trade-off energy-efficiency and performance. But these existing efforts do not include reliability against transient faults as part of their system-level NoC synthesis goals or constraints. A recent work included reliability in a design time NoC synthesis framework [14], but the approach ends up over-protecting components (e.g., protecting buffers that are not being used) which increases NoC energy costs.

Moreover, while the abovementioned design time NoC synthesis efforts are indispensable to trade-off design goals, these works do not accurately consider runtime characteristics of NoC traffic, such as unpredictable congestion due to traffic-bursts that impact both performance and energy. Changing NoC conditions at runtime also lead to a change in the fault vulnerability of various NoC components [19]. *Thus, there is a need to complement design time NoC-based MPSoC synthesis frameworks with mechanisms that can adapt to changing runtime conditions, to optimally manage the trade-off between fault tolerance, energy, and performance.*

In this paper, we propose a framework for synthesizing NoC-based MPSoCs called *HEFT*. This hybrid framework performs analysis and parameter tuning at design time and at runtime to aggressively manage trade-offs between energy, fault-tolerance, and performance. At design time, the framework solves the problem of mapping cores executing multi-application workloads on to a die while satisfying application latency and bandwidth constraints. At runtime, the framework utilizes a novel reliability predictor that dynamically estimates fault vulnerability of NoC router components, to efficiently manage energy overheads of enabling fault tolerance mechanisms in the NoC. Our main contributions in this work are:

- We create a NoC reliability model that is based on fault vulnerability analysis and improves upon prior models;
- We propose and explore runtime fault vulnerability predictors and adaptive protection management for NoC architectures;


*ESWEEK'14,* October 12 - 17 2014, New Delhi, India

http://dx.doi.org/10.1145/2656075.2656087

- We explore the impact of two different design time core-to-die mapping techniques for enabling energy-reliability trade-offs; the techniques minimize NoC energy and support tile shutdown for dark silicon driven enhancements, respectively;
- Our experimental results on several memory-intensive and compute-intensive parallel application workloads show a notable reduction in energy and increase in reliability while satisfying application performance goals, compared to multiple prior works on reliable system-level NoC design.

## 2. RELATED WORK

Several research efforts have focused on the challenges related to application core-to-die mapping and NoC architecture design for regular MPSoC architectures. In this section, we briefly present some representative examples of prior work in these areas.

In [6], Murali et al. present a greedy core-to-die mapping heuristic for a regular NoC-based MPSoC die, based on communication volumes emanating from each core. Their approach also searches different routing paths to balance NoC traffic and meet bandwidth constraints, but it does not include energy cost as an optimization goal. In [12], Ye et al. propose a model for energy consumption of NoC routers called "bit energy model" that defines the energy consumption for one bit of data transfer through a NoC router and link. Using this bit energy model, in [9], Hu et al. present a greedy framework to solve the application core-to-die mapping problem on a regular 2D mesh NoC based MPSoC. Their work aims to find a mapping that minimizes energy while meeting bandwidth goals. In [20], Srinivasan et al. use a slicing tree algorithm to search the core mapping space and arrive at a mapping that adheres to latency constraints with the goal of minimizing energy. In [21], Chou et al. introduce a traffic contention aware core mapping algorithm using integer linear programming. In [13], Leung et al. propose a genetic algorithm (GA) to find an energy-aware voltage island partitioning and core mapping. In [7] and [23], Ascia et al. extend the GA to solve a power-performance multi-objective mapping problem. In [32], Pasricha et al. propose heuristics to co-optimize the design of the memory subsystem and communication architecture fabric.

Due to reliability becoming an increasingly important goal for NoC design, some recent efforts are considering reliability as part of an overall optimization objective. Our prior work [33]-[36] proposed various fault tolerant routing schemes for 2D and 3D NoCs to overcome permanent NoC failures. In [24], Aisopos et al. use Monte Carlo simulations to model process variations and calculate reliability values for circuits. In [25], Pirretti et al. use a successful data delivery rate metric to quantify reliability and evaluate the fault tolerance of their NoC routing algorithms. In [16], Ababei et al. define the reliability of a NoC based on the path lengths between communicating cores on the die. They use this reliability model and use the core-to-die mapping framework from [9] to balance NoC reliability and energy costs. In [19], a network vulnerability factor (NVF) metric is proposed to characterize reliability of NoC components [15]. This work utilizes the idea of error masking for network flows, i.e., even if errors occur in some NoC components, as long as there is no useful data in these components, the program outcome is unaffected. In [14], an NVF based reliability model is used to characterize reliability, and a nested GA framework is proposed to solve the problem of energy-aware core-to-die mapping and fault tolerant NoC router customization for regular MPSoCs.

In this paper, we improve upon prior work (e.g., [14]) with a framework (HEFT) that uses more efficient core-to-die mapping techniques at design time, together with a runtime framework that monitors NVF and dynamically adapts protection components to trade-off energy with reliability. Our experimental results in Section 6 show the promise of our approach to balance energy, reliability, and performance goals in regular NoC based MPSoC architectures.

## 3. PROBLEM FORMULATION

In this section, two problems that impact the performance, reliability, and energy consumption of a NoC architecture are explored. The first problem is to decide the mapping of processing cores (on which application tasks have already been mapped) on to the die. The second problem is to select mechanisms to enable fault tolerance in NoC routers that support data transfers between cores. As these two problems are closely related to each other, we need to solve both of them in a unified manner.

Figure 1 shows an example of mapping each core in an application communication graph (ACG) to an MPSoC with a 2D mesh topology based NoC fabric. ACG is a directed graph, in which each vertex $C_i \in V$ represents an IP core (processor or memory), and each directed edge $comm_{ij} \in E$ represents communication dependencies between the source core $C_i$ and destination core $C_j$. The volume of communication from $C_i$ to $C_j$ is $v(comm_{ij})$ and $v(C_i) = \sum_{j \in V}(v(comm_{ij}) + v(comm_{ji}))$ . Each edge $comm_{ij}$ has a weight $B(comm_{ij})$ for application-specific communication bandwidth constraints, and a weight $L(comm_{ij})$ for latency constraints (maximum number of hops allowed between cores $C_i$ and $C_j$),

We assume that the designer can pre-define application zones on the mesh-based die, as shown in Figure 1, allowing multiple applications to co-execute on the regular MPSoC die. In general, communication occurs at an intra-application node level, i.e., only between nodes belonging to an application, and not between nodes of different applications. For a given NoC link width (e.g., 32 bits), we assume platform constraints related to the maximum bandwidth that the NoC links can support $B_{link}$, which is dependent on the CMOS technology node and chip power constraints. Thus, for all mapped cores utilizing a link, it is required that the total utilized bandwidth on that link $T_{link\_max} \leq B_{link}$.

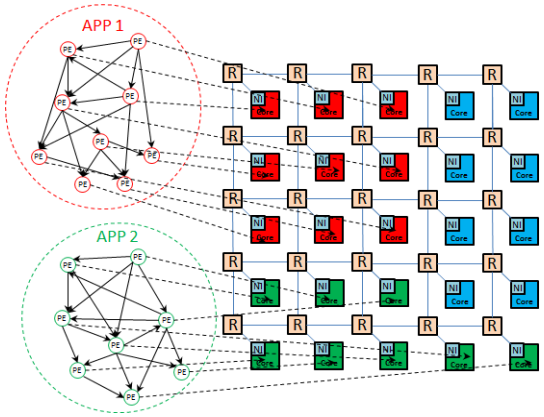



**Figure 1. Multi-application core-to-die mapping example**

The buffers in NoC routers are most susceptible to transient single event upset (SEU) errors, and we assume that errors on the relatively small footprint inter-router links are negligible. To protect NoC router buffers against SEU errors, we can employ two widely-used protection mechanisms: Hamming Error Correction Codes (HECC) and replicating components with Triple Modular Redundancy (TMR). Both mechanisms can correct SEU errors. But unlike TMR, HECC operation entails additional latency to encode and decode data. On the other hand, TMR implementations possess a larger area and power footprint than HECC. Table 1 shows the average power, latency, and area overhead trends for using either only HECC or only TMR to protect all buffers inside a NoC router, compared to a baseline NoC router without any protection. Results are obtained using an RTL model of a five staged pipelined NoC router that was enhanced with HECC and TMR mechanisms, and synthesized using Synopsis tools, with layout defined for a 45nm TSMC library.

**Table 1. HECC vs. TMR overhead comparison**

| | HECC | TMR | Integrated |
|---|---|---|---|
| **Power overhead** | 8% | 259% | 12% |
| **Latency overhead** | 150% | 10% | 75% |
| **Area overhead** | 5% | 200% | 7% |

It is apparent that using only HECC or TMR has high associated penalties. To minimize these overheads, we chose to judiciously combine HECC and TMR in a NoC router, by protecting input FIFO buffers (that can store multiple flits) in NoC routers with HECC due to its lower area and power overhead, and protecting the smaller output NoC buffers (that only store a single flit) using TMR to minimize latency overhead. This integrated HECC/TMR configuration, shown in Figure 2 and whose overheads are shown in the last column of Table 1 ("Integrated"), results in a relatively fast packet traversal with a reasonable area and power overhead.

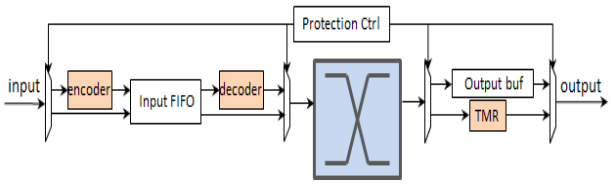



**Figure 2. Input/output NoC buffer protection control logic**

But even in our proposed NoC router configuration, protecting all the buffers in all NoC routers on the die can lead to excessive power dissipation and latency overhead. There is an opportunity to further reduce power and latency by intelligently determining an appropriate subset of buffers for protection against SEUs in the NoC routers. To this end, we require a dynamically selective protection methodology, to best match the time-varying behavior of an application, and optimally reduce the cost of enabling fault tolerance in NoCs. Our problem statement is summarized below:

---

**Given:** A set of applications (ACGs) with their unique bandwidth, latency and reliability constraints, a pre-defined island of cores on the die for each application, platform and technology constraints.

**Find:** a mapping of application cores to their pre-defined islands on the die; and an adaptive mechanism to enable energy-efficient fault tolerance in NoC routers such that all application, platform, and technology constraints are satisfied and the following problem objective is optimized:

**Problem Objective:** Minimize ($E_{network}/R_{network}$)

---

In the problem objective, $E_{network}$ is the total NoC energy consumption, which includes the energy overhead of implementing HECC and TMR in routers. $R_{network}$ is the reliability of the NoC fabric. This reliability model is discussed in the next section. By minimizing ($E_{network}/R_{network}$), we trade-off energy with reliability while simultaneously minimizing energy and maximizing reliability.

## 4. RELIABILITY AND POWER MODELING

In this section, we discuss how reliability and power are modeled for the NoC architecture considered in our work.

### 4.1 Reliability Model

Not all faults that occur on a chip eventually affect the final program outcome. For example, a bit flip in an empty input buffer of a NoC router will not impact program execution and its outcome. Based on this observation, the idea of network vulnerability factor (NVF) was proposed in [19], based on the concept of architecture vulnerability factor (AVF) for processors and memories. NVF is the probability that a transient SEU error in a network-centric structure (i.e., one that supports communication of data) finally produces a visible error in the output of a program. At any point of time, a structure's NVF can be derived via counting the important bits required for Architecturally Correct Execution (ACE) in the structure, and dividing these by the total number of bits in the structure. ACE bits are any bits in a structure that, if corrupted by transient errors, can result in application failure. In contrast, if an error occurs in any of the rest of the bits (called unACE bits), application correctness is not impacted. This phenomenon can be referred to as fault "masking" and is an inherent property of all computing components today.

In our work, we use the concept of NVF and ACE/unACE bits to define the reliability of a NoC router ($R_{router}$) as follows:

$$R_{router} = \frac{\sum_{t=1}^{t_{total}} N_{unace\_router\_t}}{t_{total} * S_{router}} \quad (1)$$

where $N_{unace\_router\_t}$ is the total number of unACE bits in a router in a clock cycle; $t_{total}$ is the total time period (clock cycles) over which reliability is being estimated; and $S_{router}$ is the total count of all buffer bits within the NoC router. We assume that all bits within each buffer have the same probability of being corrupted by SEUs.

There are several scenarios that can lead to the presence of unACE bits in a NoC buffer, as discussed in [14] which used the NVF concept as part of a reliability-aware NoC synthesis framework. However, unlike in [14] where NVF was estimated at design time, in this work we attempt to estimate NVF at run time. Data masking scenarios that require comprehensive analysis (such as "read masking" and "write-after-write masking" [14]) are impossible to detect at runtime, nevertheless two scenarios contributing to unACE bits can be detected at runtime: *(1) Idle time:* this is the most basic scenario - when there are no flits or data saved inside a buffer, we consider the buffer to be in an idle state. In this case, all the bits within the idle buffer are considered as unACE; *(2) Unused flit bits:* The flit entry in each input FIFO and output buffer of a NoC router has the same width. This assumption simplifies NoC router design and pipelined router operation with flits, which is why almost all NoC architecture implementations adhere to it. But as a result, there can be unused bits within a flit. Table 2 lists the unused bits (marked with an 'x') for the header and data flit types in our NoC architecture. The unused bits for a flit type are the unACE bits.

**Table 2. unACE bits across flit types (x indicates unACE bits)**

| | Is_tail | Pkt_id [5:0] | Flit_id [2:0] | Src_id [4:0] | Dest_id [4:0] | Data [63:0] |
|---|---|---|---|---|---|---|
| **Header** | √ | √ | √ | √ | √ | x |
| **Data** | √ | x | x | x | x | √ |

Once the reliability of individual NoC routers has been calculated by estimating the total unACE bits, the reliability of the entire NoC fabric ($R_{network}$) can be obtained by taking the product of the reliability $R_{router\_i}$ of all $N_{router}$ routers in the NoC fabric:

$$R_{network} = \prod_{0<i<N_{router}} R_{router_i} \quad (2)$$

The definition of NoC reliability in Equation (2) is different from [14] where each buffer is assigned a separate value of reliability, and the total network reliability is calculated as the product of all the buffer reliability values within the NoC. The drawback of this model from [14] is that it gives all buffers the same importance in the final reliability value and does not consider the fact that different NoC buffers can have different sizes. For example, [14] gives equal importance to input and output buffers when calculating network reliability. However, as the input buffer in our NoC router architecture is larger than the output buffer (input buffer has storage for multiple flits, and output port has storage only for a single flit), the network reliability value should be impacted to a greater degree by the reliability of the input buffer than the reliability of the output buffer. Our reliability model takes this factor into consideration, and is thus more accurate than [14].

An important motivation for considering the core-to-die mapping problem together with the problem of balancing energy and reliability during NoC-based MPSoC synthesis in our work is that the unACE bits inside NoC router buffers depend on the data traffic flowing through them, which in turn depends on manner in which cores are mapped to the die. As the unACE bits in NoC routers determine NoC router reliability, and consequently the reliability of the entire NoC architecture, core-to-die mapping has a direct impact on overall network reliability.

### 4.2 Power Model

To achieve a detailed power characterization of NoC routers with protection mechanisms, we implemented a NoC router at the RTL level, and performed logic synthesis and gate-level analysis using Synopsys Design Compiler and Primetime tools to obtain power dissipation for each sub component within a NoC router. We also implemented HECC and TMR mechanisms in the NoC router module, and considered their implementation overheads.

**Table 3. Router module power library (45nm)**

| | Input Buffer | HECC Input Buffer | Output Buffer | TMR Output Buffer | Link |
|---|---|---|---|---|---|
| **Dynamic** | 1.36 mw | 1.51 mw | 45µw | 267.55µw | 51.3µw |
| **Static** | 3.54µw | 5.18µw | 120nw | 1.43µw | 915nw |
| | **Crossbar** | **SA** | **VA** | **RC** | |
| **Dynamic** | 121µw | 105µw | 101µw | 91.5µw | |
| **Static** | 2.56µw | 2.33µw | 2.51µw | 1.02µw | |

Table 3 lists the average dynamic and static power values for the various NoC router components for the 45nm CMOS process technology node. We used a 0.5 switching probability to calculate the average power values. Using these power values, it is possible to calculate the energy of a flit flowing through a NoC router. We use knowledge of communication flows through a router after core-mapping and routing (discussed in Section 5) together with NoC router sub-component power values to generate application-specific communication energy estimates. This energy value is obviously higher if protection mechanisms are integrated in the NoC router, as can be seen from the higher power dissipation values of protected buffers in Table 3.

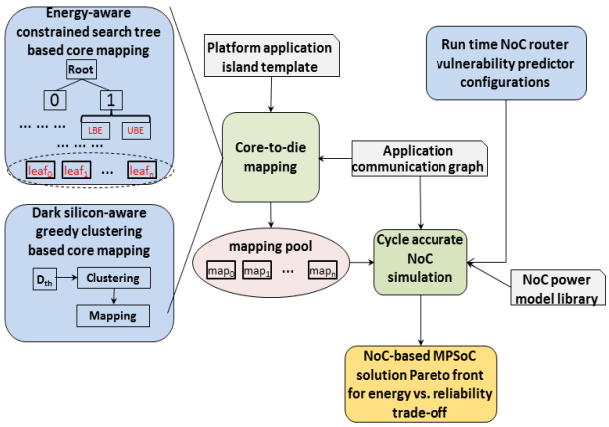



**Figure 3. Overview of HEFT synthesis framework**

## 5. HEFT SYNTHESIS FRAMEWORK

In this section, we describe our system-level framework for enabling Energy-efficient Fault-Tolerance (HEFT) in NoC-based MPSoCs. Figure 3 gives an overview of the HEFT framework. We explore two different core-to-die mapping techniques to generate a mapping pool that contains mappings optimized for either tile shutdown in dark silicon die environments using a greedy clustering approach or for overall low energy consumption using a constrained search tree approach. With the generated mappings in the mapping pool, we perform cycle accurate simulation using a NoC simulator enhanced with models for measuring energy and reliability, and runtime NoC vulnerability prediction module configurations. The simulations enable an accurate estimation of energy and reliability for the generated solutions, allowing for a Pareto front of solutions that satisfy application performance constraints while trading-off energy with reliability. In the following subsections, we discuss the core-to-die mapping techniques and runtime vulnerability prediction in detail.

### 5.1 Core-to-Die Mapping

We were interested in exploring the effectiveness of two diverse strategies for core mapping towards enabling a viable trade-off between energy and reliability. These strategies are discussed next.

#### *5.1.1 Dark Silicon-aware Core Mapping (DSCM)*

Our first core mapping strategy is designed to work in environments with dark silicon [29] constraints, where a designer may be required to shut down a subset of tiles on a die to ensure that chip-level power constraints are not violated. Shutting down a tile involves shutting down an IP core (e.g., processor, memory bank) as well as its associated NoC router. Ensuring that the shutting down of a NoC router does not end up disconnecting two communicating tiles on the die is an open problem. Our proposed mapping technique attempts to maximize the opportunities for turning off tiles to meet dark silicon power budget constraints without adversely impacting communication for the tiles that are executing workloads and communicating with each other.

**Algorithm 1: Dark silicon-aware core mapping**

**Phase 1: Clustering**

**Input**: $\mathbf{n_{core\_app\_i}}$: total core count in application i.
**v(comm$_{ij}$), B(comm$_{ij}$), L(comm$_{ij}$):** volume, bandwidth, and latency constraints for communication from core $C_i$ to core $C_j$, respectively
$\mathbf{v(C_i)} = \sum_{\forall j \text{ and } j \neq i}(v(comm_{ij}) + v(comm_{ji}))$

1: **foreach** application app_i:
2: **build** an array D of $diff_{th}$ values: $diff_{th}$ represents a threshold that influences cluster formation; the threshold values in the array are unique values calculated as: $v(C_i) - v(C_j)\ \forall i, j \text{ and } j \neq i$
3: **foreach** $d_{th}$ ∈ D:
4: **sort** cores in increasing order of $v(C_i)$ and generate a core list $\{C_{s1}, C_{s2}, \ldots, C_{sn}\}$.
5: j = 0; clu = 0;
6: **for** i=1 **to** $n_{core\_app\_i}$:
7: **if** $v(C_{si}) - v(C_{sj}) >= diff_{th}$:
8: clu++, j = i;
9: **else** add core $C_{si}$ to $cluster_{clu}$;
10: **end for**

**Phase 2: Mapping**

11: **sort** clusters in decreasing order of communication volume, and sort cores within clusters also in decreasing order of communication volume to create an ordered list of cores $\{C_{m1}, C_{m2}, \ldots, C_{mn}\}$.
12: **map** $C_{m1}$ randomly either to a corner or to the middle node.
13: **find** an unassigned tile as the next tile i with $\min(\sum_{j \in M, i \in U}(md_{ij}))$ that is connected with at least one of the mapped cores
14: **map** the next core in the ordered list of cores to the chosen tile.
15: **repeat** step 13 and 14 until all the cores are mapped.
16: **add** the generated mapping solution into mapping pool $MP_{app\_i}$. if it satisfies $B(comm_{ij})$ and $L(comm_{ij})\ \forall i, j$
17: **end foreach**
18: **end foreach**
19: **calculate** the energy cost for different combinations of application mappings from each $MP_{app\_i}$.
20: **choose** Ψ mappings with lowest energy as the final MP.

**Output: a pool of core to tile mapping solutions**

Algorithm 1 shows the steps of our dark silicon-aware core mapping technique. We perform mapping separately for different application islands (step 1). For each application, we generate a

separate mapping pool (step 16). In the end we greedily combine different mappings from the mapping pools of different applications together into full mappings and select Ψ lowest energy mappings (step 20). These mappings are eventually explored in more detail with cycle-accurate simulations to generate accurate reliability (as well as energy) estimates which are used to create a Pareto set of solutions that trade-off energy with reliability.

For the core mapping within each application, there are two phases. The first phase involves grouping cores with similar communication volume profiles into the same cluster. To form clusters, a threshold $diff_{th}$ is utilized, which represents the maximum communication volume difference within a cluster. We first build a threshold array D that contains all the differences of communication volume between any two cores in the application (step 2). We then use each element in D as a threshold value for each mapping loop (steps 3-17). In steps 4-10, starting from the core with the lowest communication volume with the other cores, we group cores into clusters based on $diff_{th}$.

The second phase involves mapping the clusters to tiles on the die. We sort the generated clusters from phase 1 in decreasing order of total communication volume of the cores inside the cluster, and then sort the cores within the clusters also in decreasing order of total communication volume of the cores to create an ordered list of cores (Step 11). Clusters with higher communication volumes and cores within them are mapped first. We randomly map the first core to a corner node or middle node of the mesh (step 12). In step 13, M represents the occupied tile set, U represents the unoccupied tile set and $md_{ij}$ is the Manhattan distance from an unassigned tile i to tile j. The cores are then iteratively mapped to the tiles (steps 13-15) and the full mapping solution for the application is added into its mapping pool of solutions as long as bandwidth and latency constraints are not violated, for an XY routing scheme (step 16).

Thus in this manner, by clustering and mapping cores with similar communication volumes together, the routers connected to the cores with less communication volume can have more opportunities to be turned off in scenarios with stringent dark silicon power budgets, as long as there is no data transfer through them.

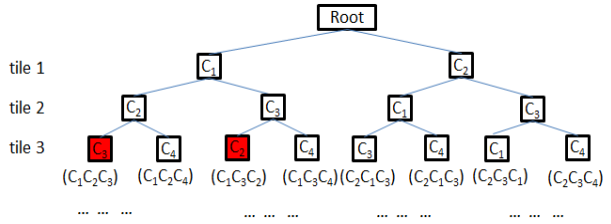



**Figure 4. Constrained search tree core mapping**

### *5.1.2 Energy-aware Core Mapping (EACM)*

In this mapping technique, our emphasis is on discovering mappings that minimize communication energy, without any optimizations for tile shutdown scenarios. By default there can be n! different ways to map n cores on n tiles. As an efficient way to represent different mappings, we build a tree structure to save different mapping solutions. In this tree, each node contains a portion of the mapping solution. By traversing the tree from the root to a leaf node, and combing the information from intermediate nodes, a unique and complete mapping solution can be obtained. Figure 4 shows an example of this tree structure. The root represents the starting point with no cores mapped. At the first level of the tree, each node represents a partial solution with one core mapped to a tile on the mesh. Subsequent levels each add one more core to the set of mapped cores and a leaf node level represents the last remaining core being mapped to the last remaining tile on the die.

An exhaustive build of our search tree to represent all possible mapping solutions is prohibitive as it leads to solution space explosion. To limit our search, we integrate two constraining techniques as part of a constrained search tree approach. First, we limit the number of children nodes to a user defined parameter k for each node. The k different cores selected for mapping at a level have the largest value for $\sum_{j\in M}(v(c_{ij}) + v(c_{ji}))$. Here M is the mapped core set and i is the unmapped core under consideration. The goal is to place highly communicating closer to each other. The manner in which tiles are selected at each successive level in the tree is shown in Figure 5, for a 25 core 2D mesh NoC topology. This tile selection pattern ensures that successive cores being mapped are close to cores that were mapped earlier. Limiting the number of child nodes reduces the total number of mapping solutions that need to be evaluated from n! to $(\sum_{i\leq k}(i)) \times k^{n-k}$. But the number of total mapping solutions is still very large, e.g., for a 9 core system (n=9) and if k=4, the total number of different mapping solutions that must be evaluated is as high as 10240. Therefore we utilize a second technique to further intelligently constrain the search process by discarding highly energy inefficient partial solutions while still considering potentially interesting solutions.

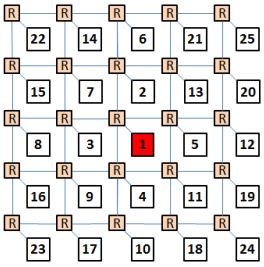



**Figure 5. Tile mapping pattern**

**Algorithm 2: Constrained search tree core mapping**

**Input:** $n_{core\_app\_i}$: total core count in application i.
**v(comm$_{ij}$), B(comm$_{ij}$), L(comm$_{ij}$):** volume, bandwidth, and latency constraints for communication from core $C_i$ to core $C_j$, respectively
$\mathbf{v(C_i)} = \sum_{\forall j \text{ and } j\neq i}(v(comm_{ij}) + v(comm_{ji}))$

1: **set** $SE_{app\_i}$ threshold for application i as the highest communication energy over *p* random mapping trials
2: **foreach** application app_i
3: **foreach** unoccupied tile:
4: **foreach** node in current level:
5: **choose** at most k cores with highest $\sum_{j\in M, i\in U}(v(comm_{ij}) + v(comm_{ji}))$ $\forall i \in U$ and sort these cores in decreasing order
6: **foreach** chosen core in sorted order:
7: **generate** a child node $n_c$ mapping the core onto the current tile
8: **if** $n_c$ is the leaf node of current application:
9: **save** mapping from root to $n_c$ to mapping pool $MP_{app\_i}$ if it satisfies $B(comm_{ij})$ and $L(comm_{ij})$ $\forall i, j$
10: **else**
11: estimate the lowest energy (LE) of $n_c$.
12: **if** LE>$SE_{app\_i}$:
13: mark this nodes as **end node** which means that the tree will not be expanded further from this node.
14: **end foreach**
15: **end foreach**
16: **end foreach**
17: **end foreach**
18: **calculate** the energy cost for different combinations of application mappings from each $MP_{app\_i}$.
19: **choose** Ψ mappings with lowest energy cost as the final MP
**Output: a pool of core to tile mapping solutions**

Algorithm 2 describes our constrained search tree core mapping algorithm in detail. At the beginning (step 1), we set a standard energy $SE_{app\_i}$ parameter for each application i, by assigning it the highest value of communication energy for *p* (chosen to be 100) random full mapping trials. This parameter will be used to constrain high energy solutions later in the algorithm. Similar to the DSCM

algorithm from the previous section, we generate mapping pools separately for different application islands (step 2) and in the end we greedily combine different mappings from the mapping pools of different applications together into full mappings and select Ψ lowest energy mappings (step 19). These mappings are eventually explored in more detail with cycle-accurate simulations to generate accurate reliability (as well as energy) estimates which are used to create a Pareto set of solutions that trade-off energy with reliability.

Starting from the root of the tree for an application, we generate k new nodes at each level of the tree (steps 4-7), with nodes being selected at each level from the set of unmapped cores that have the highest value of $\sum_{j\in M, i\in U}(v(comm_{ij}) + v(comm_{ji}))$. Here M is the set of mapped cores and U is the set of unmapped cores. For the degenerate initial case (M={null}), we select k cores with the highest values for $v(C_i)$. The tile at each level is selected based on the pattern shown in Figure 5. If a generated node is a leaf node, we add the corresponding full mapping (from root to leaf) into the application mapping pool (step 8-9) as long as no bandwidth and latency constraints are violated by the mapping. If the generated node is not a leaf node, we estimate the lowest energy (LE) that can be generated with this partial mapping information of the current node (steps 10-11). If LE is larger than $SE_{app\_i}$ for the node, it indicates that we are unlikely to find a mapping that has lower energy cost than $SE_{app\_i}$ which includes this node. Therefore we mark this node as an 'end node' and in the next level of the tree, we do not expand nodes marked as end nodes (step12-13).

The LE value for a node is calculated as follows:

$$LE = E_{static} + E_{LE(m,m)} + E_{LE(u,m)} + E_{LE(u,u)} \quad (3)$$

$$E_{LE(m,m)} = \sum_{\forall i,j\in M}(E_{dynamic_{ij}} + E_{dynamic\_ji}) \quad (4)$$

$$E_{LE(u,m)} = \sum_{\forall i\in U, j\in M}(E_{dynamic\_ij} + E_{dynamic\_ji}) \quad (5)$$

$$E_{LE(u,u)} = \sum_{\forall i,j\in U}(E_{dynamic\_ij} + E_{dynamic\_ji}) \quad (6)$$

$M: mapped\ core\ set. U: unmapped\ core\ set$

where $E_{static}$ is the static energy of the communication subsystem, $E_{LE(m,m)}$ is the theoretically lowest dynamic energy attributed to communication between msapped cores, $E_{LE(u,m)}$ is the theoretically lowest dynamic energy attributed to communication between mapped and unmapped cores, and $E_{LE(u,u)}$ is the theoretically lowest dynamic energy attributed to communication between unmapped cores. $E_{dynamic\_ij}$ is the dynamic energy attributed to communication from core $C_i$ to core $C_j$. The power model discussed in subsection 4.2 together with average estimates of packet delay through routers and hop counts along minimal path are used to calculate the energy values for communication flows between nodes. $E_{LE(m,m)}$ can be calculated easily as the required core mapping information is known. To estimate $E_{LE(u,m)}$, we randomly select unmapped cores, and greedily map each core i to an unoccupied tile that generates the lowest $\sum_{\forall j\in M}(E_{dynamic\_ij} + E_{dynamic\_ji})$. By adding the energy due to all of the newly added communication flows, $E_{LE(u,m)}$ can then be estimated. To estimate $E_{LE(u,u)}$, for each unmapped core $C_i$, we need to estimate a value for $E_{dynamic\_ij}$ where j represents the other unmapped cores. First, we map a core $C_i$ with the highest value of $v(C_i)$ to a randomly chosen unmapped tile. We then sort other unmapped cores in decreasing order of their total communication volumes with core $C_i$, and then map the cores in the sorted order to tiles in increasing order of Manhattan distance to the tile to which $C_i$ is mapped. After all the cores are mapped, $E_{LE(u,u)}$ is easily obtained.

## 5.2 Adaptive Router Protection

Application core to die mapping, discussed in the previous section, occurs at design time. In our framework, we complement the design time mapping mechanisms with a runtime technique to adapt protection in NoC routers. Our motivation for the runtime support stems from the observation that traffic between cores varies in its intensity and burstiness over time. The variation in traffic in turn creates variations in vulnerability in NoC routers. If we are able to somehow predict variations in vulnerability at runtime by observing traffic characteristics over time, it is possible to identify epochs of time during which vulnerability in one or more routers is very low. This information can then be used to turn off protection (e.g., disable HECC and TMR support) inside the low vulnerability NoC routers to save communication energy while still maintaining a relatively high on-chip communication reliability.

To achieve this objective, we propose a runtime reliability prediction manager (RPM) module within each NoC router. Figure 6 shows the block diagram of the RPM module for a pair of input and output buffers along the +x direction. For each input and output buffer, a counter is used to monitor the total ACE bits encountered within a pre-defined time window interval $t_{interval}$. The counter width depends on the value of $t_{interval}$, e.g., larger $t_{interval}$ values require more counter bits to keep an accurate count for the greater traffic volumes encountered in the interval, compared to smaller $t_{interval}$ values for which fewer counter bits may suffice. The monitored information in the interval is used to turn on/off protection modules inside a NoC router at the beginning of the next time interval based on the history-based prediction from the RPM. Figure 7 shows the state machine of the saturation counter based predictor for each buffer that guides the decision to protect that buffer. There can be $n_p + 1$ different states for each saturation counter with 1 unprotected state and $n_p$ protected states. We chose only a single state for keeping a component unprotected to give more emphasis to reliability over energy savings. Transitions between states occur when a "1" or "0" signal is received, representing "protect" or "unprotect" decisions made by the saturation counter based predictors. At the end of each $t_{interval}$ period, all counters are reset and a new round of ACE bit monitoring is initiated for the next interval (Figure 6).

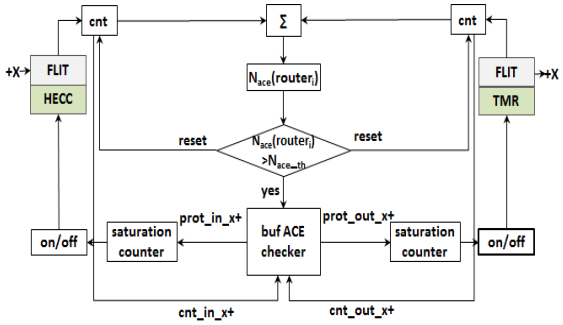



**Figure 6. Block diagram of reliability prediction manager**

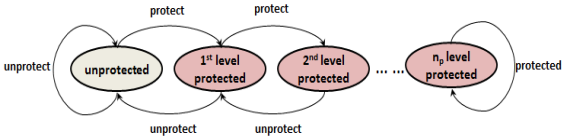



**Figure 7. State machine for saturation counter based predictor**

Algorithm 3 presents the details of how our runtime RPM is implemented inside a NoC router. The RPM module is invoked at the end of every $t_{interval}$ cycles, which is a user defined interval. A router ACE bits threshold value $N_{ace\_th\_router}$ is calculated and stored, based on the size of $t_{interval}$, the total number of buffer bits within the router $\sum_{i\in ports}(n_{buf\_i})$ and the user-defined reliability threshold $p_{th}$ (a value between 0 and 1, with 1 indicating 100% reliability) for the communication subsystem. Similarly, for each buffer, we calculate and store a buffer-level ACE bit threshold value. Both of these ACE bit threshold values typically only need to be computed once at startup (or every time the size of $t_{interval}$ is varied dynamically; but this is beyond the scope of this work).

**Algorithm 3: Reliability prediction manager**

**Input: ACE bits counter value $cnt_{buf\_i}$ for each input and output buffer.**

$N_{ace\_th\_router} = t_{interval} * \sum_{i \in ports}(n_{buf\_i}) * (1 - p_{th})$

$N_{ace\_th\_buf\_i} = t_{interval} * n_{buf\_i} * (1 - p_{th})$

1: $N_{ace}(router_i) = \sum_{i \in ports} cnt_{buf\_i}$
2: **if** $N_{ace}(router_i) > N_{ace\_th\_router}$:
3: **foreach** buf i **in** the router:
4: **if** $cnt_{buf\_i} >= N_{ace\_th\_buf\_i}$:
5: $sa\_in_i = 1$
6: **else:**
7: $sa\_in_i = 0$
8: **endif**
9: **end foreach**
10: **endif**

**Output: protection decision for each input and output buffer.**

At the end of $t_{interval}$ cycles, the RPM module checks the number of ACE bits encountered over the interval by adding together values from ACE bit counters for each buffer (step 1). The sum $N_{ace}(router_i)$ is compared to $N_{ace\_th\_router}$. If $N_{ace}(router_i)$ is larger than $N_{ace\_th\_router}$, we then proceed to check each buffer's ACE bit counter $cnt_{buf\_i}$ (steps 2-3). If the ACE bit count $cnt_{buf\_i}$ for a buffer is larger than the ACE bit threshold for this buffer, we send a "protect" signal to the buffer's saturation counter, otherwise, an "unprotect" signal is sent to the saturation counter (steps 4-7). Based on the resulting new state of each saturation counter, we decide whether or not to protect the corresponding buffer (Figure 7).

# 6. EXPERIMENTAL RESULTS

## 6.1 Experimental Setup

We modified NoC RTL models from the Netmaker library [27] with HECC and TMR protection mechanisms and RPM module support. Synopsys VCS was used to run simulations with the NoC RTL models, and with the generated signal traces, we performed logic synthesis using Synopsys Design Compiler, and gate level power analysis using Synopsys Primetime [30] to produce an activity based power library for NoC router sub-components. This power data was used to calculate communication energy, once a core mapping was established. We used a modified Gem5 full system simulator with applications from the Splash-2 benchmark suite [18] to generate instruction traces. Gem5 was configured with ALPHA processors, each with 64kB L1 instruction and data cache, and 256K L2 caches. We considered 50 million "read" and "write" transactions out of instruction traces that varied from 1-10 billion instructions depending on the benchmark. The traces were used to enable a trace-driven simulation with a modified version of the Nirgam [22] SystemC-based cycle-accurate NoC simulator (with added support for dynamic RPM-based reliability and energy management). The trace-driven simulations allowed us to determine accurate communication energy and reliability data for our proposed HEFT framework. We performed various studies to explore different configurations of our framework and also performed comparisons with relevant prior work.

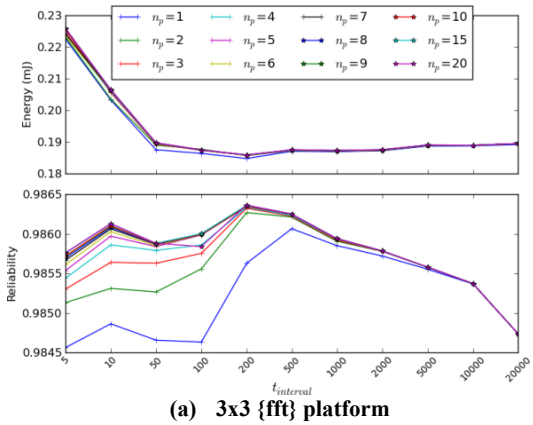
From [26], applications can in general be categorized as either compute intensive or memory intensive, based on whether an application spends more time computing than communicating with memory (compute intensive) or spends more time communicating with memory than computing (memory intensive). Thus compute intensive applications generate less communication in the network while memory intensive generate more communication traffic in the network, for the same simulation time. Table 4 shows a classification of a subset of Splash-2 benchmarks into the compute intensive and memory intensive categories. For our experimental analysis, we created single-application workloads as well as multi-application workloads that combined these applications into interesting configurations with varying compute and memory intensities. More specifically, we created the following workloads for the two platform sizes we considered: (i) 3x3 mesh: we used the compute intensive {ocean} and memory intensive {fft} single-application workloads; and (ii) 5x5 mesh: we used the {fft, barnes, raytrace} memory intensive and {ocean, fft, barnes} hybrid intensity multi-application workloads. For the 5x5 mesh workloads, we assigned 9 cores to a randomly chosen application, and 8 cores for the other applications.

**Table 4. Splash-2 workloads**

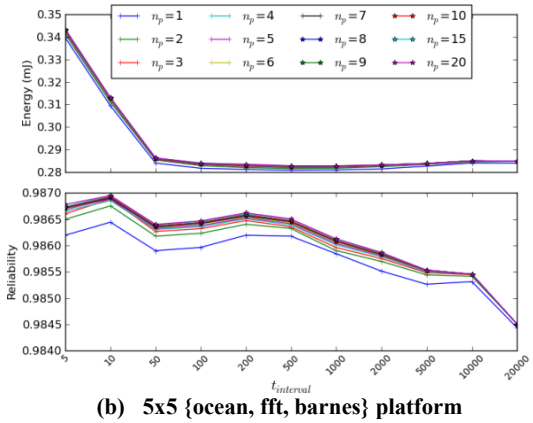
| | Applications |
|---|---|
| Compute intensive | ocean, fmm, water |
| Memory intensive | fft, barnes, raytrace |

## 6.2 RPM Parameter Sensitivity Analysis

Our first set of experiments explores the parameters that impact the operation of our runtime reliability control manager (RPM) module. There are two main design parameters in the RPM that can have an impact on the reliability and energy results: the time interval length during which vulnerability is monitored ($t_{interval}$) and the number of states in the saturating counter-based predictor ($n_p$). Figure 8 shows energy and reliability results when varying these two parameters for the EACM mapping solution with the lowest value for ($E_{network}/R_{network}$). The parameter $n_p$ is varied across a range from 1 to 20, and $t_{interval}$ values are varied across a range from 10 to 20000 cycles. Results are presented for two platforms: a 3x3 (9 core) platform with the {fft} workload and a larger 5x5 (25 core) platform running the {ocean, fft, barnes} multi-application workload.




**(a) 3x3 {fft} platform**




**(b) 5x5 {ocean, fft, barnes} platform**

**Figure 8. RPM parameter sensitivity analysis for solutions from the EACM mapping approach**

It can be observed from Figure 8 that when $t_{interval}$ is small, the overhead of frequently tabulating overall router vulnerability and more frequent decision making leads to slowdown and an increase in

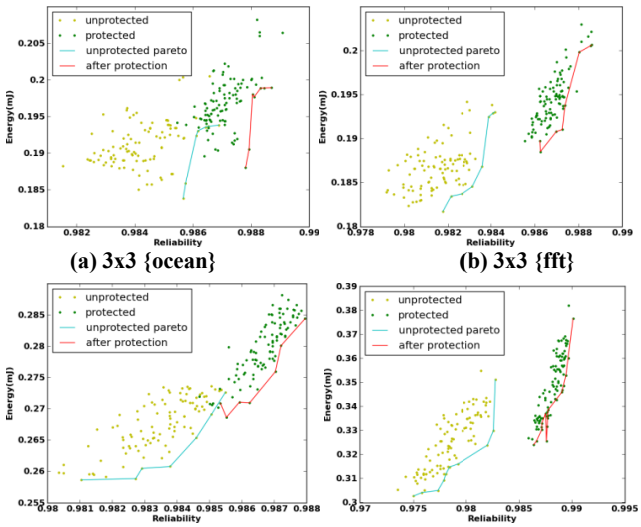
energy consumption, despite better tracking of changing traffic characteristics. As the $t_{interval}$ value increases, the overhead of RPM operation becomes more manageable, and the predictor allows opportunities for shutdown of HECC and TMR modules, resulting in a reduction in energy consumption. For very high values of $t_{interval}$, the predictor is not able to find opportunities to shutdown protection components as frequently, and therefore the energy consumption becomes larger due to inefficient runtime management of HECC and TMR modules. From a reliability perspective, high values of $t_{interval}$ end up prolonging the effect of mispredictions, which results in a reduction in reliability. For low to medium values of $t_{interval}$ the reliability is typically higher, with some variations in trends between the small and large platform sizes. We found that values of $t_{interval}$ between 200 to 500 cycles provide the best trade-off between prediction overhead and tracking effectiveness.

Figure 8 also shows the impact of varying $n_p$ values. In general, the impact of a change in $n_p$ values on energy and reliability is less pronounced than for the $t_{interval}$ parameter. For very small values of $n_p$ (1-2) both reliability and energy are lower, due to a greater bias towards energy savings over reliability. As the value of $n_p$ increases (3-5), reliability improves at a slight increase in energy consumption, due to the inherent bias of the predictor shifting form energy to reliability. For higher values of $n_p$, there is minimal impact on both reliability and energy. We found that $n_p$ values between 2 to 4 provided a reasonable tradeoff between energy and reliability.

Based on these results, we configured our RPM module with $t_{interval}$ = 300 and $n_p$ = 3. This is the RPM configuration we use when evaluating our HEFT framework in the following sections.

### 6.3 HEFT Framework Pareto Front Exploration

In this section, we explore the results generated by the HEFT framework in more detail. We were interested in understanding the value of enabling protection and RPM module based runtime management in NoC routers. Therefore we compared two scenarios: solutions generated by a core-to-die mapping framework that does not consider protection in NoC routers (i.e., a reliability un-aware synthesis framework), and solutions generated by a core-to-die mapping framework that does consider protection in NoC routers (i.e., a reliability-aware synthesis framework). For the first scenario, we only used the EACM mapping technique to generate a pool of 100 solutions, whereas for the second scenario, we additionally enabled protection and RPM-based runtime management support for the set of 100 solutions generated by EACM.




(a) 3x3 {ocean} (b) 3x3 {fft}

(c) 5x5 {fft, barnes, raytrace} (d) 5x5 {ocean, fft, barnes}

**Figure 9. Pareto front with and without protection by HEFT**

Figure 9 shows the energy and reliability results for this comparison study, for four different platform and workload combinations. The yellow dots represent the EACM-generated mapping solutions, with the blue line depicting the Pareto curve for these generated solutions. The green dots represent these same solutions after enabling protection and RPM-based runtime management, with the red line now depicting the solutions that were originally on the blue line (note: the red line is not a Pareto curve; it is simply the set of solutions that were on a Pareto front, after enabling protection support). From Figure 9, it can be clearly observed that enabling protection support boosts the overall reliability of the solution set, shifting it from the lower left quadrant up towards the upper right quadrant. There is certainly also an increase in overall energy consumption for the solutions that involve protection, which is an inevitable side-effect of increasing reliability. In general however, there is a diverse population of solutions that span a more interesting section of the design space with higher reliability guarantees, while also offering the designer flexibility to select solutions with lower energy, if lower reliability is acceptable.

**Table 5. Synthesis framework**

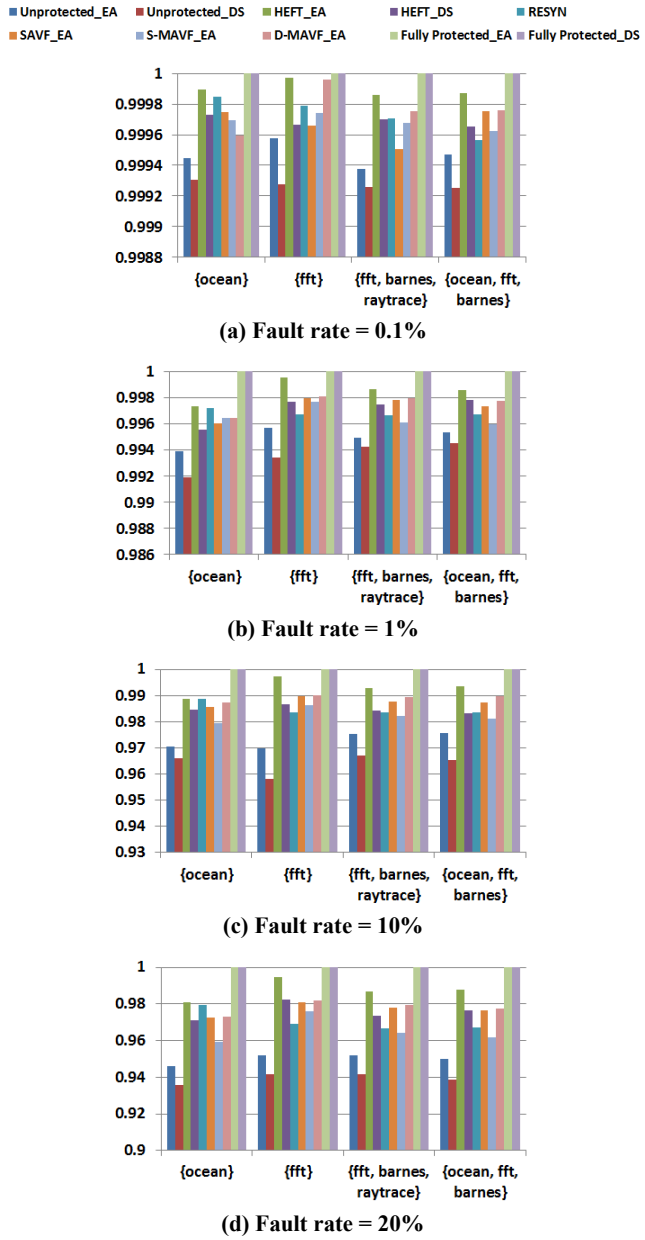
| | Mapping | Reliability configuration |
|---|---|---|
| Unprotected_EA | EACM | None |
| Unprotected_DS | DSCM | None |
| HEFT_EA | EACM | Dynamic protection |
| HEFT_DS | DSCM | Dynamic protection |
| RESYN | Genetic algorithm | GA static protection |
| SAVF_EA | EACM | SAVF |
| S-MAVF_EA | EACM | S-MAVF |
| D-MAVF_EA | EACM | D-MAVF |
| Fully Protected_EA | EACM | Full protection |
| Fully Protected_DS | DSCM | Full protection |

### 6.4 Comparison with Prior Work

In this section, we compare the effectiveness of our proposed HEFT framework with prior work in the area of reliability-aware NoC-based MPSoC design and synthesis. Table 5 shows the different synthesis frameworks that we evaluate and compare. In general, we compare various configurations derived from our HEFT framework with two prior works: RESYN [14] and "xAVFs" [4]. RESYN [14] is a design time NoC-based MPSoC synthesis framework that uses a nested genetic algorithm (GA) to perform core mapping and design time reliability configuration of NoC routers. RESYN employs an outer loop GA to explore different core mappings and with each generated core mapping solution, it uses an inner loop GA to search for unique and fixed fault tolerant configurations for all NoC routers that balance energy and reliability. "xAVF" [4] proposes runtime ECC protection management for all NoC router buffers, based solely on their utilization. Three techniques are proposed in that work: SAVF, S-MAVF, D-MAVF. SAVF statically sets the same utilization threshold for all routers in the NoC, such that if buffer utilization is higher than this threshold in a router, then ECC is enabled for all the buffers in that router. S-MAVF statically sets different utilization thresholds for routers in the NoC, based on design-time profiling, whereas D-MAVF dynamically changes the utilization thresholds for each router, based on runtime profiling. As the xAVF techniques do not discuss core mapping, for a fair comparison with HEFT, we employed the EACM mapping technique when obtaining results for each of these three NoC reliability management approaches.

We compared these prior works with the two variants of our HEFT framework: HEFT_EA which uses the EACM mapping technique, and HEFT_DS, which uses the DSCM mapping technique. In addition, we also obtained results for Unprotected_EA and Unprotected_DS which represent a subset of our HEFT

framework that use only the mapping techniques, without any NoC router protection or RPM-based management. Finally, we also obtained results for Fully_Protected_EA and Fully_Protected_DS which are subsets of our HEFT framework that use the mapping techniques but fully protect the NoC (i.e., do not employ RPM-based runtime management).




(a) Fault rate = 0.1%

(b) Fault rate = 1%

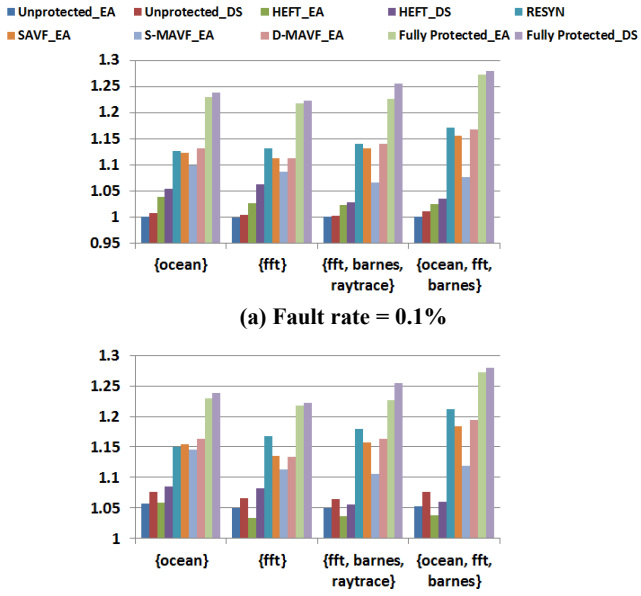
(c) Fault rate = 10%

(d) Fault rate = 20%

**Figure 10. Successful packet arrival rate comparison**

Figure 10 shows the successful packet arrival rate (along y-axes) of the generated solutions for the various frameworks listed in Table 5, for four different combinations of platforms and workloads (along x-axes), across fault rates ranging from 0.1% to 20% (Figure 10 (a)-(d)). A fault rate of 20% implies that every 5 cycles a bit fault is inserted into a randomly chosen NoC router buffer. Each bar in the figures represents results averaged over ten different trials with randomized fault insertion at the appropriate fault rate. From the results, it can be observed that, not surprisingly, protecting all buffers (Fully_Protected_EA, Fully_Protected_DS) results in the highest successful packet arrival rate. However, as will be shown later, these techniques have a high energy cost that makes them less attractive than other approaches. At the other end of the spectrum, ignoring protection for NoC routers (Unprotected_EA, Unprotected_DS) results in significantly lower reliability, which becomes more pronounced as fault rates increase.

Among the rest of the techniques, HEFT_EA can be observed to have the highest successful packet arrival rate. It turns out that optimizing for dark silicon-based shutdown with HEFT_DS is not very conducive to providing more opportunities for significantly reducing protection strengths in a subset of NoC routers. In fact, the clustering based approach used in the core-to-tile mapping in HEFT_DS detrimentally impacts traffic characteristics, resulting in reduced prediction accuracy with RPM, which in turn reduces reliability and successful packet arrival rate. Compared to RESYN, HEFT_EA can better adapt protection mechanisms dynamically to congestion and variation in traffic over time, thus avoiding scenarios where transient traffic spikes occur in NoC buffers that are left unprotected by RESYN (with this decision being made statically at design time), and cause a reduction in successful packet arrival rates in RESYN. HEFT_EA also improves upon the runtime protection mechanisms employed by SAVF, S-MAVF, and D-MAVF. The "xAVF" techniques suffer from many scenarios where either the static profiling based thresholds are too rigid (SAVF, S-MAVF) or where the dynamic prediction mechanism is unable to accurately predict buffer occupancy, compared to what the RPM is able to accomplish in our HEFT framework.




(a) Fault rate = 0.1%

(b) Fault rate = 20%

**Figure 11. Normalized energy/reliability comparison**

Figure 11 shows results for the energy/reliability metric for the various frameworks considered, across the four platform and workload combinations. Results are shown for two fault rates: a low 0.1% fault rate and a higher 20% fault rate. All energy values were normalized to the energy of the fully unprotected case and thus the energy values range from 0 to 1, similar to the range for reliability which is also from 0 to 1. Even though the fully protected approaches have the highest reliability, when considering energy costs these approaches become a lot less attractive than other approaches. For low fault rates, leaving the NoC unprotected seems to provide the best balance between energy and reliability. HEFT_EA has the lowest cost among all the other approaches. As fault rates become higher, it can be observed that HEFT_EA

provides the lowest cost and the best balance between energy and reliability compared to not just prior work but also the unprotected approaches that suffer from a steep drop in reliability. For the higher fault rates, HEFT_EA provides 8-20% improvements over RESYN [14] and 8-10% improvement over xAVF [4] approaches. Thus, HEFT_EA represents a scalable and effective system-level solution for balancing energy and reliability in emerging NoC-based MPSoCs. Our proposed framework possesses low overheads when fault rates are low and at high fault rates provides greater reliability while balancing reliability costs with energy overheads.

# 7. CONCLUSION

In this paper, we proposed a system-level framework called HEFT to trade-off energy consumption and fault-tolerance in the NoC fabric. Our hybrid framework tackled the challenge of enabling energy-efficient resilience in NoCs in two phases: at design time and at runtime. At design time, we implemented algorithms to guide the robust mapping of cores on to a die while satisfying application bandwidth and latency constraints. At runtime we devised a prediction technique to monitor and detect changes in fault susceptibility of NoC components, to intelligently balance energy consumption and reliability. Experimental results show that HEFT improves energy/reliability ratio of synthesized solutions by 8-20%, while meeting application performance goals, when compared to prior work on reliable system-level NoC design. Given the increasing importance of reliability in the deep nanometer era for MPSoCs, our work provides an important tool that can guide the reduction of energy overheads associated with reliable NoC design.

# 8. ACKNOWLEDGEMENTS

This research is sponsored in part by grants from NSF (CCF-1252500, CCF-1302693) and SRC.

# 9. REFERENCES

[1] S. Pasricha, and N. Dutt. “On-Chip Communication Architectures”, Morgan Kauffman, ISBN 978-0-12-373892-9, Apr 2008

[2] A.P. Frantz, “Dependable network-on-chip router able to simultaneously tolerate soft errors and crosstalk”, Test Conference, Oct. 2006, pp. 1-9.

[3] T. Lehtonen, P. Liljeberg, J. Plosila, “Online reconfigurable self-timed links for fault tolerant NoC”, VLSID, 2007.

[4] A. Yanamandra, S. Eachempati, N. Soundararajan, V. Narayanan, M. J. Irwin, R. Krishnan, “Optimizing power and performance for reliable on-chip networks”, ASP-DAC, 2010, pp.431-436.

[5] K. Constantinides, S. Plaza, J.Blome, B. Zhang, “Bullet Proof: a defect-tolerant CMP switch architecture”, HPCA, Feb.2006, p.5-16.

[6] S. Murali, G.D. Micheli, “Bandwidth-constrained mapping of cores onto NoC architectures”, DATE, 2004, pp. 896-901.

[7] G. Ascia, V. Catania, M. Palesi, “Mapping cores on network-on-chip”, IJCIR, 2005, pp. 109-126.

[8] K. Srinivasan, K.S. Chatha and G.Konjevod, “Linear programming based techniques for synthesis of network-on-chip architectures”, ICCD, 2004, pp.422-429.

[9] J. Hu and R. Marculescu, “Energy- and performance-aware mapping for regular NoC architectures,” IEEE TCAD, 2005, pp. 551-562

[10] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta.“The SPLASH-2 programs: characterization and methodological considerations,” International Symposium on Circuits and Systems (ISCAS), 1995, pp 24-36.

[11] N. Barrow-Williams, C. Fensch and S. Moore, “A communication characterisation of Splash-2 and Parsec”, IEEE International Symposium on Workload Characterization (IISWC), 2009, pp, 86-97.

[12] T. T. Ye, L. Benini, G. D. Micheli, “Analysis of power consumption on switch fabrics in network routers”, DAC, 2002, pp, 524-529.

[13] L.F. Leung, C.Y. Tsui, “Energy-aware synthesis of networks-on-chip implemented with voltage islands”, DAC, 2007, pp. 128-131.

[14] Y. Zou, S. Pasricha, “Reliability-aware and energy-efficient synthesis of NoC based MPSoCs”, ISQED, 2013, pp, 643-650.

[15] S.S. Mukherjee, C. Weaver, “Systematic methodology to compute the architectural vulnerability factors for a high performance microprocessor”, MICRO, 2003, p. 29-40.

[16] C. Ababei, H.S. Kia, O.P. Yadav, “Energy and reliability oriented mapping for regular networks-on-chip”, NoCS, 2011, pp. 121-128.

[17] F. Fallah, M. Pedram, “Standby and Active Leakage Current Control and Minimization in CMOS VLSI Circuits”, IEICE Transactions , 2005, pp. 509-519.

[18] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, “The SPLASH-2 programs: characterization and methodological considerations”, International Symposium on Circuits and Systems (ISCAS), 1995, pp 24-36.

[19] Y. Zou, Y. Xiang, S. Pasricha, “Characterizing Vulnerability of Network Interfaces in Embedded Chip Multiprocessors”, IEEE Embedded System Letters, 2012, pp, 41- 44.

[20] K. Srinivasan and K. S. Chatha, “A Technique for Low Energy Mapping and Routing in Network-on-Chip Architectures”, Proceedings of the 2005 International Symposium on Low Power Electronics and Design (ISLPED), 2005, pp, 387-392.

[21] C. L. Chou, R. Marculescu, “Contention-aware Application Mapping for Network-on-Chip Communication Architectures”, IEEE Intl. Conference on Computer Design (ICCD), 2008, pp, 164-169.

[22] Nirgam, http://nirgam.ecs.soton.ac.uk/

[23] G. Ascia, V. Catania, M. Palesi, “An Evolutionary Approach to Network-on-Chip Mapping Problem”, IEEE Congress on Evolutionary Computation, 2005, pp, 112-119.

[24] K. Aisoposyx, C. O. Chenx, Li-Shiuan Peh, “Enabling System-Level Modeling of Variation-Induced Faults in Networks-on-Chips”, DAC, 2011, pp, 930-935.

[25] M. Pirretti, G. M. Link, R. R. Brooks, “Fault Tolerant Algorithms for Network-On-Chip Interconnect”, IEEE Computer society Annual Symposium on VLSI (ISVLSI), 2004, pp, 46 – 51.

[26] T. Pimpalkhute, S. Pasricha, “NoC Scheduling for Improved Application-Aware and Memory-Aware Transfers in Multi-Core Systems”, IEEE International Conference on VLSI Design (VLSID), Jan. 2014.

[27] Netmaker, http://www-dyn.cl.cam.ac.uk/~rdm34/wiki/

[28] N. A. Kapadia, S. Pasricha, “VISION: a framework for voltage island aware synthesis of interconnection networks-on-chip”, GLSVLSI, 2011, pp, 31-36

[29] M. B. Taylor, “A Landscape of the New Dark Silicon Design Regime”, MICRO, 2013, pp, 8 – 19

[30] Synopsys, www.synopsys.com

[31] S. Pasricha, N. Dutt, “Trends in emerging on-chip interconnect technologies,” IPSJ Trans. on System LSI Design Methodology, 2008.

[32] S. Pasricha, N. Dutt, “A framework for cosynthesis of memory and communication architectures for MPSoC,” IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems, 26(3):408-420, 2007.

[33] Y. Zou, S. Pasricha, “NARCO: Neighbor Aware Turn Model Based Fault Tolerant Routing for NoCs”, IEEE Embedded System Letters, Vol. 2, No. 3, Sep 2010.

[34] S. Pasricha, Y. Zou, D. Connors, H. J. Siegel, "OE+IOE: A Novel Turn Model Based Fault Tolerant Routing Scheme for Networks-on-Chip", Proc. IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2010), Oct 2010.

[35] S. Pasricha, Y. Zou, "NS-FTR: A Fault Tolerant Routing Scheme for Networks on Chip with Permanent and Runtime Intermittent Faults", IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC), Jan 2011.

[36] S. Pasricha, Y. Zou, "A Low Overhead Fault Tolerant Routing Scheme for 3D Networks-on-Chip", IEEE International Symposium on Quality Electronic Design (ISQED 2011) , Santa Clara, CA, Mar 2011.