Reliability-Aware and Energy-Efficient Synthesis of NoC based MPSoCs

Yong Zou, Sudeep Pasricha
Electrical and Computer Engineering, Colorado State University
yong.zou@colostate.edu, sudeep@colostate.edu

Abstract

In sub-65nm CMOS process technologies, networks-on-chip (NoC) are increasingly susceptible to transient faults (i.e., soft errors). To achieve fault tolerance, Triple Modular Redundancy (TMR) and Hamming Error Correction Codes (HECC) are often employed by designers to protect buffers used in NoC components. However, these mechanisms to achieve fault resilience introduce power dissipation overheads that can disrupt stringent chip power budgets and thermal constraints. In this paper, we propose a novel design-time framework (RESYN) to trade-off energy consumption and reliability in the NoC fabric at the system level for MPSoCs. RESYN employs a nested evolutionary algorithm approach to guide the mapping of cores on a die, and opportunistically determine locations to insert fault tolerance mechanisms in the NoC to minimize energy while satisfying reliability constraints. Our experimental results show that RESYN can reduce energy costs by 14.5% on average compared to a fully protected NoC, while still maintaining more than a 90% fault tolerance. If higher levels of reliability are desired, RESYN can generate a Pareto set of solutions allowing designers to select the most energy-efficient solution for any reliability goal. Given the increasing importance of reliability in the nanometer era for MPSoCs, this work provides important perspectives that can guide the reduction of overheads of reliable NoC design.

Keywords

Network-on-chip, synthesis, reliability, energy

1. Introduction

As CMOS technology aggressively scales below 65nm and application complexity grows by leaps and bounds, next generation embedded processors are being driven to integrate multiple cores on a chip. Multi-processor systems on chip (MPSoCs) are thus becoming pervasive in computing systems across application domains. However these systems are becoming increasingly susceptible to transient faults (i.e., soft errors) due to a variety of factors such as higher capacitive and inductive crosstalk, elevated levels of electromagnetic noise, alpha particle strikes due to trace uranium/thorium impurities in packages, high-energy cosmic neutron particle strikes, etc. Such faults lead to single-event upsets (SEUs), causing the deposit or removal of enough charge to invert the state of a transistor, wire, or storage cell. It has been shown that SEUs will occur with even more likelihood as technology scaling continues [9], [30].

On-chip interconnect architectures constitute a single point of failure in today's MPSoCs, and are particularly susceptible to faults that can corrupt transmitted data or altogether prevent it from reaching its destination. Reliability concerns in sub-65nm nodes have in part contributed to the shift from traditional ad-hoc bus-based communication fabrics to regular network-on-chip (NoC) architectures that provide better scalability, predictability, and performance than buses [10]. Router components in a NoC fabric play the key role of routing and ensuring successful delivery of packets from the source to the destination node. Inside a router, the input buffers in each port store incoming packets. After the routing allocator (RA), virtual channel allocator (VA) and switch allocator (SA) allocate the necessary resources, packets can exit the input buffer and be sent to the allocated output port buffer. Faults in the NoC router can cause packets to be corrupted or misrouted, which can be catastrophic for applications [3].

To ensure reliable operation in NoC routers, designers often use encoding techniques such as Hamming Error Correction Codes (HECC) [1], [20], or replicate sub-components with Triple Modular Redundancy (TMR) schemes that can tolerate transient error in one of the sub-components [18], [19]. The primary emphasis is to protect input and output buffers where packets spend the majority of the time inside the router. However, such protection has an undesirable side effect: an increase in power dissipation and, consequently, energy costs. As most MPSoCs are already severely constrained by power ceilings and energy budgets (e.g., mobile devices run on batteries with limited energy capacity), the additional power and energy costs can increase time-to-market and significantly drive up costs. Thus the problem of designing a reliable NoC that is also energy-efficient is an important one. The problem of synthesizing (customizing) NoC fabrics at the system level based on application-specific goals has been addressed by many researchers [11], [12], [17], [24]-[26], where the emphasis has been on trading-off energy-efficiency and performance. However, to date, none of these efforts have integrated reliability concerns as part of their system-level NoC synthesis frameworks.

In this paper, we propose a novel NoC synthesis framework (RESYN) that, for the first time, co-optimizes reliability and energy consumption to design a robust and energy-efficient NoC fabric for MPSoCs. The major contributions of this paper are: (i) we develop a system-level power model for components in a NoC router from gate-level models at 45nm; (ii) we employ a network vulnerability factor (NVF) metric to develop a reliability model of the NoC fabric; (iii) we extract communication traces from real-world applications using full-system simulation to accurately analyze the impact of NoC customizations on real workloads; and (iv) we solve the problem of core-to-die mapping and application-specific

NoC synthesis for reliability and energy-efficiency by using a nested genetic algorithm (NGA) as part of a novel synthesis framework (RESYN). Experimental results show that RESYN can reduce energy costs by 14.5% on average, compared to a fully protected NoC, while still maintaining more than a 90% fault tolerance.

2. Related Work

The problem of synthesizing NoC communication architectures has been addressed extensively in literature. Some of the more comprehensive efforts also perform mapping of cores to a die, which can significantly improve communication power dissipation and latency, by optimizing the flow of traffic between cores. Here we briefly present some representative examples of prior work in the area. In [24], a branch and bound algorithm is proposed to map processing cores on to a mesh NoC to satisfy bandwidth constraints and minimize total energy consumption. In [25], Hu et al. proposed a multi-commodity flow based scheme to optimize NoC power consumption by customizing the topology and packet routes. In [26], Leung et al. focus on NoC implementations with voltage islands and use a genetic algorithm (GA) to find an energy-aware voltage island partitioning and assignment solution. In [11], Ascia et al. analyze and evaluate the performance of different mapping solutions in a mesh NoC topology and solve the power-performance multi-objective mapping problem with a heuristics based on GA and branch-and-bound. In [12], Murali et al. focus on the problem of mapping cores to a die, using heuristics to minimize the average communication delay. However, the focus of all these prior works to date has been on power and performance. None of the existing techniques performs core-mapping and NoC synthesis together with reliability goals, nor do they attempt to trade-off reliability with energy.

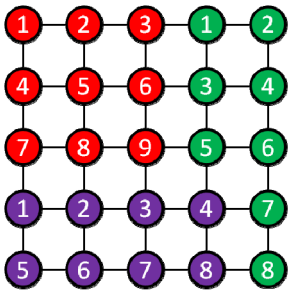

Figure 1: Example of 5x5 mesh NoC with 3 applications mapped to three unique application islands

In [10], Ababei et al. propose a branch and bound mechanism to map cores to a die for a regular mesh NoC, to balance reliability and energy efficiency. This work comes the closest to our goal of reliability-aware synthesis. However, their focus is solely on mapping cores, whereas in our work we map cores and synthesize the NoC in a manner that is cognizant of reliability, energy, and performance. Moreover, we use a more accurate methodology to estimate reliability for NoC based MPSoCs, considering the impact of fault masking, unlike [10] which ignores masking effects and thus overestimates the impact of faults.

3. Problem Formulation

In this section we present the inputs, assumptions, and objectives for our system-level synthesis problem.

Inputs:

- *n×n mesh NoC based MPSoC platform*: We assume a regular mesh NoC topology composed of a 2D regular array of identical tiles. Each tile contains a processing core (or memory) and a NoC router.
- *k applications running on the MPSoC*: Each application is set to run on a group of cores that constitute a well-defined *island* of cores in the mesh. Figure 1 shows an example of a 5×5 mesh based MPSoC on which three different applications are mapped.
- *Application constraints*: An application is defined by an application communication graph (ACG) *G(V,C)*, which is a directed graph, where each vertex $v_i \in V$ represents an IP core (processor or memory), and each directed arc $c_{ij} \in C$ represents the communication between source core v_i and destination core v_j. Each edge c_{ij} has a weight $w(c_{ij})$ to represent communication constraints. In this work, we focus on latency constraints (in terms of hop counts) that are application-specific.
- *NoC router architecture:* We consider a router architecture that contains 5 input/output ports (coming from and going to the North, South, East, West and Core directions), a crossbar switch, route-compute unit (RC), switch allocator (SA) and virtual channel allocator (VA). Figure 2 show the router architecture.
- *Protection mechanisms:* To enable fault tolerance in NoC routers, we employ two mechanisms: Hamming Error Correction Codes (HECC) and replicating sub-components with Triple Modular Redundancy (TMR). Both mechanisms can correct single bit errors in components. But unlike TMR, HECC operation entails additional latency to encode and decode data. TMR implementations however possess a larger area and power footprint, compared to HECC implementations.

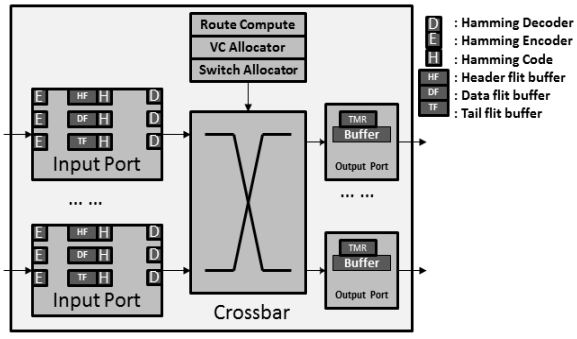

Figure 2: NoC router architecture with buffer protection

Assumptions:

- We assume that there is no communication between applications; i.e., cores only communicate with other cores within an island, and not with cores in other islands. Within an application island, all the cores communicate primarily with one or more memory nodes (tiles) via packet-switched wormhole switching.

- We assume at most a single event upset (SEU) that affects a single bit in a NoC buffer, at any given time. NoC buffers are extremely susceptible to SEUs and therefore our primary focus is to protect these buffers from SEUs that can occur unexpectedly at runtime;
- We assume that to keep implementation overheads low while still enabling fault tolerance in NoC routers, it is preferable to employ HECC for the input FIFO port buffers (that can store multiple flits), and TMR for the smaller single flit capacity output port buffers (to keep TMR implementation costs low), as shown in Figure 2.

Problem Objective: *In this work, we propose to accomplish the following objective: (i) for each application, find a mapping of cores to the 2D mesh-based die that meets application communication latency constraints; and (ii) synthesize the NoC fabric for the entire chip by customizing NoC routers at each node with appropriate protection mechanisms (discussed above) to meet a designer-specified reliability goal while minimizing energy consumption*

In the rest of this section, we discuss how we define and calculate reliability and power dissipation in the NoC fabric.

3.1. Reliability Model

Not all faults that occur on a chip eventually affect the final program outcome. For example, a bit flip in an empty translation lookaside buffer (TLB) entry will not affect the program execution. Based on this observation, Mukherjee et al. [8] defined a structure's Architectural Vulnerability Factor (AVF) as the probability that a transient error in the structure finally produces a visible error in the output of a program. At any point of time, a structure's AVF can be derived via counting the important bits required for Architecturally Correct Execution (ACE) in the structure, and dividing them by the total number of bits of the structure. Such ACE analysis has helped to derive an upper bound on AVF using performance simulation for processor buffers and caches in recent years [9].

Inspired by the AVF concept for processor and memory structures, we consider a network vulnerability factor (NVF) that aims to describe the vulnerability of structures found in NoC fabrics. Intuitively, a masking effect should certainly be present in NoC routers, but to what extent is not clear. We propose to use the idea of architecturally correct execution (ACE) bits to calculate NVF for the buffers in a NoC router, which are the key components of all NoCs. ACE bits in our case are the subsets of buffer entries that contain useful data during flit transmission. Any errors that involve these bits will cause a visible error in the final application results. In contrast, unACE bits do not affect the application results, even if soft errors cause changes to these bits. Then the NVF for a hardware component $\mathcal{H}$ of size $S_{\mathcal{H}}$ bits over an analysis window of N cycles can be expressed as

$$NVF = \frac{\sum_{i=1}^{N} (no.of\ ACE\ bits\ in\ \mathcal{H}\ at\ cycle\ i)}{N \times S_{\mathcal{H}}}$$

which allows us to calculate soft error rate (SER) for $\mathcal{H}$ as: $SER = S_{\mathcal{H}} \times (FIT/bit) \times NVF \times TVF$, where *FIT/bit* is Failure in Time per bit, and *TVF* is timing vulnerability factor which represents the fraction of each cycle that a bit is vulnerable.

There are several reasons that can lead to the presence of unACE bits in a structure that end up making it less vulnerable to soft errors. The major detectable causes of unACE bits that we found to be relevant to NoC router components in a NoC fabric are: *(i) 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, even when there are transient errors inside the buffer, as there is no critical information saved inside it at that moment, the final application result will not be affected; *(ii) Write-after-Write:* this scenario corresponds to a write-after-write event. It may so happen that a data that is propagated through the NoC router buffers is overwritten before it is used by a processor. In such a case, even if the data gets corrupted, it does not impact the correctness of the program in any way; *(iii) Read masking:* often, data that is read into a processor may not be used in its entirety. For instance, during an *xor* operation, some bits may not affect the outcome of the operation. These bits can be considered unACE. We consider all of these factors in our NVF calculations for each buffer, in each router in the NoC.

Clearly the NVF value associated with a buffer in a particular NoC router depends on the data traffic flowing through it, which in turn depends on the manner in which cores are mapped to the die and the routing algorithm used. Thus it is vital to consider the core mapping problem together with the NoC synthesis problem, which is the approach we take. The reliability value $R_{(i,j)}$ of each buffer i in NoC router j, knowing its network vulnerability factor value $NVF_{(i,j)}$, can then be calculated as:

$$R_{(i,j)} = 1 - NVF_{(i,j)}$$

Then the overall reliability R_{NoC} for the entire NoC can be expressed as the product of the reliability values of each buffer in each router in the NoC:

$$R_{NoC} = \prod_{\forall i, \forall j} R_{(i,j)}$$

We assume that there can be at most a single bit error in a NoC router buffer at any given time. So it is possible to correct the error using HECC or TMR. In other words, if a buffer is protected by an HECC or TMR mechanism, then its reliability value can be safely assumed to be 1. We use knowledge of communication flows through a router after a core-mapping and minimal routing step to generate NVF estimates for each NoC router buffer, which when combined can allow us to determine the reliability of the entire NoC.

3.2. Power model

Many research efforts make use of the Orion 2.0 [5] tool to calculate NoC router power. However, the tool does not allow us to get power overheads of protection mechanisms. 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 [7] 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 obtained the overhead of integrating these protection mechanisms into NoC routers.

Table 1 lists the average dynamic and static power values for the various NoC router components, for the 45nm CMOS process technology. We omit the power overhead for the small tail flit input buffers for brevity. As in Orion 2.0, we assume 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 the NoC router. We use knowledge of communication flows through a router after a core-mapping and minimal routing step together with NoC router sub-component power values to generate application-specific communication energy estimates. This energy value is obviously impacted by any protection mechanisms that are integrated in the NoC router.

Table 1: Router module power library (45nm)

	Input Header Buffer	Input Data Buffer	Input HECC Header	Input HECC Data	Output Buffer	TMR Output Buffer
Dynamic	216.8uw	1.36mw	425.65uw	1.51mw	45uw	267.55uw
Static	794nw	3.54uw	1.76uw	5.18uw	120nw	1.43uw
	Link	**Crossbar**	**SA**	**VA**	**RC**	
Dynamic	51.3uw	121uw	105uw	101uw	91.5uw	
Static	915nw	2.56uw	2.33uw	2.51uw	1.02uw	

4. RESYN Synthesis Framework

In this section, we present our framework for reliability-aware and energy-efficient NoC-based MPSoC synthesis. Figure 3 shows the high-level synthesis flow of our RESYN framework. The inputs consist of the application communication graph (which provides communication latency constraints), a 2D-mesh NoC platform with application islands already mapped (i.e., shape of each application island has been defined, but the cores for each application within the islands are unmapped), the NoC power library which helps estimate communication energy, and the NVF library that contains application-specific NVF data for each buffer in every NoC router. The RESYN framework employs a nested genetic algorithm (NGA) approach. The outer GA performs application core mapping to the application islands for each application, with the aim of satisfying communication latency constraints. The inner GA performs NoC synthesis for the given core mapping from the outer GA, by configuring each NoC router to meet an overall reliability constraint while minimizing energy at the system-level. In the following subsections, we provide a brief overview of genetic algorithms and then describe the NGA algorithm used in the RESYN framework.

4.1. Overview of Genetic Algorithms

Genetic algorithms (GAs) [29] generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. A GA involves the evolution of a population of individuals over a number of generations. Each individual of the population is assigned a fitness value whose determination is problem dependent. At each generation, individuals are selected for reproduction based on their fitness value. Such individuals are crossed to generate new individuals, and the new individuals are mutated with some probability. The objective of a GA is to find the optimal solution to a problem. However, because GAs are heuristics, the solution found is not always guarantied to be the optimal solution. Nevertheless, experience in applying GAs to a variety of problems has shown that often the goodness of the solutions found by GAs is sufficiently high.

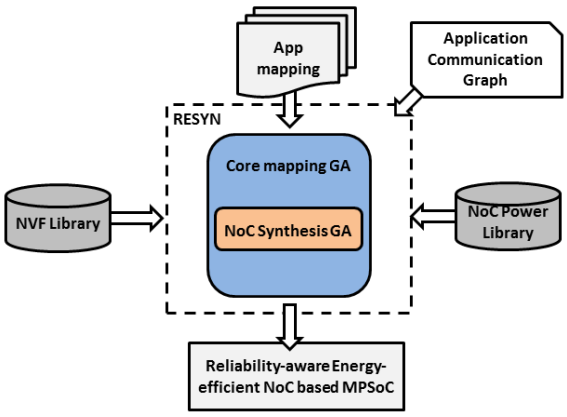

Figure 3: RESYN synthesis flow

4.2. Nested Genetic Algorithm (NGA)

The core mapping and NoC synthesis problems that we consider in this work are instances of constrained quadratic assignment problems which are known to be NP-hard [13]. We use a nested genetic algorithm (NGA) approach to accomplish the dual objectives of core mapping and NoC synthesis. In our NGA implementation, the outer loop (external) GA performs core mapping, and the inner loop (internal) GA uses the generated core mapping from the external GA to perform a search on the NoC design space, to synthesize a reliability-aware and energy-efficient NoC.

Algorithm 1: External GA

Input: Application mapping, communication latency constraints, power library, NVF library

1: Generate_core_mapping_chromosome();
2: for generation=0 to MAX_GEN_OUTER **do**
3: Internal_GA()
4: Solution_List();
5: Selection()
6: Cycle_Crossover()
7: Mutate()
8: end for

Output: core mapping solution, synthesized NoC architecture

Algorithm 1 shows the pseudo code for the external GA. Initially, we generate core mapping solution chromosomes (step 1). This chromosome is a core id array for all the cores on the die, and is encoded as:

$$coreid_{1,1}coreid_{1,2}\ldots coreid_{1,n(1)},\ldots,coreid_{m,1}coreid_{m,2}\ldots coreid_{m,n(m)}$$

where there are m applications (and thus application islands) on the die, with the number of cores in the i^{th} application given by $n(i)$. As an example, if we have a 3×3 NoC with 9 tiles, and two applications that require 4 cores (with core id's 1 to 4) and 5 cores (with core id's 5 to 9) respectively, then a mapping chromosome value of 153482976 implies that core 1 of application 1 is mapped to tile 1, core 5 of

application 2 is mapped to tile 2, and so on. Step 1 randomly generates 10 different core mapping arrays that satisfy latency hop constraints. As there are many feasible core mapping solutions that satisfy latency hop constraints, the goal is to iteratively generate new core mappings (over MAX_GEN_OUTER generations) and use the internal GA to synthesize the NoC fabric to minimize a fitness function (steps 2-8). We define the fitness function for our problem as the ratio of the NoC communication energy and the NoC reliability. Thus, minimizing the fitness value entails decreasing energy and increasing reliability, moving us towards a more desirable solution.

In step 3, we call the internal GA to calculate each chromosome's fitness value. The internal GA performs NoC synthesis to generate the NoC energy (E_{NoC}) and reliability (R_{NoC}), which allows us to calculate the fitness value of a chromosome. We use the internal GA to generate the minimum fitness values of all chromosomes. We describe the internal GA in Algorithm 2, later in this section.

In Step 4, we store the non-dominated solutions (chromosomes) in a solution list which is iteratively updated. In Step 5, we perform chromosome *selection*. A standard proportional technique is used to randomly choose whether or not drop a chromosome and generate a new one, as described next. Suppose the fitness value of the 10 chromosomes are fit_i, i=0~9. We find the sum of the 10 fitness value (fit_{sum}) and calculate the proportion of each chromosome's fitness value within the sum ($rfit_i$, i=0~9). We then calculate a cumulative number for each chromosome ($cfit_i$) such that $cfit_0 = rfit_0$, $cfit_i = cfit_{i-1}+rfit_i$. We randomly generate ten numbers $rand_i$ between 0 and 1, with i=0~9. If $rand_0<cfit_0$, then chromosome 0 is kept. If $cfit_{i-1}<rand_i<=cfit_i$, then we keep the chromosome i. We replace chromosomes that are discarded with new randomly generated chromosomes that satisfy latency constraints.

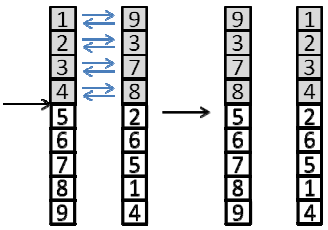

Figure 4: Traditional crossover operation

In Step 6 we perform *crossover*. Figure 4 shows a traditional crossover operation, where a crossover point is selected, and then all the entries above the point are switched between two parent chromosomes. In our core mapping problem, we note that after this crossover, in chromosome 1 core 9 is mapped twice and core 1 is not mapped at all, which creates an invalid solution. To avoid such a scenario, we replace the traditional crossover with the cycle crossover operator [27] which does allow us to generate valid crossover solutions.

After the cycle crossover we check whether the two new chromosomes that have been generated meet the latency constraint. If they both do, then we replace the two original chromosomes with the new ones. If both new chromosomes do not meet latency constraints, then we keep the original two chromosomes and move to the next step. If only one of the new chromosomes can meet the latency constraint, we replace one of the original chromosomes that has the higher fitness value with the new one. Finally, in Step 7 we perform *mutation*. For each chromosome, we randomly generate a number between 0 and 1. If the number is smaller than 0.2, then we perform a mutation on the current chromosome. This mutation is performed by randomly selecting two cores within an application island and switching them in the current chromosome. After the outer GA terminates, we obtain a final set of solutions that trade-off reliability and energy efficiency.

Algorithm 2: Internal GA

Input: Core mapping, NVF library, power library

1: Build_chromosome_routing_mask()
2: Generate_noc_synthesis_chromosome()
3: for generation=0 to MAX_GEN_INNER **do**
4: Evaluate_fitness_value();
5: Selection();
6: Crossover();
7: Mutate();
8: end for

Output: NoC configuration with minimum fitness function value

Algorithm 2 shows the pseudo code for the internal GA. For a given core mapping configuration, this internal GA explores various configurations of NoC routers, with the goal of finding a configuration that minimizes the fitness function value (E_{NoC}/R_{NoC}), as defined earlier. The encoding of the chromosome for the internal GA is shown below, where g_{ijk} is a gene which represents whether buffer k in port j of NoC router i is protected or not.

$$g_{110}g_{111}g_{112}g_{113}\cdots g_{ij0}g_{ij1}g_{ij2}g_{ij3}\cdots g_{(n*n)j0}g_{(n*n)j1}g_{(n*n-1)j2}g_{(n*n)j3}$$

$$g_{ijk} = \begin{cases} 0: router\ i\ port\ j\ buffer\ k\ is\ not\ protected \\ 1: router\ i\ port\ j\ buffer\ k\ is\ protected \end{cases}$$

$$k = \begin{cases} 0: input\ header\ buffer \\ 1: input\ data\ buffer \\ 2: input\ tail\ buffer \\ 3: output\ buffer \end{cases}$$

In Step 1, based on the core mapping configuration from the external GA, we establish minimal routing paths between communicating cores and build a mask array based on the routing paths. The mask is similar to the chromosome except that for the NoC router buffers that do not participate in any communication, the corresponding entry in the chromosome is set to 0. In step 2, we randomly generate 10 chromosomes (that represent solutions with varying degrees of protection across NoC router buffers on the chip), masked by the mask array created in step 1. Steps 3~8 attempt to perform traditional crossover, in addition to mutation and selection of chromosomes in a manner similar to the external GA. After MAX_GEN_INNER iterations, the best NoC configuration with the lowest fitness value is returned.

Ultimately, after the external GA finishes, we obtain a set of solutions, each with a unique mapping of application cores to their respective application islands on the die and

with NoC routers uniquely configured with HECC and TMR protection mechanisms, to trade-off reliability and energy efficiency for on-chip communication.

5. Experimental Results

5.1. Experimental Setup

Figure 5 shows the setup we used to generate the data needed for the RESYN synthesis framework. In our experiments we considered combinations of three SPLASH-2 benchmark applications [2] mapped to a mesh of 25 (5×5) ALPHA processors, each with 64kB L1 instruction and data caches. We assume that tiles in the mesh have already been allocated to applications (i.e., application islands are assumed to be given), based on prior designer analysis, e.g., proximity of an applications' nodes to specific I/O pins. We also assume that inter-core latency constraints (in terms of hop counts) for each application are given, based on pre-computed criticality analysis of communication flows.

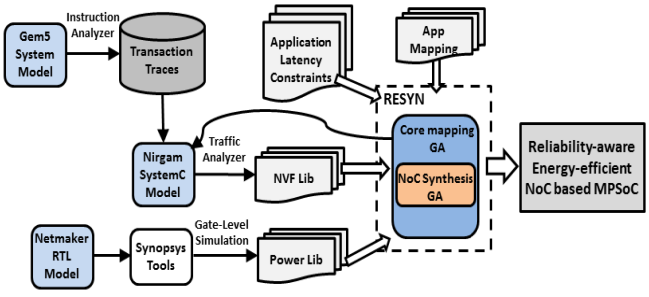

Figure 5: Experimental setup for RESYN synthesis flow

We use a modified Gem5 full system simulator [21] with the SPLASH-2 benchmark applications to generate instruction traces for each core. We parsed 50 million "memory read" and "memory write" instructions from each application, and used the Nirgam SystemC-based NoC simulator [28] for trace-driven simulation, to generate NVF values for NoC router buffers for each core mapping configuration. Every time the core mapping is changed, this simulation is run again to update the NVF library (as different core mappings cause changes in traffic flow, which in turn changes the NVF value of NoC router buffers).

We use Synopsys VCS [22] to run simulations with RTL models of the NoC built by modifying the Netmaker library [23], to get signal traces. We also perform logic synthesis using Synopsys Design Compiler, and gate level power analysis using Synopsys Primetime to generate the power library for NoC router sub-components and integrated protection mechanisms. This power data is used to calculate communication energy, once a core mapping and routing paths for communication flows are established.

5.2. RESYN Comparative Analysis

We compare the solutions generated by RESYN with two other design alternatives: *(i) unprotected*, which refers to an MPSoC with latency-aware core to die mapping but without any protection mechanisms in router architectures; and *(ii) fully protected*, which refers to an MPSoC with latency-aware core to die mapping, and with all NoC router buffers protected (HECC on input buffers, TMR on output buffers).

Figure 6 shows the energy costs for the three approaches, for three different combinations of SPLASH-2 benchmark applications. For RESYN, we select the solution with the minimum energy that still meets at least 90% reliability (i.e., R_{NoC} >= 0.9). The results are shown relative to the energy consumption of the *fully protected* case. It can be seen that energy consumption of the unprotected case is the least, which is not surprising, considering that this approach does not protect NoC router buffers from transient faults. Compared to the energy consumption of the fully protected case, our RESYN framework enables savings of 14.5% in communication energy costs on average.

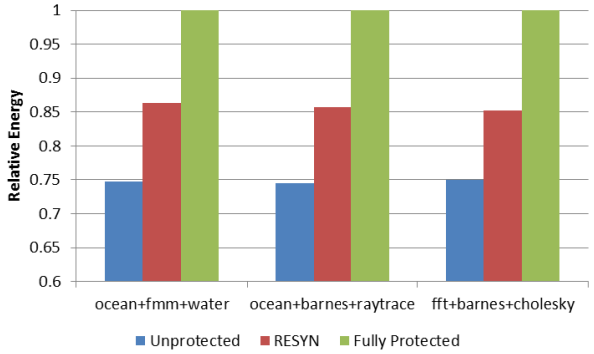

Figure 6: Energy cost comparison for RESYN with *unprotected* and *fully protected* design alternatives

Figure 7 shows the reliability values for the same solutions for which energy costs are shown in Figure 6. It can be seen that the unprotected case is not very viable, given its low reliability in the face of transient faults. The solutions generated by RESYN all have a reliability of greater than 90%. The results in Figures 6 and 7 demonstrate how RESYN allows a convenient mechanism for trading off reliability with energy at the system-level, allowing designers to minimize energy given a reliability goal. Techniques such as end to end ACK/NACK flow control can be utilized to complement the NoC fabric generated by RESYN, to ensure that no error goes undetected.

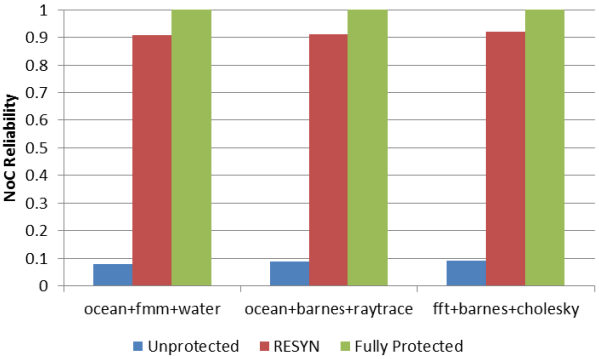

Figure 7: Reliability comparison for RESYN with *unprotected* and *fully protected* design alternatives

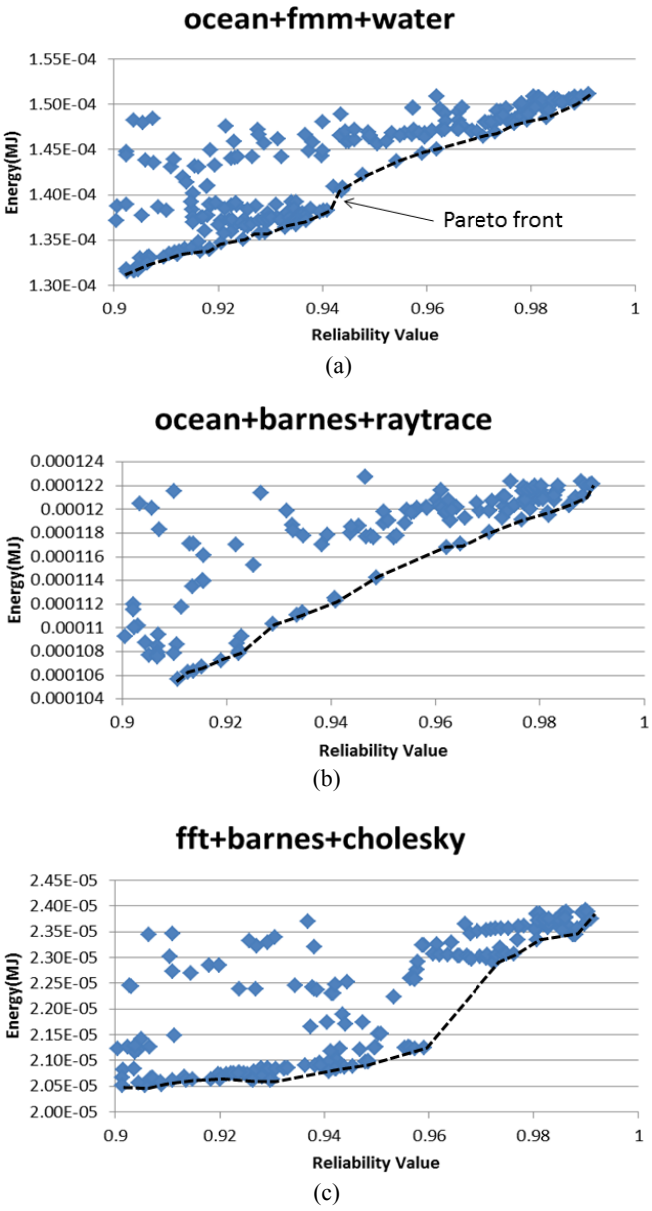

Figure 8: Generated solution set using RESYN with Pareto front for the three applications (a) ocean+fmm+water, (b) ocean+barnes +raytrace, (c) fft+barnes+cholesky

In some scenarios, a reliability goal of 90% may be too low, and designers may require the ability to trade-off reliability with energy efficiency over a broad spectrum of higher reliability values. Figures 8(a)-(c) show the solution set generated by RESYN for the three different combinations of applications that we study. The figures show the energy and reliability of several solutions whose reliability varies from 90% to 100%. Designers can then select the appropriate energy-efficient solutions along the Pareto front, for any desired value of reliability. Even for higher values of reliability (i.e., more stringent reliability constraints), it is possible to get significant energy savings. For example, for a 95% reliability constraint in *fft+barnes+ cholesky*, RESYN generates a solution that still saves a significant 12% energy consumption over the fully protected case. RESYN thus represents an invaluable system-level tool for designers in the era of nanometer design uncertainty, providing a design space exploration environment that enables comprehensive trade-offs between reliability and energy-efficiency for NoC based MPSoC applications.

6. Conclusion

In this paper, we propose a novel design-time framework (RESYN) to trade-off energy consumption and reliability in NoC-based MPSoCs. RESYN employs a nested genetic algorithm (NGA) approach to guide the mapping of cores on a die, and opportunistically determine the locations to insert fault tolerance mechanisms in NoC routers to minimize energy consumption while satisfying reliability constraints. Our experimental results show that RESYN can reduce communication energy costs by 14.5% on average compared to a fully protected NoC design, while still maintaining more than a 90% fault tolerance. RESYN also enables a comprehensive trade-off between reliability and energy-efficiency for more stringent reliability constraints, generating a Pareto set of solutions which may reveal opportunities for energy savings even for high reliability configurations; e.g., for a higher 95% reliability constraint, a notable 13% energy savings can still be achieved for some applications. Given the increasing importance of reliability in the nanometer era for NoC-based MPSoCs, this work provides important perspectives that can guide the reduction of energy overheads associated with reliable NoC design.

References

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

[2] 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.

[3] M.H. Neishaburi, Z. Zilic, "ERAVC: Enhanced reliability aware NoC router", ISQED, Mar. 2011, pp. 1-6.

[4] F. Refan, H. Alemzadeh, S. Safari, P. Prinetto, Z. Navabi, "Reliability in application specific mesh-based noc architectures", IOLTS, 2008, pp. 207-212.

[5] A. Kahng, et al., "ORION 2.0: A fast and accurate noc power and area model for early-stage design space exploration" DATE, 2009, pp. 1-6.

[6] T.T. Ye, L. Benini, G.D. Micheli, "Analysis of power consumption on switch fabrics in network routers", DAC, 2002, p. 524-529.

[7] Synopsys DC and Primetime, www.synopsys.com

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

[9] N.J. George, C.R. Elks, "Transient fault models and AVF estimation revisited", Dependable Systems and Networks (DSN), 2010, pp. 477-486.

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

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

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

[13] M.R. Garey, D.S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness", W. H. Freeman & Co. New York, NY, USA, 1990.

[14] J.C. Hu, R. Marculescu, “Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures”, DATE, 2003, pp. 688-693.
[15] Y.F. Hu, Y. Zhu, H.Y. Chen, R. Graham, “Communication latency aware low power noc synthesis”, DAC, 2006, pp. 574-579.
[16] Y.F. Hu, H.Y. Chen, Y. Zhu, A.A. Chien, C.K. Cheng, “Physical synthesis of energy efficient networks on chip through topology exploration and wire style optimization”, ICCD, 2005, pp.111-118.
[17] K. Srinivasan, K.S. Chatha and G.Konjevod, “Linear programming based techniques for synthesis of network-on-chip architectures”, ICCD, 2004, pp.422-429.
[18] A. Yanamandra, S. Eachempati, N. Soundararajan, “Optimizing power and performance for reliable on-chip networks”, ASP-DAC, 2010, pp.431-436.
[19] K. Constantinides, S. Plaza, J.Blome, B. Zhang, “Bullet Proof: a defect-tolerant CMP switch architecture”, HPCA, Feb.2006, p.5-16.
[20] T. Lehtonen, P. Liljeberg, J. Plosila, “Online reconfigurable self-timed links for fault tolerant NoC”, VLSID, 2007,
[21] N. Binkert, et al., “The gem5 simulator,” SIGARCH Comput. Archit. News 39(2), 2011, pp. 1-7.
[22] Synopsys, VCS®/VCSi™ User Guide 2011
[23] Netmaker, http://www-dyn.cl.cam.ac.uk/~rdm34/wiki/
[24] J.C. Hu, R. Marculescu, “Energy-aware mapping for tile-based NoC architectures under performance constraints”, ASP-DAC, 2003, pp. 233-239.
[25] Y.F. Hu, H.Y. Chen, “Physical synthesis of energy-efficient networks-on-chip through topology exploration and wire style optimization”, ICCD, 2005, pp. 111-118.
[26] L.F. Leung, C.Y. Tsui, “Energy-aware synthesis of networks-on-chip implemented with voltage islands”, DAC, 2007, pp. 128-131.
[27] A.E. Eiben, J.E. Smith, “Introduction to evolutionary computing”, Springer, 2008.
[28] Nirgam, http://nirgam.ecs.soton.ac.uk/
[29] D. Goldberg, “Genetic algorithms in search, optimization and machine learning,” Kluwer Academic Publishers, Boston, MA, 1989.
[30] S. Shamshiri, A. Ghofrani, “End-to-End Error Correction and Online Diagnosis for On-Chip Networks”, ITC, 2011, pp. 1-10