A Flexible and Parallel Simulator for Networks-on-Chip with Error Control

Qiaoyan Yu, Student Member, IEEE, and Paul Ampadu, Member, IEEE

Abstract—We present a fast parallel simulator to evaluate the impact of different error control methods on the performance of networks-on-chip (NoCs). The simulator, implemented with message passing interface (MPI) language to exploit multiprocessor server environments, models characteristics of intellectual property (IP) cores, network interfaces (NIs), and routers. Moreover, different error control schemes can be inserted to the simulator in a plug-and-play manner for evaluation. A highly tunable fault injection feature allows estimation of the latency, throughput and energy consumption of an error control scheme against different fault types, fault injection locations and traffic injections. Case NoC studies are presented to demonstrate how the simulator assists in NoC design space exploration, evaluating the impact of different error control methods on NoC performance, and providing design guidelines for NoCs with error control capabilities.

Index Terms—Fault tolerance, reliability, performance analysis and design aid, simulator, error control, networks-on-chip.

I. INTRODUCTION

NETWORKS-ON-CHIP (NoCs) provide an elegant approach to managing interconnect complexity [1]-[5], facilitating integration of heterogeneous intellectual property (IP) cores by separating computation and communication [6]. Fig. 1 shows the three essential components of NoCs—links, network interfaces (NIs), and routers. Links facilitate communication between routers: NIs transform streams of bits from IP cores into packets for transmission to routers and vice versa; routers extract the destination address from each received flow control unit (flit) and pass the flit to its intended destination.

Fig. 1. Key components of a network-on-chip

Because of their modular design and regular topologies, NoCs achieve better scalability and more predictable electrical parameters than traditional buses [7]-[9]. Packet switching and intelligent routing algorithms improve NoC throughput and latency at the expense of a minor area penalty [1], [9].

NoCs are a good solution to managing integration of a large number of cores in nanoscale systems. In these deeply scaled technologies, reliability emerges as a critical parameter [38], [40], as increased coupling noise affects information transmission on the NoC links [10]. The reduced critical charge accompanying technology scaling makes registers and logic circuits more susceptible to noise [7], [8], [11], while process and environmental variations exacerbate delay uncertainties [8]. Thus, error control methods are needed to improve reliability.

Incorporating error control in NoCs leads to performance degradation, energy consumption increasing and area overhead. Consequently, NoC simulator that capable of assessing error control methods is needed. Adding error control feature to an existing NoC simulator is not as simple as including an error control module. Instead, the flow control, routing algorithms, and buffer management need modification to support error control. Moreover, various noise condition modeling are required to facilitate the extensive evaluation.

II. BACKGROUND

A. Previous Simulators

General network simulators (e.g. ns-2) were previously used to estimate NoC performance [14]-[16]. Recently, many research groups have proposed specific modeling methods and frameworks to assist in NoC development. Orion is proposed to evaluate network power and latency with different traffic patterns and flow control parameters [17]. On-chip communication network (OCCN) [18] employs a multi-layer NoC modeling methodology; furthermore, many application programming interfaces (APIs) are well defined in OCCN to assist communication between different layers. The framework for power/performance exploration (PIRATE) can be used to examine the impacts of different topologies and traffic injection rates on the average throughput and power for on-chip interconnects [19]. A system-level NoC modeling framework integrates a transaction-level model and an analytical wire model to evaluate throughput and power of NoCs [21]. Thid [22] introduced a layered-network simulator kernel, named Semla. Using the Semla kernel, Lu et al further proposed the
Nostrum NoC simulation environment (NNSE) to explore the design space for Nostrum NoC [23] [24]. Commercial design kits (e.g. NoCexplorer and NoCcompiler [25] proposed by Arteris) allow defining NoC interfaces, quality of service (QoS) requirements, and topologies; they are also capable of estimating area and performance in the network. Various routing algorithms, NoC sizes and arbitrary traffic injection rates are supported in the NoXim NoC simulator [26]. Different QoS levels and parameters for network traffic are investigated in the Nirgam NoC interconnect routine and application modeling tool [27]. Those simulators are useful to explore design space and power estimation; however, they have not explored error control features in NoCs.

B. Previous Platform for Evaluation of Error Control Scheme

To improve reliability of on-chip communication, NoCs have employed various error control methods [10]-[13], [28]-[36]—forward error correction (FEC), error detection combined with retransmission (using automatic repeat request (ARQ) protocol), and hybrid ARQ. Retransmission protocols includes stop-and-wait, go-back-N and selective repeat fashions. Most of the work [11], [13], [28], [30]-[32] [34]-[36] evaluate the error control scheme in a point-to-point communication architecture, without considering the impact of traffic injection on performance and energy efficiency. Although realistic traffic injections have been considered in the evaluation in [10], [12], the probability of fault presence is used to estimate the energy consumption and the fault injection location is limited at the link between IP core and router. In [29], Ali et al have modified the ns-2 simulator to assess their fault tolerant routing protocol; unfortunately, only fault injection rate is tunable in their simulation. Indeed, researchers have noticed that error on header or payload flits of a packet need different error protection strength [30], [31]. Consequently, it is urgent to develop a comprehensive NoC simulator that allows evaluation of error control schemes against different traffic injection, fault injection rate, fault types and faulty flit property.

C. Contributions of This Work

To fill the gap between NoC simulator implementation and NoC error control exploration, we develop an NoC simulator that allows more comprehensive investigation of the impact of different error control methods on NoC performance and energy consumption. The proposed simulator integrates plug-and-play error control coding (ECC) insertion, a variety of different error recovery methods, and various fault injection scenarios. The flexibility provided by the proposed simulator fastens the evaluation of different error control schemes in NoCs.

The powerful simulation environment potentially requires unaffordable simulation time and system resources, which becomes worse and worse as the number of NoC nodes increases [24]. To resolve this challenge, we use C and message passing interface (MPI) languages to schedule a parallel simulation on a multiprocessor server, which dramatically improving the speed of Monte Carlo simulations and improve the simulation flexibility. We also propose an index-based method to inject faults with different injection rates and duration in a memory-efficient way. Because of our parallel simulation framework, the multiple independent fault injection locations are supported, as well.

Using a multiprocessor server environment, each IP core, router and network interface (NI) can be modeled independently. As a result, the performance of a NoC-based system comprised of heterogeneous IP cores can be efficiently evaluated.

The remainder of this paper is organized as follows: Section II presents the simulation flow and modeling methods. In Section III, energy estimation and improvements on simulation speed and memory consumption are analyzed. Experimental results are provided in Section IV. Conclusions and future directions are presented in Section V.

III. SIMULATOR IMPLEMENTATION

A. Simulator Overview

Fig. 2 shows the four steps to perform a simulation in our proposed simulator: (1) compile simulator source codes; (2) specify an NoC and initialize the simulation; (3) run simulation and obtain key metrics; (4) invoke simulator accessories to process the internal output data.

Plug-and-play error control coding insertion is realized by adding an ECC module before the first step. Two free compilers are available for simulator source codes—gcc for C files compilation; mpicc for MPI and C files (parallel codes running on multi-processor server) compilation. Options makefile and other (shell commend), make the compiling easier, if some special libraries (e.g. GNU Scientific Library [37]) are needed.

In the second step, a parameterized NoC is emulated in the simulator, in which one can define the number of NoC nodes (NoC size), select an NoC topology, choose a routing algorithm and retransmission protocol, and assign the round-trip delay for switch-to-switch retransmission. Two popular NoC topologies—mesh and torus, are available in the simulator. Deadlock-free XY deterministic routing and north-last partially adaptive routing [Refs] are built-in. In the future, more routing algorithms will be included, but the simulator is also simple to expand by adding user-generated routing modules. The retransmission protocols include stop-and-wait and go-back-N ARQ and HARQ. In addition to specify an NoC, total simulation time, traffic pattern of each IP core, fault type, fault injection location and fault duration are user inputs (the second step). Using those parameters, our simulator emulates an NoC and creates packets injected by IP cores, as well as a fault injection profile. Fig. 3 shows more details for traffic and fault injection. Traffic injection in IP cores is further described in Section II.B. The index-based fault injection method that efficiently generates fault vectors is discussed in Section III.

The desired output metrics (such as average latency, throughput, reliability and energy—shown in Fig. 2(a)) are displayed as soon as the simulation finishes in the third step.
The output files may be individually named for external data processing. One example of available internal data is the switching factor for the error control encoder/decoder, buffers, crossbar switch, and routing block. In addition to manually run simulation, scripts can be used to run multiple Monte Carlo simulations.

In the last step, several simulator accessories can be invoked to process the customized internal data obtained in the previous step. NoC visualizer (Fig. 2(b)) is used to visually show the data flow over the emulated NoC. GUIDE (Fig. 2(c)) is a MATLAB tool to process the obtained data and plot statistic curves.

The NoC framework modeled in the proposed simulator consists of routers, network interfaces (NIs), and IP cores. These three components can be connected using a number of different topologies. In this paper, we are interested in the tours topology, which has been shown to be one of the most energy-efficient topologies [2]. The operation of each component can be modeled in a physically parallel manner using a multiprocessor server (e.g., TeraGrid IA-64 [41]). Thus, the concurrent operation of routers can be achieved; each router/IP core can be independently controlled in simulation, rather than employing the same error control to the overall design. This feature is particularly useful because IP cores are typically heterogeneous in NoC-based CMP and have different characteristics (e.g., data format, send/receive data, error resilience requirement, and I/O bandwidth).

B. Error Control Modeling in Router and Network Interface

In torus NoCs, the router typically has five ports—north, south, east, west (connected to neighboring routers) and local (connected to an IP core)—plus a crossbar switch for routing and input/output ports connections. Each port has one input channel and one output channel, consisting of incoming (and outgoing) data buffers and error control modules. The crossbar extracts the destination address from the received packet and directs the packet to the appropriate output channel for its next hop. Meanwhile, the crossbar also detects the availability of the destination port (i.e., sink) to avoid buffer overflow. If a resource contention occurs, a round-robin channel-reservation method is employed to ensure the fairness of port-accessing.

1) Error Control in Router

Unlike previous simulators (e.g., [17], [19], [23], [26], [27]), the proposed NoC simulator embeds an ECC module in routers and NIs to support different error control schemes, including various forward error correction (FEC), error detection combined with ARQ and HARQ. The ECC module is extendable. New error control codec can be easily added to the ECC module in a plug-and-play fashion. For easy-to-use purpose, typical error control codes have been provided in the ECC module, such as even-parity check code (PAR), Hamming code (HM), extended Hamming code (EHM), cyclic redundancy code (CRC), duplicated-add-parity (DAP) [Refs], SEC-DED code [Refs] and configurable error control codes [Refs]. Users can specify their desired error control codec using a global parameter or convey ECC selection information in the header flit of a packet. The various error control strategies are integrated with flow control, routing algorithm and buffer management, as well. To evaluate the performance of an NoC without error control, ECC-related features can be easily disabled using bypass paths.

Error detection/correction is executed in the input channel, which accepts data from neighboring routers or its local IP core via the Flit_in vector if no error is detected. As shown in Fig. 4(a), the input channel first executes ECC decoding if the incoming buffer is not full yet; otherwise no decoding is necessary. Only the “error-free” data can be propagated to the buffer and then be forwarded to the next hop. Here, the term “error-free” has different meaning from the viewpoint of different error control methods. For the case of no ECC or FEC, all the received data are treated as error-free no matter whether error bits exist or not, since no retransmission is allowed. In contrast, for the case of ECC combined ARQ, flits that without
errors or containing undetectable errors will be flagged as “error-free”; otherwise, $Err$ is set to high and the buffer control module is informed of the this error event. The HARQ scheme asserts $Err$ only when the detected error cannot be corrected by the current decoder. Because of ECC insertion, writing to the buffer has one more constraint that the input data should be error-free, in addition to available buffer space and the permission of using the headed output port. The non-propagated flit requests retransmission via NACK signal.

The output channel processes the incoming data in a similar but opposite fashion. As shown in Fig. 4 (b), the ECC encoder is placed after the buffer, so that a smaller buffer is needed because the check bits of the coded flit are not stored. The buffer structure in the output channel changes in different error control schemes. For instance, the error recovery scheme using go-back-N retransmission is easier to implement with a circular shifter, so the previous flit can be retransmitted at the moment that the feedback NACK arrives. If no retransmission is needed in the employed error control scheme, traditional FIFO buffers work well for buffering incoming flits from the crossbar switch. Consequently, our simulator provides multiple buffer structures for different error control schemes. Note that the module describing the buffer structure is extendable, too. Thus, the exploration of new buffer structures is supported.

The crossbar switch creates an interconnect path between the input and output channels, according to the address field indicated in the packet header and the routing algorithm employed. As shown in Fig. 4(c), the crossbar has five destination-port computation blocks ($Dest_{port\ Comp}$), one for each port. The $Dest_{port\ Comp}$ block detects the header/tail flit and computes the intended output port ID based on the employed routing algorithm. If a header is detected, a port-reservation request is sent to the destination port reservation block ($Dest_{port\ Reservation}$), in which the algorithm that resolves network contention can be explored. In the simulator, we implement a round-robin method to achieve the fairness of accessing output ports. The port reservation is cancelled as soon as the tail flit of the current packet is successfully transferred.

2) Error Control in Network Interface

The network interface (NI) is the gateway between IP cores and network. As shown in Fig. 5, NIs have only one pair of input and output channels and do not have a crossbar switch. In addition, NIs use buffers to manage the frequency difference between IP cores and NoC links switching. In packet switching NoCs, NIs packetize the data stream into packets with user defined packet length and provide each packet with a header and a tail. In our simulator, we assume the packet format as shown in Fig. 6. A header flit includes information such as source address ($Src\ ID$), destination address ($Dest\ ID$), error control coding method and routing algorithm. Payload and tail flits carry data. The tail flit has a tail bit to indicate the end of a packet.

End-to-end ECC ($End\ ECC$) is usually executed on the original packets before packetization/after depacketization; hop-to-hop error control ($Hop\ ECC$) is executed in the input/output channel, which is similar to routers. Typically, the code used in end-to-end ECC is more powerful than that in hop-to-hop ECC, because the former one is used in a forward error correction manner. Unlike end-to-end ECC, simple code combined with retransmission is more efficient for hop-to-hop ECC [Refs]. Both end-to-end and hop-to-hop ECC can be disabled by bypassing the data path.

C. Flexible Fault Injection and Traffic Injection

The fault injection profile is generated based on four key parameters—fault injection location, fault type, faulty flit property and fault injection rate. According to those parameters, various noise scenarios can be modeled to evaluate different error control schemes.
1) Fault Injection Location

In this simulator, the faults occurring on interconnect links are interested. Fault injection location is used to differentiate router-to-router interconnects from router-to-NI interconnects. Global link fault represents a fault existing in the global link between routers; local link fault is the fault injected on the link between a router and a module of IP core combined with NI. These two locations for fault injection are depicted in Fig. 7.

In the simulator interface, one can select either a global link or a local link, or both. The parameter, All_Links shown in Fig. 2(a) indicates all the links suffering from faults with the same fault injection rate; the parameter, Link_IDs (shown in Fig. 2(a)), is used to specify the number of the links that are affected by faults. According to Link_IDs, our simulator randomly chooses one set of links, on which faults will be injected in Monte Carlo simulation. During the simulation, each link reads its fault injection profile (if applicable) and manages the faults in the way indicated by the employed error control scheme. Local link faults are detected at the input channel of NI and the local ports in routers. Global link faults are examined in the input channels of east, south, west and north ports of routers.

2) Fault Type

The proposed simulator is capable of simulating four fault types: (1) transient independent faults; (2) permanent independent faults; (3) transient coupling faults (leading to multi-bit errors); (4) permanent coupling faults. The fault type parameter describes the fault lasting time and the number of bits that a fault event affects on the link. The lasting time of the transient fault is defined by fault duration. Permanent fault exists on the link through the entire simulation. For the independent faults, the time that a wire is injected a fault is independent. For the coupling faults, one fault affects multiple neighboring wires.

3) Faulty Flit Property

Faulty flit property means whether the faulty flit is header, payload or tail. Faulty flit property is used to investigate the impact of erroneous-bit position on the reliability, performance and energy consumption of an NoC.

Uncorrectable error on the header flit leads to packet lost and wrong destination. If the fields of Hop ECC and End ECC are incorrect, the link noise condition may be exaggerated. The residual errors on the routing fields result in the performance being less than expected. Unrecognized errors on the tail flit make the packet transmission endless, which prevents the following packet passing through the router. Consequently, the average latency and throughput is reduced.

4) Multiple Traffic Patterns

The IP core emulated in our simulator acts like a traffic generator, which provides the data streams injected to the NoC. Following the flowchart shown in Fig. 3, we create a traffic engine to generate sufficient input streams that replicate the characteristics of different random processes and random distributions, as well as a wide range of packet injection rates.

We can further obtain the input streams with different clock frequencies, emulating an NoC-based CMP system. Based on the input file generated by our traffic engine, we insert invalid flits among the valid data to model different clock frequencies.

Assume that the switching frequency for the NoC link is $f_L$, and the frequencies for IP core_0, IP core_1 and IP core_2 are $f_L$, $1/2f_L$, and $1/3f_L$, respectively. The original data stream is directly injected to the NoC through the network interface. For the IP core running on a slower frequency, invalid packets are inserted before the valid packet enters the network.

As shown in Fig. 8, for an IP core operating at half of the link frequency, one invalid packet is inserted before the valid packet. If the IP core frequency is higher than the link frequency, no modification is needed at this point because the store and forward mechanism applied in the network interface will manage the unmatched frequency. For simplicity, we restrict ourselves to cases where the link period is an integer times the IP core period. The adjustment for non-integer rates frequency will be accomplished in future.

D. Parallel Fault Injection and Traffic Injection

The behavior of router or the module of IP core combined with NI is described in C and MPI language, which allows each microprocessor in a multiprocessor server representing a router or IP core. Thus, all the routers and IP cores can operate in parallel. MPI_Send and MPI_Recv functions perform a standard-mode blocking send and receive, modeling the on-chip communication between routers and IP cores.

1) Fault Injection on Links

For an NoC without error control, one set of MPI_Send/Recv is sufficient to deliver flits (Flit_s/r). To facilitate investigating ECC feature, one more MPI_Send/Recv function is needed to transfer error control feedback (NACK_s/r), as shown in Fig. 9(a). Sink and source processor IDs are indicated by DestID and SrcID, respectively. In the server, message transferred by the MPI_Send is first saved in the processor buffer until the matched MPI_Recv fetches it, as shown in Fig. 9(b). If a fault is injected for the purpose of evaluating ECC, one or some bits of the flit fetched by MPI_Recv from the microprocessor buffer are flipped using XOR logic ‘1’. The
exact amount of the inverted bits and the time period the fault lasting depend on the fault vector given by a fault injection profile (faultinjection.dat shown in Fig. 3).

2) Fault Injection Rate

Fault injection rate is the probability of a faulty flit presents during the simulation. To efficiently generate the fault injection profile based on the given fault parameters, we propose an index-based fault injection method that indicates when and where a wire encounters an error. Fig. 10 shows the execution flowchart of the parallel simulation in \( N \) microprocessors. The main steps are described below:

1. The root processor first generates a list of time indices for each wire; these indices indicate when an error occurs in the overall simulation.

2. Random numbers are generated. In our simulator, we use the GSL [37] to select many random number generators (e.g. taus, MT19937, and ranlxd1 from the GSL library). The wire ID is first used as a random seed for each generator; thus different random sequences can be generated.

3) Parallel Traffic Injection

Using the MPI language, we are able to schedule a parallel simulation in a multiprocessor server environment, where all the processors physically work in parallel. In MPI simulation, each real processor can be assigned one or more processor IDs; if the maximum ID is greater than the realistic amount of processors; virtual processors are created using transparent time-division. Using of this feature, we let each virtual processor model an IP core, loading a traffic file during the simulation. This allows flexible assignment of any traffic injection file to any IP core.

IV. ENERGY ESTIMATION AND FAULT INJECTION EFFICIENCY

A. Energy Estimation

Energy is an important constraint in NoC designs. In previous NoC simulators [26] [27], the router energy is simply categorized into busy mode and idle mode. This is coarse-grain estimation. First, energy in busy mode varies with the number of input/output channels used in each cycle. Second, energy varies in different technologies; in the deep scaled technologies, leakage cannot be ignored. When the error control methods are employed in an NoC, energy consumption should be estimated in a fine-grain way to improve the estimation accuracy.

The total energy for NoC (expressed in equation (1)) includes the energy for router, network interface and links. Router energy shown in (3) is comprised of input/output channel, and crossbar module. Because of the inserted error control module, the energy consumed by the router is not constant for different noise scenarios and reliability requirement. Consequently, the energy for each port (including both input and output channel) consists of buffer energy and error control energy, as expressed in (4).

\[
E_{\text{NoC}} = \sum_{i=1}^{N} \left( E_R + E_{NI} \right) + \sum_{k=1}^{N} \alpha_k \times E_{\text{Link}}
\]

\[
E_R = \sum_{j=1}^{S} E_{\text{Port} - j} + E_{\text{Xbar}}
\]

\[
E_{\text{Port} - j} = \beta_{\text{input}} \times E_{\text{Buffer} - \text{input}} + \beta_{\text{output}} \times E_{\text{Buffer} - \text{output}} + \gamma_{\text{ECC}} \times E_{\text{ECC}} + \gamma_{\text{DeECC}} \times E_{\text{DeECC}}
\]
Here, symbols $\alpha$, $\beta$, $\gamma$ are switching factor for links, buffer per port, and error control coding block, respectively. $E_{\text{NoC}}$, $E_{\text{R}}$, $E_{\text{Src}}$, $E_{\text{link}}$, $E_{\text{Xbar}}$, $E_{\text{port}}$, $E_{\text{buffer}}$, $E_{\text{ECC/DeECC}}$ are energy for the overall NoC, router, network interface, bus between two nodes, crossbar module (including crossbar switch, routing block and port reservation block), input/output channel and error control module, respectively. As a result, one can estimate the energy consumption for a given specification of the NoC structure and the characteristics of the fault injection.

B. Speed and Memory Consumption for Fault Injection

Reduction in simulation time and memory consumption is achieved by means of MPI-based parallel simulation in two ways—multiple links simulation per cycle and index-based fault injection. Unlike simulation performed on a single pair of switch-to-switch links, our method simulates multiple pairs of transmitters/receivers in parallel. As a result, the time for modeling the probability that the link is affected by noise can be shortened. Further, the index-based fault injection method requires only a small memory to indicate when and where a wire encounters an error, rather than a large table to indicate the error status for each cycle.

The bit error rate is a measure of a link’s susceptibility to noise from external and internal sources. In theoretical analysis, the Gaussian pulse function (4) is widely used to model the bit error rate $\varepsilon$ [12, 13].

$$
\varepsilon = \sqrt{\frac{V_{dd}}{2\sigma}} = \frac{1}{2\sigma} \int_{-\infty}^{\infty} e^{-y^2/2\sigma^2} dy
$$

(4)

The noise voltage is a normal distribution with standard deviation $\sigma$; $V_{dd}$ is the supply voltage. In a switch-to-switch simulation, the bit error rate can be modeled as the ratio of the number of cycles $n_e$ containing erroneous bits over the total simulation time $n_T$, as shown below

$$
\varepsilon = \frac{\sum_{i=1}^{W_{\text{flit}}} n_{e,i} / n_T}{W_{\text{flit}}}
$$

(5)

$W_{\text{flit}}$ is the flit width in bits. We estimate the bit error rate as the average value of all bit error rates of switch-to-switch links in the NoC. Suppose each node has $N_L$ outgoing links, the bit error rate for the overall NoC links is given below

$$
\varepsilon = \frac{\sum_{i=1}^{W_{\text{flit}}} n_{e,i} / n_T}{W_{\text{flit}}} = \frac{\sum_{i=1}^{W_{\text{flit}}} n_{e,i} / n_T}{W_{\text{flit}}} = \frac{\sum_{i=1}^{W_{\text{flit}}} n_{e,i} / n_T}{W_{\text{flit}}}
$$

(6)

To obtain a good approximation for $\varepsilon$, the total simulation time $T_{\text{sim}}$ is given by

$$
T_{\text{sim}} = n_T \times N_L \times N_{\text{node}}
$$

(7)

Typically, $\varepsilon$ is below $10^{-5}$; thus, $n_T$ is greater than $10^6$. Examining more fault patterns requires an increase in the overall simulation time. As the bit error rate decreases, these simulations would become prohibitively time-consuming in uniprocessor simulators.

To speed up simulations, we divide and distribute the overall simulation to $N_p$ microprocessors, while maintaining the same bit error rate with (6). Equation (8) shows the revised definition for the bit error rate of the NoC links.

$$
\varepsilon = \frac{\sum_{j=1}^{N_{\text{node}}} \varepsilon_j}{(N_L \times N_{\text{node}}) / N_p}
$$

(8)

where,

$$
\varepsilon_j = \frac{n_e \times N_L \times N_{\text{node}}}{n_T \times N_{\text{node}} \times W_{\text{flit}} / N_p}
$$

(9)

and $(N_L \times N_{\text{node}}) / N_p$ is an integer. In each processor, $\varepsilon_j$ is modeled. Consequently, the simulation time for each microprocessor is reduced to

$$
T_{\text{sim}} = \frac{n_T \times N_L \times N_{\text{node}}}{N_p}
$$

(10)

A straightforward way to obtain the bit error rate is to (1) create an empty table, with each entry for one simulation cycle;

(2) randomly select a number of entries to mark as faulty;

(3) in the marked entries of the table, randomly choose the bit positions to inject faults;

(4) check the memory table each cycle, and flip the marked bit. This is called table-based fault injection.

Suppose each entry costs one integer memory space (e.g. four bytes), the total memory required for fault injection table is $4 \times S$ (Bytes) given by (11) below.

$$
\text{MEM}_{\text{table}} = 4 \times \frac{1}{\varepsilon} S \quad \text{(bytes)}
$$

(11)

Here, $S$ is the number of fault patterns tested in the simulation. For $\varepsilon=10^{-7}$, the memory consumed for fault injection table is 400 MB. As the bit error rate further decreases, the memory consumption in this table-based method becomes unaffordable.

In contrast, in a index-based fault injection method indicates when and where a wire encounters an error. The proposed index-based fault injection method requires memory given blow

$$
\text{MEM}_{\text{index}} = 4 \times W_{\text{flit}} \times \frac{S}{N_{\text{sim}}} \quad \text{(bytes)}
$$

(12)

Here, $N_{\text{sim}}$ is the number of microprocessors employed in parallel simulation. As shown in (13), the memory consumption for the index-based fault injection method only depends on the flit width and the number of fault patterns simulated in each microprocessor ($S / N_{\text{sim}}$). Here, $S / N_{\text{sim}}$ is many orders of magnitude less than $1 / \varepsilon$. Thus memory consumption in this index-based method is much smaller than in the table-based method. Index-based fault injection compresses the size of the table indicating the fault location, compared to table-based fault injection. Because the index-based method does not depend on the bit error rate, it is convenient to simulate a scenario in which the bit error rate is extremely low and simulation error is strictly constrained (i.e. massive number of fault patterns should be tested in the simulation).
V. Experiments

A. Experimental Setup and Evaluation Metrics

The proposed simulator has been successfully applied to the investigation of the impact of error control on NoC performance, as well as NoC design space exploration. In the following experiments, we performed simulation on a 10x10 torus NoC, which is comprised of five-port routers and single-port network interface. The input buffer depth is eight flits; the output buffer depth is four flits, each packet contains six flits; retransmission delay is set to four for single cycle link delay. XY routing algorithm is employed in the emulated NoC. Two traffic injection patterns, Poisson and Uniform, are modeled. The energy for each module is reported by Synopsys Design Compiler using 180nm and 65nm TSMC technologies; link switching energy is simulated in Cadence. The switching factors are reported by the proposed simulator. Parallel simulation has been examined in SDSC TeraGrid IA-64 server [41] that has 524 Intel® Itanium®2 1.5 GHz processors, and a Linux server that has four Intel(R) Xeon(TM) 3.06GHz processors. Total simulation cycle for each experiment is 1000000.

In the following experiments, we evaluate an NoC using the metrics—average flit latency, average throughput, switching factor (for link, buffer, ECC and crossbar), and energy per useful flt. The definitions of these metrics are given below.

\[ \text{Avg.FltLatency} = \frac{1}{M} \sum_{i=1}^{M} \left( T_{\text{flt_received}} - T_{\text{flt_send}} \right) \]  
(13)

\[ \text{AvgThroughput} = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{\text{Total Flt Received}}{\text{Total Simulation Time}} \right) \]  
(14)

\[ \text{SwitchingFactor} = \frac{1}{C} \sum_{i=1}^{C} \sum_{j=1}^{N} \left( \sum_{k=1}^{b} \text{SwitchedInPorts} \right) \]  
(15)

\[ \text{EnergyPerUsefulFlit} = \frac{\sum_{i=1}^{C} \sum_{j=1}^{N} \text{Energy}_i}{\text{Total ErrorFree Flits Received}} \]  
(16)

B. ECC Exploration

Design space exploration for an NoC without error control has been accomplished in [Refs]. Our focus is the impact of error control on the performance and energy of NoC designs.

1) Impacts of Packet and Fault Injection on NoC Performance

Increasing traffic injection rate typically deteriorates network congestion. In spite of improving NoC reliability, error detection combined with retransmission further increases the congestion if the noise increases. In this section, we examine the combination impacts of traffic and fault injection rates on NoC performance. IP cores inject packets following uniform traffic patterns, and our simulator brings in errors with the specified flit error rate, which is the number of the erroneous flits over the total number of injected flits. For simplicity, we assume the error detection codec can detect all the injected errors and go-back-n retransmission is applied to recovery the errors.

As shown in Fig. 11, the average flit latency increases with the traffic injection rate because of increasing network congestion. As the flit error rate increases, more retransmissions are requests to improve on-chip communication reliability; thus, the latency increases too. It can be noticed that the impact of increasing traffic injection rate is more significant than that of increasing flit error rate. Fig. 12 shows the impact of traffic and flit error injection on throughput. Increasing flit error rate dramatically degrades the throughput of error-free flts in low traffic region. High traffic injection rate compensates for the throughput degradation. This case study shows that traffic injection rate should be moderately reduced to maintain the desired latency or throughput, if error control schemes are employed. The proposed simulator assists in quantifying the traffic reduction percent for the target QoS.

2) Impact of Error Control on Performance and Energy

Three categories of error control approaches have been employed in our analysis of reliable NoC design—FEC, error detection combined with ARQ, and HARQ. FEC always attempt to correct the detected error, even when the error is beyond its correction capability. ARQ requests a retransmission when the error is detected. HARQ first attempts to correct the detected error; if the number of errors is beyond the codec’s error correction capability, retransmission is requested. As a result, HARQ leads to less retransmission than ARQ, at the cost of more redundant bits and the complexity of codec.
To demonstrate the impact of different error control schemes on NoC performance and energy, we employ six error control schemes in the proposed simulator—FEC1 (correct 1-bit error), FEC2 (correct 1- and 2- adjacent errors), ARQ1 (detect 1- and 2-bit errors), ARQ2 (detect 1-, 2-, 4-bit adjacent errors), HARQ1 (correct 1-bit and detect 2-bit errors), and HARQ2 (correct 1- and 2-bit adjacent errors and detect 2- and 4-bit adjacent errors). For simplicity, Hamming and extended Hamming codes are used for error detection and correction. Table 1 shows the dynamic and leakage power of each module in the NoC. FEC2, implemented with two groups of Hamming (21,16) with interleaving, consumes less codec power than FEC1 because of shorter critical path; cost more link switching power than FEC1 because of more redundant wires. So does the comparison of ARQ1 and ARQ2 (HARQ1 and HARQ2). HARQ scheme has higher error resilience than FEC and ARQ at the cost of more codec and link switching power. Table 1 also shows that the ratio of leakage power over dynamic power in 65nm node is three orders of magnitude higher than that in 180nm node. Thus, it is not suitable to ignore the impact of leakage on the energy comparison among different error control schemes.

Fig. 13 and Fig. 14 shows the average flit latency and throughput comparison of these six ECC schemes in three noise scenarios—1-bit, 2-bit adjacent and 4-bit adjacent errors. In each noise case, only the specific error type is injected to flits.

In the first noise scenario (Fig. 13 left), the 1-bit error can be detected or corrected by all the ECC schemes employed here, although ARQ schemes result in the highest latency. In low flit error rate region (i.e. $10^{-4}$ in Fig. 13), the latency of those ECC schemes are very close. As the flit error rate increases, the latency of ARQ increases by 3%, 22%, 27% compared to that in low noise region. In contrast, FEC and HARQ can correct the errors without retransmission; thus maintains the latency as the

<table>
<thead>
<tr>
<th>ECC schemes*</th>
<th>Codec</th>
<th>Router</th>
<th>Input buffer</th>
<th>Output buffer</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEC1</td>
<td>23.8</td>
<td>4.51</td>
<td>102.8</td>
<td>11.9</td>
<td>15.0</td>
</tr>
<tr>
<td>FEC2</td>
<td>18.5</td>
<td>4.51</td>
<td>102.9</td>
<td>12.0</td>
<td>16.6</td>
</tr>
<tr>
<td>ARQ1</td>
<td>24.0</td>
<td>4.51</td>
<td>102.9</td>
<td>12.0</td>
<td>15.4</td>
</tr>
<tr>
<td>ARQ2</td>
<td>16.0</td>
<td>4.51</td>
<td>102.9</td>
<td>12.3</td>
<td>17.0</td>
</tr>
<tr>
<td>HARQ1</td>
<td>40.5</td>
<td>4.51</td>
<td>102.9</td>
<td>12.0</td>
<td>15.8</td>
</tr>
<tr>
<td>HARQ2</td>
<td>33.5</td>
<td>4.51</td>
<td>102.9</td>
<td>12.3</td>
<td>17.8</td>
</tr>
</tbody>
</table>

**Dynamic Power at 180 nm node (mW)**

<table>
<thead>
<tr>
<th>ECC schemes*</th>
<th>Codec</th>
<th>Router</th>
<th>Input buffer</th>
<th>Output buffer</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEC1</td>
<td>0.83</td>
<td>0.12</td>
<td>1.86</td>
<td>0.39</td>
<td>0.0068</td>
</tr>
<tr>
<td>FEC2</td>
<td>0.34</td>
<td>0.12</td>
<td>1.86</td>
<td>0.39</td>
<td>0.0076</td>
</tr>
<tr>
<td>ARQ1</td>
<td>0.74</td>
<td>0.12</td>
<td>1.86</td>
<td>0.39</td>
<td>0.0070</td>
</tr>
<tr>
<td>ARQ2</td>
<td>0.21</td>
<td>0.12</td>
<td>1.86</td>
<td>0.39</td>
<td>0.0070</td>
</tr>
<tr>
<td>HARQ1</td>
<td>1.05</td>
<td>0.12</td>
<td>1.86</td>
<td>0.39</td>
<td>0.0072</td>
</tr>
<tr>
<td>HARQ2</td>
<td>0.35</td>
<td>0.12</td>
<td>1.86</td>
<td>0.39</td>
<td>0.0081</td>
</tr>
</tbody>
</table>

**Leakage Power at 180 nm node (µW)**

<table>
<thead>
<tr>
<th>ECC schemes*</th>
<th>Codec</th>
<th>Router</th>
<th>Input buffer</th>
<th>Output buffer</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEC1</td>
<td>8.7</td>
<td>1.12</td>
<td>15.79</td>
<td>3.25</td>
<td>2.40</td>
</tr>
<tr>
<td>FEC2</td>
<td>5.6</td>
<td>1.12</td>
<td>15.81</td>
<td>6.3</td>
<td>1.99</td>
</tr>
</tbody>
</table>

**Dynamic Power at 65 nm node (mW)**

<table>
<thead>
<tr>
<th>ECC schemes*</th>
<th>Codec</th>
<th>Router</th>
<th>Input buffer</th>
<th>Output buffer</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEC1</td>
<td>44.0</td>
<td>13.8</td>
<td>89.1</td>
<td>6.1</td>
<td>1.68</td>
</tr>
<tr>
<td>FEC2</td>
<td>26.0</td>
<td>13.8</td>
<td>89.3</td>
<td>6.2</td>
<td>1.86</td>
</tr>
<tr>
<td>ARQ1</td>
<td>31.5</td>
<td>13.8</td>
<td>89.2</td>
<td>6.1</td>
<td>1.73</td>
</tr>
<tr>
<td>ARQ2</td>
<td>23.5</td>
<td>13.8</td>
<td>89.3</td>
<td>6.3</td>
<td>1.90</td>
</tr>
<tr>
<td>HARQ1</td>
<td>49.5</td>
<td>13.8</td>
<td>89.2</td>
<td>6.2</td>
<td>1.77</td>
</tr>
<tr>
<td>HARQ2</td>
<td>44.5</td>
<td>13.8</td>
<td>89.4</td>
<td>6.3</td>
<td>1.99</td>
</tr>
</tbody>
</table>

**Leakage Power at 65 nm node (µW)**

*Codec used in each scheme—FEC1: Hamming(38,32) for error correction; FEC2: two groups of Hamming(21,16) with interleaving [Refs] for error correction; ARQ1: Hamming(38,32) for error detection; ARQ2: two groups of Hamming(21,16) with interleaving for error detection; HARQ1: Hamming(39,32) for error detection and correction; HARQ2: two groups of Hamming(22,16) with interleaving for error detection and correction.

**NoC Verilog codes are synthesized in 180nm TSMC CMOS technology. Supply voltage = 1.8 V, frequency = 500 MHz, link switching factor = 0.5.

**NoC Verilog codes are synthesized in 65nm TSMC CMOS technology. Supply voltage = 1 V, frequency = 1 GHz, link switching factor = 0.5.

![Fig. 13 Average latency in different noise conditions with traffic injection rate=0.15 packet/cycle/node](image1)

![Fig. 14 Average throughput in different noise conditions with traffic injection rate=0.15 packet/cycle/node](image2)
flit error rate increases. In the second noise scenario (Fig. 13 middle), HARQ1 fails to correct the detected error, resulting in the same latency with ARQ schemes. In the third noise scenario (Fig. 17 right), HARQ2 is not able to correct the detected 4-bit errors and requests as many retransmissions as ARQ1, ARQ2 and HARQ1 schemes. In contrast, the FEC latency is not affected by the increasing flit error rate and the number of error bit, but the received flits cannot be guaranteed all error free because of error correction failure.

Retransmission in ARQ and HARQ schemes leads to throughput degradation as shown in Fig. 14. As the flit error rate increases, the throughput of ARQ decreases by 3%, 26%, 33% compared to that in low noise region. Since the same retransmission protocol (i.e. go-back-N) is employed in ARQ and HARQ, the throughput degradation of HARQ in the second and third noise condition is same with that of ARQ. Similar to latency comparison, HARQ1/HARQ2 scheme cannot maintain its throughput in 2-bit/4-bit adjacent error scenario because of the limited error correction capability of extended Hamming (38,32)/two groups of extended Hamming(38,32).

The synthetic error injection conditions provided in our simulator facilitate the performance evaluation of a specific error detection/correction codec and error recovery policy. Note, other noise scenarios that mix different number of error bits are also supported in the simulator, by loading faultinjection.dat file (shown in Fig. 3).

To consider energy consumption, reliability and throughput simultaneously, energy per useful flit in a given time is used to evaluate error control schemes in different noise conditions. The useful flit means the flit reaching the destination without errors. As shown in Fig. 15, the energy per useful flit of FEC schemes is constant in 1-bit error scenario, since the number of received error-free flits does not change with the increasing flit error rate. Because of using more redundant links, FEC2 consumes more energy per useful flit than FEC1. Although the codec power of FEC2 is less than that of FEC1, the sum of link switching energy exceeds the codec switching energy (the redundant links switch when the codec is not used during buffer full and retransmission period). As the error bit increases to 2-bit, FEC1 yields less useful flits and thus higher energy per useful flit than FEC2. In 4-bit adjacent error scenario, both FEC1 and FEC2 cannot correct the errors, so their energy performance increases with the flit error rate. Different with FEC schemes, the energy per useful flit of ARQ schemes increases with increasing flit error rate regardless of the number of error bits. In the moderate flit error rate region, ARQ2 consumes more link switching energy than ARQ1, resulting in worse energy efficiency. In the high flit error rate region and 4-bit error scenario, ARQ2 achieves better energy efficiency than ARQ1, because of less codec complexity. In the condition of 1-bit error, HARQ schemes achieve constant energy per useful flit as FEC, since all the detected errors can be corrected without retransmission. Different with HARQ2, HARQ1 fails to correct 2-bit adjacent errors and requires retransmission to maintain reliability, resulting in more energy consumption. When 4-bit error presents, both HARQ1 and HARQ2 use

---

Fig. 15 Average energy per useful flit at 180nm node

Fig. 16 Average energy per useful flit at 65nm node
injection error. It is assumed that retransmission is used to recover erroneous flits. The header flit contains the information indicating the direction of the next hop. As a result, header errors only affect buffer releasing in the previous hop. In contrast, errors on tail flits postpone releasing both the output buffer in previous hop and output port reservation in the current hop. As a result, errors on tail flits lead to potential resource contention. Error on payload delays releasing the output buffer in previous hop and may affect the output port reservation in current hop, which depends on the input buffer depth of current hop. In our experiment, the input buffer depth is larger than the packet length; thus, error on payload does not affect output port reservation. The flit error rate used in the simulation is $10^{-3}$.

### C. Fault Injection Exploration

In addition to flit error rate, NoC performance is affected by the type of faulty flits (header, payload, or tail), the location of faulty links (router-to-router links or router-to-NI links) and the fault types (transient or permanent).

1) **Faulty Flit Property**

Errors injected on header, payload, or tail flits yield different performance degradation. We assume that retransmission is used to recover the erroneous flits. Header flit contains the information indicating the direction of the next hop. As a result, header errors only affect buffer releasing in the previous hop. In contrast, errors on tail flits postpone releasing both the output buffer in previous hop and output port reservation in current hop. As a result, errors on tail flits lead to potential resource contention. Error on payload delays releasing the output buffer in previous hop and may affect the output port reservation in current hop, which depends on the input buffer depth of current hop. In our experiment, the input buffer depth is larger than the packet length; thus, error on payload does not affect output port reservation. The flit error rate used in the simulation is $10^{-3}$.

2) **Fault Location**

There are two types of links in NoC—router-to-router interconnect (i.e., global link) and router-to-NI (i.e., local link). Error control (error detection combined with retransmission) for global link reliability improvement typically deteriorates network congestion, especially in high traffic injection region. In contrast, retransmission to recover fault on local links merely stops the packet injecting to the network and does not cause additional network congestion. As shown in Fig. 18(a), global fault increases up to 20% more latency than local fault. However, the effect that local faults decrease the packet injection to network results in a lower throughput than the global fault, as shown in Fig. 18(b). If the global fault is large, the corresponding throughput degradation is significant since there are more global links than local links.

3) **Fault Type**

In the previous sections, we have examined the impact of transient faults on NoC performance and energy. In addition to transient faults, the proposed simulator can facilitate the impact of permanent faults on performance. Suppose error detection combined with retransmission is employed to improve link reliability. We further assume that no adaptive routing is available to reroute the flit in the presence of permanent faults. As shown in Fig. 19, more sets of faulty links yield less
D. Memory and Time for Fault Injection

In this section, we compare the memory consumptions for the table-based and index-based fault injection over a wide range of bit error rates. Here, we examine 100 and 1000 fault patterns. Typically, more fault patterns require more memory consumption, because a larger number of simulation cycles are needed.

As shown in Fig. 20, the memory required in the conventional table-based fault injection method increases exponentially (log-plot) with decreasing bit error rate. When the bit error rate is below $10^{-8}$, the memory requirement exceeds 10 gigabytes, which is unaffordable for most machines. Furthermore, such large memory requirement limits the number of fault patterns in each simulation.

In contrast, our proposed index-based fault injection method consumes near-constant memory for fault injection. These simulation results match the predicted analytical estimations in (10). It can be seen in Fig. 20 that the index-based fault injection method can reduce the memory consumption for fault injection by several orders of magnitude. In parallel simulation, the memory requirement for the index-based method can be further reduced, since each processor only handles one segment of the overall simulation.

To quantize the impact of fault injection on simulation time, times for fault index generation and fault injection in different simulation environments are compared in Fig. 21 and Fig. 22, respectively. Here, 1000 fault patterns are examined. In this experiment, 1, 10 and 100 processors are used to generate fault indices, which obey uniform distribution over the total simulation time. As shown in Fig. 20, the time for index generation using 100 processors in the parallel simulation is improved about 100X over that for a uniprocessor; indeed, decreasing bit error rate does not result in increasing time for fault index generation. In contrast, decreasing bit error rate leads to an exponential increase in time for the fault injection method, as shown in Fig. 22.

E. Investigation of NoC-Based CMP System

In a CMP system, different IP cores may operate at different frequencies. Our parallel simulation environment facilitates multi-frequency simulation. Different IP core frequencies are modeled with the method discussed in Section II.B.2).

To demonstrate the impact of IP core placement on the system performance, four different scenarios have been examined—(a) all IP cores working at the same frequency (uni-freq.), (b) the slowest IP cores are placed in the center of the system (center low freq.); (c) the fastest IP cores are placed in the center of the system (center high freq.); (d) the entire CMP system is divided into four regions, each region using one frequency (localized freq.). These four cases are shown in Fig. 24.

The traffic injection by each IP core follows Poisson arrival process, and the expected number of packets arriving per unit time varies from 0.05 to 0.5. Here, we examine two extreme on-chip communication patterns—(a) each IP core has the same
The impact of IP core placement in the CMP system on latency in Fig. 25(a), the uni-freq. case yields the highest latency. As shown in Fig. 24(c), the CMP system using the IP core placement shown in Fig. 24 yields the highest network congestion and thus highest latency. In contrast, the case shown in Fig. 24(a) inject packets to network with the same probability to send data to other IP cores in the CMP system (i.e., uniform destination); (b) each IP core only communicates with IP cores in the quarter region where it belongs to (i.e., localized destination). Fig. 25 shows the latency comparison of the CMP system, in which the IP cores is targeting the localized destination. As can be seen, the latency of uni-freq CMP system is comparable with that of the center low freq. CMP system. This is because most of IP cores in the center low freq. CMP system run at f_0, which results in comparable traffic injection to the network.

VI. CONCLUSION AND FUTURE DIRECTIONS

In deeply scaled technologies, reliability emerges as a critical parameter in circuit design. To improve reliability, error control has been integrated with conventional NoC designs. Although many simulators exist, they do not incorporate error control coding features. Consequently, evaluating performance of nanometer scale NoCs that employ various error control modules becomes increasing difficult. In this work, we fill this gap by presenting a fast parallel NoC simulator with configurable error control. To improve simulation speed, a parallel simulation method based on MPI has been proposed, and a case study shows that the simulation speed can be linearly improved with the number of microprocessors used. An index-based fault injection method has been introduced to achieve memory efficiency. Simulation results show that the memory requirement for our index-based method does not increase with improving reliability requirement; in contrast, the table-based fault injection method often used requires exponentially increasing memory. Using this simulator, we investigate the NoC design space, and evaluate the dependence of flit error rate and packet injection rate on NoC performance. Further, the performance degradation caused by different error control approaches is examined. NoCs promise significant performance improvement in on-chip communication, as designers move to chip multiprocessors (CMPs) to squeeze yet more performance from scaled technologies. Using this simulator, one can readily estimate the performance of NoC-based CMP systems.

ACKNOWLEDGMENT

We wish to thank Dr. Amitava Majumdar, Dr. Mahidhar Tatineni, and others at the San Diego Supercomputer Center (SDSC) for many helpful suggestions.

REFERENCES

IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS


http://www.arteris.com
http://noxim.sourceforge.net/
http://nirgam.ecs.soton.ac.uk/


G. De Micheli and L. Benini, Networks On Chips, Morgan Kaufmann, Chapter 4, pp. 81–90, 2007.


http://www.gnu.org/software/gsl/


http://www.sdsc.edu/us/resources/ia64/