NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures

Andreas Prodromou, Andreas Panteli, Chrysostomos Nicopoulos, Yiannakis Sazeides

University of Cyprus

The Multicore Computer Architecture Laboratory (multiCAL)
Ξ - Computer Architecture Research Group (Ξ - CARCH)
EuroCloud FP7 Project

International Symposium on Microarchitecture, December 3 2012, Vancouver, Canada
Outline

- Necessity of Networks-on-Chip (NoCs)
- Reliability and NoCs
- The NoCAAlert Approach: Invariance Checking
- Identifying invariances and examples
- Evaluation
- Results
- Conclusion – Future Work
Outline

- Necessity of Networks-on-Chip (NoCs)
- Reliability and NoCs
- The NoCAAlert Approach: Invariance Checking
- Identifying invariances and examples
- Evaluation
- Results
- Conclusion – Future Work
The Network-on-Chip (NoC) paradigm

- On-chip interconnection fabric (backbone) to connect all nodes
- Modular design
- Structured Interconnect Layout
- *Scalable* and *efficient*
- Packet-based communication
Core Number Increases

Following Moore’s law the number of transistors/chip double approx. every 18-24 months

→ Designers turn into integrating more cores to take advantage of parallelism

Intel 4004

4-bit

1971

Graph courtesy of www.crn.com
Core Number Increases

Intel Pentium 4
1 Core

1971

2000
Core Number Increases

Intel Core 2 Duo
2 Cores

1971
...
2000
2007
Core Number Increases

Intel Core i7 (Nehalem)
4 Cores
Core Number Increases

AMD Opteron 2400
6 Cores

Intel 4004
1 Core (4-bit)

1971

... 2000 2007 2008 2009

Intel Pentium 4
1 Core

Intel Core 2 Duo
2 Cores

Intel Core i7
4 Cores

AMD Opteron 2400
6 Cores
Core Number Increases

IBM POWER7
8 Cores

1971
...
2000
2007
2008
2009
2010

Intel 4004
1 Core (4-bit)

Intel Pentium 4
1 Core

Intel Core 2 Duo
2 Cores

Intel Core i7
4 Cores

AMD Opteron 2400
6 Cores
Core Number Increases

Intel Xeon Westmere-EX
10 Cores
Core Number Increases

Intel Xeon Phi Coprocessor
64 Cores

Core Number Increases

Intel Single-Chip Cloud Computer
48 Cores

Intel Polaris Chip
80 Cores

Near Future

Intel Single-Chip Cloud Computer
48 Cores

Intel Polaris Chip
80 Cores
It’s already happening!

Intel Polaris Chip

- Router is becoming part of the core design
- NoCs are becoming necessary

Tilera TILE64 – 64 Cores
- 2D mesh NoC comprising
- 5 independent networks
  - one for each of 5 message classes

1971…2000
2007
2008
2009
2010
2011
2012
Near Future
Outline

- Necessity of Networks-on-Chip (NoCs)
- Reliability and NoCs
- The NoCAAlert Approach: Invariance Checking
- Identifying invariances and examples
- Evaluation
- Results
- Conclusion – Future Work
Reliability in the nano era

- Aggressive transistor downsizing
  - Increasing hardware *variability*
  - Susceptibility to *wear-out* (accelerated aging effects)

- **Permanent** faults
  - *Static* (occurring at manufacture-time)
    - Process Variability (PV), Manufacturing imperfections
  - *Dynamic* (occurring at run-time, prolonged stressing $\rightarrow$ component wear-out)
    - Electro-Migration (EM), Negative Bias Temperature Instability (NBTI), Oxide breakdown, Stress-Induced Voiding (SIV), Hot Carrier Injection (HCI), etc.

- **Transient** faults (or Soft Errors – Single-Event Upsets, SEU)
  - Alpha particles (impurities in packaging/interconnect), Cosmic-ray-induced neutrons, Neutron-induced $^{10}\text{B}$ fission (interconnect layer insulator)
  - Traditionally associated with *memories*
    - Error Correcting Codes (ECC) widely used in DRAM modules
Ominous predictions regarding **reliability**

<table>
<thead>
<tr>
<th>Technology</th>
<th>Inverter $\zeta$</th>
<th>Latch $\zeta$</th>
<th>SRAM $\zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>45nm</td>
<td>$\approx 0$</td>
<td>$\approx 0$</td>
<td>6.1e-13</td>
</tr>
<tr>
<td>32nm</td>
<td>$\approx 0$</td>
<td>1.8e-44</td>
<td>7.3e-09</td>
</tr>
<tr>
<td>22nm</td>
<td>$\approx 0$</td>
<td>5.5e-18</td>
<td>1.5e-06</td>
</tr>
<tr>
<td>16nm</td>
<td>2.4e-58</td>
<td>5.4e-10</td>
<td>5.5e-05</td>
</tr>
<tr>
<td>12nm</td>
<td>1.2e-39</td>
<td>3.6e-07</td>
<td>2.6e-04</td>
</tr>
</tbody>
</table>

Impact of NBTI on failure probability trends


- This recent study from DATE-2010 signifies increases in failure probabilities by **tens of orders of magnitude** at 12 nm, as opposed to 45 nm.
- Each new technology generation decreases IC lifetime by **half** [ITRS 2011]

**Challenge: “Designing reliable systems from unreliable components”** *

NoC (un)reliability implications

- A single fault in the NoC can cause:
  - Network disconnections
  - Deadlocks (Network and Protocol-level)
  - Lost packets
  - Degraded performance

→ A single fault can paralyze the entire system (CMP)

- Protecting the NoC is of paramount importance
Outline

- Necessity of Networks-on-Chip (NoCs)
- Reliability and NoCs
- The NoCAAlert Approach: Invariance Checking
- Identifying invariances and examples
- Evaluation
- Results
- Conclusion – Future Work
NoCAAlert: The Big Picture

NoCAAlert:
- Lightweight distributed invariance checkers
- Checkers behave like hardware assertions
- Checks for legality, not correctness
- Network’s operation is never interrupted
- Provides almost instantaneous fault detection

Assumption:
- Packet/flit contents are protected with ECC

NoCAAlert protects against faults in the control logic

Interesting observation
- Erroneous but legal module outputs are always benign
NoCAAlert’s Terminology

• Invariance violation:
  – The breaking of a fundamental functional rule within the context of a component’s operation
  – e.g., the routing computation unit outputs an illegal direction

• Legality:
  – Illegal is an output that is impossible to occur, based on the set of functional correctness rules of a given component

• Instantaneous fault detection:
  – Detect a fault as soon as it manifests (same clock cycle)
  – Easier to recover
  – Localized information could identify faulty location
Invariance Checking

• System is continuously (on-line) examined for *illegal* outputs
  – An illegal output can be the result of some kind of fault

• Emulates **assertions** used in software

• Example: Assume a variable X cannot get the value 5
  – \texttt{assert(X! = 5)}
  – In hardware this would be achieved with a **comparison unit** that raises a flag
Outline

- Necessity of Networks-on-Chip (NoCs)
- Reliability and NoCs
- The NoCAAlert Approach: Invariance Checking
- Identifying invariances and examples
- Evaluation
- Results
- Conclusion – Future Work
A generic (typical) NoC router micro-architecture

* Network packets are broken into multiple flits. A flit is a flow control unit and it is the smallest unit of flow control in the NoC.
Identifying invariances within the NoC Router

• Identification of invariances relies on the *modularity* and *hierarchy* of the NoC Router

• The functional algorithm of each module is exhaustively inspected using a *bottom-up approach*
  – Identification of all the *functional rules*
  – Identification of all the *functionally illegal outputs*

• End-to-end invariances at the network-level are identified
Invariance categorization

• **32 invariances** have been identified through detailed exploration of the router’s microarchitecture

• Identified invariances are categorized based on the router module they are associated with
  
  – Routing Computation unit (3)
  
  – Arbiters (10)
  
  – Crossbar (3)
  
  – Buffer State (12)
  
  – Port-Level (3)
  
  – End-to-End (network-level) (1)
Ensuring network correctness

• Which conditions must a reliable network satisfy?

• **Four** main conditions that **ensure** functional correctness within the network*
  – No packets are dropped
  – Delivery time is bounded
  – No data corruption occurs
  – No new packet is generated within the network

• **Additional requirement:**
  Intra-packet flit ordering

---

* D. Borrione et al. A generic model for formally verifying NoC communication architectures: A case study. In NOCS 2007
Routing Algorithm

• Routing algorithms forbid some turns to avoid deadlocks and livelocks in the network

• E.g., Dimension-order **XY routing**
Invariance Example – Routing Algorithm

• Invariance violation due to forbidden turn according to the specification of the XY routing algorithm
Invariance Example - Arbiters

- Grant is not allowed without a corresponding request

- Arbiter’s output must *always* be 1-hot
- Can’t assign one resource (output port or VC) to multiple contestants

- If at least one request exists, the arbiter *must* grant one of the contestants
Faults that do not cause invariance violations

- Invariance checking only detects illegal outputs
- Does not necessarily detect incorrect outputs
Faults that do not cause invariance violations (Cont.)

• Two elemental questions arising by this kind of faults:

1. If such non-invariant upsets cause some other functional violation later on in the network, will the fault be caught by subsequent NoCAAlert checkers?

2. If these non-invariant upsets are never caught by any subsequent NoCAAlert checker, do they end up affecting the overall network correctness?
Outline

- Necessity of Networks-on-Chip (NoCs)
- Reliability and NoCs
- The NoCAAlert Approach: Invariance Checking
- Identifying invariances and examples
- Evaluation
- Results
- Conclusion – Future Work
Evaluation framework

• Tools used:
  – **GARNET** cycle-accurate NoC simulator
    • Used for extensive experimentation under fault presence
  – **Synopsys Design Compiler** (for hardware synthesis)
    • Used to assess the hardware overhead of NoCAAlert
    • Based on full Verilog HDL implementation of NoCAAlert and synthesis using **65 nm** commercial standard-cell libraries

• Compared against **ForEVeR***, the current state-of-the-art

---

The ForEVeR mechanism *

• Epoch-based on-line fault detection mechanism
  – Achieved with the help of an additional lightweight checker network that is assumed to be 100% reliable
  – Contains run-time checks for arbitration stages and End-to-End coverage

• Counter-based scheme that uses notification packets
• Fault assessment occurs at the end of each epoch
  – If counter values not reconciled, a recovery mechanism is triggered
  – In-flight data delivered to the intended destination via the checker network

• BUT, how to choose epoch duration (sensitivity to traffic injection rate, application-specific, etc.) \(\rightarrow\) False positives even in a fault-free environment!

Fault-injection framework

- Fault model: Single-bit, single-event transient faults
- Faults were injected at the *inputs* and *outputs* of every control module of a router
  - RC units, VA and SA arbiters, Crossbar, and Status Tables
  - One fault injected in each experiment
- **Total number of fault locations:**
  - **11,808** for 8x8 2D mesh network
The *Golden Reference* report

• *For each experiment*, generate a *Golden Reference* (GR) report:
  – A log of the entire network’s output (flit ejections), under a *fault-free* run.
  – “Oracle” knowledge of what should *normally* happen

• “*Contaminated*” Logs are compared against the GR
  – If all flits were delivered *correctly* (remember the *four rules*), *and* intra-packet order was maintained, the fault was benign (no system-level effect)
  – Note that the global order of packets *is allowed* to change
Network’s State Affects Fault Detection

• The state of the network can influence fault detection behavior:
  – Faults in an empty network are less likely to be masked
  – Warmed-up networks might “hide” faults

• Need for testing at different states
  – 7 different traffic injection ratios (10-40% in 5% increments)
  – 3 different fault injection instances
    • Faults were injected at cycle 0 (empty network), cycle 32 K, and cycle 64 K (warmed-up network)
  – Resulting in a total of approx. 248 K simulations
Classification of NoCAlert’s detection outcomes

• Four main fault detection categories:
  – True positive: Detected non-benign fault
  – True negative: Non-detected benign fault
  – False positive: Detected benign fault
    • Can cause unnecessary fault recovery triggering
  – False negative: Non-detected non-benign fault
    • Worst case
    • Ideally, this should be ZERO

• Non-benign: Comparison against GR failed
• Benign: Successful comparison against GR
Outline

- Necessity of Networks-on-Chip (NoCs)
- Reliability and NoCs
- The NoCAler t Approach: Invariance Checking
- Identifying invariances and examples
- Evaluation

- Results
- Conclusion – Future Work
Fault coverage breakdown

- **Observations:**
  - **0% false negatives** for both schemes (no malicious faults escape)
  - False-positive percentages higher in a warmed-up network
    - More faults are masked
  - NoCAAlert behaves slightly worse than ForEVeR in terms of false positives (ForEVeR is an epoch-based mechanism → some faults vanish by end of epoch)
• Observations:
  – 97% of NoCAlert’s fault detections were *instantaneous*
  – Significant fault detection latency improvement
    • Up to 100x
  – NoCAlert: on-line mechanism $\rightarrow$ instantaneous detection
Hardware **overhead** (synthesis using 65 nm libraries)

- DMR-CL: Use Double Modular Redundancy (DMR) protection for control logic
- NoCAAlert: Use NoCAAlert protection for control logic

- **Area** overhead: ranges from 1.38% to 4.42% (3% average)
- **Power*** overhead: 0.3% - 1.3% (0.7% average)
- **Critical path** overhead: At most 3% (1% average)

* The power numbers were extracted from the Synopsys Design Compiler power report, with switching activity set to 50% for all nets. The operating voltage is 1 V.
Outline

- Necessity of Networks-on-Chip (NoCs)
- Reliability and NoCs
- The NoCAAlert Approach: Invariance Checking
- Identifying invariances and examples
- Evaluation
- Results
- Conclusion – Future Work
Conclusion

• Comprehensive on-line and real-time fault detection mechanism
• Ensures 0% false negatives within the NoC
• Based on the concept of invariance checking
  – A collection of micro-checker modules dispersed throughout the router’s control logic modules
  – Real-time hardware assertions
• Tremendous improvement in detection delay
• Extremely lightweight nature of NoCAlert in terms of area/power/timing overhead
Future Work

• Formally prove NoCAAlert’s full coverage

• Fault localization
  – Take advantage of the localized information provided by NoCAAlert
  – Try to pinpoint the faulty location

• Fault Recovery
  – After fault localization
  – Techniques to bypass faulty modules
QUESTIONS?