Deadlock-Recovery Support for Fault-tolerant Routing Algorithms in 3D-NoC Architectures

Akram Ben Ahmed, Achraf Ben Ahmed, Abderazek Ben Abdallah
The University of Aizu
Graduate School of Computer Science and Engineering,
Adaptive Systems Laboratory, Aizu-Wakamatsu, Japan.

Email: d8141104@u-aizu.ac.jp
Outline

• Background
• Motivation and goal
• Look-Ahead-Fault-Tolerant routing
• RAB mechanism for deadlock-recovery
• Evaluation
• Conclusion and future work
Outline

• Background
• Motivation and goal
• Look-Ahead-Fault-Tolerant routing
• RAB mechanism for deadlock-recovery
• Evaluation
• Conclusion and future work
Background: 3D-NoC systems

• 2D-NoC limitations:
  – Large diameter
  – High power

• 3D-NoC merits
  – High scalability
  – Low interconnect power
  – Heterogenous integration

Typical 3D-NoC structure (3D-OASIS-NoC)*

• 3D-NoC systems are complex and they are susceptible to a variety kinds of faults that can be caused by different factors:
  – Physical damage
  – Crosstalk
  – Thermal power etc..

• Types: permanent, transient, and intermittent.

• Faults can cause the information corruption or the entire system failure
Fault tolerant Routing algorithms can be used to redirect the flits to non-faulty links. Since fault tolerant routing algorithms are adaptive, the deadlock problem is one of the main concerns.
Background: Deadlock
(Example)

Permanent Fault link
Valid link

The University of Aizu
Adaptive systems lab
Background: Deadlock
(Example)

- Permanent Fault link
- Valid link

The University of Aizu
Adaptive systems lab
Background: Deadlock
(Example)
Background: Virtual Channels

- **Permanent Fault link**
- **Valid link**

Diagram showing network channels and destinations:

- Dest 200
- Dest 201
- Dest 202
- Dest 211
- Dest 212
- Dest 203
- Dest 212
- Dest 201
- Dest 222

The University of Aizu
Adaptive systems lab
Background: Virtual Channels

Permanent Fault link
Valid link
Background: Virtual Channels

- Permanent Fault link
- Valid link

The University of Aizu
Adaptive systems lab
Background: Virtual Channels

- Permanent Fault link
- Valid link

The University of Aizu
Adaptive systems lab
Background: Virtual Channels

Permanent Fault link

Valid link

211 → Dest 211 → Dest 212

201 → Dest 201 → Dest 202

212 → Dest 212

202
Outline

• Background

• Motivation and goal

• Look-Ahead-Fault-Tolerant routing

• RAB mechanism for deadlock-recovery

• Evaluation

• Conclusion and future work
Motivation and Goal

• Previously, we presented a high throughput fault tolerant routing algorithm named Look-Ahead-Fault-Tolerant (LAFT).

• LAFT is an adaptive routing that takes advantage of look-ahead routing to enhance the system performance while guaranteeing fault tolerance.

LAFT is susceptible to deadlock
Motivation and Goal

• Virtual Channels (VCs) are used in most systems to solve the deadlock
  – Expensive to implement
  – Require additional clock cycles for arbitration

• We present Random-Access-Buffer mechanism to solve the deadlock problem at very low cost
Outline

• Background
• Motivation and goal
• Look-Ahead-Fault-Tolerant routing
• RAB mechanism for deadlock-recovery
• Evaluation
• Conclusion and future work
1- The current out-port is read from the flit and the next-node address is computed
Look-Ahead-Fault-Tolerant: Example

2- The three possible direction are calculated: North, East, and Up
3- When verifying the link status of the three directions, two possible directions are computed: North and UP (East is faulty)
Look-Ahead-Fault-Tolerant: Example

4- When calculating the diversity value of each direction, North has the highest one: North=3 (North, east, and up); Up=2 (North and east)
Look-Ahead-Fault-Tolerant: Example

5- North is selected as the Next out-port and it is embedded in the flit to be used in the next downstream node
Outline

• Background
• Motivation and goal
• Look-Ahead-Fault-Tolerant routing
• RAB mechanism for deadlock-recovery
• Evaluation
• Conclusion and future work
**Random-Access-Buffer mechanism:**

**Architecture**

- **FIFO manager**
  - Manages the input buffer when no deadlock is detected.
  - If the flit’s request is not served after a period of time, a flag is issued.

- **RAB manager**
  - When receiving the flag, it drops the request of the blocking flit and searches for another one.

**Timer**

- If the flit’s request is not served after a period of time, a flag is issued.

**Diagram**

- Data in: P2 South, P1 North, P1 North
- Data out: Next_port
- Rd_adr, Wr_adr
- Timer
- RAB_cntrl
- Select_Wr, Select_Rd
- Wr_addr, Rd_addr
- Tail, Head
- RAB_Wr_addr, RAB_Rd_addr
- Deadlock_flag
- Sw_grnt
Random-Access-Buffer mechanism: Example

Status-register
Used to keep the status of the blocking flits
Random-Access-Buffer mechanism: Example

- Timer informs that the flit being processed did not get the grant and it is blocked.
Random-Access-Buffer mechanism: Example

-The request is dropped and the status-register is updated for the entire packet
-RAB cntrl reads the next packet Next-port
Random-Access-Buffer mechanism:
Example

-The next flit is checked and served
Random-Access-Buffer mechanism: Example

- When assigning the \textit{Wrt-adr}, the RAB cntrl check the status register and assign an unoccupied slot to avoid flit overwriting
Random-Access-Buffer mechanism: Example

- Slots are freed and the input buffer can host new incoming flits
Random-Access-Buffer mechanism: Example

- The previously blocking packet is given the grant and served
- The Status-register is updated
Outline

• Background
• Motivation and goal
• Look-Ahead-Fault-Tolerant routing
• RAB mechanism for deadlock-recovery
• Evaluation
• Conclusion and future work
• **We evaluate:**
  – Hardware complexity
    • Area(ALUTs)
    • Power
    • Speed
  – System performance
    • Latency/flit
    • Throughput

• **Benchmarks:**
  – Transpose
  – Uniform
  – Matrix Multiplication

• **We use:**
  – Verilog HDL
  – Quartus II ver. 12.0
  – Target device: Stratix III
  – Modelsim ver. 6.5
## Evaluation:
### Evaluation parameters

### Simulation configuration

<table>
<thead>
<tr>
<th>Parameters / System</th>
<th>LAFT-based</th>
<th>XYZ-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Size Mesh</td>
<td>Matrix</td>
<td>Matrix</td>
</tr>
<tr>
<td>Transpose &amp; Uniform</td>
<td>3x6x6</td>
<td>3x6x6</td>
</tr>
<tr>
<td>flit size</td>
<td>34 bits</td>
<td>31 bits</td>
</tr>
<tr>
<td>Header size</td>
<td>13 bits</td>
<td>10 bits</td>
</tr>
<tr>
<td>Payload size</td>
<td>21 bits</td>
<td>21 bits</td>
</tr>
<tr>
<td>Buffer Depth</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Switching</td>
<td>Wormhole-like</td>
<td>Wormhole-like</td>
</tr>
<tr>
<td>Flow control</td>
<td>Stall-Go</td>
<td>Stall-Go</td>
</tr>
<tr>
<td>Scheduling</td>
<td>Matrix-Arbiter</td>
<td>Matrix-Arbiter</td>
</tr>
<tr>
<td>Routing</td>
<td>LAFT</td>
<td>XYZ</td>
</tr>
</tbody>
</table>
**Evaluation:**

**Hardware complexity**

**Hardware complexity results**

<table>
<thead>
<tr>
<th>System/ Parameter</th>
<th>Area</th>
<th>Power (mW)</th>
<th>Speed (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Static</td>
<td>Dynamic</td>
</tr>
<tr>
<td>LAFT+RAB</td>
<td>3522</td>
<td>1334.5</td>
<td>188.42</td>
</tr>
<tr>
<td>LAFT</td>
<td>3272</td>
<td>1296.92</td>
<td>178.09</td>
</tr>
<tr>
<td>XYZ</td>
<td>2809</td>
<td>1258.01</td>
<td>165.00</td>
</tr>
</tbody>
</table>

+7.1% Vs. LAFT
+20% Vs. XYZ
+3.5% Vs. LAFT
+6.3% Vs. XYZ
unchanged Vs. LAFT
-6% Vs. XYZ
Evaluation: Performance (Latency/flit)

Average Latency/flit results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>XYZ</th>
<th>LAFT (0%)</th>
<th>LAFT (5%)</th>
<th>LAFT (10%)</th>
<th>LAFT (15%)</th>
<th>LAFT (20%)</th>
<th>LAFT+RAB (20%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transpose</td>
<td>9700</td>
<td>6250</td>
<td>8050</td>
<td>9450</td>
<td>11300</td>
<td>X</td>
<td>13124</td>
</tr>
<tr>
<td>Uniform</td>
<td>101000</td>
<td>68500</td>
<td>86500</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>104080</td>
</tr>
<tr>
<td>Matrix</td>
<td>980</td>
<td>720</td>
<td>825</td>
<td>1090</td>
<td>X</td>
<td>X</td>
<td>1223</td>
</tr>
</tbody>
</table>

- The proposed RAB mechanism has almost the same latency/flit as LAFT with deadlock recovery mechanism
- At low fault rate, LAFT provides a latency/flit reduction that can reach 35%
- RAB mechanism managed to recover from deadlock while the previous system at 10% fault-rate
The proposed RAB mechanism has almost the same throughput as LAFT with deadlock recovery mechanism. At low fault rate, LAFT provides a throughput enhancement that can reach 47%.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>XYZ</th>
<th>LAFT (0%)</th>
<th>LAFT (5%)</th>
<th>LAFT (10%)</th>
<th>LAFT (15%)</th>
<th>LAFT (20%)</th>
<th>LAFT+RAB (20%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transpose</td>
<td>14.05</td>
<td>26.15</td>
<td>19.6</td>
<td>14.85</td>
<td>11.65</td>
<td>X</td>
<td>10.1</td>
</tr>
<tr>
<td>Uniform</td>
<td>13.15</td>
<td>22.4</td>
<td>13.5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>12.58</td>
</tr>
<tr>
<td>Matrix</td>
<td>9.8</td>
<td>16.35</td>
<td>16.3</td>
<td>7.65</td>
<td>X</td>
<td>X</td>
<td>6.02</td>
</tr>
</tbody>
</table>
Outline

• Background
• Motivation and goal
• Look-Ahead-Fault-Tolerant routing
• RAB mechanism for deadlock-recovery
• Evaluation
• Conclusion and future work
Conclusion

• Proposal of an Efficient deadlock recovery mechanism named Random-Access-Buffer (RAB mechanism)
• RAB was implemented with high throughput fault tolerant routing algorithm called Look-Ahead-Fault-Tolerant (LAFT)
• Complexity and performance evaluation
Conclusion

• The proposed mechanism provides an average of 27% latency/flit reduction and 35% throughput enhancement when compared to XYZ based system at the absence of faults.

• At high fault rates, RAB mechanism guarantees deadlock freedom while the previous system fails starting from 10% fault-rate.
Conclusion

- RAB mechanism exhibits only 7.1% area overhead and 3% increasing power with almost the same behavior in terms of speed when compared to the previous LAFT-based architecture with no deadlock support.
Future work

• Use larger benchmarks to illustrate the real behavior of the proposed mechanism.
• Optimize the area and power overhead
• Extend RAB to be able to detect and recover form different types faults in the input buffer
Thank you
for your attention