Toward Exascale Resilience

Part 5:
Processors and networks

Mattan Erez
The University of Texas at Austin

July 2015
Processors are expensive
– Redundant processors vs. redundant memory chips
Expectation of high reliability
What’s in a processor?

– Lots of SRAM
  • Large SRAM arrays in caches
  • Smaller arrays in local caches, TLBs, predictors, …

– Lots of smaller latch-arrays
  • Buffers, registers, …

– Datapaths
  • Logic for doing compute – processing the data
  • Pipelining means a lot of scattered latches

– Control logic
  • Pipelining and FSMs mean a lot of scattered latches

– Communication
  • Buses, interconnection networks, …
SRAM reliability dominates processor reliability

– Much more SRAM than anything else
– SRAM more vulnerable

• Smaller transistors
• Much less masking in memories than logic
SRAM faults and error

– Particle-strikes
– Retention errors
– Read and write errors
Particle strike faults
– Ionizing particle leads to spurious current flowing
– Can overcome feedback
Retention and read/write errors

- Transistor mismatches lead to imbalance in symmetric SRAM cell
  - *Worse as transistors shrink because of variation*
  - *Worse as voltage decreases*
  - *Vcc_crit limits min voltage*
- Stability of feedback compromised
- Writes have longer tails
- Some reads too slow
Figure 2. Probability of failure (Pfail) for a cell vs. Vcc [8]

From Wilkerson et al., ISCA’08
Circuit techniques

– Make cells bigger?
  • Use more transistors or bigger ones
  • Reduces error rates
  • Sacrifices too much area

– Clever read/write circuits
  • Help, but not with stability
It’s memory – use ECC!

– ECC has low redundancy
  • SECDED is 10 bits for 512b cache-line
  • DECTED just 17 bits

– Some ECC is simple to compute
  • Bit-wise hamming codes, in particular
How much ECC protection do we need?
How many bits can be wrong?
- Multiple particle-strikes very unlikely
- Variations very random
- But, multi-cell upsets exist
Correlations reduce protections costs
– Physical bit interleaving for large arrays (on board)
The try-try-again approach

- Detection → retry
  - Works great for transient errors that did not affect previous state

Backward (rollback) recovery!
Data in many arrays is replicated
  – TLBs
  – Write-through L1
  – Clean cache lines, in general
Data in some arrays affects only microarch
  – Predictors
Is detection enough?
  – Yes! Can re-fill from elsewhere or regenerate
  – Simple parity often enough
What about tags?
Remember the bear chasing you

– Balance protection with expected rates and impact
– Smaller structures often protected less
– No need to catch extremely rare events
Latch arrays
– Similar to SRAM arrays
– Bigger transistors $\implies$ lower inherent fault rate
– Too many latches to ignore (in the future)
Use better circuits – **hardened latches**

- Design latches that check and correct themselves
- Basically replicate the latch
- Not cheap
  - More area and power, but manageable
- Can reduce by 10x or so

- Enough for exascale?
  - Not yet clear
ECC for latches?

– ECC is great, but need arrays to make effective
– Parity may be good enough for detection
  • Lower fault rate
  • More distance between cells (if not in array)
– Parity still sometimes hard to apply
Datapath

- Combination of logic and latches
- Used for numerical/logic operations
- Lots of masking
  - Often not a concern today
  - More expensive processors take care of it
Protection through duplication

– Do everything twice

– In space?
  • Double the cost

– In time?
  • Double energy and time (of arithmetic)
Protection through reduced duplication

– Do we need to check everything to detect errors?
– Residue checking

\[ |a \oplus b|_A = ||a|_A \oplus |b|_A |_A \]
Residue can be extended to protect registers
– Can this also catch other errors?
Protecting control
– Protect the logic
– Protect the semantics
Protecting logic

- Without clear arithmetic, rely on design
- Harden the circuits
- Partial duplication $\rightarrow$ parity prediction
- $>20\%$ overhead
- Coverage hard to estimate, but assume good

- Used in extreme designs
Protecting the semantics

– Check for symptoms of errors
  • Branching to an address that doesn’t start a basic block
  • Illegal instructions
  • Out-of-bounds accesses
  • Using registers that haven’t been defined
  • …

– Collection of symptoms may have excellent coverage
  • Evaluation tricky

– More on the board
Communication

- Buses move data between components
  - No changes to the data
- Parity (or other error detection) + retry
- More when we talk about networks
- Not problematic today
  - Because parity/retry is cheap and effective
Conclusion: cost / reliability tradeoff

– We can build reliable processors for exascale
– Not clear that we should (more later)
– Processor count still a concern
Fujitsu SPARC64 X

- Arrays + arithmetic + some control
- Not cheap
- My guess:
  - ~25% overhead in non-arrays

[Table: SPARC64 X RAS Functions]

<table>
<thead>
<tr>
<th>Error detection</th>
<th>Error Correction</th>
<th>Recording</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level1 cache</td>
<td>Multiplicity Parity+ECC</td>
<td>Retry, ECC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dynamic way</td>
</tr>
<tr>
<td></td>
<td></td>
<td>degradation(*2)</td>
</tr>
<tr>
<td>Level2 cache</td>
<td>ECC</td>
<td>ECC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dynamic way</td>
</tr>
<tr>
<td></td>
<td></td>
<td>degradation(*2)</td>
</tr>
<tr>
<td>Arithmetic Logic Unit</td>
<td>Parity (*1) +Residue</td>
<td>ECC, Hardware</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instruction retry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Core degradation</td>
</tr>
</tbody>
</table>

Circuits in Green: one bit error correctable
Circuits in yellow: one bit error detectable
Circuits in gray: no problem to continue operations even if this circuit fail
Network

– Generally, multi-hop networks
– Source $\rightarrow$ NIC $\rightarrow$ routers/switches $\rightarrow$ NIC $\rightarrow$ dest
What’s in a network?

– “Processor” + memory + links
– Backward recovery (retry) prominent
Network router (processor)

– Decides on routing
– Kind of looks like a simple processor
– Small part of overall network and typically well-protected with duplication and symptom checks
Memory is memory $\rightarrow$ use ECC
Links are like buses, carry the data

- Use error detection and retry
- Typically strong CRC for detection
  
  • Long-symbol codes that don’t attempt to correct
End-to-end checks?
– Can protect the message too
What about failed links?

– Unfortunately, common
– Mechanical failures of connectors 😊
Path diversity and rerouting until repair!
Other hard faults

- Corrosion
- Mechanical stress
- Accidents
- Current stress

Mechanical $\rightarrow$ power $\rightarrow$ connectors

- Lots of redundancy can be put in
- Recently, scaling in the chip faster than outside
An example: Blue Waters

From Di Marini et al., DSN’14

TABLE III: Failure Statistics. The last row refers to the statistics calculated across all the failure categories.

<table>
<thead>
<tr>
<th>Failure Category</th>
<th>count</th>
<th>%</th>
<th>MTBF [h]</th>
<th>MTTR [h]</th>
<th>$\sigma_{TBF}$ [h]</th>
<th>$\sigma_{TTTR}$ [h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Failure (No Interrupt)</td>
<td>164</td>
<td>11%</td>
<td>35.17</td>
<td>13.5</td>
<td>70.8</td>
<td>35.3</td>
</tr>
<tr>
<td>2) Interrupt (Failover)</td>
<td>99</td>
<td>6.6%</td>
<td>58</td>
<td>14.7</td>
<td>92</td>
<td>42.2</td>
</tr>
<tr>
<td>3) Link &amp; Node Failure (Job Failed)</td>
<td>19</td>
<td>1.3%</td>
<td>297.7</td>
<td>6.1</td>
<td>427.3</td>
<td>5.4</td>
</tr>
<tr>
<td>4) Link Failure (No Job Failed)</td>
<td>285</td>
<td>19.1%</td>
<td>19.9</td>
<td>32.7</td>
<td>51.9</td>
<td>91.2</td>
</tr>
<tr>
<td>5) Link Failure (Job Failed)</td>
<td>19</td>
<td>1.3%</td>
<td>291.6</td>
<td>16</td>
<td>444</td>
<td>26.7</td>
</tr>
<tr>
<td>6) Single/Multiple Node Failure</td>
<td>868</td>
<td>58.2%</td>
<td>6.7</td>
<td>26.7</td>
<td>6.3</td>
<td>72</td>
</tr>
<tr>
<td>7) Interruption (system-wide outage)</td>
<td>39</td>
<td>2.62%</td>
<td>159.2</td>
<td>5.16</td>
<td>174.2</td>
<td>8.1</td>
</tr>
<tr>
<td>ALL</td>
<td>1490</td>
<td>100%</td>
<td>4.2</td>
<td>34.5</td>
<td>13.3</td>
<td>50.5</td>
</tr>
</tbody>
</table>
An example: Blue Waters

From Di Marino et al., DSN’14

TABLE III: Failure Statistics. The last row refers to the statistics calculated across all the failure categories.

<table>
<thead>
<tr>
<th>Failure Category</th>
<th>count</th>
<th>%</th>
<th>MTBF [h]</th>
<th>MTTR [h]</th>
<th>$\sigma_{TBF}$ [h]</th>
<th>$\sigma_{TTTR}$ [h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Failure (No Interrupt)</td>
<td>164</td>
<td>11%</td>
<td>35.17</td>
<td>13.5</td>
<td>70.8</td>
<td>35.3</td>
</tr>
<tr>
<td>2) Interrupt (Failover)</td>
<td>99</td>
<td>6.6%</td>
<td>58</td>
<td>14.7</td>
<td>92</td>
<td>42.2</td>
</tr>
<tr>
<td>3) Link &amp; Node Failure (Job Failed)</td>
<td>19</td>
<td>1.3%</td>
<td>297.7</td>
<td>6.1</td>
<td>427.3</td>
<td>5.4</td>
</tr>
<tr>
<td>4) Link Failure (No Job Failed)</td>
<td>285</td>
<td>19.1%</td>
<td>19.9</td>
<td>32.7</td>
<td>51.9</td>
<td>91.2</td>
</tr>
<tr>
<td>5) Link Failure (Job Failed)</td>
<td>19</td>
<td>1.3%</td>
<td>291.6</td>
<td>16</td>
<td>444</td>
<td>26.7</td>
</tr>
<tr>
<td>6) Single/Multiple Node Failure</td>
<td>868</td>
<td>58.2%</td>
<td>6.7</td>
<td>26.7</td>
<td>6.3</td>
<td>72</td>
</tr>
<tr>
<td>7) Interruption (system-wide outage)</td>
<td>39</td>
<td>2.62%</td>
<td>159.2</td>
<td>5.16</td>
<td>174.2</td>
<td>8.1</td>
</tr>
<tr>
<td>ALL</td>
<td>1490</td>
<td>100%</td>
<td>4.2</td>
<td>34.5</td>
<td>13.3</td>
<td>50.5</td>
</tr>
</tbody>
</table>
An example: Blue Waters

From Di Marino et al., DSN’14
### TABLE IV: Breakdown of the count of the

<table>
<thead>
<tr>
<th>Failure (No Interrupt)</th>
<th>Interrupt (SWO)</th>
<th>Interrupt (Failover)</th>
<th>L/F</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HW</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSU</td>
<td>20</td>
<td>EPO</td>
<td>1</td>
</tr>
<tr>
<td>IPMI</td>
<td>15</td>
<td>Compute Blade</td>
<td>2</td>
</tr>
<tr>
<td>Fan tray assy</td>
<td>14</td>
<td>Storage module</td>
<td>2</td>
</tr>
<tr>
<td><strong>SW</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moab/TORQUE</td>
<td>33</td>
<td>Lustre</td>
<td>18</td>
</tr>
<tr>
<td>CLE/kernel</td>
<td>17</td>
<td>Moab/TORQUE</td>
<td>6</td>
</tr>
<tr>
<td>Warm swap</td>
<td>5</td>
<td>Gemini</td>
<td>3</td>
</tr>
<tr>
<td>Disks</td>
<td></td>
<td>IPMI</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Storage module</td>
<td>2</td>
</tr>
</tbody>
</table>

From Di Marino et al., DSN’14

---

The top 3 hardware and software failure root causes

<table>
<thead>
<tr>
<th>Link Failure (User Job Failed)</th>
<th>Link &amp; Node Failure (User Job Failed)</th>
<th>Single/Multiple Node Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optic 12</td>
<td>GPU 2</td>
<td>Processor 160</td>
</tr>
<tr>
<td>RAM 9</td>
<td>Gemini ASIC 1</td>
<td>RAM 158</td>
</tr>
<tr>
<td>Gemini voltage regulator 8</td>
<td>Compute blade 2</td>
<td>GPU 38</td>
</tr>
<tr>
<td>Lustre net (Lnet) 2</td>
<td>Lustre 8</td>
<td>Lustre 30</td>
</tr>
<tr>
<td></td>
<td>CLE/kernel 1</td>
<td>CLE/Kernel 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sonexion/Storage 5</td>
</tr>
</tbody>
</table>