

Merrimac Reliability

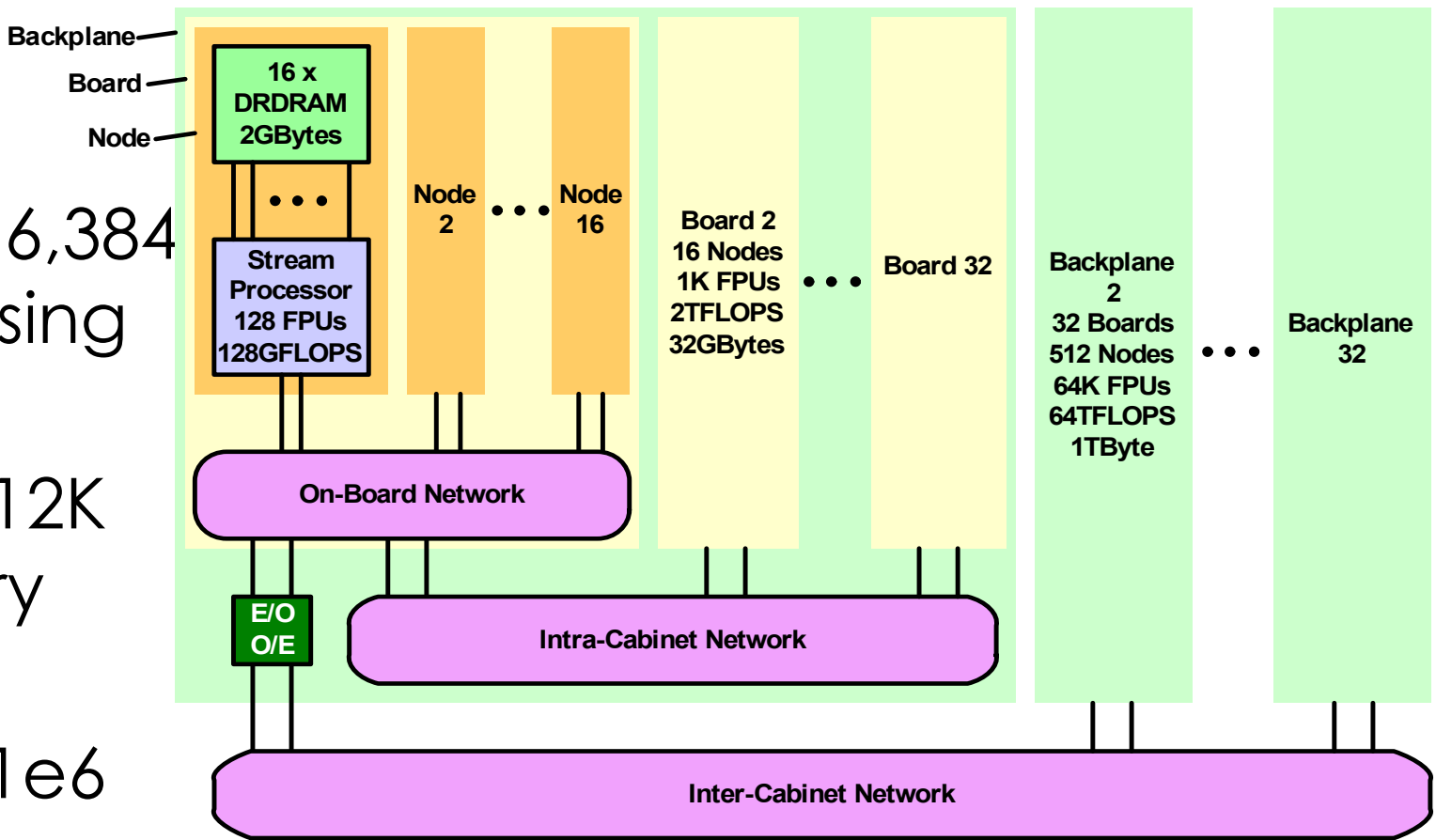
Thoughts and Plan of Record

Mattan Erez

September 30, 2003

Why do we care?

- Up to 16,384 processing nodes
- Over 512K memory chips
- MTTF ~1e6 hours per component



$\text{Pr}\{\text{no error in a 10 hour computation}\} < 0.06$

Fault-tolerance, Availability, and Reliability

- **Fault-tolerance** – property of a system that allows it to produce correct results in the presence of faults
- **Availability $A(t)$** – $\Pr\{\text{system available for useful computation at time } t\}$
- **Reliability $R(t)$** – $\Pr\{\text{system not faulted until time } t \mid \text{system functioned at } t=0\}$

We need fault-tolerance and high availability

Fault Tolerance

- Fault Detection
 - Multiple instances of a computation
 - Redundant operations
 - ECC
 - Self-checking logic
- Fault Correction
 - Retry an operation
 - Correct error using extra information
 - ECC
 - Repair and restart
 - Graceful degradation

Redundancy is the key

Space and Time Redundancy

- Time redundancy
 - Perform the separate operations using a single hardware over time
 - Low hardware cost
 - Increased use of system resources
 - Long *fault latencies*
- Space redundancy
 - Perform separate operations on different dedicated hardware
 - High overhead
 - Short fault latency

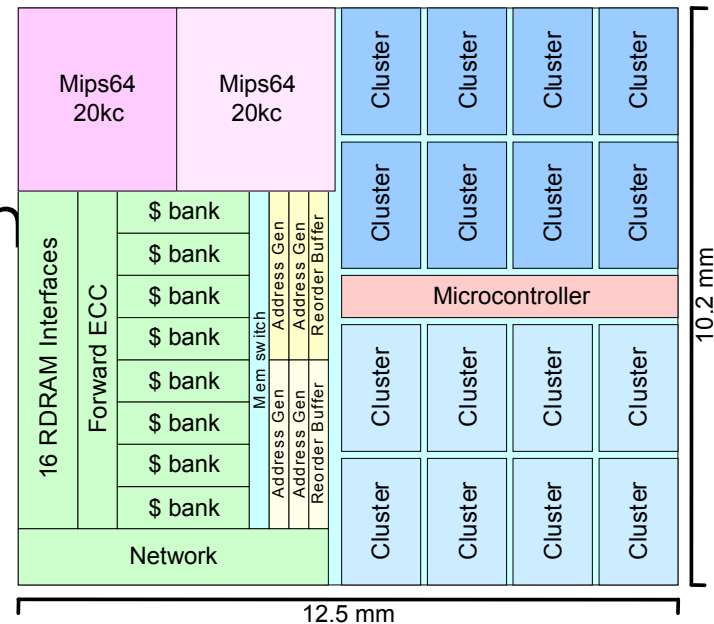
Both forms are appropriate for different purposes

Fault Detection

- Stream Processor
 - Space redundancy using duplication and ECC
- Memory
 - Standard off the shelf with ECC
- Boards
- I/O and mass storage
 - Standard off the shelf I/O systems (RAID)
- Network and back planes
 - Network is inherently fault tolerant
- Power and cooling
 - Physical sensors

Stream Processor Fault Detection

- Arrays and buses use ECC
 - Low implementation overhead
 - Used in commercial processors today
 - ~1.125 area overhead typical
 - Allows to correct 1 and detect 2 errors in a 64 bit word
- Complex logic uses duplication
 - Two scalar processors in parallel with result comparison
 - Split clusters into two groups and compare results



**Data only read once from memory,
full utilization of on-chip memories**

Fault Detection Uses Spatial Redundancy

- Low fault latencies
 - Faults are discovered quickly
 - Faults can be contained
 - Online “self-healing” possible
- Efficient use of system resources
 - No need to re-use critical resources
 - Conserves off-chip bandwidth
- Higher fault detection *coverage*
 - Lower probability of faults masking one another
 - Reduced probability that faults go unfixed
 - Increased availability

Fault Correction

- Stream Processor
 - Fault masking of transient (and some permanent) faults in arrays
- Memory
- Board
- I/O and mass storage
 - No fault tolerance (on top of the RAID system)
- Network and back planes
 - Network is inherently fault tolerant
- Power and cooling
 - Online redundancy masks faults
 - Diode voting and temperature sensors

If masking doesn't work need to restart

Checkpoint and Restart

- Checkpoint all transient state every T_c time
- Upon unmasked fault
 - Diagnose failed component
 - Repair if diagnostic failed
 - Rollback state
 - Restart computation
 - On the same components (retry)
 - On repaired components
- Assumes perfectly reliable disk system
- Performance impact depends on I/O capabilities, repair times, and probability of fault

Checkpoint-Restart Evaluation

- Simple model assuming
 - No faults during checkpoint or rollback
 - Calculation based on expected times for failure and recovery
 - Once system is restarted it is like new (in terms of $\Pr\{f\}$)

$$\text{slow down} = \frac{T_{cp,i} + T_{cp,d}}{T_{cp,i}} + \frac{T_{cp,i}}{T_f} \cdot \left(\frac{T_{cp,i}}{2} + T_r \right)$$

$$\frac{\partial \text{slow down}}{\partial T_{cp,i}} = -\frac{T_{cp,d}}{T_{cp,i}^2} + \frac{T_{cp,i} + T_r}{T_f}$$

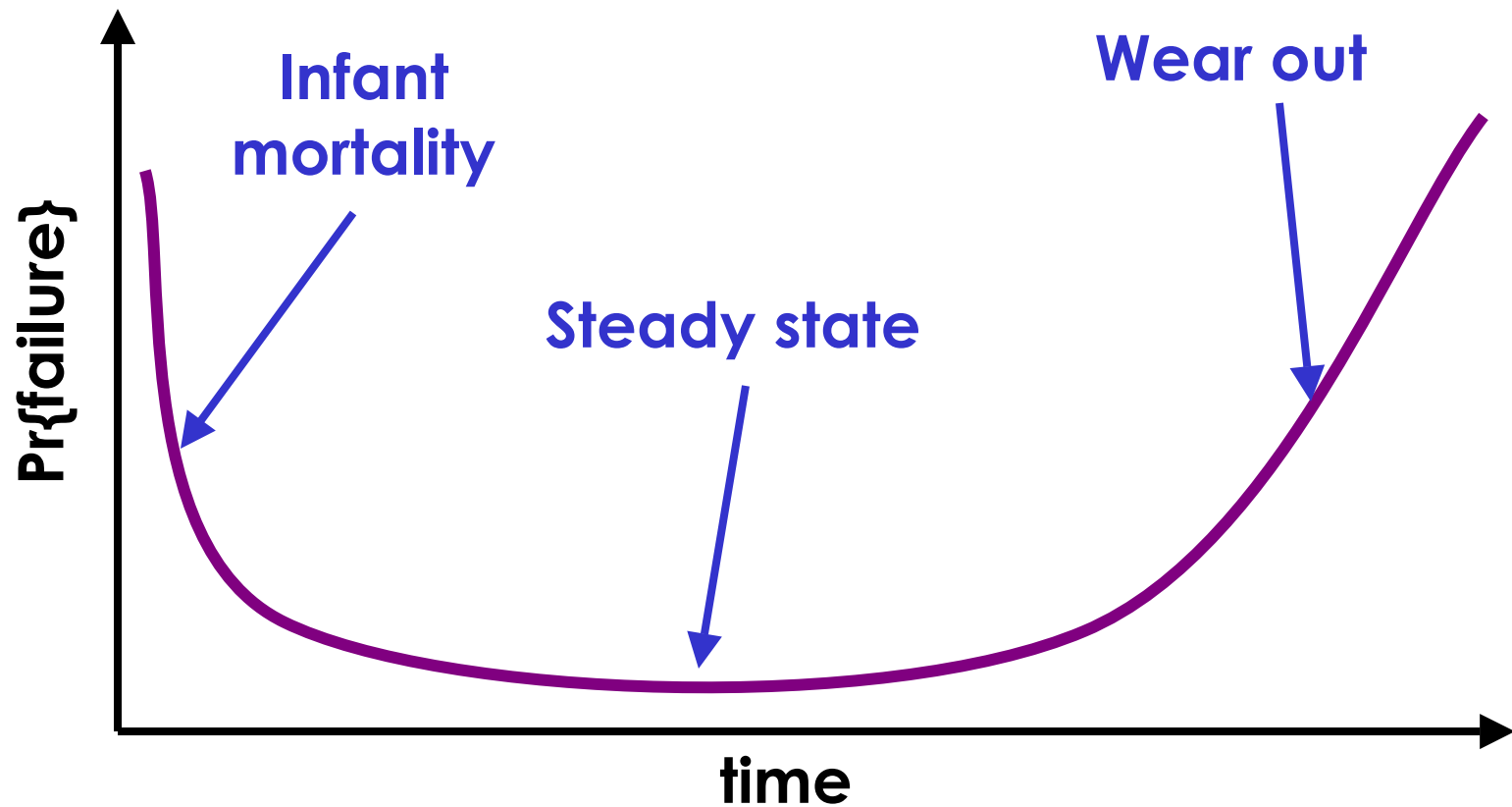
$T_{cp,i}$ – checkpoint interval
 $T_{cp,d}$ – checkpoint duration
 T_f – mean time between
 T_r – recovery time

$$\frac{\partial \text{slow down}}{\partial T_{cp,i}} = 0$$

$$\frac{1}{T_f} T_{cp,i}^3 + \frac{T_r}{T_f} T_{cp,i}^2 - T_{cp,d} = 0$$

In this evaluation slowdown = availability

Fault Probability

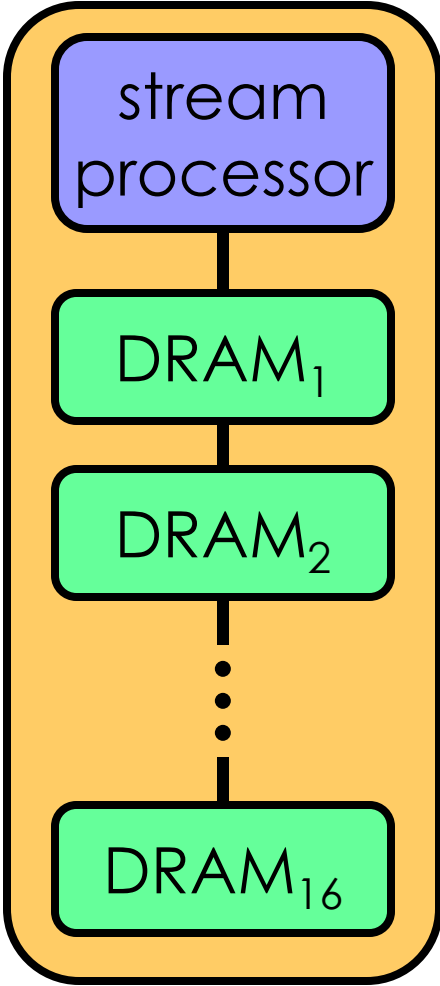


During steady state model $R(t) = e^{-\lambda t}$

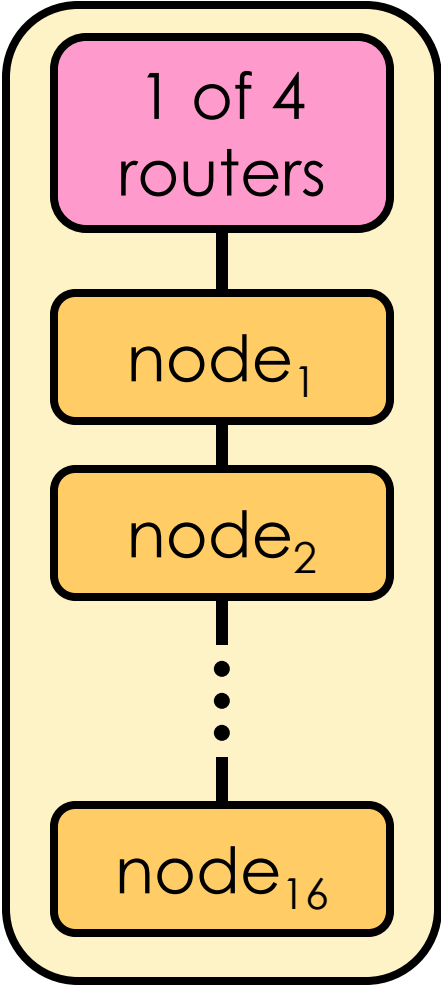
System Fault Probability

- Model system using a graph of components
 - On-line redundancy – components in parallel
 - $R_{par}(t) = 1 - \Pi(1-R_i(t))$
 - No redundancy – components in serial
 - $R_{ser}(t) = \Pi(R_i(t))$
 - M of N redundancy – generalization of parallel
 - $R_{NofM}(t) = \Sigma(C_i^N R^{N-i} (1-R)^i)$
- Assume failure of components is independent
- Assume steady state constant instantaneous probability of fault

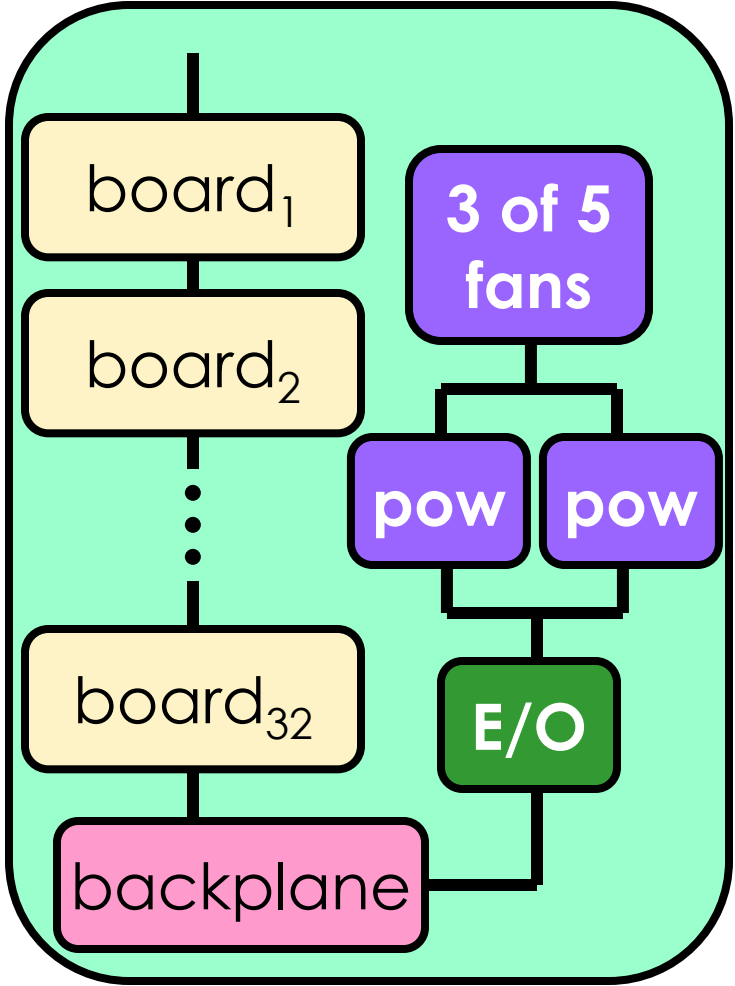
System Fault Probability



Node



Board



Cabinet

Calculating $R_{sys}(t)$

- Use models and formulae from previous slide
 - Simple
 - Decent estimate
 - In our system
- Use automated tools
 - Input model diagram
 - Input component reliabilities
 - Allows time-variant probabilities
 - More accurate
 - At least one tool freely available (SHARPE from Duke)

System Requirements

- $A(t)$ - high system availability for all t
 - $\lim A(t) = \text{availability}$
- In our case we can look at slowdown
 - Single application usage model

To Do

- Identify and evaluate ECC options
- Take into account cost of comparators in area estimates
- Evaluate use of CHIPKILL type memory architecture
- Gather data on component reliability
- Use reliability software for analysis
- Evaluate network fault tolerance
- Quantify tradeoffs of space/time redundancy in the stream processor
- Do we need the fast non-reliable option?
- Cost analysis (power and cooling especially)

Comments

- Power redundancy should go on the board
 - Voltage converters on board
 - Redundant 48V lines feeding in to each case
- Analyze soft and hard errors separately
- Board reliability should be fairly high in our system
 - Few components
 - PC numbers reflect very low cost and perhaps generic faults
- Look into shifting fault-detection to software
 - Spatial partitioning on entire nodes
 - Compare results of stream stores