On Determining a Viable Path to Resilience at Exascale

Frank Mueller
Dept. of Computer Science
North Carolina State University

NC STATE UNIVERSITY
Department of Computer Science

Resilience in HPC

System	# CUPs	MTBF
ASCI White	8,192	5/40 hrs
Google	1,5000	20 reboots/day
ASC BD/L	212,992	7 hrs
Jaguar	300,000	5/52 hrs

- HPC: 10k-100k nodes
  - Some component failure likely
  - System MTBF becomes shorter
  - processor/memory/IO failures
- MPI widely used for scientific apps
  - Problem w/ MPI: no recovery from faults in the standard
- Currently FT exist but…
  - not scalable
  - mostly reactive: process checkpoint/restart
  - restart entire job → inefficient if only one/few node(s) fail
  - overhead: re-execute some of prior work
  - issues: checkpoint at what frequency?
- 100 hr job → +150 hrs for chkpt / 55%-85% time wasted [Philp'05,Daly'08]

Exascale Resilience

- 1 billion cores
- ~ 1 million components
- MTBF/node 50 yrs
  (52 hrs for Jaguar)

System attributes	2010	"2015"	"2018"
System peak FLOPS	2 Peta	200 Peta	1 Exa
Power	6 MW	~15 MW	~20 MW
System memory	0.3PB	5 PB	32-64PB
Node performance	125 GF	0.5TF or 7 TF	1 TF or 10x
Node memory BW	25GB/s	0.1TB/s or 10x	0.4TB/s or 10x
Node concurrency	12	O(100)	O(1k) or 10x
TotalNode Interconn BW	1.5 GB/s	20 GB/s or 10x	200GB/s or 10x
System size (nodes)	18,700	50,000 or 1/10x	O(100,000) or 1/10 x
MTTI	days	O(1day)	O(1 day)

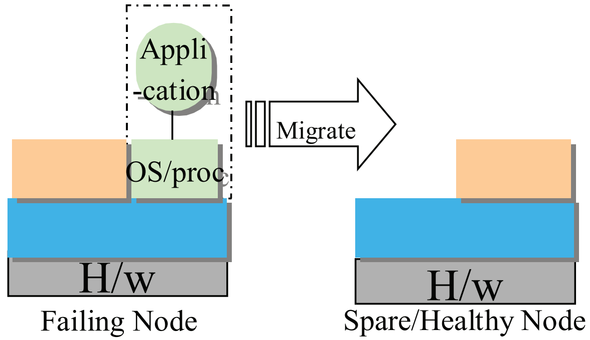
- Goal: MTBF ~ 1 day
- 10x-100x > components
- Reliability ~ # components

➢ need 10x-100x reliability improvement
  - H/w: 10x (or less → smaller fabs)
  - S/w: 10x (or more → this talk)

- **How can this be achieved?**

Proactive Resilience: Live Migration

- OpenIPMI health monitoring → predict node failure
- takes preventive action (instead of "reacting" to a failure)
  - Live migration of process/OS → healthy node
  - transparent to app/process/OS)
- OS vs. process level: Abstraction vs. overhead tradeoff
  - Copy pages while running
  - Then stop & copy rest
  - Kill src, continue dst



- Implemented over
  1. Xen
  2. Ours: Open MPI/LAM + BLCR + Linux kernel
     - BLCR extensions
     - Kernel enhancements (dirty bit tracking in PTEs)
     - Add'l MPI support

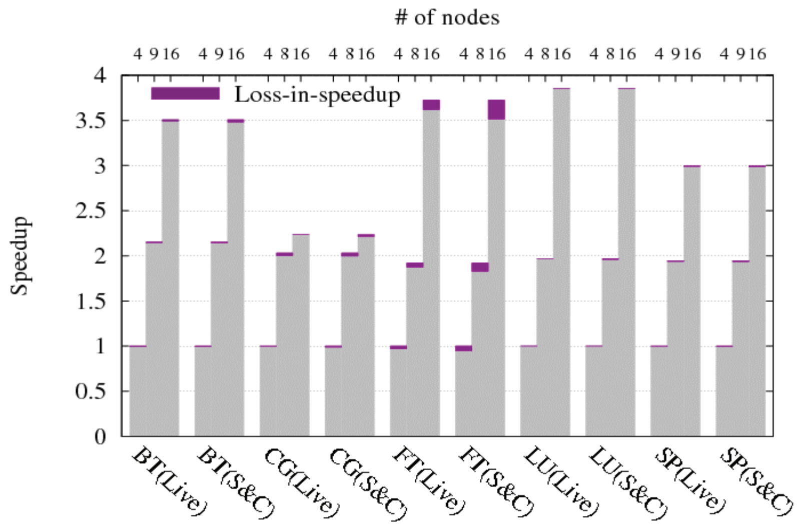
Process vs. OS Migration [ICS'07+SC'08]

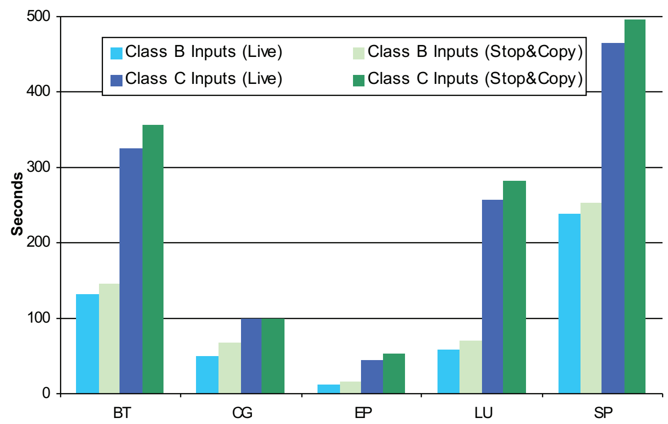
Process-level

- 2.6-6.5 sec live migration
- 1-1.9 sec frozen migration
  - xfer subset of OS image
- 1-6.5 secs prior warning

Xen virtualization

- 14-24 sec live migration
- 13-14 sec frozen migration
  - xfer entire VM image
- 13-24 sec prior warning





Proactive FT Complements Reactive FT

$$T_c = \sqrt{2 \times T_s \times T_f}$$ [J.W.Young Commun. ACM '74]

Tc: time interval between checkpoints

Ts: time to save checkpoint information (mean Ts for BT/CG/FT/LU/SP Class C on 4/8/16 nodes is 23 seconds)

Tf: MTBF, 1.25hrs [I.Philp HPCRI'05]

$$T_c = \sqrt{2 \times 23 \times (1.25 \times 60 \times 60)} = 455$$

70% faults [R.Sahoo et.al KDD '03] can be predicted and handled proactively

$$T_c = \sqrt{2 \times 23 \times (1.25/(1-0.7) \times 60 \times 60)} = 831$$

Cut the number of chkpts in half: 455→831 seconds

Contributions (2)

- Reactive FT
  - Save restart cost: **70%** < job queuing, MPI startup
- Novel, proactive fault resilient scheme w/ process live migration
  - Provides transparent & automatic FT for arbitrary MPI apps
  - Less overhead than reactive
  - Also complements reactive → lower checkpoint frequency
  - Process-level: **½ overhead** of OS-level
  - **½ the chkpts** when 70% faults handled proactively
- Incr. Chkpt → less overhead & I/O pressure, **needs garbage coll.**
- Back migration → original performance, **wins if >10% work left**

Resilience Advances in HPC (3)

Redundancy: double/triple each MPI task

- Either need 2x/3x more nodes (and 2x/3x # msgs) [our work]
- Or need 2x/3x more bandwidth [SNL]

- Why? (*) [Ferreira at al. SC'11]

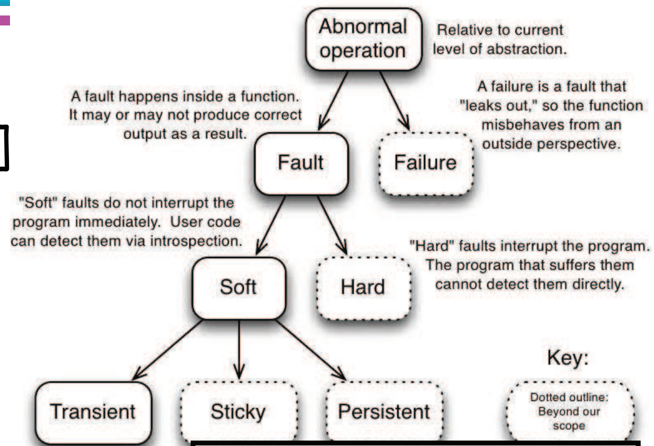
No. of Nodes	Work	Checkpoint	Re-computation	Restart
100	96%	1%	3%	0%
1,000	92%	7%	1%	0%
10,000	75%	15%	6%	4%
100,000	35%	20%	10%	35%

- C/R not scalable: > 50% of time spent in C/R
  - (maybe less due to C/R optimizations)

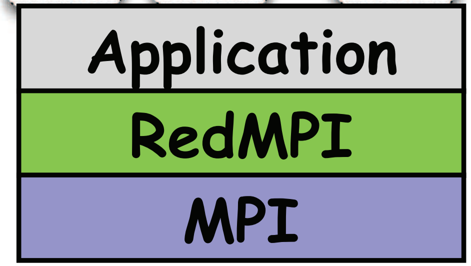
Sandia National Laboratories

Design of Redundancy: RedMPI [SC'12]

- RedMPI library, related to
  - MR-MPI [Engelmann&Boehm PDCN'11]
  - rMPI [Ferreira et al. SC'11]



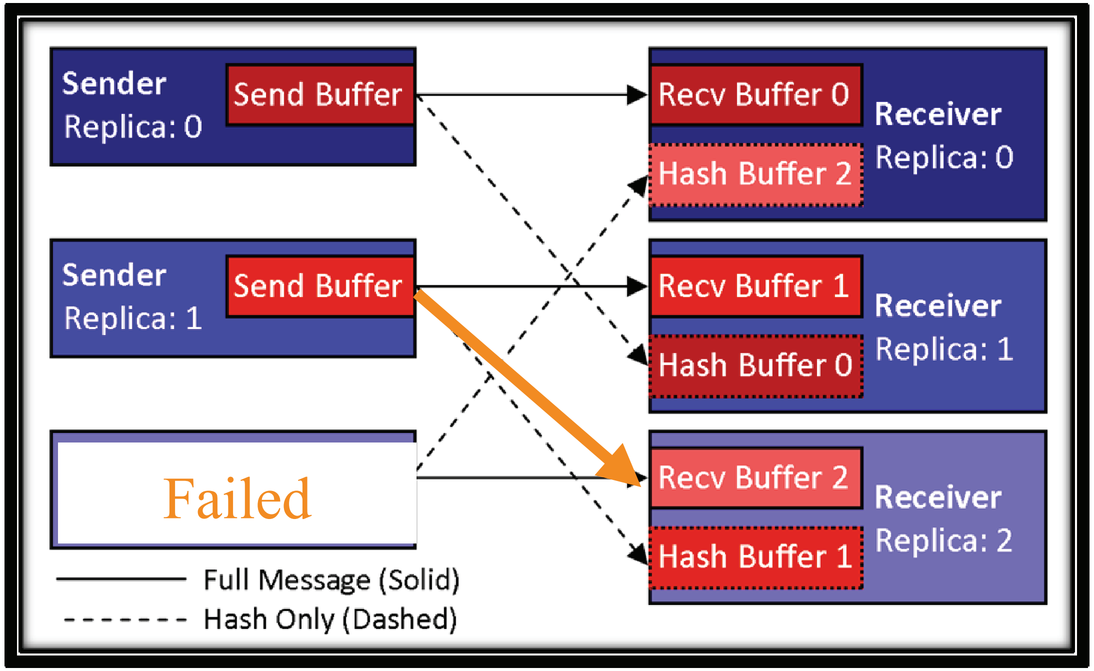
- Works at profiling layer
- Goal: guard faults that leak into msgs (IO)
  - file IO also handled [Engelmann PDP'12]
- Intercepts MPI function calls



- Each redundant copy needs to receive same messages in same order
- Each message is sent/received r number of times
  - opt. hashes to detect silent data corruption (SDC)
  - Why?Multi-bit flips,DRAM err in 2% of DIMMs/year [Schroeder'11]

RedMPI – MsgPlusHash Protocol [SC'12]

- optimization for critical path : msg not corrupt
- Send r msgs + r small hash messages: $(r_{data}+r_{hash})$



- faster than r^2 msgs
- Patches faulty nodes
- SDC: detect&correct

- Main objective: catch memory errors (interconnects have CRC)

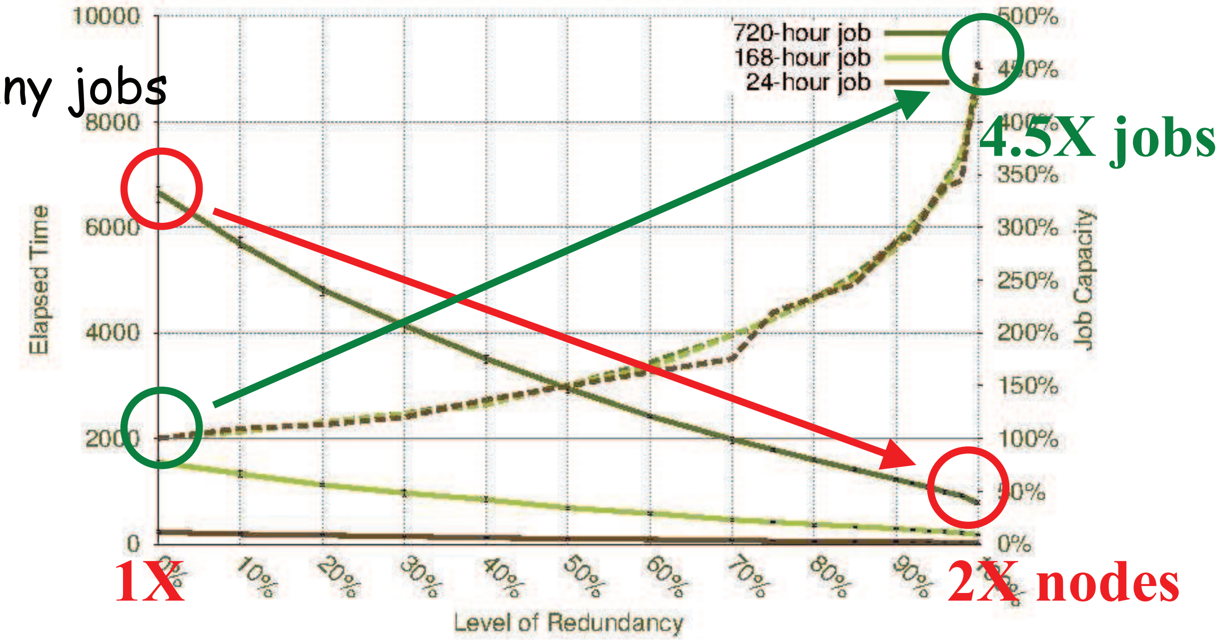
RedMPI Overhead & Benefit

- Overhead: 1-11% time

	Dual Redundancy	Triple Redundancy
NPB CG	6%	11%
NPB LU	8%	10%
SWEEP3D	0%	1%

- Benefit:
  at 2X # nodes
  → run 4.5X as many jobs

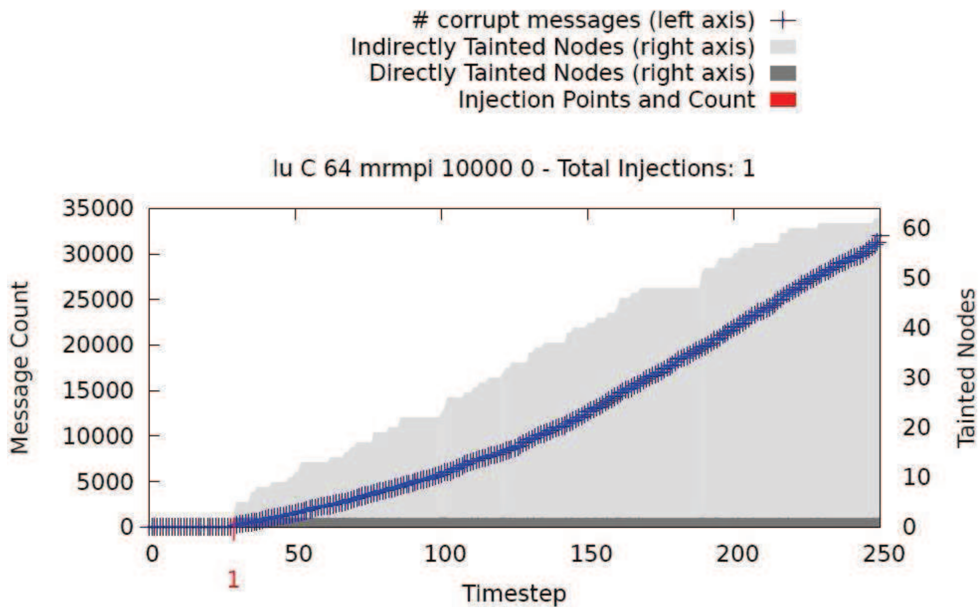
Caveat:
simplistic model
→ fixed next

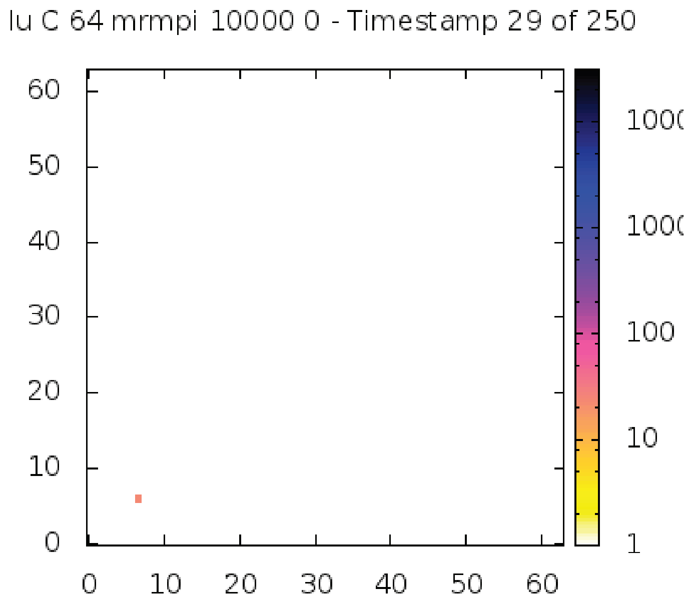


- Exascale: capacity computing ✓, ~~capability computing~~

Fault Injection: SDC Correction (TMR)

- Inject 1 bit flip / 5M msgs
  1. Keep on running: single, corrected msg → 90% of cases, others:
  2. > 1 sent corrupt msgs simultaneously → detected & job failed
  3. Tainted buffer reuse, propagates





Modeling Preliminaries [ICDCS'12]

- A physical process (node) follows an exponential failure distribution
  - θ - Mean Time Between Failures (MTBF)
- A system of virtual processes has an exponential failure distribution
  - Θ - system MTBF
  - r - Degree of Redundancy
  - α - Communication to Computation ratio
- Failures arrive following a Poisson process
- Redundancy increases the system reliability.

Modeling Preliminaries

- Effect of Redundancy on Execution Time
  - Application execution time ≥ base execution time
  - Dependent upon many factors
    - Placement of processes, communication to computation ratio, degree of redundancy, relative speed, etc.
  - Consider ideal execution environment:

$$\underbrace{t}_{Total\ time} = \underbrace{\alpha t}_{Communication} + \underbrace{(1-\alpha)t}_{Computation}$$

$$t_{Red} = (\alpha t)r + (1-\alpha)t$$

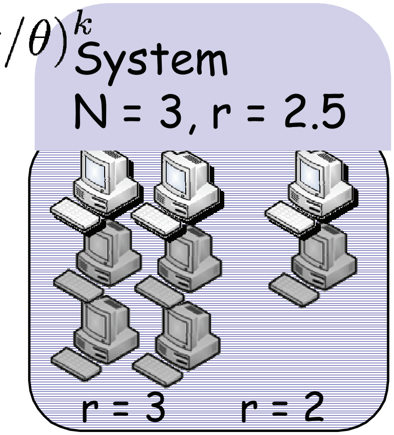
System Reliability Model

- Probability of failure of a physical node:

$$\Pr(Node\ Failure) = 1 - e^{\frac{-t}{\theta}} = 1 - (1 - t/\theta) = t/\theta$$

- Probability of survival of a virtual node with some integer k degree of redundancy

$$\Pr(Virtual\ Node\ Survival) = 1 - \prod_{i=1}^{k} t/\theta = 1 - (t/\theta)^k$$



- Partition N virtual processes into sets of real-world redundancy levels

$$N = N_{\lfloor r \rfloor} + N_{\lceil r \rceil}$$

- Reliability of the system may be expressed as

$$\Pr(All\ Virtual\ Processes\ Survive)$$

$$\Pr(All\ N_{\lfloor r \rfloor}\ Processes\ Survive \text{ and } All\ N_{\lceil r \rceil}\ Processes\ Survive)$$

$$R_{sys} = \left[1 - (t_{Red}/\theta)^{\lfloor r \rfloor}\right]^{N_{\lfloor r \rfloor}} \times \left[1 - (t_{Red}/\theta)^{\lceil r \rceil}\right]^{N_{\lceil r \rceil}}$$

System Reliability Model

- Assuming an Exponential distribution,

$$R_{sys} = e^{-\lambda_{sys} t_{Red}}$$

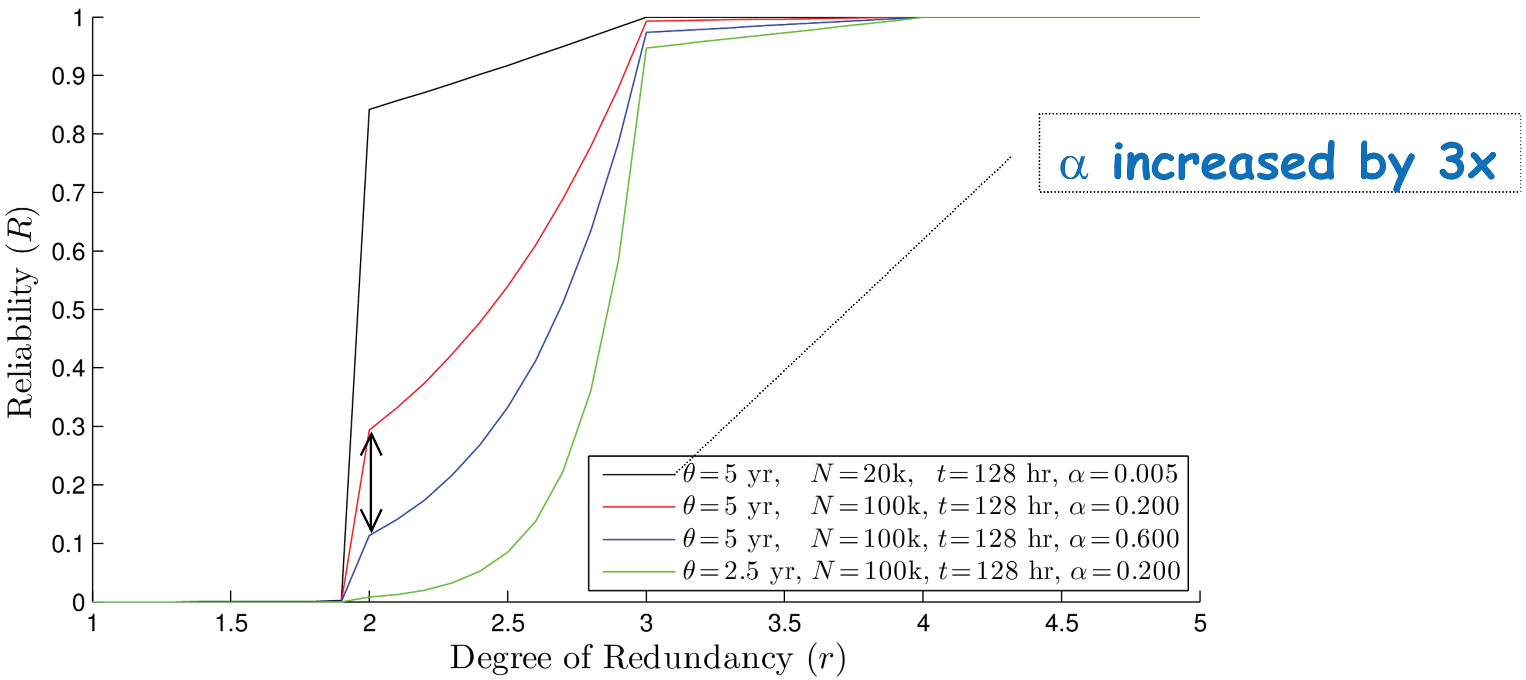
- The system failure rate is

$$\lambda_{sys} = -\ln R_{sys} / t_{Red}$$

- System MTBF is

$$\Theta_{sys} = \frac{1}{\lambda_{sys}}$$

Effect of Redundancy on Reliability [ICDCS'12]

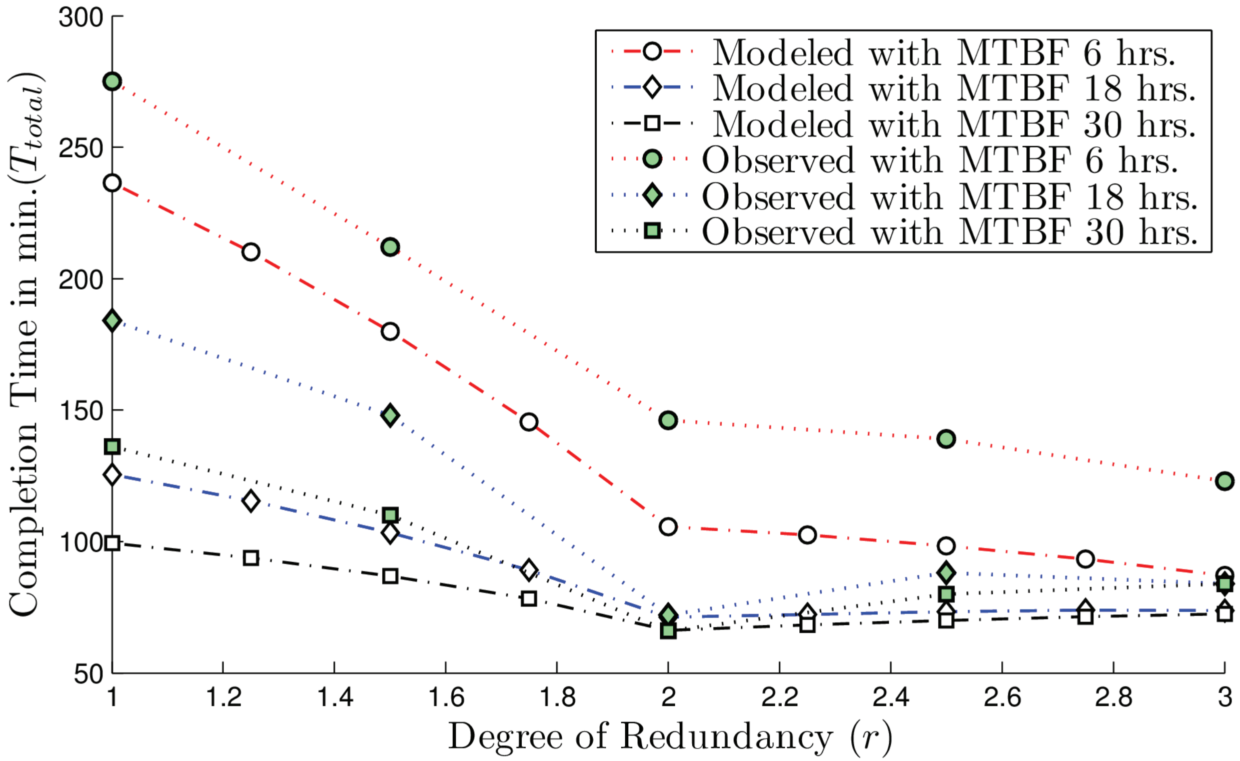


Quantify how redundancy increases system reliability

- Reliability spikes at whole number redundancy levels
- Reliability now depends on α = communicate/compute ratio
  - **Time is a function of alpha**

$$R_{sys} = \left[1 - (t_{Red}/\theta)^{\lfloor r \rfloor}\right]^{N_{\lfloor r \rfloor}} \times \left[1 - (t_{Red}/\theta)^{\lceil r \rceil}\right]^{N_{\lceil r \rceil}}$$

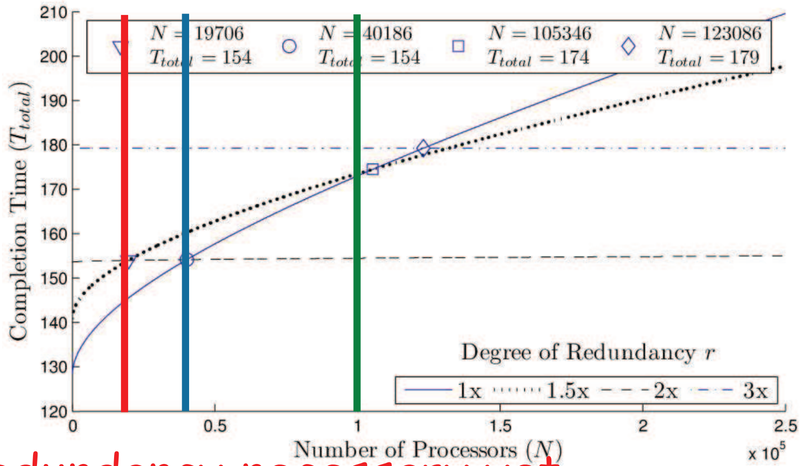
Results – Model vs. Experiment



- Experiments agree with model (+ additive const)
  - **minimum runtime always achieved at 2x redundancy**

Results – Extrapolation based on Jaguar

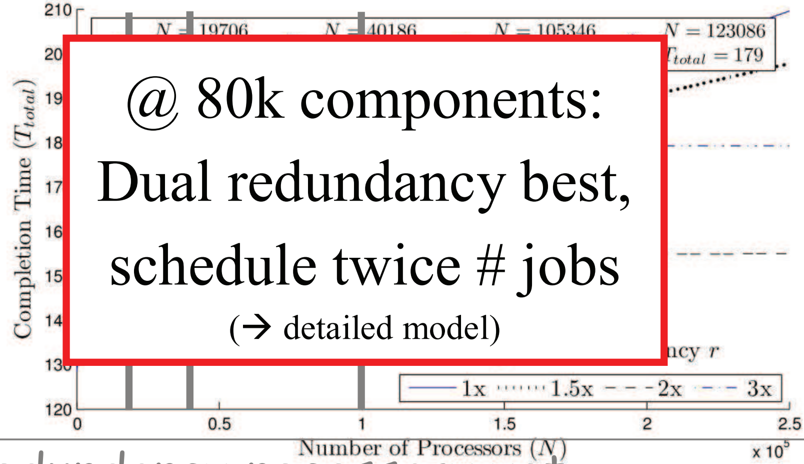
- Jaguar: node MTBF ~ 50 years (on 18,688 nodes)
- K-Computer: has 2.3X more components (equiv. 44,064)
- Exascale lane 1: ~100k nodes



- Jaguar: No redundancy necessary yet
- Titan maintains node count/component
  - **increases core count by 33%, adds GPUs→effect?**
- K-Computer: Dual redundancy break-even
- Exascale: 12% faster under dual redundancy than single,
  - close to triple redundancy for free (free SDC correction)

Results – Extrapolation based on Jaguar

- Jaguar: node MTBF ~ 50 years (on 18,688 nodes)
- K-Computer: has 2.3X more components (equiv. 44,064)
- Exascale lane 1: ~100k nodes



@ 80k components:
Dual redundancy best,
schedule twice # jobs
(→ detailed model)

- Jaguar: No redundancy necessary yet
- Titan maintains node count/component
  - **increases core count by 33%, adds GPUs→effect?**
- K-Computer: Dual redundancy break-even
- Exascale: 12% faster under dual redundancy than single,
  - close to triple redundancy for free (free SDC correction)

Silent Data Corruption (SDC) -- Revisited

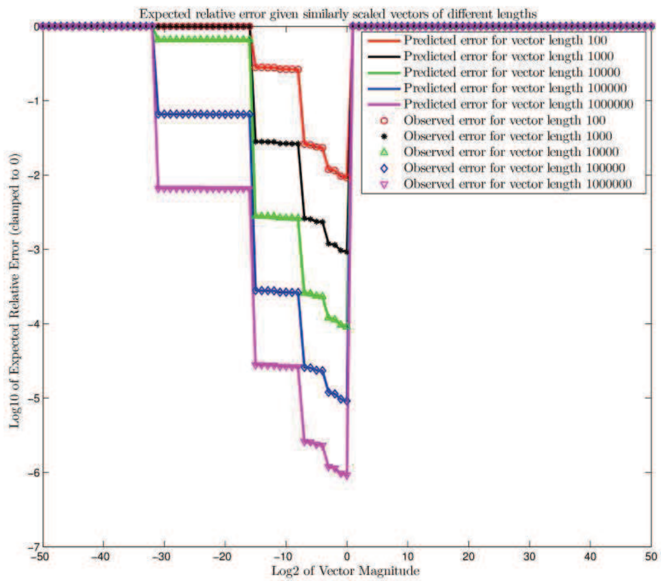
SDC @ ORNL Titan:

- Many single bit flips
- **1 double bit flip/24 hrs**
- 20 faults/ hr
  - 1 missed heartbeat/3min.
  - 4 kernel panics/day
- Common approaches:
  - replication+voting
  - algo.-based FT (ABFT)
- Problem: bit identical
  → vs. numerical convergence
- **Goal: precision-awareness**
  →Tolerate small errors (due to SDCs)
  - Focus data : IEEE float awareness

Quantify Impact of SDC on FP Ops [SIAM/TR'13]

- Model likelihood of bit flip to affect results
  - Mantissa vs. exponent
- For vector dot product (DP): $\vec{u} \cdot \vec{v} = \sum_{i=1}^{N} c_i; \quad \text{where} \quad c_i = u_i v_i.$
- Plot for same |vector|:
  Expected relative error [y axis]
  over vector magnitude [x axis]:
  - (DP – flipped DP) / DP
- Flip lower 10 bits of exponent
  - Spikes due to patterns:
    1023 vs. 1024
    many 1s, few 0s vs. …

➢ Model fits experiments

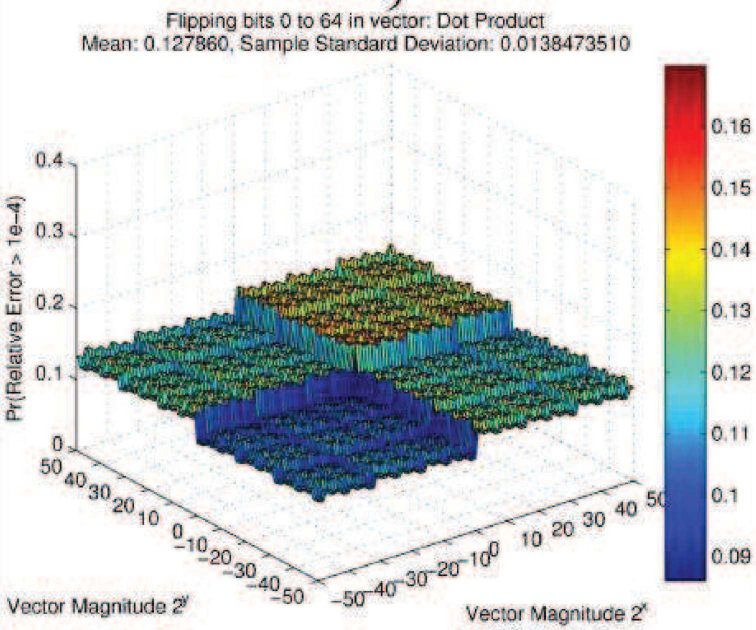


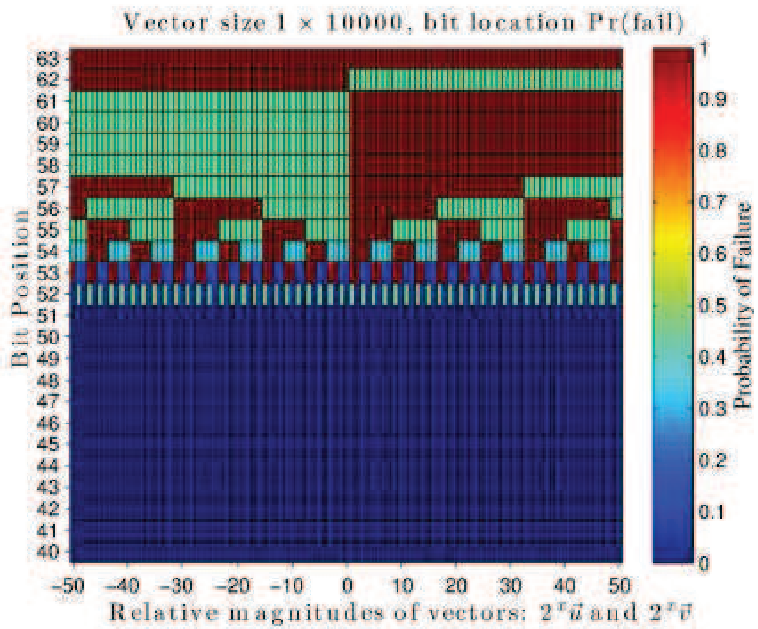
Quantify Impact of SDC on FP Ops (2)

- Monte-Carlo sampling
  - via random # gen.
  - Expected # flips
    → Pr(Error>10^{-4}) [y axis]
- Slice across
  - Similar magnitudes (front to back)
- Shows bit position of error

➢ Should scale # to max precision→few flips affect you

1x10,000

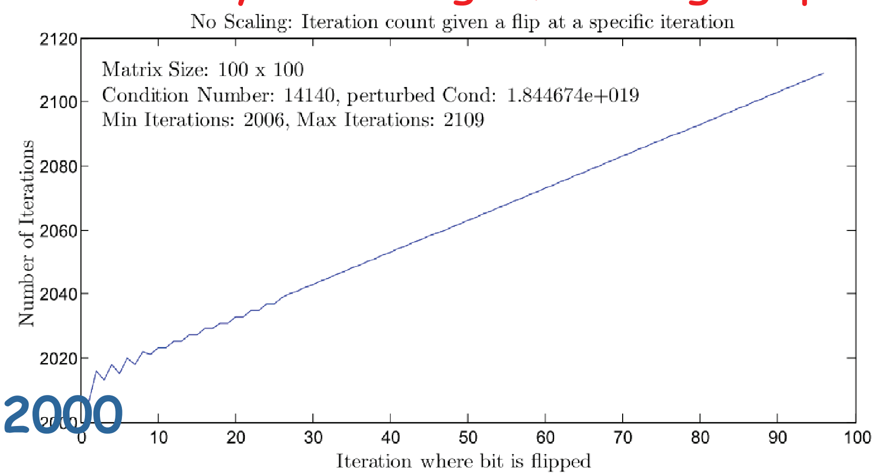


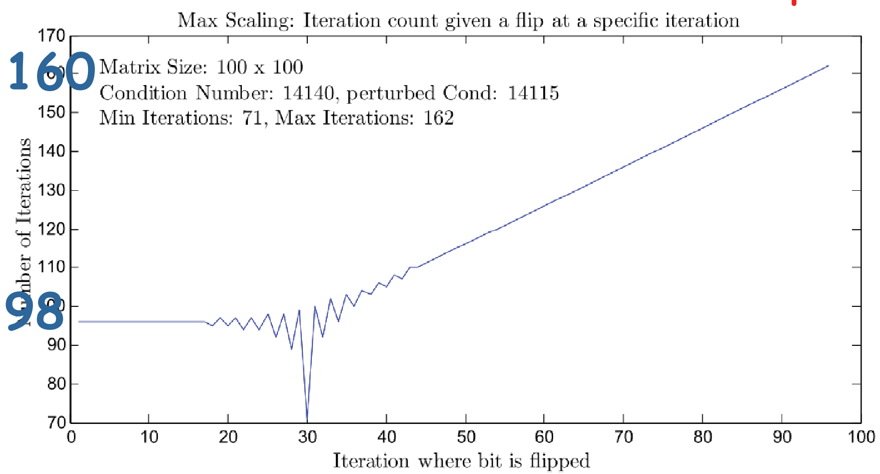


Quantify Impact of SDC on FP Ops (3)

- Order-one iterative methods:
  - Always converges after bit flip
  - How about stationary methods?
- Case study: Jacobi unscaled
  - No fault → 98 iterations
  - Bit fliped @ iteration X
    → converges but 2000+ iters (20x)
- Jacobi max. scaled
  - Anomaly → flips can help !
  - Converges
    → 0-65 more iterations (1x-1.6x)

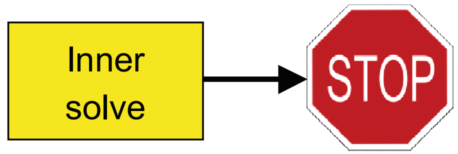
➢ always converges, scaling helps a lot → reduces overhead after flip





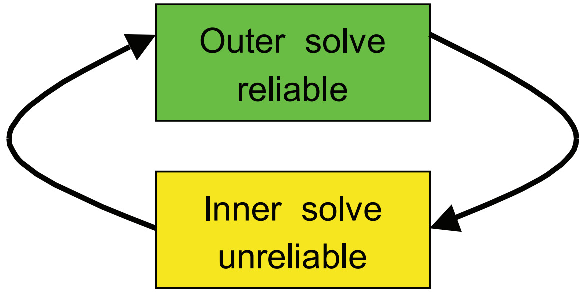
Reliability Models: Sandboxing [IPDPS'14]

- Current Model: Fail-stop
  Roll-back recovery → redo work



  - System tries to detect all soft faults → **bit identity**
  - Turn all detected soft faults into hard faults
  - Detected local faults become global
  - Checkpoint / restart is only recovery model
- Or ABFT → **bit identity**
  - Often unnecessary, hard for sparse

- **Better Model: Sandbox**
  Run thru errors → fwd resilience

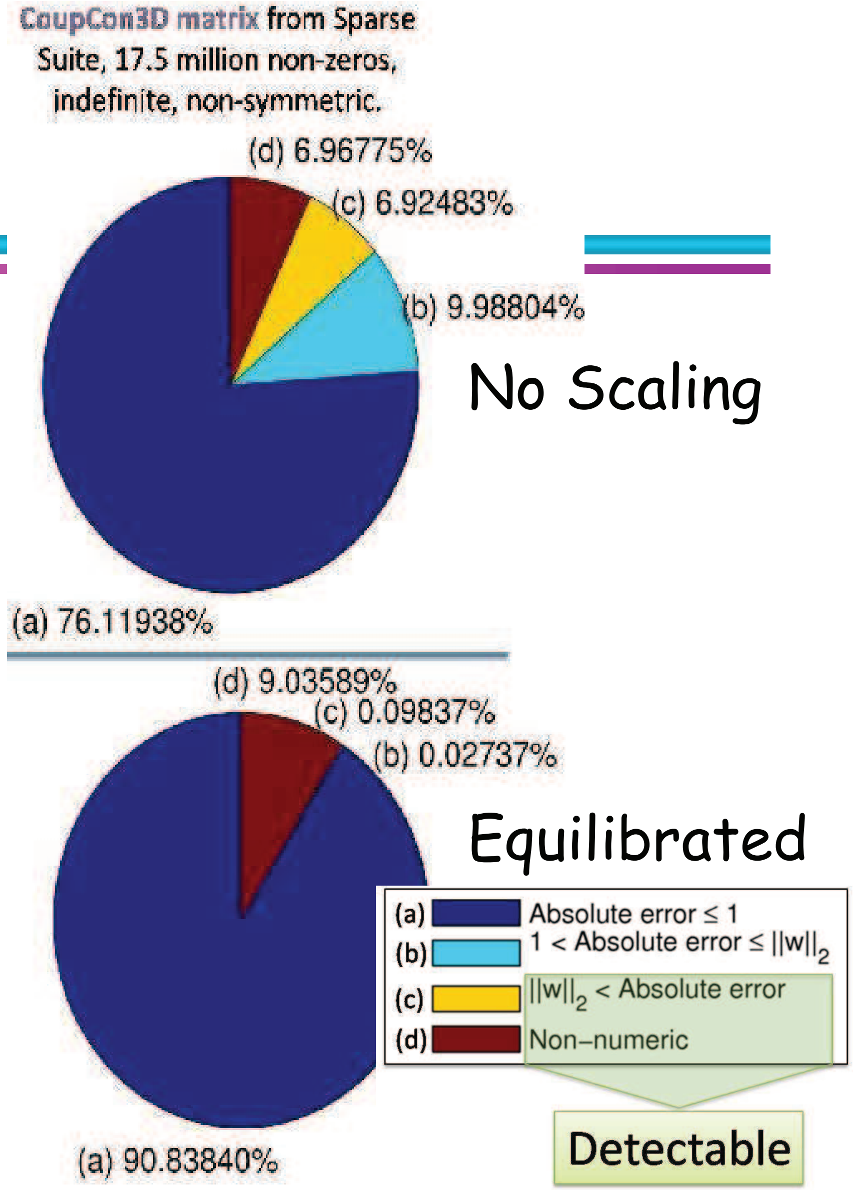


  - Isolate unreliable data & computation in a box
- Reliable code invokes box
  - Local faults stay local
  - App gets flexibility to define recovery model

Fault Tolerant GMRES

- Use Sandbox model
  - GMRES as inner solve (unreliable preconditioner)
- Check result: compute residual
  - Drive theoretical bounds on Arnoldi process (inner kernel over matrix A)
  - Based on l_2 "L two" norm:
  - Requires determining largest singular value of A
  - allows detection of large perturbations
- Experiment: Faults injected as perturbations to matrix

➢ **Bounded error ignored:** 99.97% of flips detected/irrelevant after equilibration



Selective Reliability: FGMRES

Algorithm 1 Flexible GMRES (FGMRES)

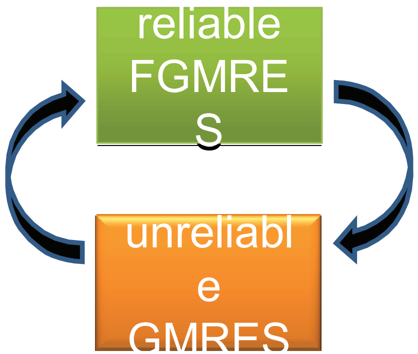
Input: Linear system $Ax = b$ and initial guess x_0
Output: Approximate solution x_m for some $m \geq 0$

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$ ▷ Unpreconditioned initial residual
$\beta := \|\mathbf{r}_0\|_2$, $\mathbf{q}_1 := \mathbf{r}_0/\beta$
for $j = 1, 2, \ldots$ until convergence **do**
  Solve $\mathbf{q}_j = \mathbf{M}_j \mathbf{z}_j$ ▷ Apply current preconditioner
  $\mathbf{v}_{j+1} := \mathbf{A}\mathbf{z}_j$ ▷ Apply the matrix $\mathbf{A}$
  for $i = 1, 2, \ldots, k$ **do** ▷ Orthogonalize
    $h_{i,j} := \mathbf{q}_i \cdot \mathbf{v}_{j+1}$
    $\mathbf{v}_{j+1} := \mathbf{v}_{j+1} - h_{i,j}\mathbf{q}_i$
  end for
  $h_{j+1,j} := \|\mathbf{v}_{j+1}\|_2$
  Update rank-revealing decomposition of $\mathbf{H}(1{:}j, 1{:}j)$
  if $\mathbf{H}(j+1, j)$ is less than some tolerance **then**
    if $\mathbf{H}(1{:}j, 1{:}j)$ not full rank **then**
      Did not converge; report error
    else
      Solution is $\mathbf{x}_{j-1}$ ▷ Happy breakdown
    end if
  else
    $\mathbf{q}_{j+1} := \mathbf{v}_{j+1}/h_{j+1,j}$
  end if
  $\mathbf{y}_j := \arg\min_y \|\mathbf{H}(1{:}j+1, 1{:}j)\mathbf{y} - \beta \mathbf{e}_1\|_2$
  $\mathbf{x}_j := \mathbf{x}_0 + [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_j]\mathbf{y}_j$ ▷ Compute solution update
end for

M_j are the preconditioners:

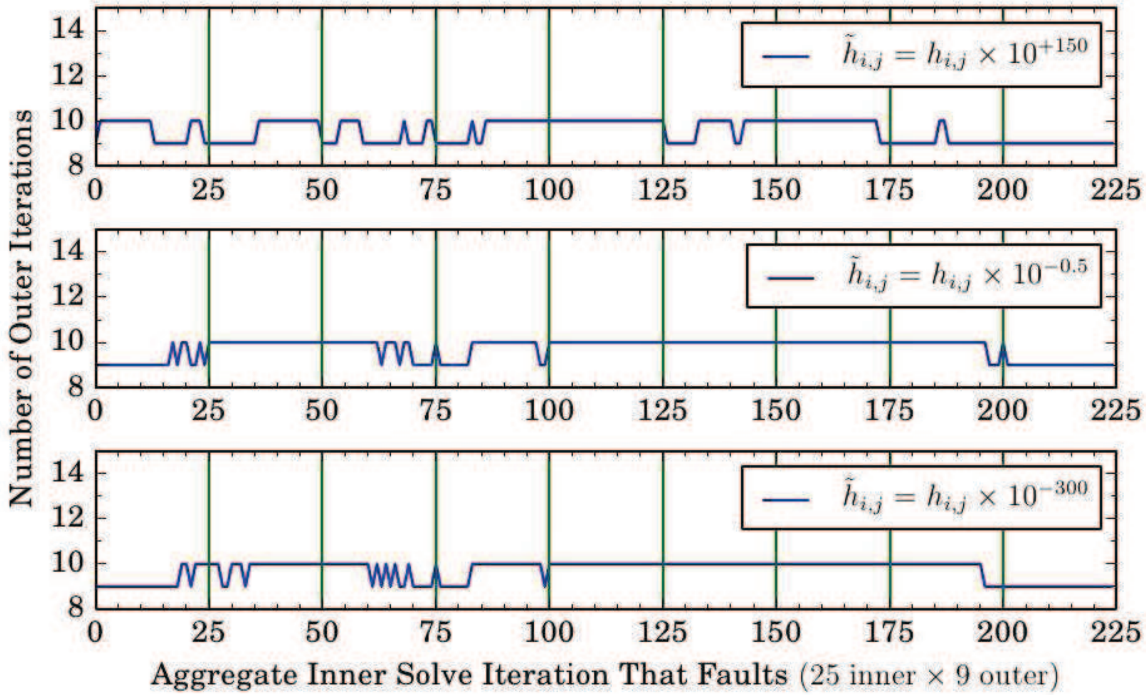
$$\mathbf{z}_j = \text{gmres}(\mathbf{A}, \mathbf{q}_j)$$

M_j represents using GMRES as a preconditioner…
inside FGMRES.



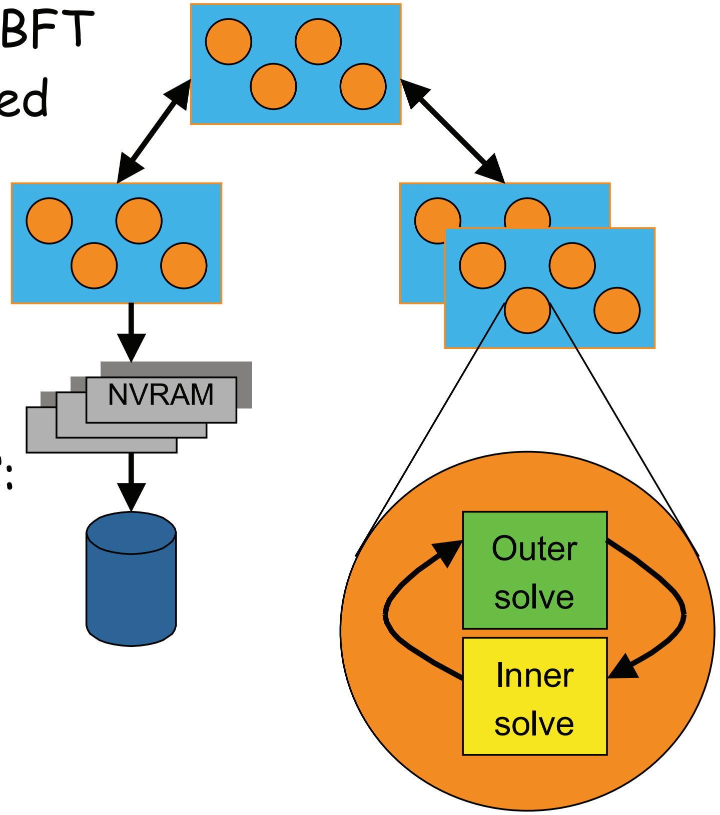
Fault Tolerant GMRES

- Sandbox has little overhead
  - **1 extra outer iteration**
  - Except for fault during 1st iteration → 2-5 extra iterations
    - Depends on magnitude of error



Exascale Vision

- Skeptical Programming for SDC → bounded error ABFT sandbox
  - for solvers/numerical libs → fwd recovery
  - Sandbox bounds vs. bit-identical ABFT
  - cheap; but not everything protected
- Checkpointing for Fail-Stop (FS):
  - Hierarchical: coord+uncoord
  - Incremental
  - NVRAM to bleed off to PFS
- Redundancy for extreme scale FS+SDC:
  - When chkpts too costly
  - Duality is enough [submitted]



64 Cores/Chip: Scalable+Predictable Runtime

Core Scalability Limitations

- Shared bus: contention
  → MESI coherence, max. 4-8 cores?
- Hypertransport/Quickpath/Rings:
  same, max. 16-32 cores?
- Memory Controllers → more contention

Network-On-Chip (NoC): Mesh

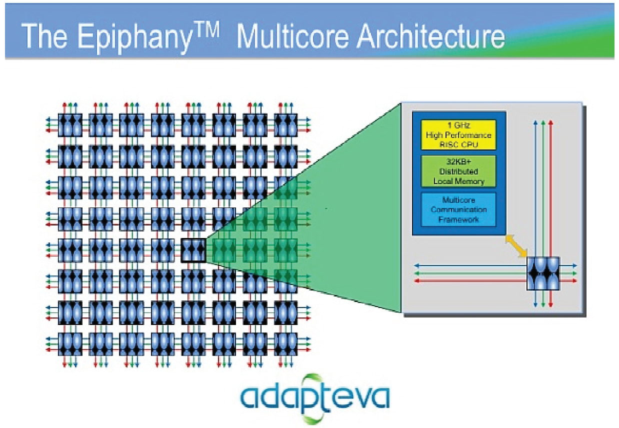
- High speed packetized memory request
- NUMA design→ more memory bandwidth

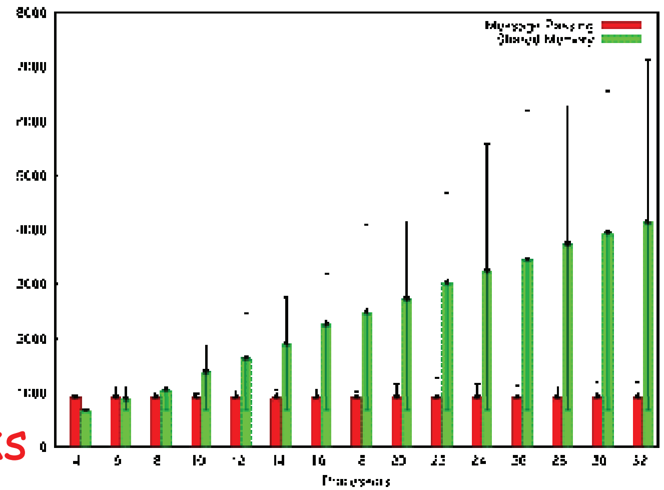
Objectives: redesign micro/pico-kernel OS

- Eliminate coherence → more predictable
- Reduce memory contention

Methods: new NoCMsg abstraction

- Bare-metal comm. (poll), prevent deadlocks
- Eliminate flow control when possible





64 Cores/Chip: Scalable+Predictable Runtime

Tilera TilePro 64 / Maestro 49 (Boeing/DoD for Satellites) [ccGrid'14]

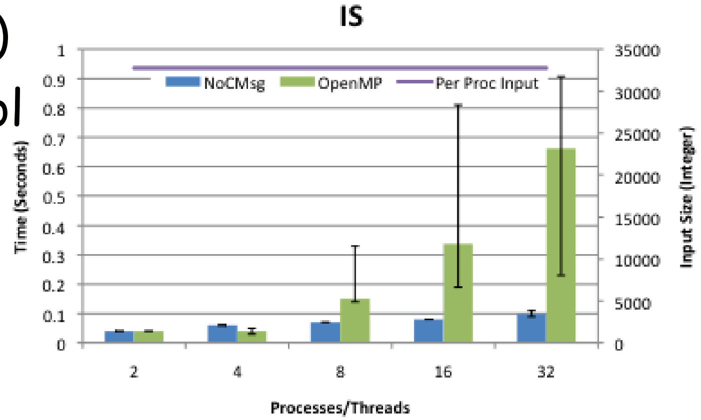
- 10x speedup IS: int app → comm. heavy)

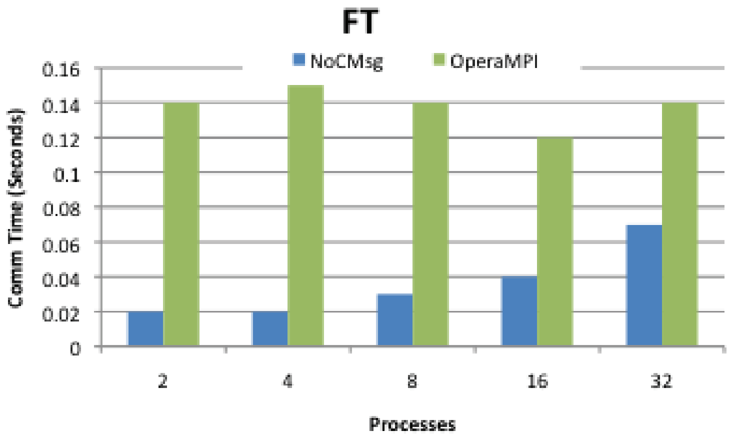
NocMsg/Opera MPI: Eliminate flow control

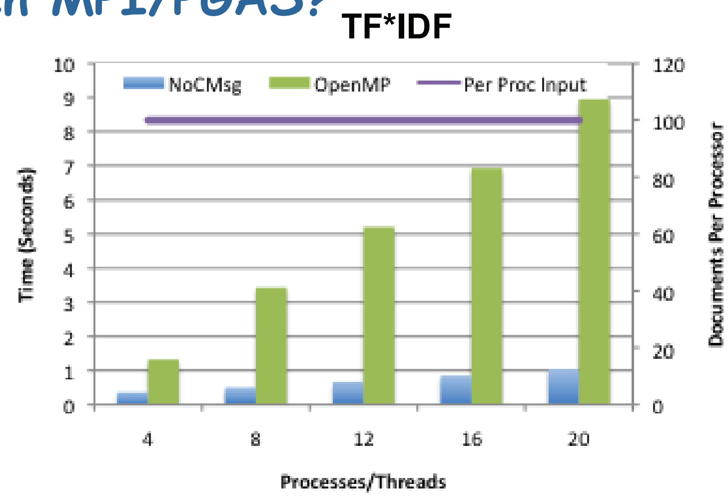
- ~7x speedup FT: float app, reduced comm. Latency, scalable

~10x for TF*IDF Document Clustering

- Heap intensive, locks hurt OpenMP
- **Future: OpenMP: max 16 cores, then MPI/PGAS?**

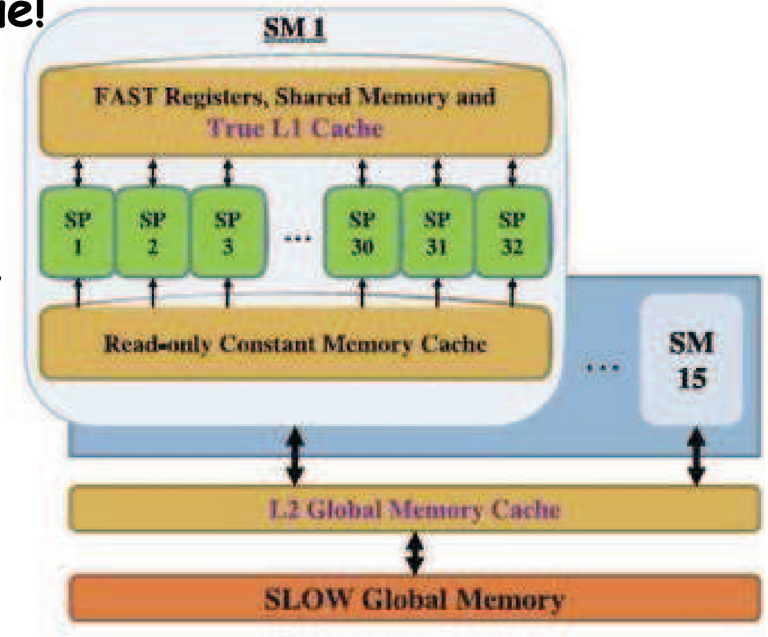






Memory Variability: Case for Auto-Tuning [ISPASS'14]

- **GPUs memory reconfigurable → unique!**
  - s/w-managed
    - -GPU "shmem", KNL near memory
  - h/w-managed: L1 D-cache
- Always use L1 D-cache? → simple but…
- shmem advantage: MLP+coalescing
  - matmult, fft
- D-cache advantage: TLP+reg stores
  - Marching cubes, pathfinder



- ➢ Depends on GPU generation!
- most benchmarks favor sh-mem
  → justified s/w complexity to manage them
- **More complex memory hierarchy → auto-tuning: perf.+power!**
  - General-purpose languages vs. DSLs

Contributions

1. Scalable network overlay (ICS'06)
   - track live nodes, group communication
2. Reactive fault tolerance (IPDPS'07, Linux'11, ICPADS'11)
   - job pause → 70% reduced resubmit overhead
   - Incr. Chkpts → 1:9 full/incr. Ratio best, reduce I/O
3. Proactive fault tolerance (ICS'07, SC'08, JPDC'12)
   - process virt. → ½ overhead of OS, health monitor
   - live migration → ½ # chkpts
   - back migration → wins if >10% work left
4. Redundancy + SDC Handling (ICDCS'12, SC'12)
   - 2x # nodes → 2x # jobs: capacity not capability comp.
   - dual for SDC check / triple SDC correction (msgs, RAM, I/O)
5. Algorithm-based Fault tolerance (IPDPSP'14. Chen&others, subm.)
   - Complements above, sign. less overhead, only dense linear algebra
   - Model SDC for numerical algorithms → Sandbox: run thru errors

- Code contributed to BLCR, available for Open MPI, later RedMPI

Acknowledgement

Supp. in part by DOE/NFS grants, Humboldt fellowship

DOE DE-FG02-05ER25664, DE-FG02-08ER25837, DE-AC05-00OR22725, NFS 0237570, 0410203, 0429653, 1058779, 0958311, 0937908

DOE DE-AC04-94AL85000 (SNL) , DOE DE-AC05-00OR22725 (ORNL) , LBL-6871849 (LBL)

- NCSU: J. Varma, A. Nagarajan, Chao Wang (ORNL), M. Vasavada, K. Kharbas, D. Fiala, J. Elliott
- ORNL collaborators: Engelmann, Scott
- LBL: Hargrove,Roman
- SNL: Kurt Ferreira, Ron Brightwell, Mark Hoemmen
- IBM: Rolf Riesen