CS420: Fault Tolerance

Laxmikant V. Kale

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Faults, Errors and Failures

- Fault
  - The cause of an error (e.g. a bug, stuck bit, alpha particle)
- Error
  - The part of total state that *may* lead to a failure (e.g. a bad value)
- Failure:
  - A transition to incorrect service (an event, e.g. the start of an unplanned service outage, premature job termination)

Transient, Intermittent, and Permanent Faults

- **Transient**
  - Usually uncontrollable, environmentally influenced – cosmic radiation
- **Intermittent**
  - Marginal or failing hardware
  - Through aging, parameter of a device drifts in value, exceeds built-in margin
  - E.g. intermittency of contacts at solder joints, threshold voltage of a MOSFET, etc.
- **Permanent**
  - Irreversible physical changes
  - Usually cause device to be inoperable
  - May be the evolution of intermittent errors, also extreme environmental conditions

Hard vs. Soft

- **“Hard” usually refers to a hard stop failure**
  - ~detectable by the system/application/hardware
- **“Soft” usually refers to data corruption**
  - ~undetectable by the system/application/hardware

Where Do Errors in Supercomputers Come From?

- **HPC systems of today are extremely complex systems made from hardware and software components that were never designed to work together as one complete system**
  - Dielectric breakdown and electrical breakdown
  - Temperature (extremes and variations)
  - Aging
  - Manufacturing defects
  - Stress
  - Extreme conditions
  - Voltage fluctuation
  - Electro-magnetic interference
  - Terrestrial neutrons
  - Cosmic radiation
  - Alpha particles

How Do Errors Manifest in Supercomputers?

- Hardware or software crashes
  - System reboot usually fixes this
  - Application usually crashes, must be restarted
- Performance variation
  - Terribly hard to diagnose and fix
  - Usually wasteful but not destructive
  - Much worse for tightly-coupled numerical simulations
- Data corruption
  - Clearly a wrong answer in a calculation – must re-run some of the simulation again
  - Silently corrupted calculation – result is corrupted, but in a way that we cannot tell

Failures on Titan

Failure Category	Failure Type	Count	Percentage
GPU	GPU DBE GPU DPR GPU Bus SXM power off SXM warm temp	51 66 11 14 2	16.1% 20.8% 3.5% 4.4% 0.6%
Processor	Machine check exception bank 0,2,6	31	9.8%
Memory	Machine check exception Bank 4 MCE	120	37.9%
Blade	Voltage fault Module failed	12 10	3.8% 3.1%

Typical Fault-Tolerance Problem

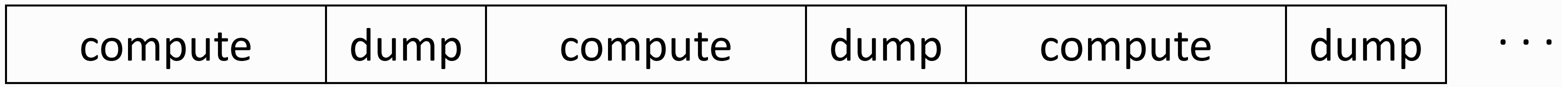
- Assume:
  - A problem that needs to run for a long time (e.g. days) …
  - On a system in which the *MTBF* (Mean Time Between Failures) is relatively small (e.g. hours)
- Problem:
  - How to get a complete execution ?

Typical Fault-Tolerance Solution

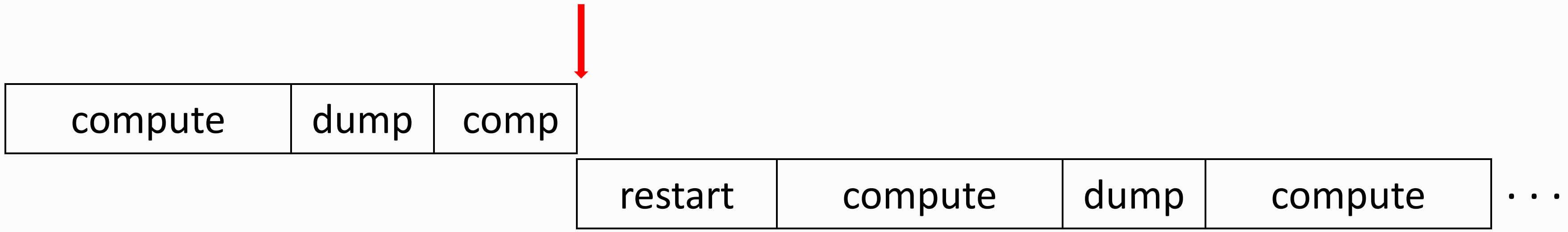
- Checkpoint/Restart
  - Explore iterative/periodic pattern in applications
  - After running for a given period, *checkpoint* the application (i.e. save minimal state required to be able to *restart*, if there is a failure)
- Basic Idea:
  - Do some work; save/dump state; do more work; save state, do more work, etc., etc.
  - In case of failure, restart from last checkpoint taken

Typical Fault-Tolerance Solution

- Execution without failures:



- Execution with a failure:



- Dump (Checkpoint) phase: save essential state
  - typically saving data to disk (checkpoint file)

- Restart phase: recover essential state

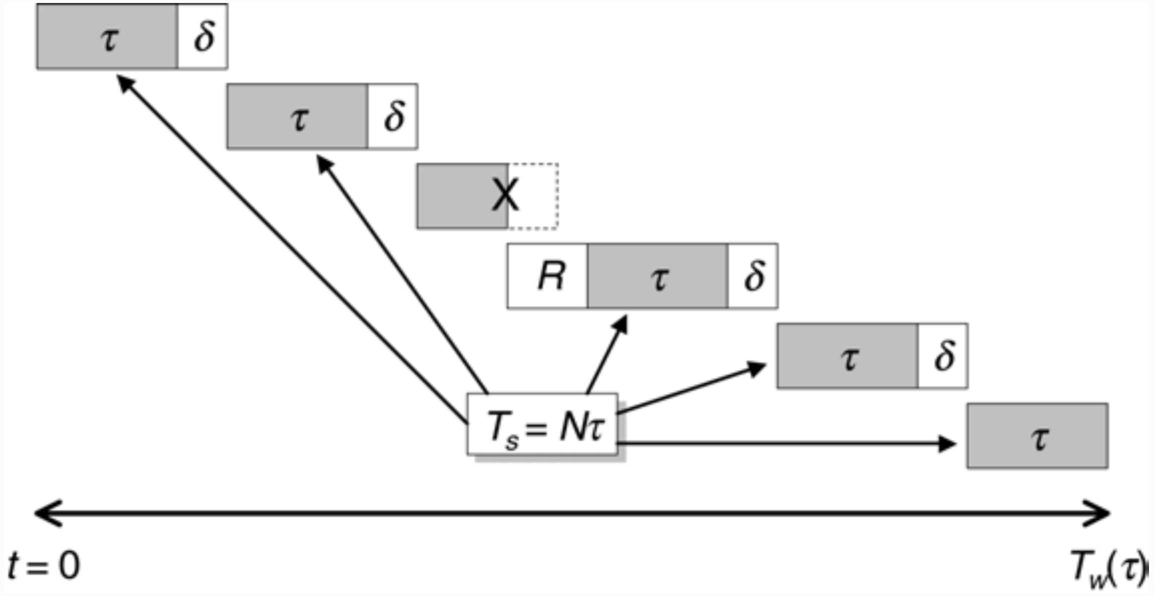
How Often to Checkpoint?

Tradeoffs in Dump Period Selection:

- If T(compute) >> T(dump)
  - Less overhead imposed by dumping data
  - More work likely to be lost when a failure occurs
- If T(compute) ≈ T(dump)
  - More overhead due to dumping data
  - Less work is lost in case of failure
- Classical checkpoint decision:
- What is the checkpoint period that will minimize the *total* application execution time ?
- Ref: J.Daly – *A higher order estimate of the optimum checkpoint interval for restart dumps.* Future Generation Computer Systems, 22(2006), pp.303-312

Standard Fault-Tolerance Model

- A simple model
  - τ: regular computation
  - δ: dump of checkpoint
  - X: failure,   R: recovery time, M: MTBF
  - Ts: Total "useful" execution = N τ
  - Tw: Total walltime of execution



Standard Fault-Tolerance Model

A simple model (cont.):

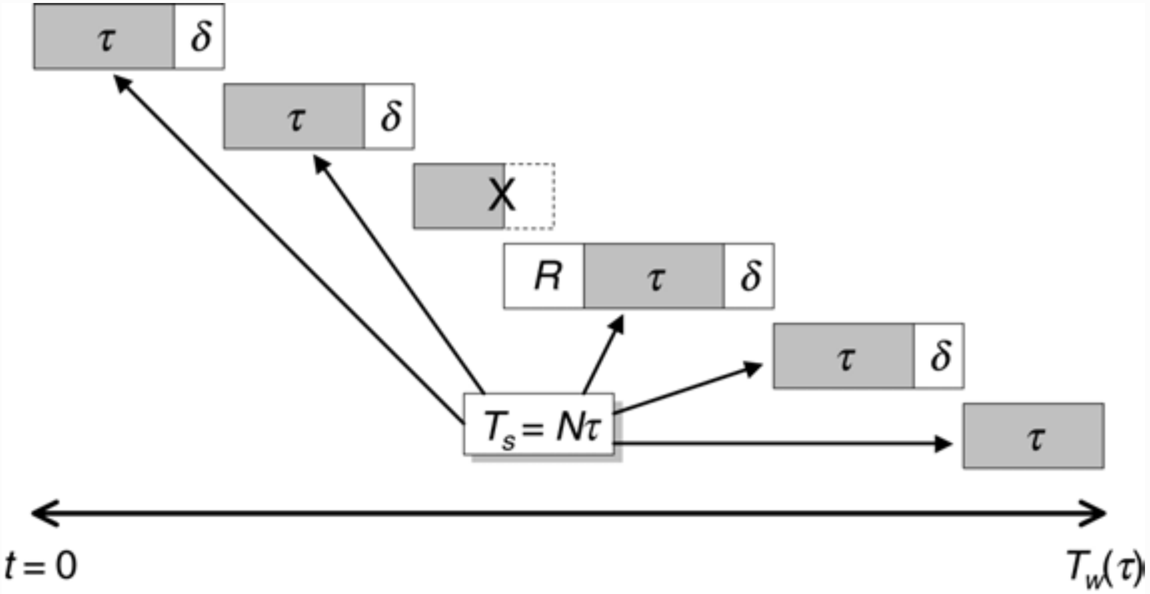
$T_w(\tau)$ = computation time + dump time + rework time + recovery time

$= \quad T_s \quad + (T_s/\tau - 1)\,\delta + [\tau+\delta]\,\phi\, n(\tau) \; + \quad R\, n(\tau)$

where:

ϕ: fraction of work lost, on average

$n(\tau)$: number of failures, on average



Standard Fault-Tolerance Model

A simple model:

- Assumptions:
  - Only one failure per compute segment
  - No failures during dump and recovery
- Approximations (see reference):
  - $\phi = ½$
  - $n(\tau) \approx Ts\ [(\tau+\delta)/M]\ /\ \tau$

$Tw(\tau) = Ts + (Ts/\tau - 1)\delta + [(\tau + \delta)/2 + R]\ Ts/\tau\ (\tau+\delta)\ /\ M$

To minimize $Tw(\tau)$: $d(Tw)/d\tau = 0$

$\Rightarrow \tau\ (opt) = [\ 2\ \delta\ (M+R)]^{½}$ for $(\tau+\delta) << M$

Example: M=1 hour, R=δ=1 min. $\Rightarrow \tau$ (opt) $\approx$ 11 min. , $\approx$ 9% overhead!

But for checkpoints to disk, δ can be 10+ minutes (esp. if almost all memory is being dumped)

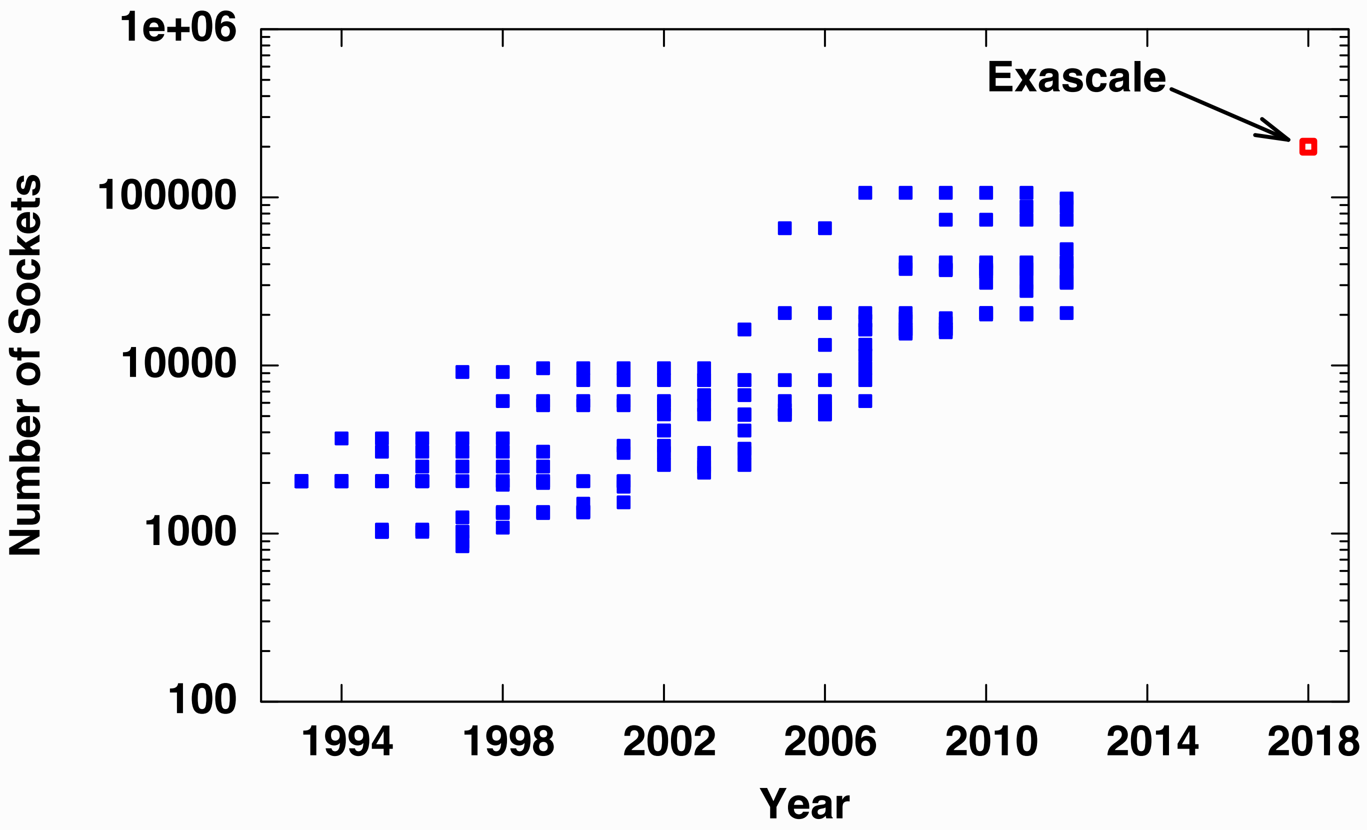
Higher Order Fault-Tolerance Model

Note: (ignore for the exam)

- This comes from a simple, first order model
- A higher order model (see Ref.):
  - $\tau\,(\text{opt}) = (2\,\delta\,M)^{1/2} - \delta \quad \text{if } \delta < M/2$
  - $\tau\,(\text{opt}) = M \quad \text{if } \delta \geq M/2$
- In practice, checkpoint/restart is largely used by real applications
  - Tolerance to failures *and* to execution scheduling
  - Job "failure" = Job is aborted by the system scheduler
  - New executions simply restart from last checkpoint
  - Dump phase can be accelerated with local disks/filesystems

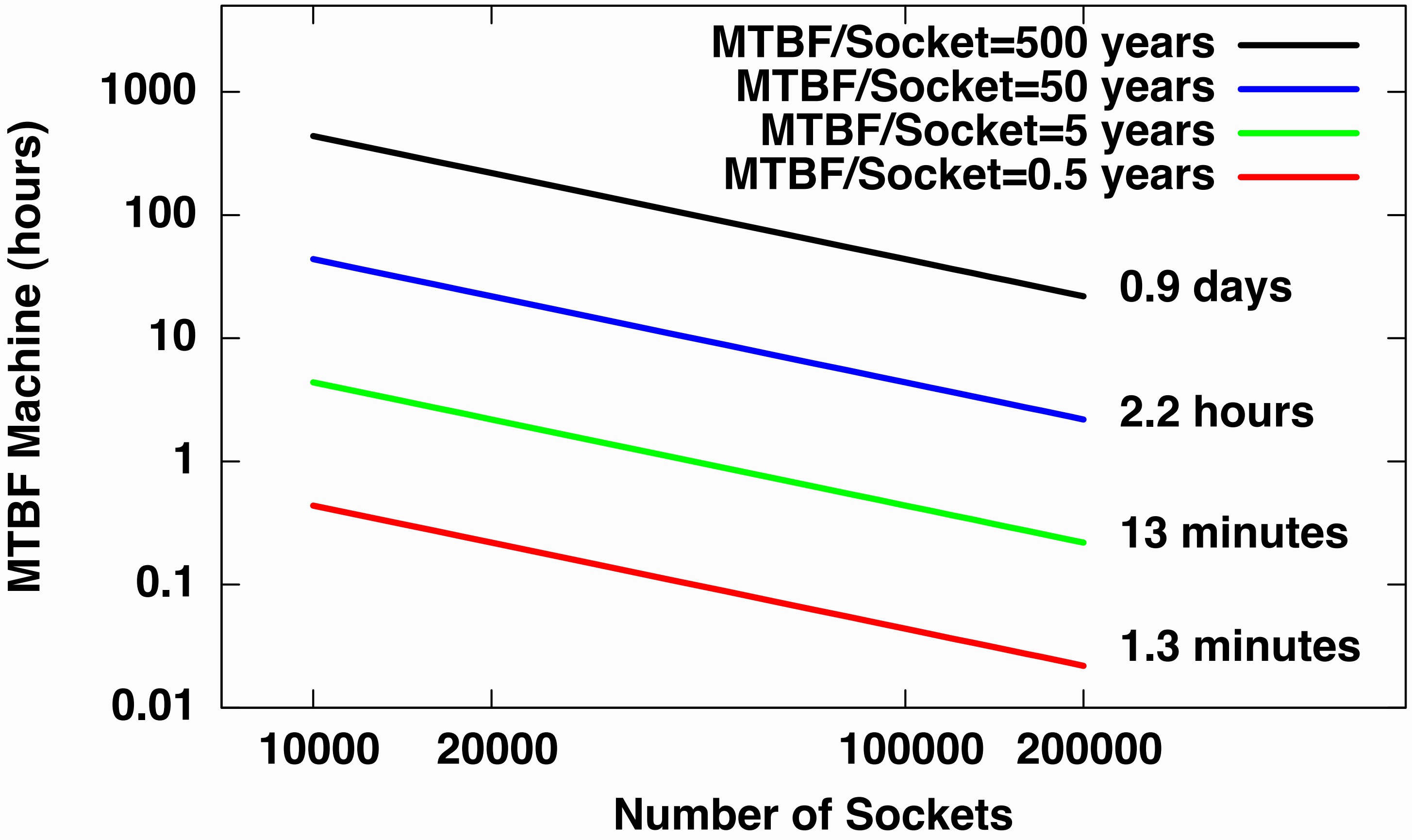
Fault Trends in Large Systems

No matter how reliable the components are, a large system *will* be likely to suffer a failure



Fault Trends in Large Systems

No matter how reliable the components are, a large system *will* be likely to suffer a failure



Fault Tolerance in Parallel Systems

- As machines grow in size
  - MTBF decreases
  - Applications have to tolerate faults
- Checkpoint/Restart may not scale
  - All nodes are rolled back just because one crashed
  - Even nodes independent of the crashed node are restarted
  - Typically requires same configuration for restart

Fault Tolerance References

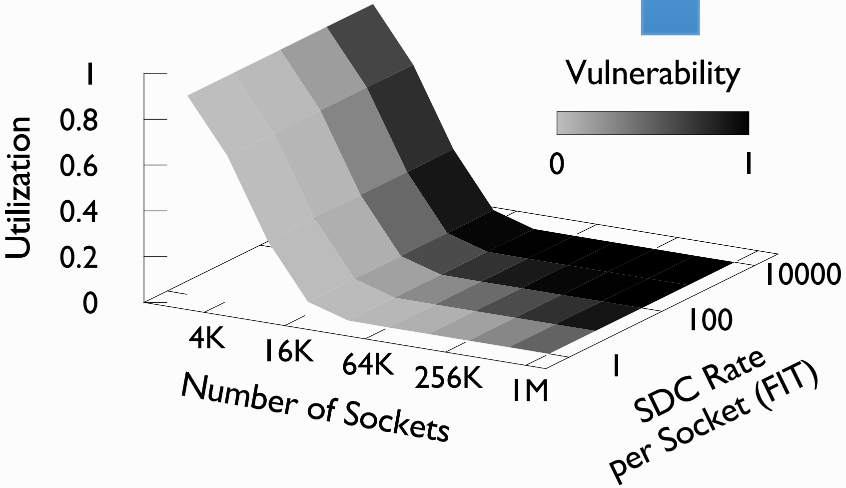
- Checkpoint-based methods
  - Coordinated – Blocking [Tamir84], Non-blocking [Chandy85] Co-check, Starfish, Clip – fault tolerant MPI
  - Uncoordinated – suffers from rollback propagation
  - Communication – [Briatico84], doesn't scale well
- Message-Logging schemes
  - Basic idea: only roll back the failed processors
  - Pessimistic – MPICH-V1 and V2, SBML [Johnson87]
  - Optimistic – [Strom85] unbounded rollback, complicated recovery
  - Causal Logging – [Elnozahy93] Manetho, complicated causality tracking and recovery
  - Charm++ based methods :
    - Message-logging.. Actually benefits performance because you can parallelize the restart

Silent Data Corruption

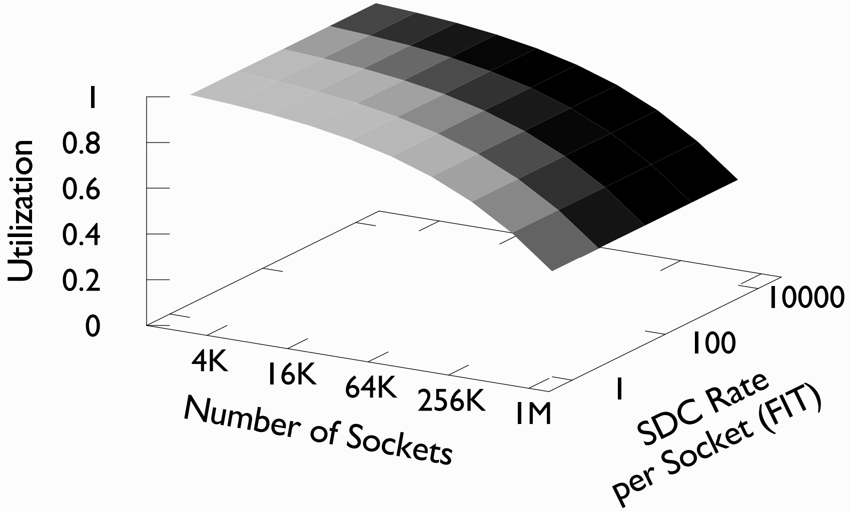
- **Cosmic Rays from Outer Space!**
  - **Muons (very heavy electrons)**
    - Most abundant particle in shower
    - Deposits energy in matter in an even distributed manner
    - Like throwing a baseball at a stack of pillows
    - They don’t do much damage to you or electrical circuits
  - **Neutrons**
    - ~70per hour per square centimeter in Los Alamos
    - Only “see” nuclei
    - Most matter is nearly invisible to a neutron – just goes right through
    - However, when it hits something, it hits it HARD!
  - **Radiation and you**
    - 3.5 billion years of evolution has equipped you to repair yourself
    - Computers aren’t as good at self-repair

Impact of silent data corruption

Probability of incorrect results



No Fault-tolerance



Checkpoint/restart

Dealing with silent data corruption

- How do you know if happened??
- How to prevent it in any case?
- Redundancy is one answer:
  - TMR: triple modular redundancy. Applying in parallel computations is tricky.
    - You can compare messages among 3 copies. Note floating point comparisons cannot be exact
  - Take advantage of continuity of “field” data
    - Nearby temperatures/pressures and such physical quantities being simulated don’t normally differ by a huge amount. Check, and if they are found to be different, fix them
    - In addition, for control variables, such as loop control variables, indices, etc. : protect them via replication and duplicate computations (or triplicate, if you really want correction)
- In the meanwhile, practical checkpoint/restart, with use of Daly’s formula, is good enough
  - Possibly with automation (e.g. how AMPI or Charm++ does it)

Fault Tolerance Research: Thoughts

- Fault tolerance is a really interesting area of research
  - With very “nice” and deep challenges
- However, improved engineering keeps making this research unnecessary
  - Its forever ”we may need this in future” mode
- But it is still worth while continuing research
- E.g. low-threshold voltage components may be necessary in future to drastically reduce power consumption
  - But they increase failure probabilities
  - If we can handle some failures in software, a wider variety of design options can be considered