

# SC3260 / SC5260

## Fault-tolerant Techniques for HPC

Lecture by: Ana Gainaru

# Table of content

Slides heavily based on the SC tutorial given by George Bosilca

<https://fault-tolerance.org/downloads/sc14tutorial.pdf>

- ▶ Failures and faults in HPC
- ▶ Checkpointing
  - ▶ Theory
  - ▶ System and application level solutions currently used
- ▶ Fault tolerant MPI
- ▶ Designing a resilient application
- ▶ Silent errors



VANDERBILT  
UNIVERSITY

# Failures and Faults in HPC

Top ranked supercomputers in the US (June 2017)

Rank	Name	Laboratory	Technology	Cores	PFlops/s	MTBF
4	Titan	ORNL	Cray XK7	560,640	17.59	$\approx 1$ day
5	Sequoia	LLNL	BG/Q	1,572,864	17.17	$\approx 1$ day
6	Cori	LBNL	Cray XC40	622,336	14.01	$\approx 1$ day
9	Mira	ANL	BG/Q	786,432	8.59	$\approx 1$ day

**Fail-stop errors** Node failure, resource crashes

**Silent errors or silent data corruptions (SDCs)** Double bit flips, soft faults



VANDERBILT  
UNIVERSITY

# Failures and Faults in HPC

Top ranked supercomputers in the US (June 2017)

Rank	Name	Laboratory	Technology	Cores	PFlops/s	MTBF
4	Titan	ORNL	Cray XK7	560,640	17.59	$\approx 1$ day
5	Sequoia	LLNL	BG/Q	1,572,864	17.17	$\approx 1$ day
6	Cori	LBNL	Cray XC40	622,336	14.01	$\approx 1$ day
9	Mira	ANL	BG/Q	786,432	8.59	$\approx 1$ day

**Fail-stop errors** Node failure, resource crashes

**Silent errors or silent data corruptions (SDCs)** Double bit flips, soft faults

**Exascale computing** (1000 PFlops/s)

- Larger core count: millions or even billions of cores
- Shorter Mean Time Between Failures (MTBF)  $\mu$



VANDERBILT  
UNIVERSITY

# Failures and Faults in HPC

Top ranked supercomputers in the US (June 2017)

Rank	Name	Laboratory	Technology	Cores	PFlops/s	MTBF
4	Titan	ORNL	Cray XK7	560,640	17.59	$\approx 1$ day
5	Sequoia	LLNL	BG/Q	1,572,864	17.17	$\approx 1$ day
6	Cori	LBNL	Cray XC40	622,336	14.01	$\approx 1$ day
9	Mira	ANL	BG/Q	786,432	8.59	$\approx 1$ day

**Fail-stop errors** Node failure, resource crashes

**Silent errors or silent data corruptions (SDCs)** Double bit flips, soft faults

**Exascale computing** (1000 PFlops/s)

- ▶ Larger core count: millions or even billions of cores
- ▶ Shorter Mean Time Between Failures (MTBF)  $\mu$

**Coping with faults**

- ▶ Build more reliable hardware!
- ▶ Make applications more fault tolerant!
- ▶ Design better resilience techniques/algorithms!



VANDERBILT  
UNIVERSITY

# Exascale platforms

## ► Hierarchical

- $10^5$  or  $10^6$  nodes
- Each node equipped with  $10^4$  or  $10^3$  cores

## ► Failure-prone

MTBF – one node	1 year	10 years	120 years
MTBF – platform of $10^6$ nodes	30sec	5mn	1h

More nodes → Shorter MTBF (Mean Time Between Failures)



VANDERBILT  
UNIVERSITY

# Exascale platforms

## ► Hierarchical

- $10^5$  or  $10^6$  nodes
- Each node equipped with  $10^4$  or  $10^3$  cores

## ► Failure-prone

MTBF – one node	1 year	10 years	120 years
MTBF – platform of $10^6$ nodes	30sec	5mn	1h

More nodes → Shorter MTBF (Mean Time Between Failures)

Exascale  $\neq$  Petascale x 1000 !!



VANDERBILT  
UNIVERSITY

# Even for today's platforms (courtesy F. Cappello)

Joint Laboratory for Petascale Computing

## Also an issue at Petascale

INRIA NCSA

Fault tolerance becomes critical at Petascale (MTTI  $\leq 1$  day)  
Poor fault tolerance design may lead to huge overhead

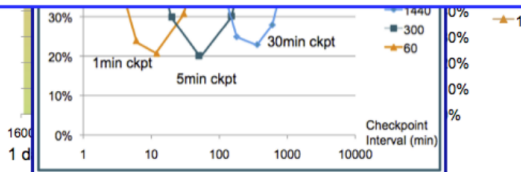
Overhead of checkpoint/restart

Cost of non optimal checkpoint intervals:

100%  
0%

Today, 20% or more of the computing capacity in a large high-performance computing system is wasted due to failures and recoveries.

Dr. E.N. (Mootaz) Elnozahy et al. *System Resilience at Extreme Scale, DARPA*



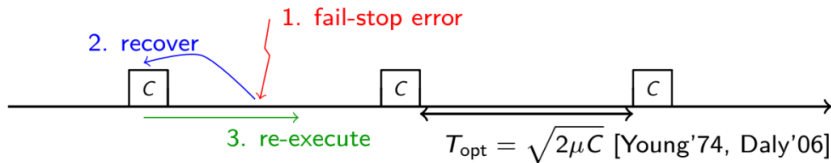
VANDERBILT  
UNIVERSITY



# Classic Resilience Techniques for HPC

**Fail-stop errors** (instantaneous error detection)

**Standard approach** periodic checkpointing, rollback and recovery

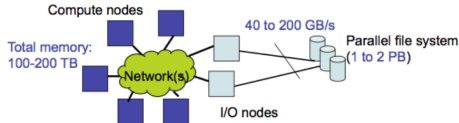
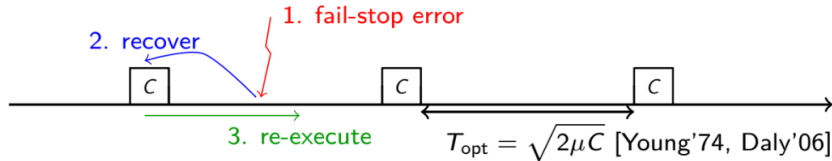


VANDERBILT  
UNIVERSITY

# Classic Resilience Techniques for HPC

**Fail-stop errors** (instantaneous error detection)

**Standard approach** periodic checkpointing, rollback and recovery



**Without optimizations, currently C/R could take up to one hour !**



VANDERBILT  
UNIVERSITY

# Sources of failures

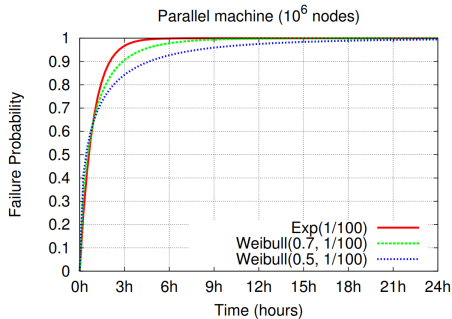
- ▶ Many types of faults: software error, hardware malfunction, memory corruption
  - ▶ Software errors: Applications, OS bug (kernel panic), communication libs, File system error, etc
  - ▶ Hardware errors: Disks, processors, memory, network
- ▶ Many possible behaviors: silent, transient, unrecoverable
- ▶ **Restrict to faults that lead to application failures**
  - ▶ This includes most hardware faults, and some software ones
- ▶ Will use terms fault and failure interchangeably
- ▶ **Silent errors (SDC) addressed later in the lecture**



VANDERBILT  
UNIVERSITY

# Values from real systems

- ▶ MTBF of one processor: between 1 and 125 years
- ▶ MTBF for systems: between several hours to one day



Failure trace archive from INRIA (<http://fta.inria.fr>)

Computer Failure Data Repository from LANL (<http://institutes.lanl.gov/data/fdata>)



# Process Checkpointing

## Goal

Save the current state of the process

- ▶ FT Protocols save a possible state of the parallel application

## Two techniques

- ▶ User-level checkpointing
- ▶ System-level checkpointing
  
- ▶ Blocking call
- ▶ Asynchronous call



VANDERBILT  
UNIVERSITY

# User-level checkpointing

## User code serializes the state of the process in a file

- ▶ Usually small(er than system-level checkpointing)
- ▶ Portability
- ▶ Diversity of use
  
- ▶ Hard to implement if preemptive checkpointing is needed
- ▶ Loss of the functions call stack
  - ▶ code full of jumps
  - ▶ loss of internal library state

## FTI (Fault Tolerance Interface)

Multi-level checkpointing tool for MPI applications



VANDERBILT  
UNIVERSITY

# System-level checkpointing

- ▶ Different possible implementations: OS syscall; dynamic library; compiler assisted
- ▶ Create a serial file that can be loaded in a process image. Usually on the same architecture, same OS, same software environment.
- ▶ **Entirely transparent**
- ▶ Preemptive (often needed for library-level checkpointing)
- ▶ Lack of portability
- ▶ Large size of checkpoint ( $\approx$  memory footprint)

## BLCR

Requires linking with the code

## CRIU

User-space tool that can freeze a running container (or an individual application) and checkpoint its state to disk



VANDERBILT  
UNIVERSITY

# Blocking / Asynchronous call

## Blocking Checkpointing

- ▶ Relatively intuitive: `checkpoint(filename)`
- ▶ Cost: no process activity during the whole checkpoint operation.
- ▶ Can be linear in the size of memory and in the size of modified files

## Asynchronous Checkpointing

- ▶ System-level approach: make use of copy on write or fork syscall
- ▶ User-level approach: critical sections, when needed



VANDERBILT  
UNIVERSITY



## Remote Reliable Storage

- ▶ Intuitive. I/O intensive. Disk usage.

## Memory Hierarchy

- ▶ local memory
- ▶ local disk (SSD, HDD)
- ▶ remote disk
  - ▶ Scalable Checkpoint Restart Library
  - ▶ FTI

Checkpoint is valid when finished on reliable storage



VANDERBILT  
UNIVERSITY

# How to use checkpointing for your application?

## 1 Choose the checkpointing tool that fits your application

- ▶ Do you have access to the code of the application?
- ▶ What is the memory footprint?
- ▶ How much time do you want to invest in changing your application?
- ▶ What overhead is acceptable?
- ▶ Is there a plugin in SLURM capable of taking checkpoints for you?

## 2 Choose the storage space for your application

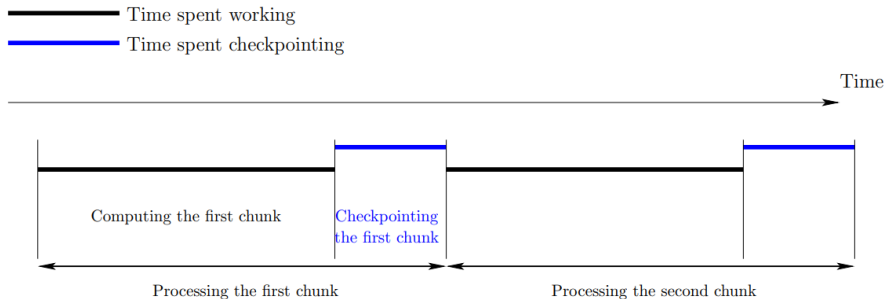
- ▶ Permanent storage or local memory

## 3 Choose how often you take checkpoints



VANDERBILT  
UNIVERSITY

# Checkpointing cost

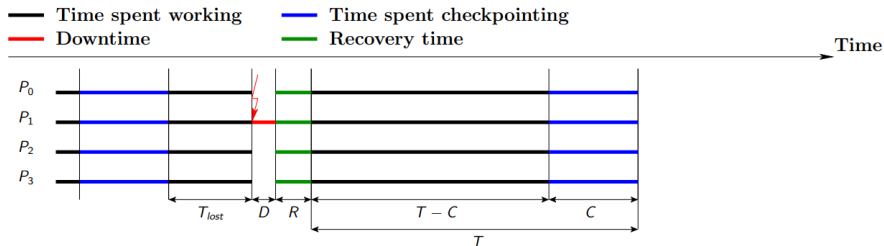


**Blocking model:** while a checkpoint is taken, no computation can be performed



VANDERBILT  
UNIVERSITY

# Checkpointing cost



## Time lost

- ▶ Downtime required to fix the error
- ▶ Time it takes to restart the application
- ▶ Average lost time is half the time between two checkpoints
- ▶ In addition to the time to take the checkpoints



# Checkpointing interval

## Optimization problem

- ▶ Studied by multiple research projects
  - ▶ Failure rate of different hardware components
  - ▶ Application behavior
- ▶ **Daly's optimal checkpoint sequence from 2006 is still valid**  
 $\sqrt{2\mu C}$



VANDERBILT  
UNIVERSITY

# Lesson learned for fail-stop failures

- ▶ Tsubame 2: 962 failures during last 18 months so  $\mu = 13$  hrs
- ▶ Blue Waters: 2-3 node failures per day
- ▶ Titan: a few failures per day
- ▶ Tianhe 2: wouldn't say

$$T_{\text{opt}} = \sqrt{2\mu C} \Rightarrow \text{WASTE}[\text{opt}] \approx \sqrt{\frac{2C}{\mu}}$$

Petascale:	$C = 20$ min	$\mu = 24$ hrs	$\Rightarrow \text{WASTE}[\text{opt}] = 17\%$
Scale by 10:	$C = 20$ min	$\mu = 2.4$ hrs	$\Rightarrow \text{WASTE}[\text{opt}] = 53\%$
Scale by 100:	$C = 20$ min	$\mu = 0.24$ hrs	$\Rightarrow \text{WASTE}[\text{opt}] = 100\%$



# Table of content

- ▶ Failures and faults in HPC
- ▶ Checkpointing
  - ▶ Theory
  - ▶ System and application level solutions currently used
- ▶ **Fault tolerant MPI**
- ▶ Designing a resilient application
- ▶ Silent errors



VANDERBILT  
UNIVERSITY

**Message Passing Interface** Most popular middleware for multi-node programming in HPC

MPI Standard 3.0, p. 21, l. 24:25

This document does not specify the state of a computation after an erroneous MPI call has occurred.

**Open MPI** (<http://www.open-mpi.org>) **MPICH** (<http://www.mcs.anl.gov/mpi/mpich/>)

- ▶ Default: on failure detection, the runtime kills all processes. Can be de-activated by a runtime switch
- ▶ Errors might be reported to MPI processes. In that case MPI might be partly usable.



VANDERBILT  
UNIVERSITY



# MPI-Next-FT proposal: ULFM

## Goal

Resume Communication Capability for MPI (and nothing more)

- ▶ Error reporting indicates impossibility to carry an operation
  - ▶ State of MPI is unchanged for operations that can continue (i.e. if they do not involve a dead process)
- ▶ Errors are non uniformly returned
  - ▶ (Otherwise, synchronizing semantic is altered drastically with high performance impact)

## New APIs

- ▶ REVOKE allows to resolve non-uniform error status
- ▶ SHRINK allows to rebuild error-free communicators



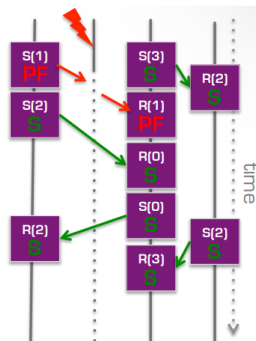
VANDERBILT  
UNIVERSITY

# MPI-Next-FT proposal: ULFM

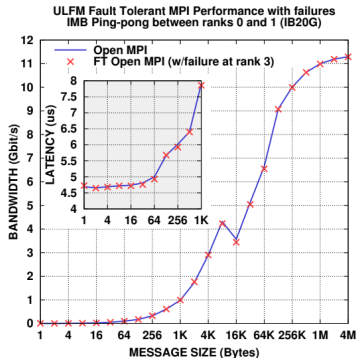
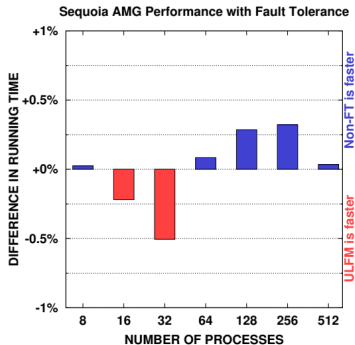
Errors are visible only for operations that cannot complete

## Error Reporting

- ▶ Operations that cannot complete return `ERR_PROC_FAILED`, or `ERR_PENDING` if appropriate
- ▶ State of MPI Objects is unchanged (communicators etc.)
- ▶ Operations that can be completed return `MPI_SUCCESS` point to point operations between non-failed ranks can continue



# MPI-Next-FT proposal: ULFM



OpenMPI support

Branch of Open MPI ([www.open-mpi.org](http://www.open-mpi.org))



VANDERBILT  
UNIVERSITY

# Table of content

- ▶ Failures and faults in HPC
- ▶ Checkpointing
  - ▶ Theory
  - ▶ System and application level solutions currently used
- ▶ Fault tolerant MPI
- ▶ **Designing a resilient application**
- ▶ Silent errors



VANDERBILT  
UNIVERSITY

# Application level resiliency

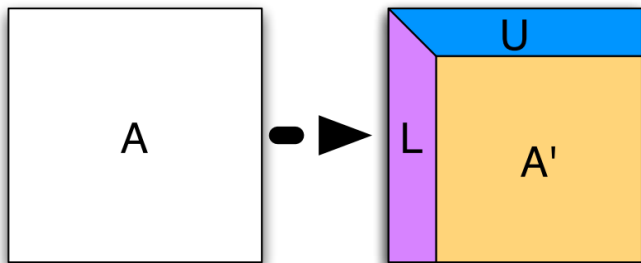
Applications can build customized methods for tolerating failures

- ▶ Depending on the application behavior some can mask failures all together
  - ▶ Neural networks
  - ▶ Heat equation
- ▶ **Algorithm based fault tolerance**
  - ▶ ABFT for Linear Algebra applications



VANDERBILT  
UNIVERSITY

## Example: block LU/QR factorization



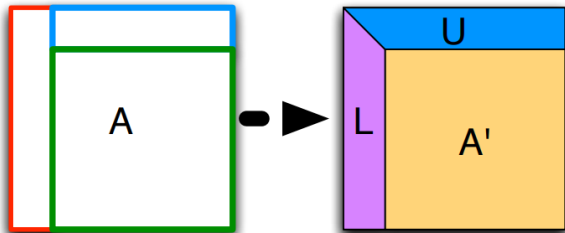
- ▶ Solve  $A \cdot x = b$  (hard)
- ▶ Transform  $A$  into a LU factorization
- ▶ Solve  $L \cdot y = B \cdot b$ , then  $U \cdot x = y$



VANDERBILT  
UNIVERSITY

# Example: block LU/QR factorization

TRSM - Update row block



GETF2: factorize a column block  
GEMM: Update the trailing matrix

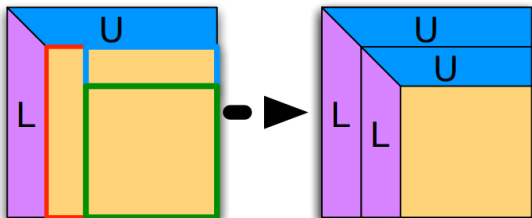
- ▶ Solve  $A \cdot x = b$  (hard)
- ▶ Transform  $A$  into a LU factorization
- ▶ Solve  $L \cdot y = B \cdot b$ , then  $U \cdot x = y$



VANDERBILT  
UNIVERSITY

# Example: block LU/QR factorization

TRSM - Update row block



GETF2: factorize a column block  
GEMM: Update the trailing matrix

- ▶ Solve  $A \cdot x = b$  (hard)
- ▶ Transform  $A$  into a LU factorization
- ▶ Solve  $L \cdot y = B \cdot b$ , then  $U \cdot x = y$

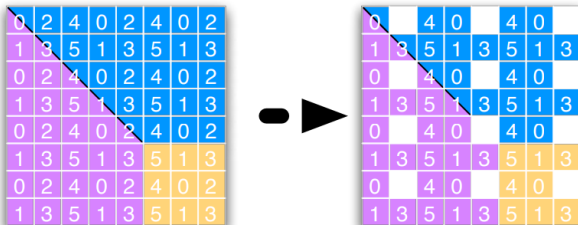


VANDERBILT  
UNIVERSITY



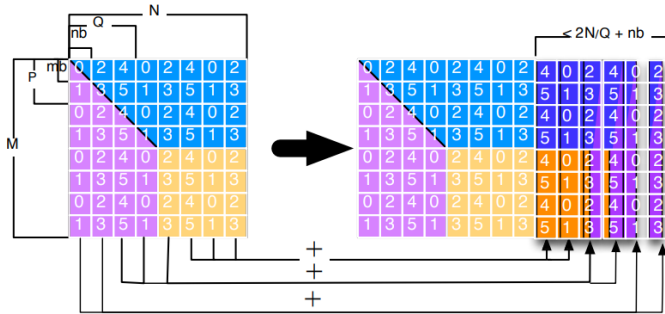
## Example: block LU/QR factorization

Failure of rank 2



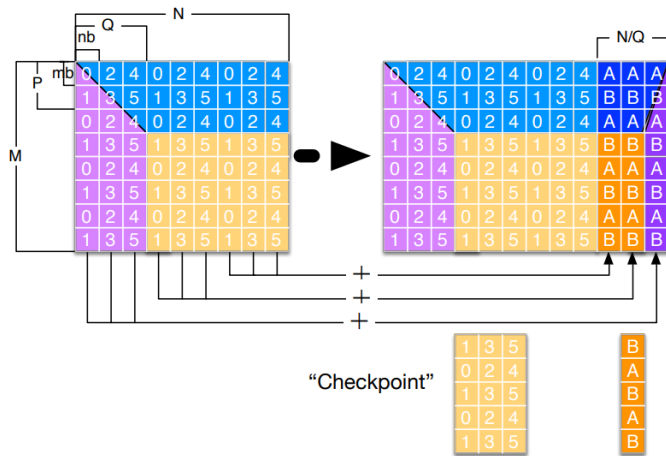
- ▶ 2D Block Cyclic Distribution (here 2 x 3)
- ▶ A single failure → many data elements will be lost

# Algorithm Based Fault Tolerant QR decomposition



- Checksum: invertible operation on the data of the row / column
- Checksum blocks are doubled, to allow recovery when data and checksum are lost together

# Algorithm Based Fault Tolerant QR decomposition

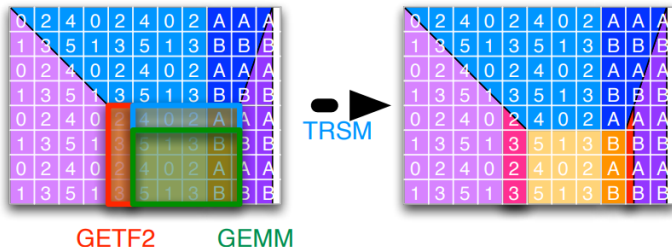


- ▶ Checksum: invertible operation on the data of the row / column
- ▶ Checksum replication can be avoided by dedicating computing resources



VANDERBILT  
UNIVERSITY

# Algorithm Based Fault Tolerant QR decomposition



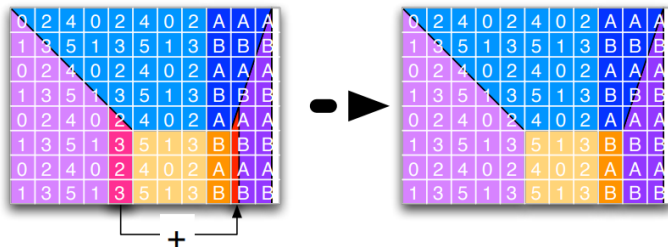
## Idea of ABFT

Applying the operation on data and checksum preserves the checksum properties



VANDERBILT  
UNIVERSITY

# Algorithm Based Fault Tolerant QR decomposition



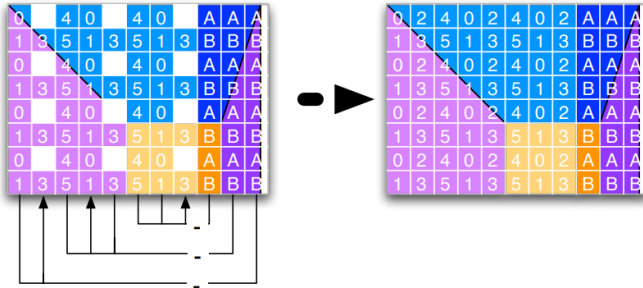
## Idea of ABFT

For the part of the data that is not updated this way, the checksum must be re-calculated



VANDERBILT  
UNIVERSITY

## In case of a failure

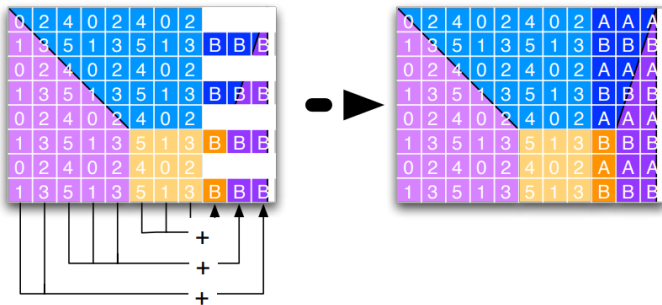


- In case of failure, conclude the operation, then
  - Missing Data = Checksum - Sum(Existing Data) s



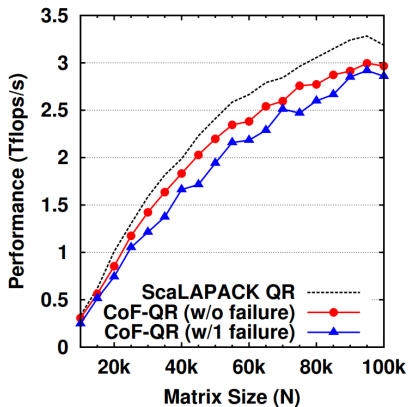
VANDERBILT  
UNIVERSITY

## In case of a failure



- ▶ In case of failure, conclude the operation, then
  - ▶ Missing Data = Checksum - Sum(Existing Data) s

# ABFT QR decomposition: performance



Open MPI; Kraken supercomputer;



VANDERBILT  
UNIVERSITY



# Table of content

- ▶ Failures and faults in HPC
- ▶ Checkpointing
  - ▶ Theory
  - ▶ System and application level solutions currently used
- ▶ Fault tolerant MPI
- ▶ Designing a resilient application
- ▶ **Silent errors**



VANDERBILT  
UNIVERSITY

# What are silent failures?

- ▶ Instantaneous error detection → fail-stop failures, e.g. resource crash
- ▶ Silent errors (data corruption) → detection latency

## Silent error detected only when the corrupt data is activated

- ▶ Includes some software faults, some hardware errors (soft errors in L1 cache), double bit flip
- ▶ Cannot always be corrected by ECC memory



VANDERBILT  
UNIVERSITY

# Should we be afraid? (courtesy Al Geist)

## Fear of the Unknown

**Hard errors** – permanent component failure either HW or SW  
(hung or crash)

**Transient errors** – a blip or short term failure of either HW or SW

**Silent errors** – undetected errors either hard or soft, due to lack of detectors for a component or inability to detect (transient effect too short). Real danger is that answer may be incorrect but the user wouldn't know.

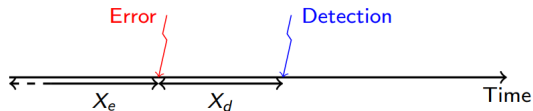
**Statistically, silent error rates are increasing.  
Are they really? Its fear of the unknown**

Are silent errors really a problem  
or just monsters under our bed?



VANDERBILT  
UNIVERSITY

# Coupling checkpointing with verification



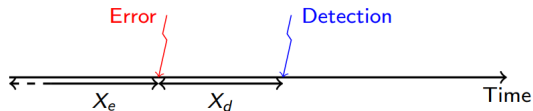
Error and detection latency

- ▶ Last checkpoint may have saved an already corrupted state
- ▶ Possible solution: Saving  $k$  checkpoints (Lu, Zheng and Chien):
  - ❶ Critical failure when all live checkpoints are invalid



VANDERBILT  
UNIVERSITY

# Coupling checkpointing with verification



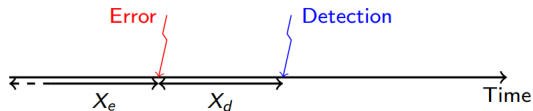
Error and detection latency

- ▶ Last checkpoint may have saved an already corrupted state
- ▶ Possible solution: Saving  $k$  checkpoints (Lu, Zheng and Chien):
  - ❶ Critical failure when all live checkpoints are invalid  
Assume unlimited storage resources
  - ❷ Which checkpoint to roll back to?



VANDERBILT  
UNIVERSITY

# Coupling checkpointing with verification



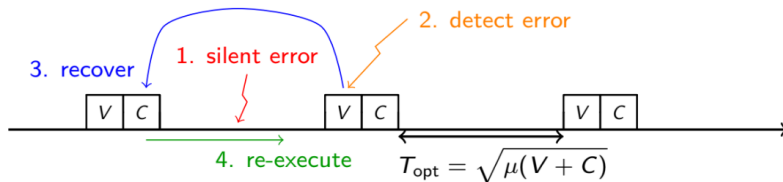
Error and detection latency

- ▶ Last checkpoint may have saved an already corrupted state
- ▶ Possible solution: Saving  $k$  checkpoints (Lu, Zheng and Chien):
  - ❶ Critical failure when all live checkpoints are invalid  
Assume unlimited storage resources
  - ❷ Which checkpoint to roll back to?  
Assume verification mechanism



VANDERBILT  
UNIVERSITY

# Coupling checkpointing with verification



- ▶ The most promising approach: checkpointing + verification (error detection)
- ▶ Use verification to correct some errors (ABFT)

## A lot of unknowns

- ▶ Error rate? MTBE?
- ▶ Selective reliability?
- ▶ New algorithms beyond iterative? matrix-product, FFT



- ▶ **Multiple approaches to Fault Tolerance**
- ▶ **Application-Specific Fault Tolerance will always provide more benefits:**
  - ▶ Checkpoint Size Reduction (when needed)
  - ▶ Portability (can run on different hardware, different deployment, etc..)
  - ▶ Diversity of use (can be used to restart the execution and change parameters in the middle)
- ▶ **General Purpose Fault Tolerance is a required feature of the platforms**
  - ▶ Not every computer scientist needs to learn how to write fault-tolerant applications
- ▶ **Requirements of a more Fault-friendly programming environment**
  - ▶ MPI-Next evolution
  - ▶ Other programming environments?





### Fault-Tolerance Techniques for High-Performance Computing

Springer Book, Computer Communications and Networks series, 2015

Editors: Thomas Herault and Yves Robert

### Checkpointing tools

- ▶ FTI: Fault Tolerance Interface, application level checkpointing using multi-level storage, <https://github.com/leobago/fti>
- ▶ BLCR: Berkeley Lab Checkpoint/Restart for Linux (system level checkpointing tool), <https://crd.lbl.gov/departments/computer-science/CLaSS/research/BLCR/>
- ▶ CRIU: Checkpoint and Restore in Userspace, system level tool to checkpoint/restore Linux tasks, <https://github.com/checkpoint-restore/criu>

### More info on the ABFT example

Algorithm-based fault tolerance applied to high performance computing,  
Bosilca G. et al., JPDC 69, 4 (2009)



VANDERBILT  
UNIVERSITY