Fault tolerance techniques for high-performance computing Part 4

Anne Benoit

ENS Lyon

Anne.Benoit@ens-lyon.fr

http://graal.ens-lyon.fr/~abenoit

CR02 - 2016/2017

Outline

1. In-memory checkpointing

2. Probabilistic models for advanced methods
   - Failure prediction
   - Replication

3. Forward-recovery techniques
   - Introduction: Matrix-Matrix Multiplication
   - ABFT for Linear Algebra applications
   - Composite approach: ABFT & Checkpointing

4. Conclusion

Outline

Outline

Outline

Outline

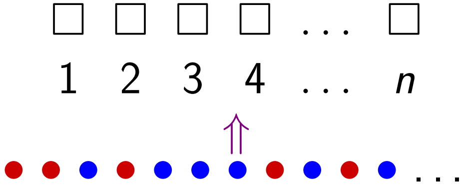
Replication

- Systematic replication: efficiency $< 50\%$
- Can replication+checkpointing be more efficient than checkpointing alone?
- Study by Ferreira et al. [SC'2011]: yes

Model by Ferreira et al. [SC' 2011]

- Parallel application comprising N processes
- Platform with $p_{total} = 2N$ processors
- Each process replicated $\rightarrow$ N *replica-groups*
- When a replica is hit by a failure, it is not restarted
- Application fails when both replicas in one replica-group have been hit by failures

Correct analogy



N bins, red and blue balls

Mean Number of Failures to Interruption (bring down application)
$MNFTI$ = expected number of balls to throw
until one bin gets one ball of each color

Exponential failures

Theorem: $MNFTI = \mathbb{E}(NFTI|0)$ where

$$\mathbb{E}(NFTI|n_f) = \begin{cases} 2 & \text{if } n_f = N, \\ \frac{2N}{2N-n_f} + \frac{2N-2n_f}{2N-n_f}\,\mathbb{E}\left(NFTI|n_f+1\right) & \text{otherwise.} \end{cases}$$

$\mathbb{E}(NFTI|n_f)$: expectation of number of failures to kill application, knowing that

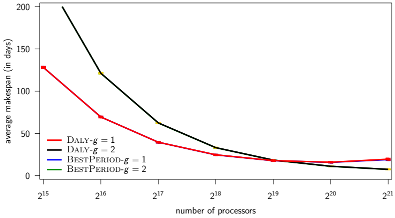
- application is still running
- failures have already hit n_f different replica-groups

How do we obtain this result?

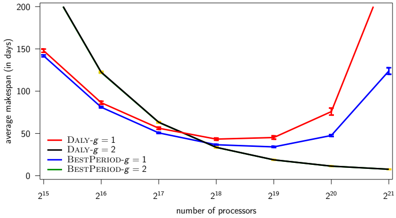
Comparison

- 2N processors, no replication
  $\textsc{Throughput}_{\mathsf{Std}} = 2N(1 - \textsc{Waste}) = 2N\left(1 - \sqrt{\frac{2C}{\mu_{2N}}}\right)$
- N replica-pairs
  $\textsc{Throughput}_{\mathsf{Rep}} = N\left(1 - \sqrt{\frac{2C}{\mu_{\mathsf{rep}}}}\right)$
  $\mu_{\mathsf{rep}} = MNFTI \times \mu_{2N} = MNFTI \times \frac{\mu}{2N}$
- Platform with $2N = 2^{20}$ processors $\Rightarrow MNFTI = 1284.4$
  $\mu = 10$ years $\Rightarrow$ better if C shorter than 6 minutes

Failure distribution



(a) Exponential

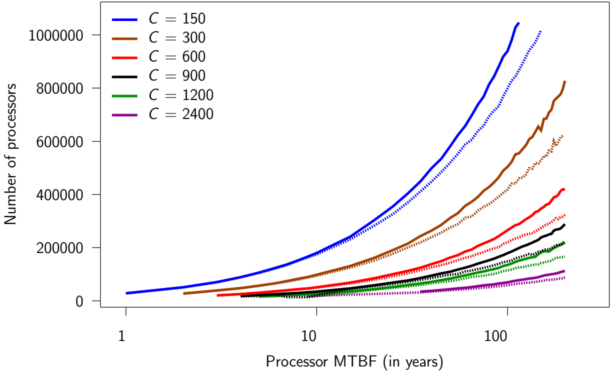


(b) Weibull, $k = 0.7$

Crossover point for replication when $\mu_{ind} = 125$ years

Weibull distribution with $k = 0.7$

Dashed line: Ferreira et al.   Solid line: Correct analogy



- Study by Ferrreira et al. favors replication
- Replication beneficial if small μ + large C + big p_{total}

Outline

Outline

Generic vs. Application specific approaches

Generic solutions

- Universal
- Very low prerequisite
- One size fits all (pros and cons)

Application specific solutions

- Requires (deep) study of the application/algorithm
- Tailored solution: higher efficiency

Backward Recovery vs. Forward Recovery

Backward Recovery

- Rollback / Backward Recovery: returns in the history to recover from failures
- Spends time to re-execute computations
- Rebuilds states already reached
- Typical: checkpointing techniques

Backward Recovery vs. Forward Recovery

Forward Recovery

- Forward Recovery: proceeds without returning
- Pays additional costs during (failure-free) computation to maintain consistent redundancy
- Or pays additional computations when failures happen
- General technique: Replication
- Application-Specific techniques: Iterative algorithms with fixed point convergence, ABFT, ...

Algorithm Based Fault Tolerance (ABFT)

Principle

- Limited to Linear Algebra computations
- Matrices are extended with rows and/or columns of checksums

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 12 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Algorithm Based Fault Tolerance (ABFT)

Principle

- Limited to Linear Algebra computations
- Matrices are extended with rows and/or columns of checksums

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 12 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Algorithm Based Fault Tolerance (ABFT)

Principle

- Limited to Linear Algebra computations
- Matrices are extended with rows and/or columns of checksums

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 12 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

ABFT and fail-stop errors

Missing checksum data

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Simple recomputation: 4+3+5 = 12.

Missing original data

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & & 5 & 12 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Simple recomputation: 12-(4+5) = 3.

ABFT and fail-stop errors

Missing checksum data

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Simple recomputation: $4+3+5 = 12$.

Missing original data

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & & 5 & 12 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Simple recomputation: $12-(4+5) = 3$.

ABFT and fail-stop errors

Missing checksum data

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Simple recomputation: $4+3+5 = 12$.

Missing original data

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & & 5 & 12 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Simple recomputation: $12-(4+5) = 3$.

ABFT and fail-stop errors

Missing checksum data

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Simple recomputation: $4+3+5 = 12$.

Missing original data

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & & 5 & 12 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Simple recomputation: $12-(4+5) = 3$.

ABFT and silent data corruption

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 13 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

Error detection: $4 + 3 + 5 \neq 13$

Limitations

- The following matrix would have successfully passed the sanity check:

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 5 & 3 & 5 & 13 \\ 4 & 6 & 9 & 19 \end{pmatrix}$$

- Can detect **one** error and correct **zero**.

ABFT and silent data corruption

$$M = \begin{pmatrix} 5 & 1 & 7 & \color{blue}{13} \\ 4 & 3 & 5 & \color{red}{13} \\ 4 & 6 & 9 & \color{blue}{19} \end{pmatrix}$$

Error detection: $4 + 3 + 5 \neq 13$

Limitations

- The following matrix would have successfully passed the sanity check:

$$M = \begin{pmatrix} 5 & 1 & 7 & \color{blue}{13} \\ 5 & 3 & 5 & \color{blue}{13} \\ 4 & 6 & 9 & \color{blue}{19} \end{pmatrix}$$

- Can detect **one** error and correct **zero**.

ABFT and silent data corruption

One row and one column of checksums

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 11 \\ 4 & 6 & 9 & 19 \\ 13 & 9 & 21 & 43 \end{pmatrix}$$

Checksum recomputation to look for silent data corruptions:

$$\begin{pmatrix} 5 & + & 1 & + & 7 & = & 13 \\ 4 & + & 3 & + & 5 & = & 12 \\ 4 & + & 6 & + & 9 & = & 19 \\ 13 & + & 10 & + & 21 & = & 44 \end{pmatrix}$$

Checksums do not match !

ABFT and silent data corruption

One row and one column of checksums

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 11 \\ 4 & 6 & 9 & 19 \\ 13 & 9 & 21 & 43 \end{pmatrix}$$

Checksum recomputation to look for silent data corruptions:

$$\begin{pmatrix} 5 & + & 1 & + & 7 & = & 13 \\ 4 & + & 3 & + & 5 & = & 12 \\ 4 & + & 6 & + & 9 & = & 19 \\ 13 & + & 10 & + & 21 & = & 44 \end{pmatrix}$$

Checksums do not match !

ABFT and silent data corruption

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 11 \\ 4 & 6 & 9 & 19 \\ 13 & 9 & 21 & 43 \end{pmatrix} \qquad \begin{pmatrix} 5 & + & 1 & + & 7 & = & 13 \\ 4 & + & 3 & + & 5 & = & 12 \\ 4 & + & 6 & + & 9 & = & 19 \\ 13 & + & 10 & + & 21 & = & 44 \end{pmatrix}$$

Both checksums are affected, giving out the location of the error.

We solve:

$$4 + x + 5 = 11 \qquad 1 + x + 6 = 9$$

Recomputing the checksums we find that:

$$\begin{pmatrix} 5 & + & 1 & + & 7 & = & 13 \\ 4 & + & 2 & + & 5 & = & 11 \\ 4 & + & 6 & + & 9 & = & 19 \\ 13 & + & 9 & + & 21 & = & 43 \end{pmatrix}$$ Checksums match ☺

Can detect **two** errors and correct **one**

ABFT and silent data corruption

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 11 \\ 4 & 6 & 9 & 19 \\ 13 & 9 & 21 & 43 \end{pmatrix} \qquad \begin{pmatrix} 5 & + & 1 & + & 7 & = & 13 \\ 4 & + & 3 & + & 5 & = & 12 \\ 4 & + & 6 & + & 9 & = & 19 \\ 13 & + & 10 & + & 21 & = & 44 \end{pmatrix}$$

Both checksums are affected, giving out the location of the error.

We solve:

$$4 + x + 5 = 11 \qquad 1 + x + 6 = 9$$

Recomputing the checksums we find that:

$$\begin{pmatrix} 5 & + & 1 & + & 7 & = & 13 \\ 4 & + & 2 & + & 5 & = & 11 \\ 4 & + & 6 & + & 9 & = & 19 \\ 13 & + & 9 & + & 21 & = & 43 \end{pmatrix}$$ Checksums match ☺

Can detect **two** errors and correct **one**

ABFT and silent data corruption

$$M = \begin{pmatrix} 5 & 1 & 7 & 13 \\ 4 & 3 & 5 & 11 \\ 4 & 6 & 9 & 19 \\ 13 & 9 & 21 & 43 \end{pmatrix} \qquad \begin{pmatrix} 5 & + & 1 & + & 7 & = & 13 \\ 4 & + & 3 & + & 5 & = & 12 \\ 4 & + & 6 & + & 9 & = & 19 \\ 13 & + & 10 & + & 21 & = & 44 \end{pmatrix}$$

Both checksums are affected, giving out the location of the error.

We solve:

$$4 + x + 5 = 11 \qquad 1 + x + 6 = 9$$

Recomputing the checksums we find that:

$$\begin{pmatrix} 5 & + & 1 & + & 7 & = & 13 \\ 4 & + & 2 & + & 5 & = & 11 \\ 4 & + & 6 & + & 9 & = & 19 \\ 13 & + & 9 & + & 21 & = & 43 \end{pmatrix}$$ Checksums match ☺

Can detect **two** errors and correct **one**

ABFT for Matrix-Matrix multiplication

Aim: Computation of $C = A \times B$

Let $e^T = [1, 1, \cdots, 1]$, we define

$$A^c := \begin{pmatrix} A \\ e^T A \end{pmatrix}, \; B^r := \begin{pmatrix} B & Be \end{pmatrix}, \; C^f := \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix}.$$

Where A^c is the *column checksum matrix*, B^r is the *row checksum matrix* and C^f is the *full checksum matrix*.

$$\begin{aligned} A^c \times B^r &= \begin{pmatrix} A \\ e^T A \end{pmatrix} \times \begin{pmatrix} B & Be \end{pmatrix} \\ &= \begin{pmatrix} AB & ABe \\ e^T AB & e^T ABe \end{pmatrix} = \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix} = C^f \end{aligned}$$

ABFT for Matrix-Matrix multiplication

Aim: Computation of $C = A \times B$

Let $e^T = [1, 1, \cdots, 1]$, we define

$$A^c := \begin{pmatrix} A \\ e^T A \end{pmatrix}, \; B^r := \begin{pmatrix} B & Be \end{pmatrix}, \; C^f := \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix}.$$

Where A^c is the *column checksum matrix*, B^r is the *row checksum matrix* and C^f is the *full checksum matrix*.

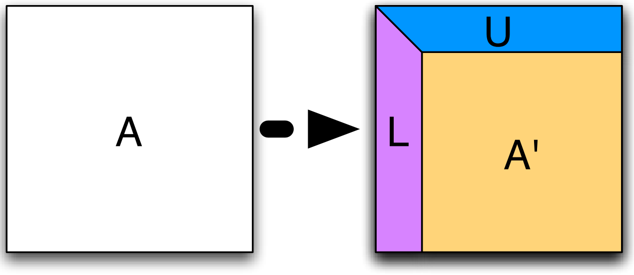
$$\begin{aligned} A^c \times B^r &= \begin{pmatrix} A \\ e^T A \end{pmatrix} \times \begin{pmatrix} B & Be \end{pmatrix} \\ &= \begin{pmatrix} AB & ABe \\ e^T AB & e^T ABe \end{pmatrix} = \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix} = C^f \end{aligned}$$

In practice... things are more complicated!

- When do errors strike? Are all data always protected?
- Computations are approximate because of floating-point rounding
- Error detection and error correction capabilities depend on the number of checksum rows and columns

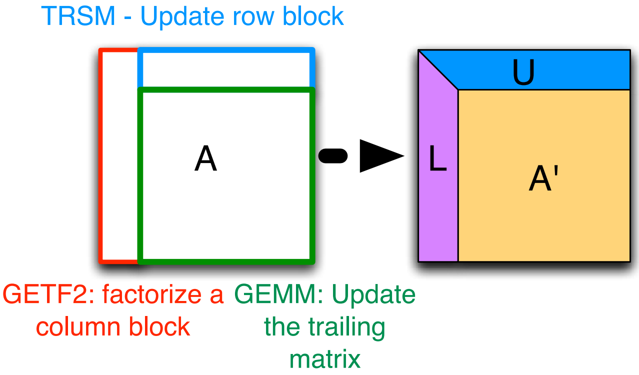
Outline

Example: block LU factorization



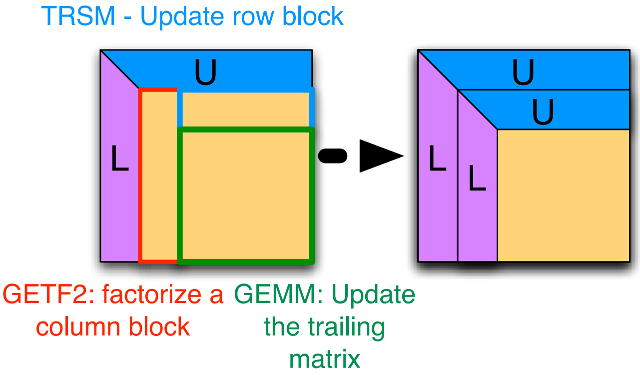
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$

Example: block LU factorization



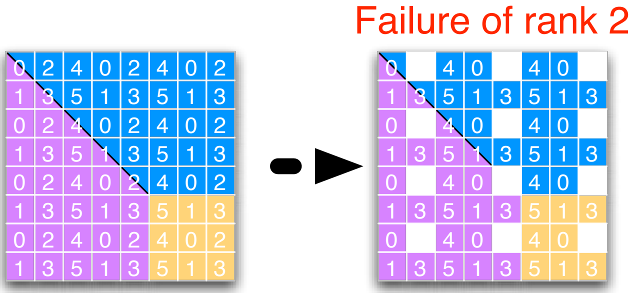
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$

Example: block LU factorization



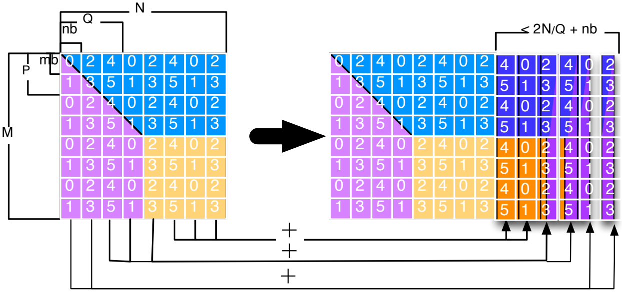
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$

Example: block LU factorization



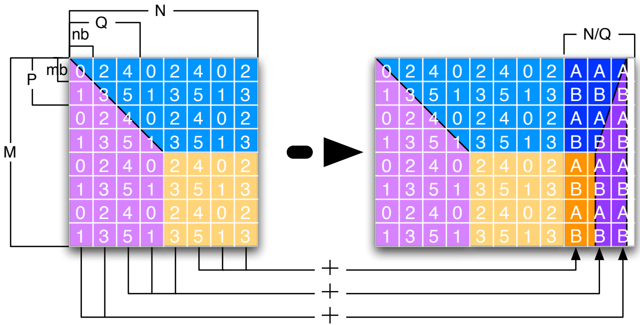
- 2D Block Cyclic Distribution (here 2×3)
- A single failure $\Rightarrow$ many data lost

Algorithm Based Fault Tolerant LU decomposition



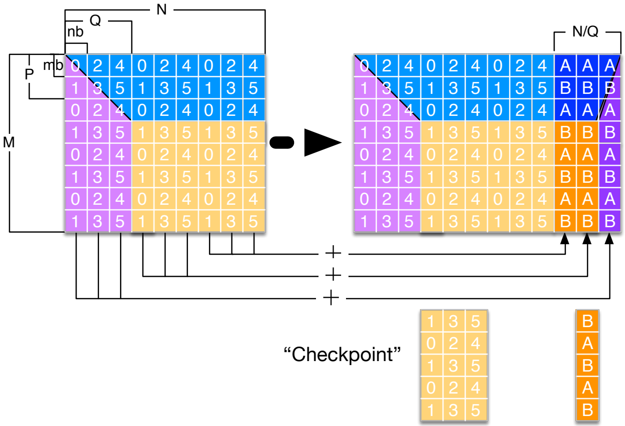
- Checksum: invertible operation on the data of the row / column
  - Checksum blocks are doubled, to allow recovery when data and checksum are lost together

Algorithm Based Fault Tolerant LU decomposition



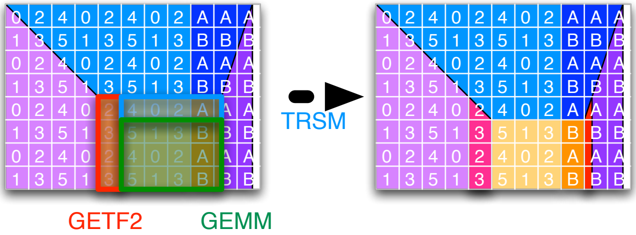
- Checksum: invertible operation on the data of the row / column
  - Checksum replication can be avoided by dedicating computing resources to checksum storage

Algorithm Based Fault Tolerant LU decomposition



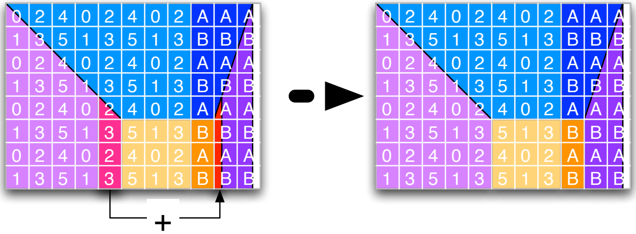
- Checkpoint the next set of Q-Panels to be able to return to it in case of failures

Algorithm Based Fault Tolerant LU decomposition



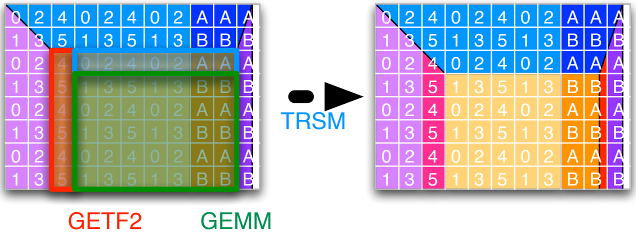
- Idea of ABFT: applying the operation on data and checksum preserves the checksum properties

Algorithm Based Fault Tolerant LU decomposition



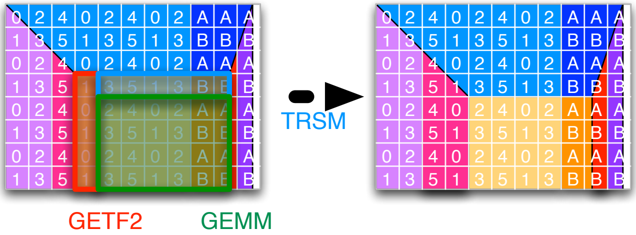
- For the part of the data that is not updated this way, the checksum must be re-calculated

Algorithm Based Fault Tolerant LU decomposition



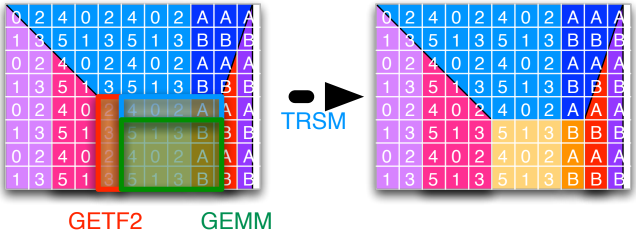
- To avoid slowing down all processors and panel operation, group checksum updates every Q block columns

Algorithm Based Fault Tolerant LU decomposition



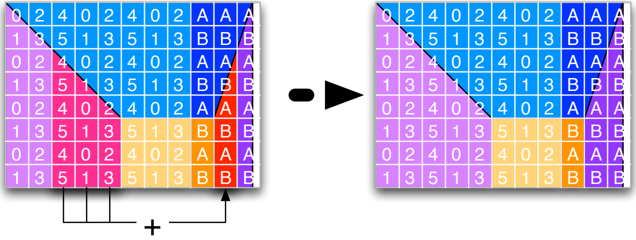
- To avoid slowing down all processors and panel operation, group checksum updates every Q block columns

Algorithm Based Fault Tolerant LU decomposition



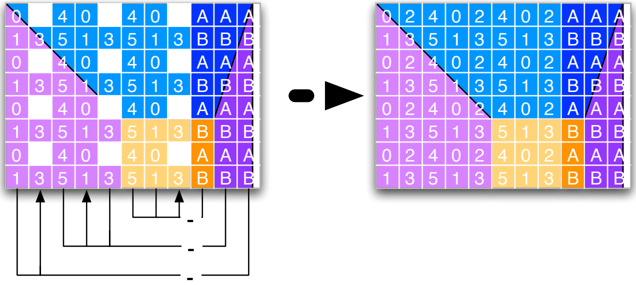
- To avoid slowing down all processors and panel operation, group checksum updates every Q block columns

Algorithm Based Fault Tolerant LU decomposition



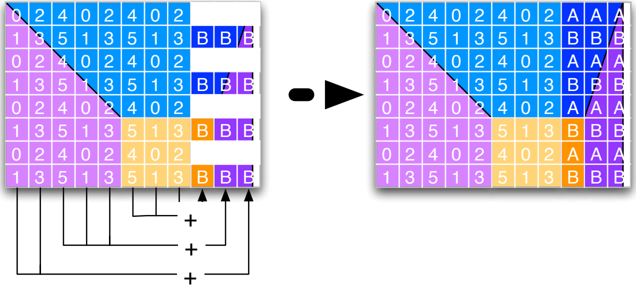
- Then, update the missing coverage. Keep checkpoint block column to cover failures during that time

Algorithm Based Fault Tolerant LU decomposition



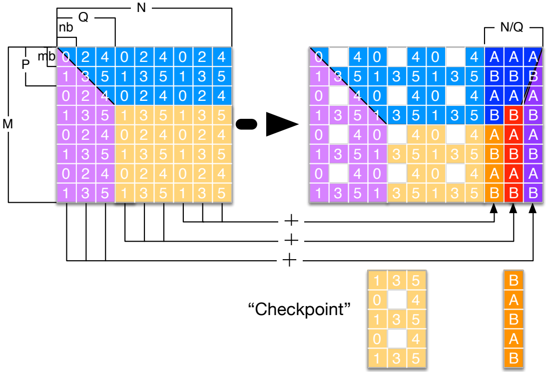
- In case of failure, conclude the operation, then
  - Missing Data = Checksum - Sum(Existing Data)

Algorithm Based Fault Tolerant LU decomposition



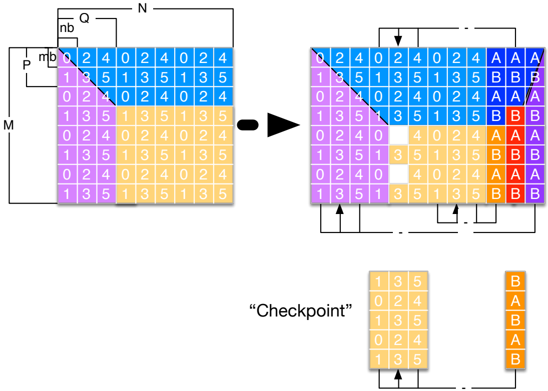
- In case of failure, conclude the operation, then
  - Missing Checksum = Sum(Existing Data)

Failure inside a $Q-$panel factorization



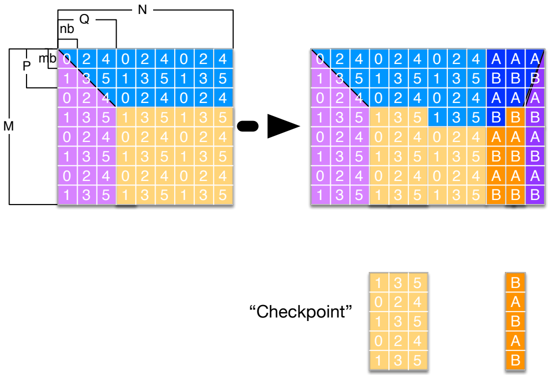
- Failures may happen while inside a $Q-$panel factorization

Failure inside a $Q-$panel factorization



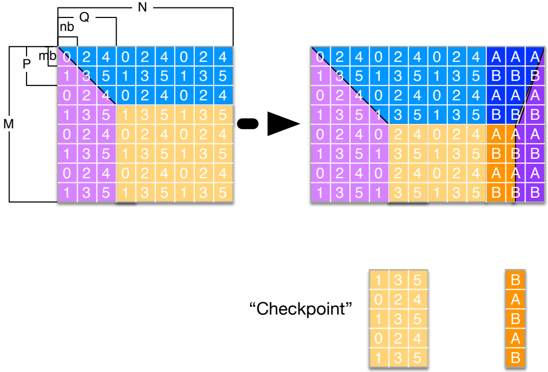
- Valid Checksum Information allows to recover most of the missing data, but not all: the checksum for the current $Q-$panels are not valid

Failure inside a $Q-$panel factorization



- We use the checkpoint to restore the $Q-$panel in its initial state

Failure inside a $Q-$panel factorization



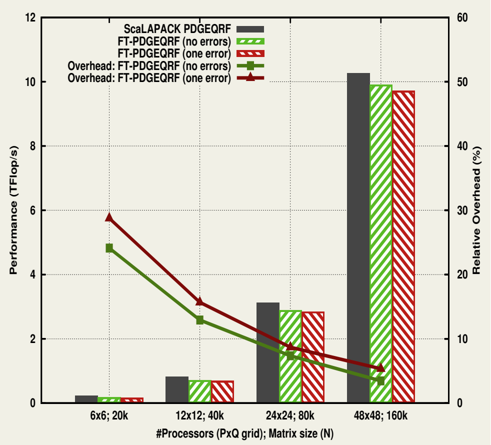
- and re-execute that part of the factorization, without applying outside of the scope

ABFT LU decomposition: implementation

MPI Implementation

- PBLAS-based: need to provide "Fault-Aware" version of the library
- Cannot enter recovery state at any point in time: need to complete ongoing operations despite failures
  - Recovery starts by defining the position of each process in the factorization and bring them all in a consistent state (checksum property holds)
- Need to test the return code of each and every MPI-related call

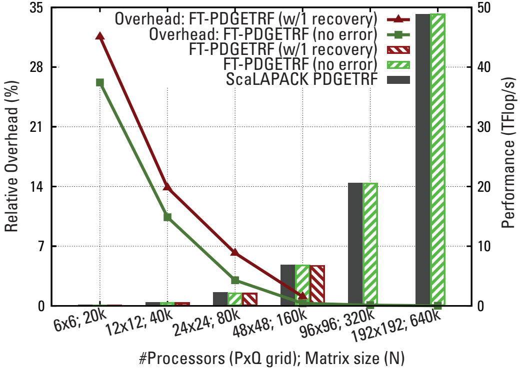
ABFT QR decomposition: performance



MPI-Next ULFM Performance

- Open MPI with ULFM; Kraken supercomputer;

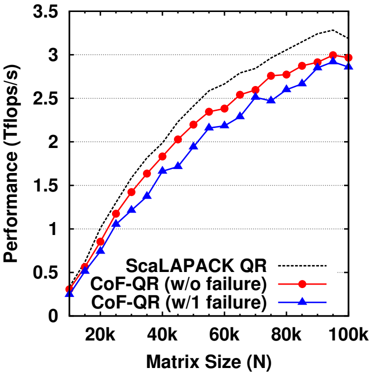
ABFT LU decomposition: performance



MPI-Next ULFM Performance

- Open MPI with ULFM; Kraken supercomputer;

ABFT QR decomposition: performance



Checkpoint on Failure - MPI Performance

- Open MPI; Kraken supercomputer;

Outline

Fault tolerance techniques

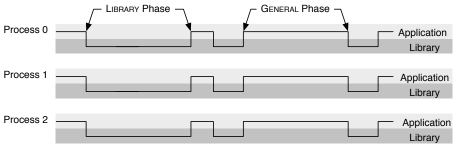
General techniques

- Replication
- Rollback recovery
  - Coordinated checkpointing
  - Uncoordinated checkpointing & Message logging
  - Hierarchical checkpointing

Application-specific techniques

- Algorithm Based Fault Tolerance (ABFT)
- Iterative convergence
- Approximated computation

Application



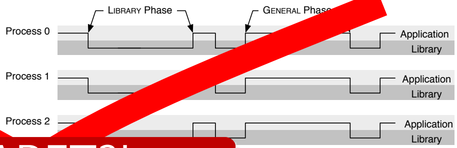
Typical Application

```
for( aninsanenumber ) {
  /* Extract data from
   * simulation, fill up
   * matrix */
  sim2mat();

  /* Factorize matrix,
   * Solve */
  dgeqrf();
  dsolve();

  /* Update simulation
   * with result vector */
  vec2sim();
}
```

Characteristics

- ☺ Large part of (total) computation spent in factorization/solve
- Between LA operations:
  - ☹ use resulting vector / matrix with operations that do not preserve the checksums on the data
  - ☹ modify data not covered by ABFT algorithms

Application



Typical Applicatio

```
for( aninsanenumber ) {
  /* Extract dat
   * simulation ,
   * matrix */
  sim2mat();

  /* Factorize matri
   * Solve */
  dgeqrf();
  dsolve();

  /* Updat  imulation
   * with  esult vector */
  vec2s   ();
}
```

Goodbye ABFT?!

- Larg part of (total) compu tion spent in factorizat /solve
- Between LA erations:
  - use resultin ector / matrix with operatio that do not preserve the che sums on the data
  - modify data not cov d by ABFT algorithms

Application

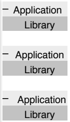
Problem Statement

How to use fault tolerant operations$^{(*)}$ within a non-fault tolerant$^{(**)}$ application?$^{(***)}$

(*) ABFT, or other application-specific FT
(**) Or within an application that does not have the same kind of FT
(***) And keep the application globally fault tolerant...

Typica

```
for( a
  /* I
   * s
   * r
  sim2

  /* I
   * S
  dge(
  dsol

  /* Update simulation
   * with result vector */
  vec2sim();
}
```

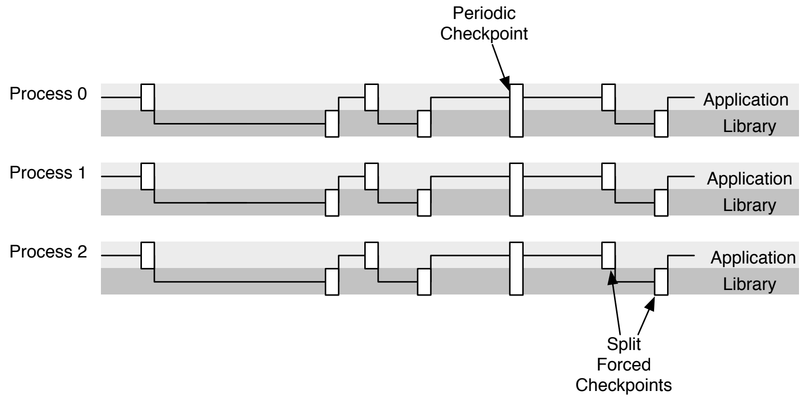


use resulting vector / matrix with operations that do not preserve the checksums on the data

☹ modify data not covered by ABFT algorithms

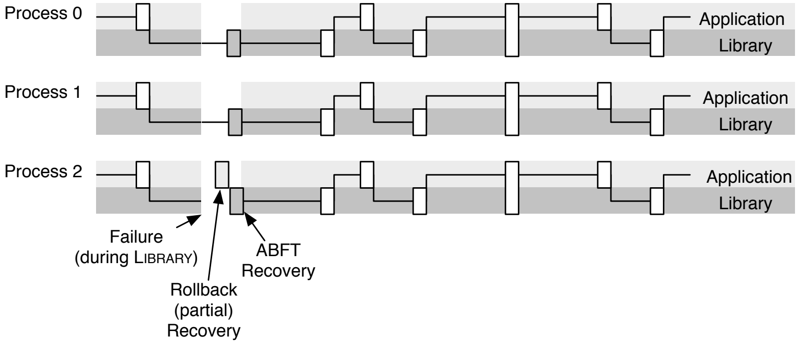
ABFT&PeriodicCkpt

ABFT&PeriodicCkpt: no failure



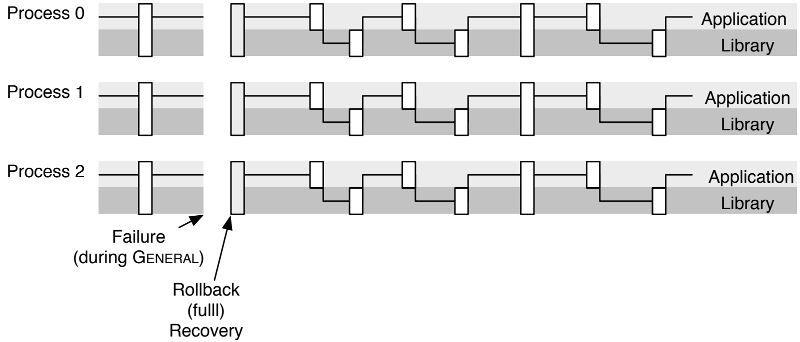
ABFT&PeriodicCkpt

ABFT&PeriodicCkpt: failure during Library phase

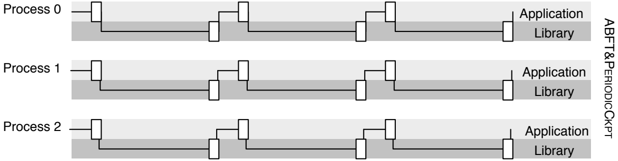


ABFT&PeriodicCkpt

ABFT&PeriodicCkpt: failure during General phase



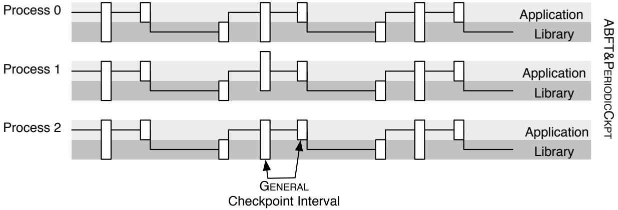
ABFT&PeriodicCkpt: Optimizations



ABFT&PeriodicCkpt: Optimizations

- If the duration of the General phase is too small: don't add checkpoints
- If the duration of the Library phase is too small: don't do ABFT recovery, remain in General mode
  - this assumes a performance model for the library call

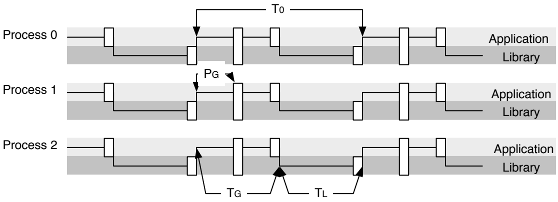
ABFT&PeriodicCkpt: Optimizations



ABFT&PeriodicCkpt: Optimizations

- If the duration of the General phase is too small: don't add checkpoints
- If the duration of the Library phase is too small: don't do ABFT recovery, remain in General mode
  - this assumes a performance model for the library call

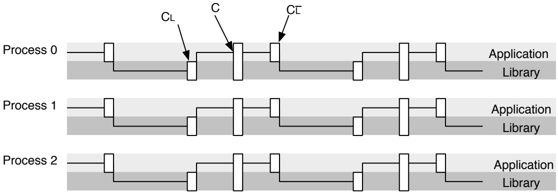
A few notations



Times, Periods

T_0: Duration of an Epoch (without FT)
$T_L = \alpha T_0$: Time spent in the LIBRARY phase
$T_G = (1 - \alpha)T_0$: Time spent in the GENERAL phase
P_G: Periodic Checkpointing Period
T^{ff}, T_G^{ff}, T_L^{ff}: "Fault Free" times
t_G^{lost}, t_L^{lost}: Lost time (recovery overhreads)
T_G^{final}, T_L^{final}: Total times (with faults)

A few notations



Costs

$C_L = \rho C$: time to take a checkpoint of the LIBRARY data set
$C_{\bar{L}} = (1-\rho)C$: time to take a checkpoint of the GENERAL data set
$R, R_{\bar{L}}$: time to load a full / GENERAL data set checkpoint
D: down time (time to allocate a new machine / reboot)
$\mathrm{Recons}_{\mathrm{ABFT}}$: time to apply the ABFT recovery
ϕ: Slowdown factor on the LIBRARY phase, when applying ABFT

Overall

Overall

Time (with overheads) of LIBRARY phase is constant (in P_G):

$$T_L^{\text{final}} = \frac{1}{1 - \frac{D + R_{\bar{L}} + \text{Recons}_{\text{ABFT}}}{\mu}} \times (\alpha \times T_L + C_L)$$

Time (with overehads) of GENERAL phase accepts two cases:

$$T_G^{\text{final}} = \begin{cases} \frac{1}{1 - \frac{D+R+\frac{T_G + C_{\bar{L}}}{2}}{\mu}} \times (T_G + C_L) & \text{if } T_G < P_G \\ \frac{T_G}{(1-\frac{C}{P_G})(1-\frac{D+R+\frac{P_G}{2}}{\mu})} & \text{if } T_G \geq P_G \end{cases}$$

Which is minimal in the second case, if

$$P_G = \sqrt{2C(\mu - D - R)}$$

Waste

From the previous, we derive the waste, which is obtained by

$$\textsc{Waste} = 1 - \frac{T_0}{T_G^{\text{final}} + T_L^{\text{final}}}$$

Toward Exascale, and beyond!

Let's think at scale

- Number of components $\nearrow \Rightarrow$ MTBF $\searrow$
- Number of components $\nearrow \Rightarrow$ Problem size $\nearrow$
- Problem size $\nearrow \Rightarrow$
  Computation time spent in LIBRARY phase $\nearrow$

☺ ABFT&PERIODICCKPT should perform better with scale

By how much?

Competitors

FT algorithms compared

PeriodicCkpt Basic periodic checkpointing

Bi-PeriodicCkpt Applies incremental checkpointing techniques to save only the library data during the library phase

ABFT&PeriodicCkpt The algorithm described above

Weak Scale #1

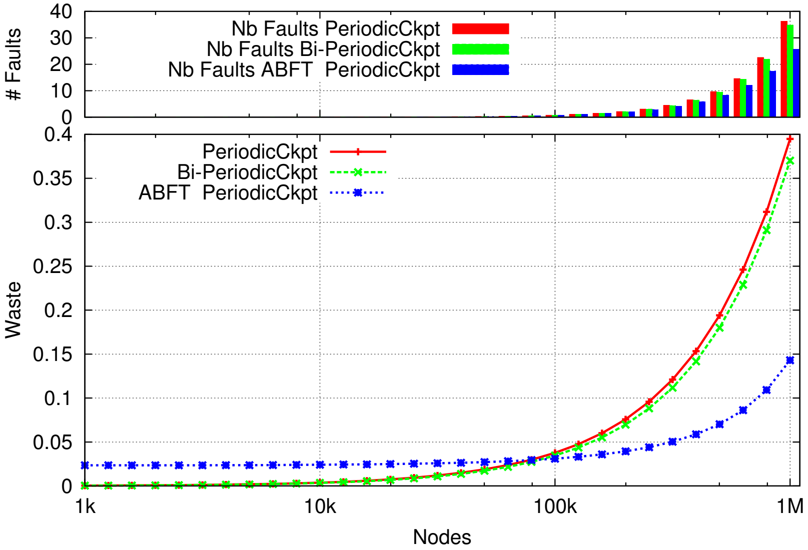
Weak Scale Scenario #1

- Number of components, n, increase
- Memory per component remains constant
- Problem size increases in $O(\sqrt{n})$ (e.g. matrix operation)

- μ at $n = 10^5$: 1 day, is in $O(\frac{1}{n})$
- C $(=R)$ at $n = 10^5$, is 1 minute, is in $O(n)$
- α is constant at 0.8, as is ρ.

  (both LIBRARY and GENERAL phase increase in time at the same speed)

Weak Scale #1



Faults
40
30
20
10
0
Nb Faults PeriodicCkpt
Nb Faults Bi-PeriodicCkpt
Nb Faults ABFT PeriodicCkpt
Waste
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
PeriodicCkpt
Bi-PeriodicCkpt
ABFT PeriodicCkpt
1k
10k
100k
1M
Nodes

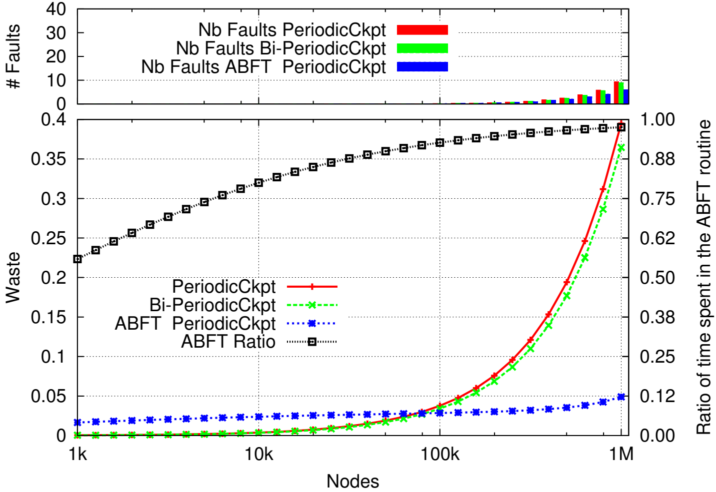
Weak Scale #2

Weak Scale Scenario #2

- Number of components, n, increase
- Memory per component remains constant
- Problem size increases in $O(\sqrt{n})$ (e.g. matrix operation)

- μ at $n = 10^5$: 1 day, is $O(\frac{1}{n})$
- C ($=R$) at $n = 10^5$, is 1 minute, is in $O(n)$
- ρ remains constant at 0.8, but LIBRARY phase is $O(n^3)$ when GENERAL phases progresses in $O(n^2)$ (α is 0.8 at $n = 10^5$ nodes).

Weak Scale #2


Nb Faults PeriodicCkpt
Nb Faults Bi-PeriodicCkpt
Nb Faults ABFT PeriodicCkpt
Faults
40
30
20
10
0
PeriodicCkpt
Bi-PeriodicCkpt
ABFT PeriodicCkpt
ABFT Ratio
Waste
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1.00
0.88
0.75
0.62
0.50
0.38
0.25
0.12
0.00
Ratio of time spent in the ABFT routine
1k
10k
100k
1M
Nodes

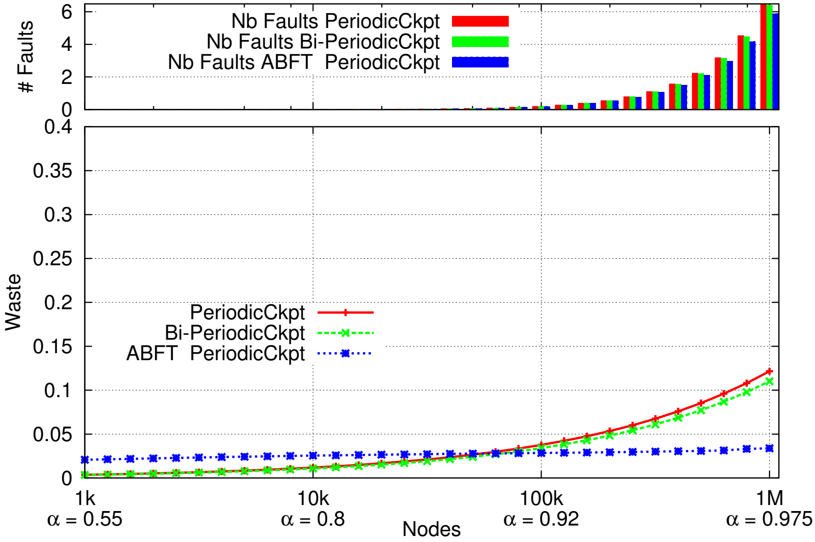
Weak Scale #3

Weak Scale Scenario #3

- Number of components, n, increase
- Memory per component remains constant
- Problem size increases in $O(\sqrt{n})$ (e.g. matrix operation)

- μ at $n = 10^5$: 1 day, is $O(\frac{1}{n})$
- C $(=R)$ at $n = 10^5$, is 1 minute, stays independent of n $(O(1))$
- ρ remains constant at 0.8, but LIBRARY phase is $O(n^3)$ when GENERAL phases progresses in $O(n^2)$ (α is 0.8 at $n = 10^5$ nodes).

Weak Scale #3


6
4
2
0
Faults
Nb Faults PeriodicCkpt
Nb Faults Bi-PeriodicCkpt
Nb Faults ABFT PeriodicCkpt
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Waste
PeriodicCkpt
Bi-PeriodicCkpt
ABFT PeriodicCkpt
1k
α = 0.55
10k
α = 0.8
100k
α = 0.92
1M
α = 0.975
Nodes

Outline

Leitmotiv

Resilient research on resilience

Models needed to assess techniques at scale
without bias ☺

Conclusion

- 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)

Conclusion

- Multiple approaches to Fault Tolerance
- General Purpose Fault Tolerance is a required feature of the platforms
  - Not every computer scientist needs to learn how to write fault-tolerant applications
  - Not all parallel applications can be ported to a fault-tolerant version
- Faults are a feature of the platform. Why should it be the role of the programmers to handle them?

Conclusion

Application-Specific Fault Tolerance

- Fault Tolerance is introducing redundancy in the application
  - replication of computation
  - maintaining invariant in the data
- Requirements of a more Fault-friendly programming environment
  - MPI-Next evolution
  - Other programming environments?

Conclusion

General Purpose Fault Tolerance

- Software/hardware techniques to reduce checkpoint, recovery, migration times and to improve failure prediction
- Multi-criteria scheduling problem
  execution time/energy/reliability
  add replication
  best resource usage (performance trade-offs)
- Need combine all these approaches!

Several challenging algorithmic/scheduling problems ☺

Bibliography

Exascale

- Toward Exascale Resilience, Cappello F. et al., IJHPCA 23, 4 (2009)
- The International Exascale Software Roadmap, Dongarra, J., Beckman, P. et al., IJHPCA 25, 1 (2011)

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

Coordinated Checkpointing Distributed snapshots: determining global states of distributed systems, Chandy K.M., Lamport L., ACM Trans. Comput. Syst. 3, 1 (1985)

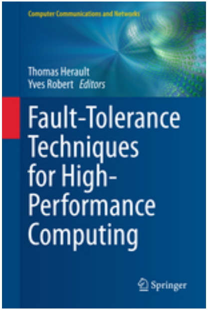
Message Logging A survey of rollback-recovery protocols in message-passing systems, Elnozahy E.N. et al., ACM Comput. Surveys 34, 3 (2002)

Replication Evaluating the viability of process replication reliability for exascale systems, Ferreira K. et al, SC'2011

Models

- Checkpointing strategies for parallel jobs, Bougeret M. et al., SC'2011
- Unified model for assessing checkpointing protocols at extreme-scale, Bosilca G et al., INRIA RR-7950, 2012

Bibliography



New Monograph, Springer Verlag 2015

Thanks Yves Robert, Thomas Hérault, George Bosilca, Aurélien Bouteiller and Frédéric Vivien for the slides (SC'15 tutorial, JLESC'16 summer school)