Algorithm-Based Fault Tolerance

Thomas Hérault[1], Yves Robert[1,2] & Frédéric Vivien[2]

1 – University of Tennessee Knoxville, USA
2 – ENS Lyon & INRIA, France

frederic.vivien@inria.fr

3rd JLESC Summer School

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/algorihtm
- 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

Algorithm Based Fault Tolerance (ABFT)

Principle

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

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

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

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

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: $4+3+5=12$.

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

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

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

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

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

One row and one column of checksums

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

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.

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:

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

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*.

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*.

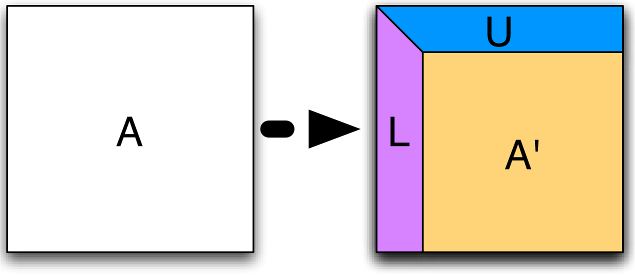
$$\begin{aligned} A^c \times B^r &= \begin{pmatrix} A \\ e^T A \end{pmatrix} \times \begin{pmatrix} B & Be \end{pmatrix} \\ &= \begin{pmatrix} AB & ABe \\ e^T AB & e^T ABe \end{pmatrix} = \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix} = C^f \end{aligned}$$

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

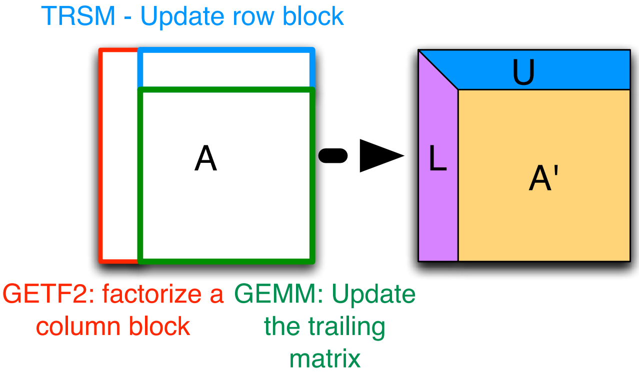
Outline

Block LU factorization



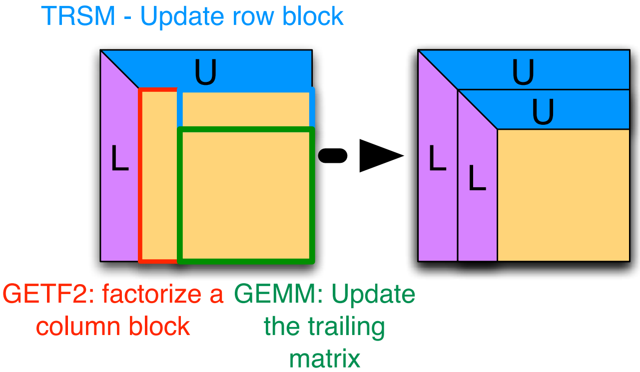
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$

Block LU factorization



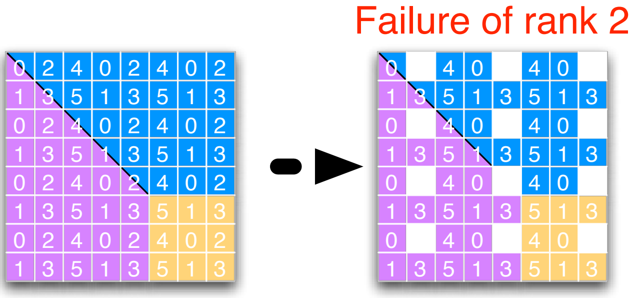
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$

Block LU factorization



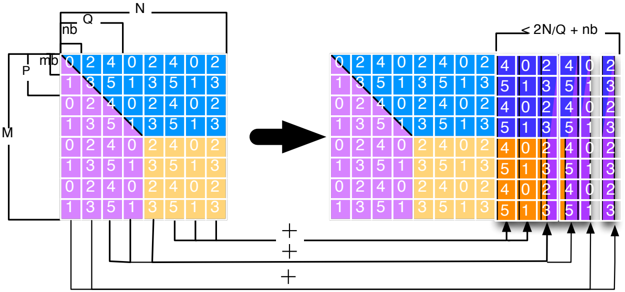
- Solve $A \cdot x = b$ (hard)
- Transform A into a LU factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$

Block LU factorization



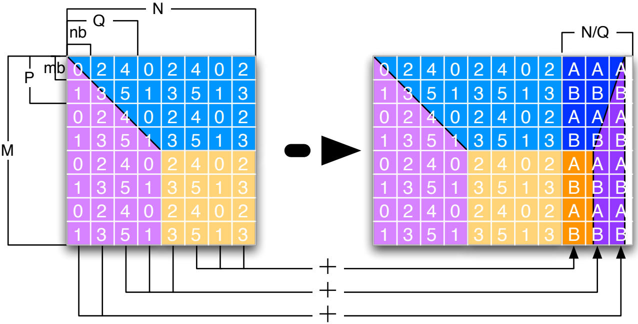
- 2D Block Cyclic Distribution (here 2×3)
- A single failure $\Rightarrow$ many data lost

Algorithm Based Fault Tolerant LU decomposition



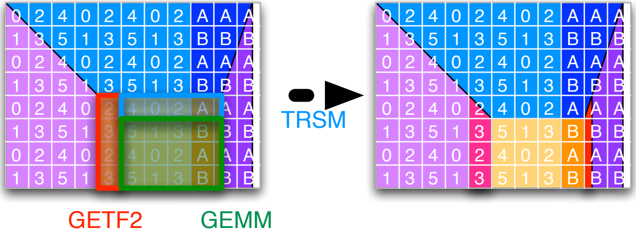
- Checksum:
  invertible operation on the data of the row / column
  - Checksum blocks are doubled, to allow recovery when data and checksum are lost together

Algorithm Based Fault Tolerant LU decomposition



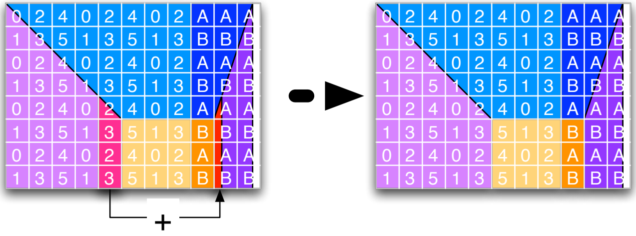
- Checksum:
  invertible operation on the data of the row / column
  - Checksum replication can be avoided by dedicating computing resources to checksum storage

Algorithm Based Fault Tolerant LU decomposition



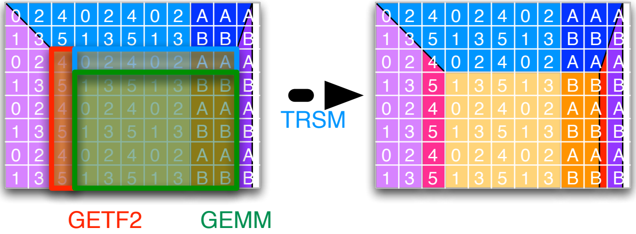
- Idea of ABFT: applying the operation on data and checksum preserves the checksum properties

Algorithm Based Fault Tolerant LU decomposition



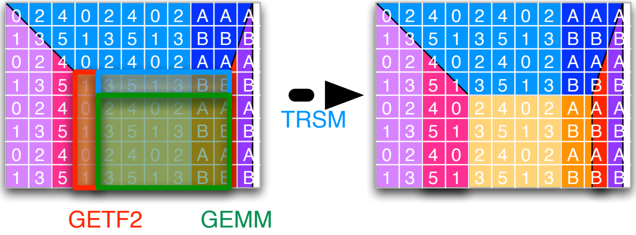
- For the part of the data that is not updated this way, the checksum must be re-calculated

Algorithm Based Fault Tolerant LU decomposition



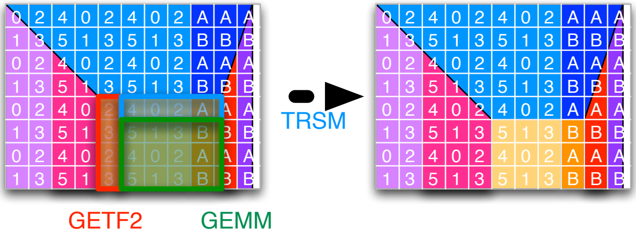
- To avoid slowing down all processors and panel operation, group checksum updates every Q block columns

Algorithm Based Fault Tolerant LU decomposition



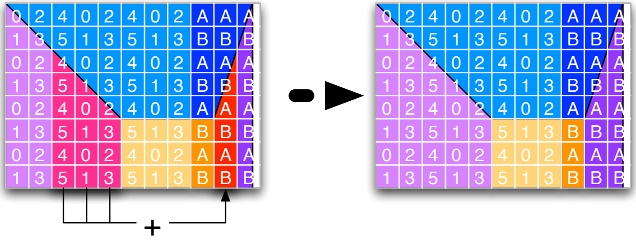
- To avoid slowing down all processors and panel operation, group checksum updates every Q block columns

Algorithm Based Fault Tolerant LU decomposition



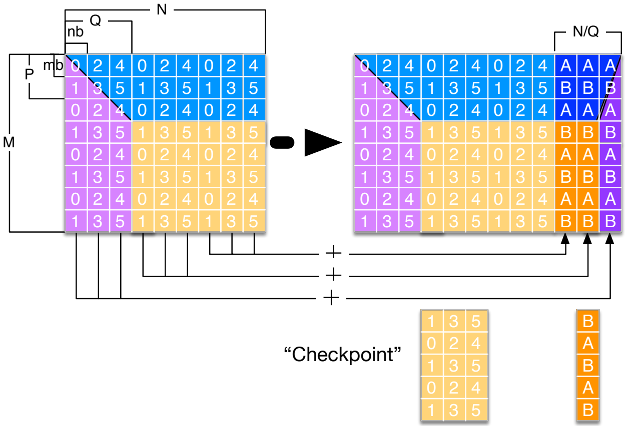
- To avoid slowing down all processors and panel operation, group checksum updates every Q block columns

Algorithm Based Fault Tolerant LU decomposition



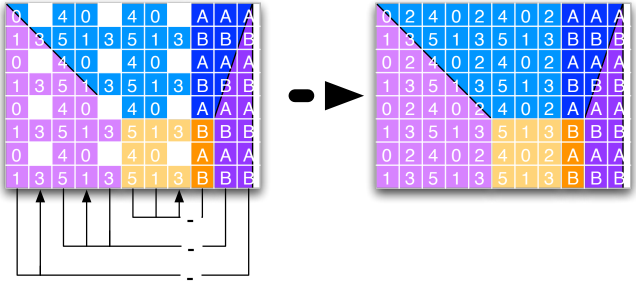
- Then, update the missing coverage.
  Keep checkpoint block column to cover failures during that time

Algorithm Based Fault Tolerant LU decomposition



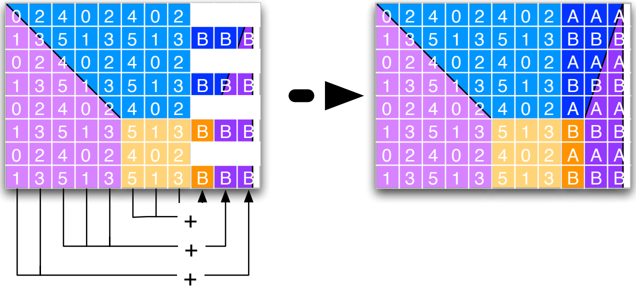
- Checkpoint the next set of Q-Panels to be able to return to it in case of failures

Algorithm Based Fault Tolerant LU decomposition



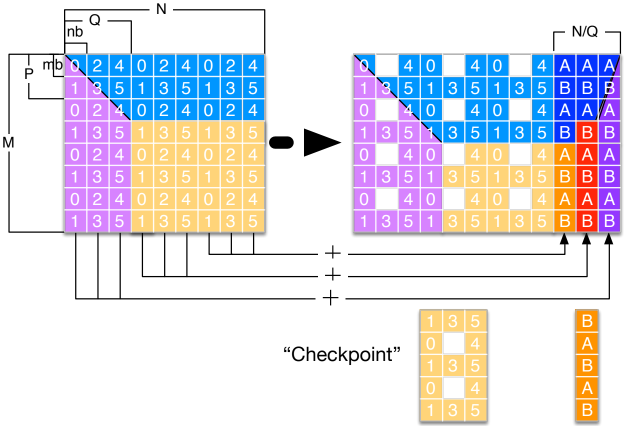
- In case of failure, conclude the operation, then
  - Missing Data = Checksum - Sum(Existing Data) s

Algorithm Based Fault Tolerant LU decomposition



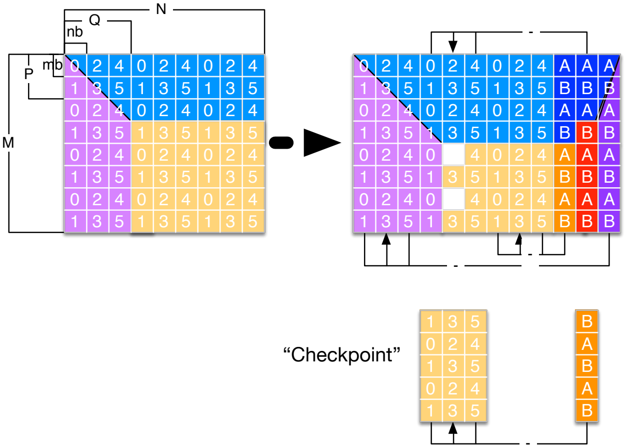
- In case of failure, conclude the operation, then
  - Missing Checksum = Sum(Existing Data)s

Failure inside a $Q-$panel factorization



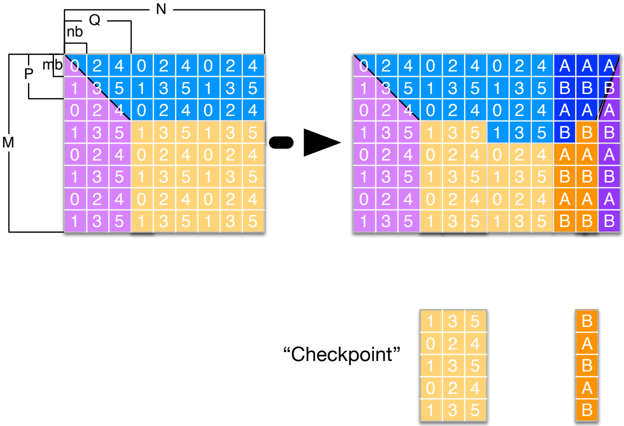
- Failures may happen while inside a $Q-$panel factorization

Failure inside a $Q-$panel factorization



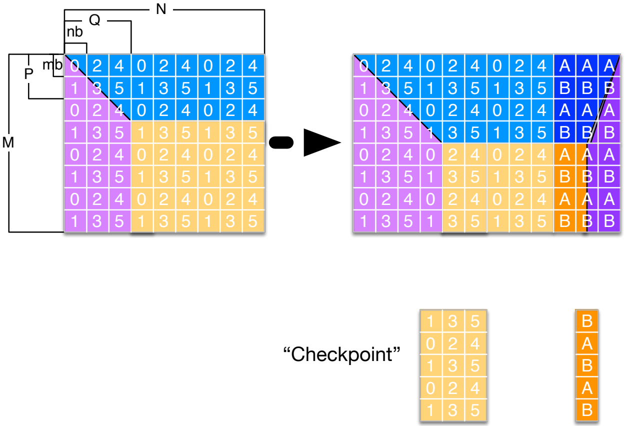
- Valid Checksum Information allows to recover most of the missing data, but not all: the checksum for the current $Q-$panels are not valid

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



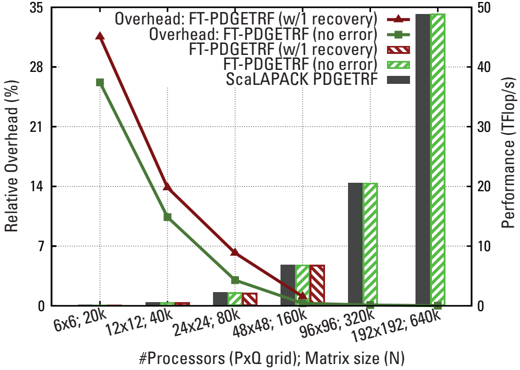
- and re-execute that part of the factorization, without applying outside of the scope

ABFT LU decomposition: implementation

MPI Implementation

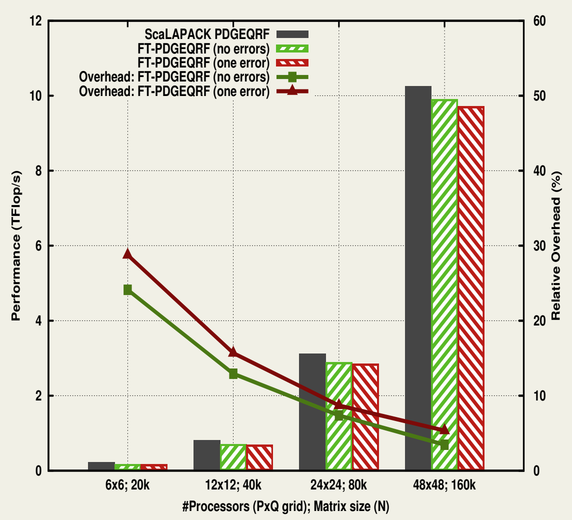
- PBLAS-based: need to provide "Fault-Aware" version of the library
- Cannot enter recovery state at any point in time: need to complete ongoing operations despite failures
  - Recovery starts by defining the position of each process in the factorization and bring them all in a consistent state (checksum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

ABFT LU decomposition: performance



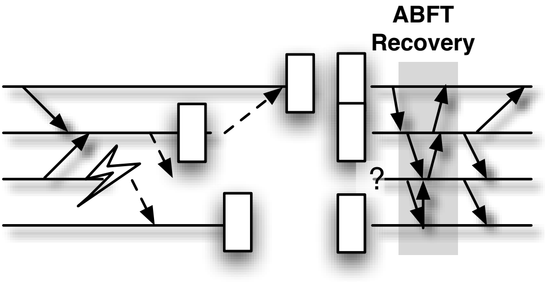
Open MPI with ULFM; Kraken supercomputer.

ABFT QR decomposition: performance



Open MPI with ULFM; Kraken supercomputer.

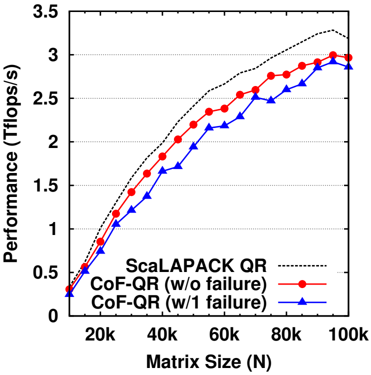
ABFT LU decomposition: implementation



Checkpoint on Failure - MPI Implementation

- FT-MPI / MPI-Next FT: not easily available on large machines
- Checkpoint on Failure = workaround

ABFT QR decomposition: performance



Checkpoint on Failure - MPI Performance

- Open MPI; Kraken supercomputer;

Outline

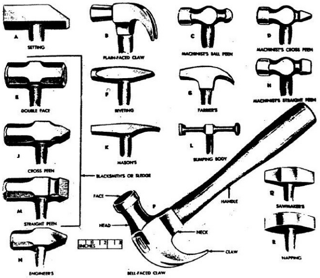
Fault Tolerance Techniques

General Techniques

- Replication
- Rollback Recovery
  - Coordinated Checkpointing
  - Uncoordinated Checkpointing & Message Logging
  - Hierarchical Checkpointing
  - Multilevel Checkpointing

Application-Specific Techniques

- Algorithm Based Fault Tolerance (ABFT)
- Iterative Convergence
- Approximated Computation



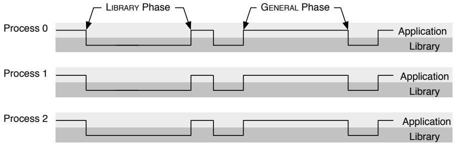
Application

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

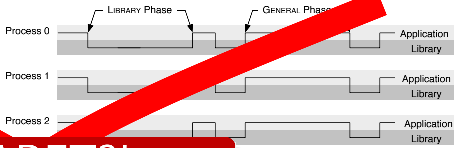
Application

Typical Application

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

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

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



Goodbye ABFT?!

- ☺ 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

Application
Library
Application
Library
Application
Library

Typica

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

  /* I
   * S
  dge(
  dsol

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

Problem Statement

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

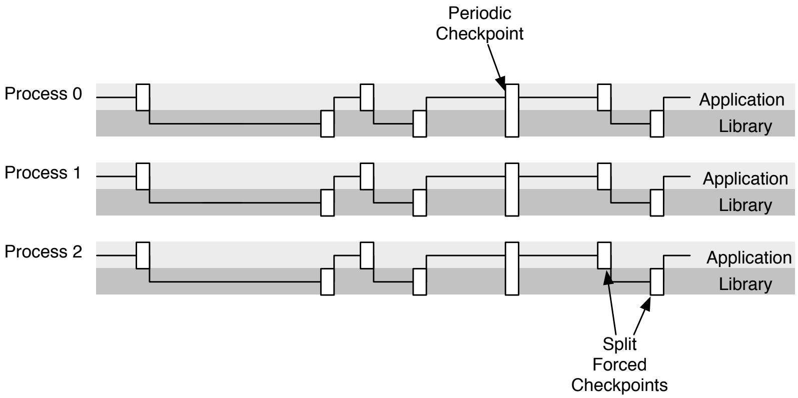
(*) ABFT, or other application-specific FT
(**) Or within an application that does not have the same kind of FT
(***) And keep the application globally fault tolerant...

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

☹ modify data not covered by ABFT algorithms

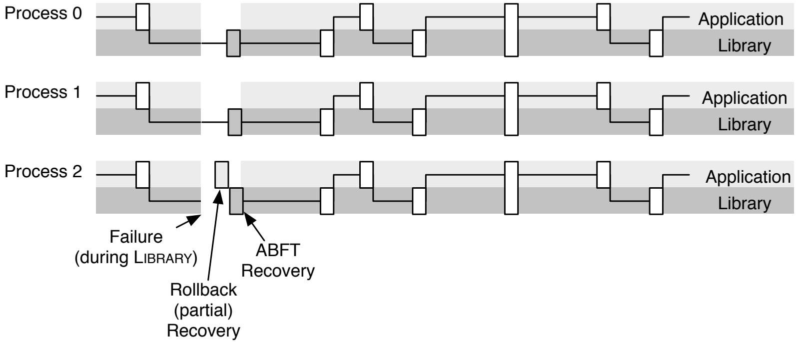
ABFT&PeriodicCkpt

ABFT&PeriodicCkpt: no failure



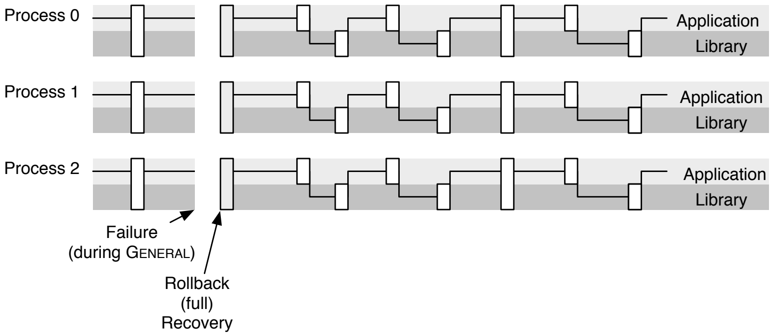
ABFT&PeriodicCkpt

ABFT&PeriodicCkpt: failure during Library phase

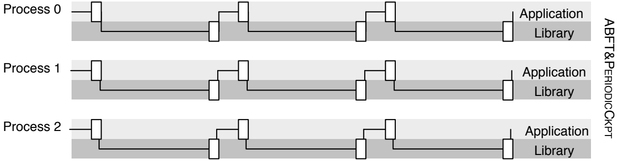


ABFT&PeriodicCkpt

ABFT&PeriodicCkpt: failure during General phase



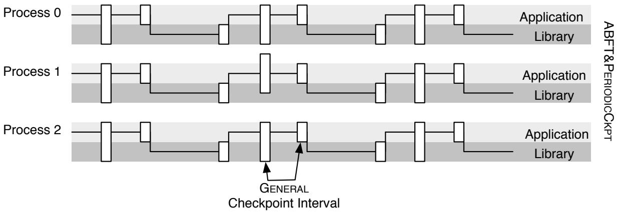
ABFT&PeriodicCkpt: Optimizations



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

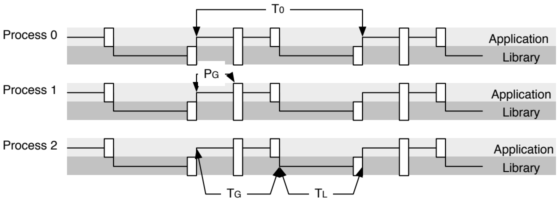
ABFT&PeriodicCkpt: Optimizations



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

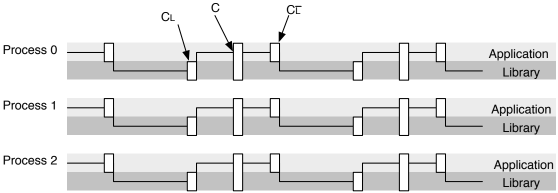
A few notations



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
$T^{\text{ff}}, T_G^{\text{ff}}, T_L^{\text{ff}}$: "Fault Free" times
$t_G^{\text{lost}}, t_L^{\text{lost}}$: Lost time (recovery overheads)
$T_G^{\text{final}}, T_L^{\text{final}}$: Total times (with faults)

A few notations

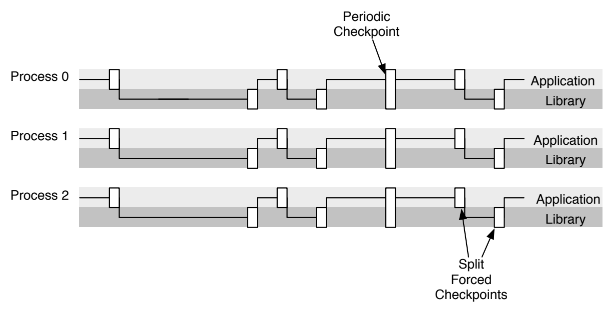


Costs

$C_L = \rho C$: 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)
$\text{Recons}_{\text{ABFT}}$: time to apply the ABFT recovery
ϕ: Slowdown factor on the LIBRARY phase, when applying ABFT

GENERAL phase, fault free waste

GENERAL phase

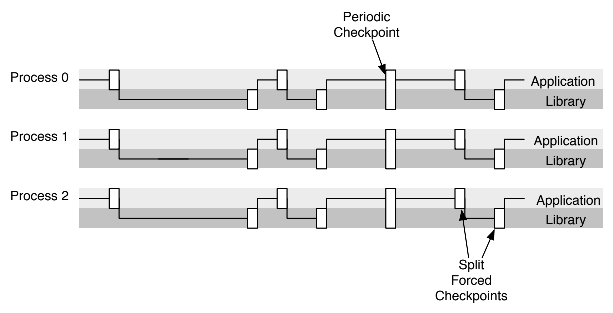


Without Failures

$$T_G^{\text{ff}} = \begin{cases} T_G + C_{\bar{L}} & \text{if } T_G < P_G \\ \frac{T_G}{P_G - C} \times P_G & \text{if } T_G \geq P_G \end{cases}$$

Library phase, fault free waste

Library phase

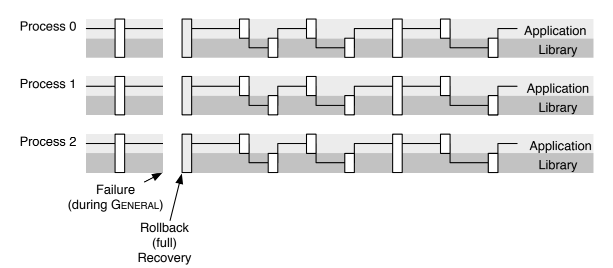


Without Failures

$$T_L^{\text{ff}} = \phi \times T_L + C_L$$

GENERAL phase, failure overhead

GENERAL phase

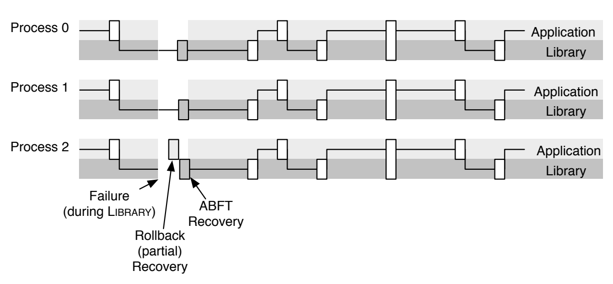


Failure Overhead

$$t_G^{\text{lost}} = \begin{cases} D + R + \frac{T_G^{\text{ff}}}{2} & \text{if } T_G < P_G \\ D + R + \frac{P_G}{2} & \text{if } T_G \geq P_G \end{cases}$$

LIBRARY phase, failure overhead

LIBRARY phase



Failure Overhead

$$t_L^{\text{lost}} = D + R_{\bar{L}} + \text{Recons}_{\text{ABFT}}$$

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

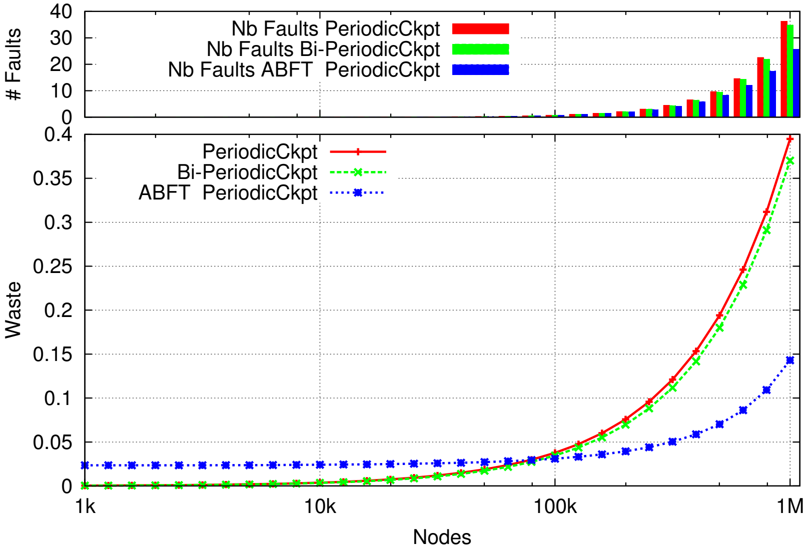
Weak Scale #1

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

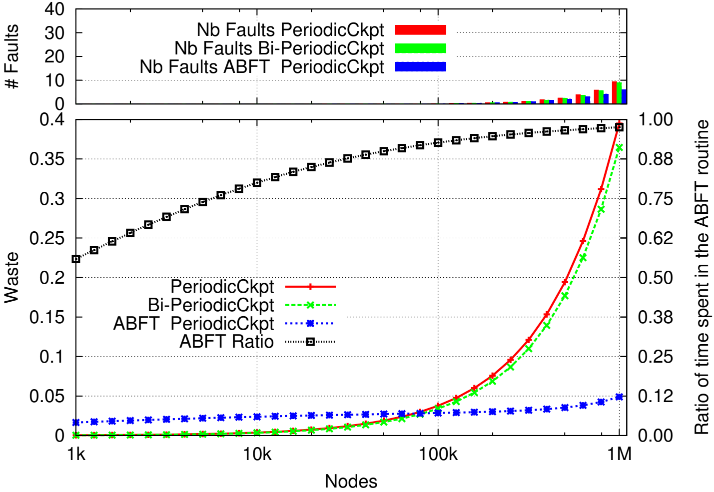
Weak Scale #2

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



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
ABFT Ratio
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

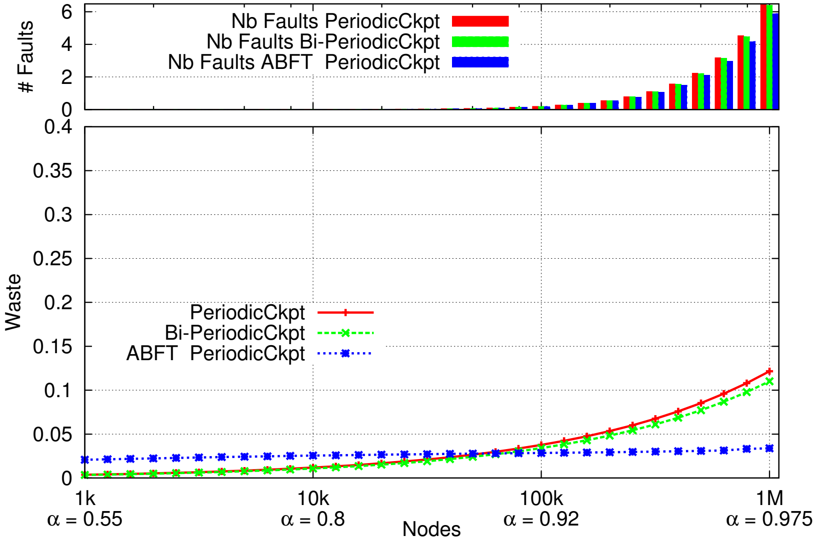
Weak Scale #3

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



Faults
6
4
2
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
α = 0.55
10k
α = 0.8
100k
α = 0.92
1M
α = 0.975
Nodes

Conclusion

Algorithm-Based Fault Tolerance

- Application-specific solution for linear algebra kernels
- Low-overhead forward-recovery solution
- Used alone or in conjunction with backward-recovery solutions

Going further

- **Algorithm-Based Fault Tolerance for Dense Matrix Factorizations, Multiple Failures and Accuracy.** A. Bouteiller, Th. Herault, G. Bosilca, P. Du, J. Dongarra. ACM Transactions on Parallel Computing 1(2), 2015.
- **Composing resilience techniques: ABFT, periodic and incremental checkpointing.** G. Bosilca, A. Bouteiller, Th. Herault, Y. Robert, J. Dongarra. IJNC 5(1), 2015.
- **Fault tolerance techniques for high-performance computing.** J. Dongarra, Th. Herault, Y. Robert. http://www.netlib.org/lapack/lawnspdf/lawn289.pdf