Fault-tolerant matrix factorisation: a formal model and proof

Camille Coti, Laure Petrucci, Daniel Alberto Torres González

Laboratoire d'Informatique de Paris Nord, CNRS UMR 7030,
Université Paris 13, Sorbonne Paris Cité
99, avenue Jean-Baptiste Clément
F-93430 Villetaneuse, FRANCE

camille.coti@lipn.univ-paris13.fr
laure.petrucci@lipn.univ-paris13.fr
daniel.torres@lipn.univ-paris13.fr

April 6, 2019

Content

Motivation

Matrix operations

- Addition, transposition, matrix multiplication
- Row operations, submatrix
- Diagonal matrix, triangular matrix, identity matrix, orthogonal matrix
- Determinant, eigenvalues, eigenvectors

Motivation

Matrix operations

- Addition, transposition, matrix multiplication
- Row operations, submatrix
- Diagonal matrix, triangular matrix, identity matrix, orthogonal matrix
- Determinant, eigenvalues, eigenvectors

Decompositions

- QR, LU, Cholesky
  - TSQR: iterative methods use it
    - Linear systems with multiple right-hand sides
    - Block iterative eigensolvers
    - s-step Krylov methods

Motivation

Fault tolerance in HPC

- System-level
  - Transparent for the application
  - Specific middleware to ensure coherent state of the application
- Application-level
  - The application itself handles the failures and adapt to them
  - The middleware must be robust enough to provide primitives

High Performance Systems

Platforms at large scale

- Have their own technical challenges
  - The total number of hardware and software components grows exponentially
  - Platforms needed to manage and handle complex computational problems
  - Hardware or software failures may occur anytime during the execution of high parallel applications
- System reliability, availability and scalability are factors to deal with
  - Failures may result in a high execution times and high cost

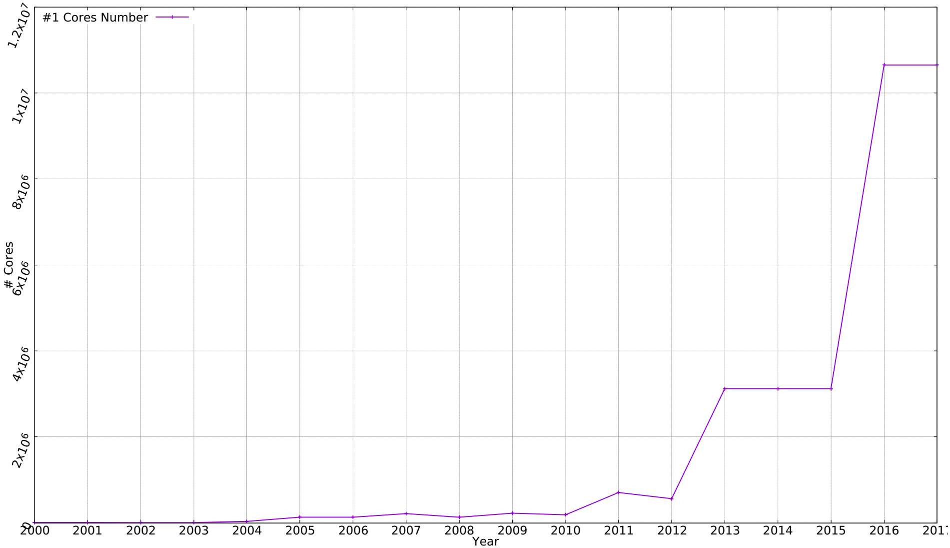
Top500.org

Top500

- A statistical list with ranks and details of the 500 world's most powerful supercomputers
- It shows that performance has almost doubled each year

Rank	System	**Cores**	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	**2,397,824**	143,500.0	200,794.9	9,783
2	Sierra - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	**1,572,480**	94,640.0	125,712.0	7,438
3	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	**10,649,600**	93,014.6	125,435.9	15,371
4	Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	**4,981,760**	61,444.5	100,678.7	18,482
5	Piz Daint - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	**387,872**	21,230.0	27,154.3	2,384

#1 Cores Number



Failures

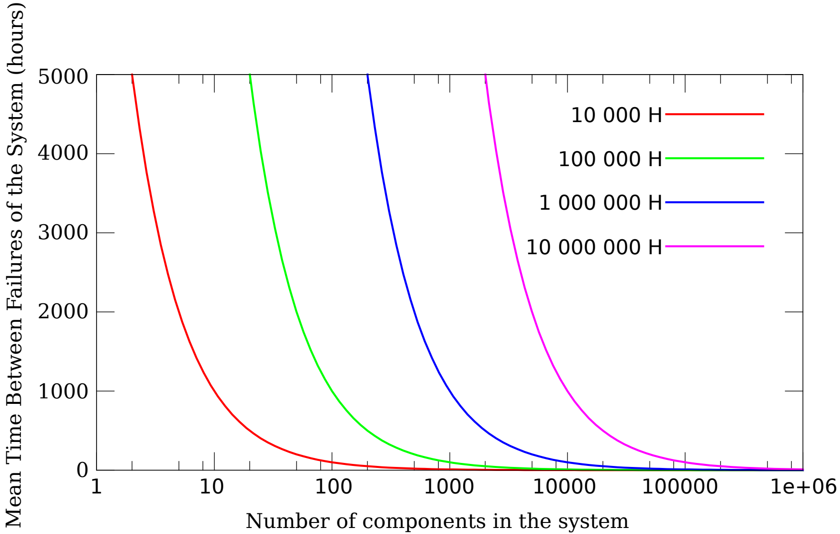
Failures in High Performance Systems

- Node increase in HPC $\Rightarrow$ platforms more subject to failures
  - Mean Time Between Failures (MTBF): measure of system reliability
  - Defined as the probability that the system performs without deviations from agreed-upon behavior for a specific period of time

$$MTBF_{\mathcal{T}} = \left(\sum_{i=0}^{n-1} \frac{1}{MTBF_i}\right)^{-1}$$

- Failures arise anytime
  - Stops partially or totally the execution (crash-type failures)
  - Provides incorrect results (bit errors)
- With an increase in the number of components, the system will experience a component failure every few hours or even minutes

MTBF


Mean Time Between Failures of the System (hours)
5000
4000
3000
2000
1000
0
1
10
100
1000
10000
100000
1e+06
Number of components in the system
10 000 H
100 000 H
1 000 000 H
10 000 000 H

Fault tolerance

Challenges in HPC

- HPC algorithms should be designed to:
  - expect failures: very difficult to predict all possible failures
  - take suitable actions: ensure that intensive applications run smoothly with reduced overhead
- Fault tolerant solutions are being incorporated
  - Have the ability to contain failures
  - Minimize the impact of failures
- Provide a fault tolerant environment
  - Enhance the utilization of the system at high scale
  - Ensure the failure-free execution of critical algorithms
- Hard to describe and verify the system's properties: how to simplify it?

Formal models

Coloured Petri Nets (CPN)

- Better understanding of the system
- The system ensures mathematically that it is correct
- Modelling, validating properties and synchronizing communications of parallel and distributed algorithms
- Allow for better readability and understandability

FT-TSQR Formal Model

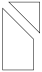
Formal model and associated verifications

- Proves it tolerates the failures
- Guarantees that the final results are correct
  - Data flow is correct
  - Each process calculates and shares its results
  - At the end, all the process have the same result

Los Tres Amigos

QR factorization: $A = QR$

- R upper triangular
- Q orthogonal



LU decomposition: $A = LU$

- L lower triangular
- U upper triangular

Cholesky factorization: $A = LL^T$

- A symmetric, positive definite
- L lower triangular

Tall and Skinny QR

Tall and Skinny QR (TSQR) Factorisation

- It calculates the QR factorisation of a tall and skinny matrix A, i.e. a matrix with m rows and n columns, $m \gg n$
- Linear algebra applications depend on the algorithm:
  - $Ax = b$
  - numerically stable: eigenvalues computation is sensitive to the accuracy of the orthogonalization

Fault-Tolerant TSQR

t processes

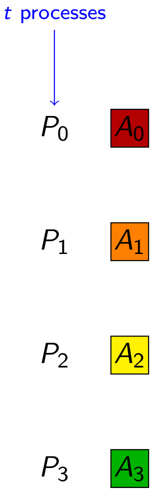
P_0

P_1

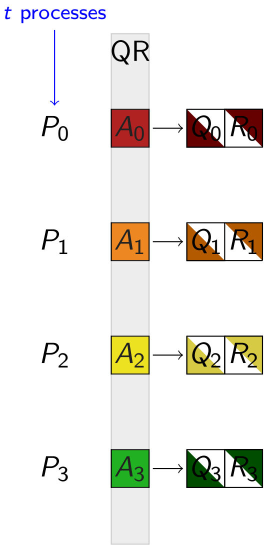
P_2

P_3

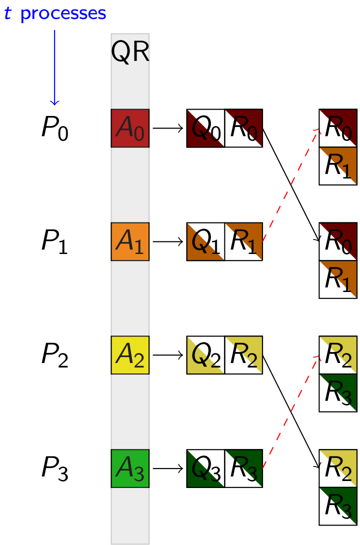
Fault-Tolerant TSQR



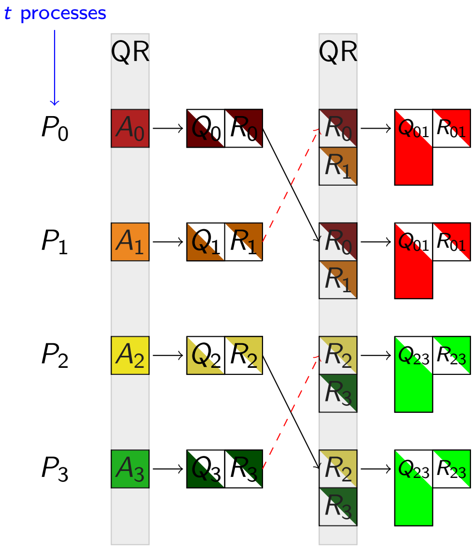
Fault-Tolerant TSQR



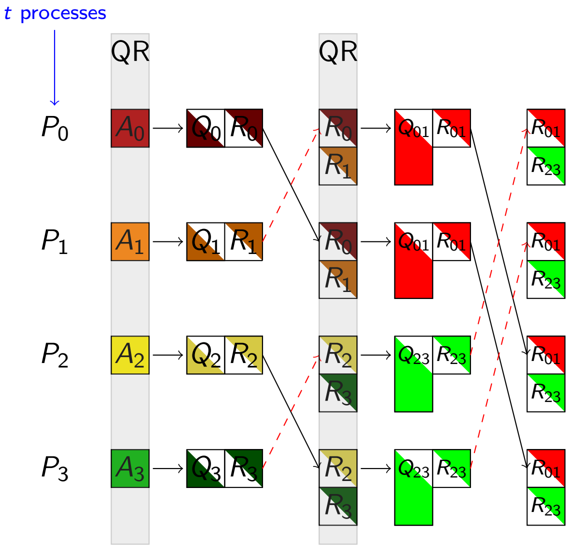
Fault-Tolerant TSQR



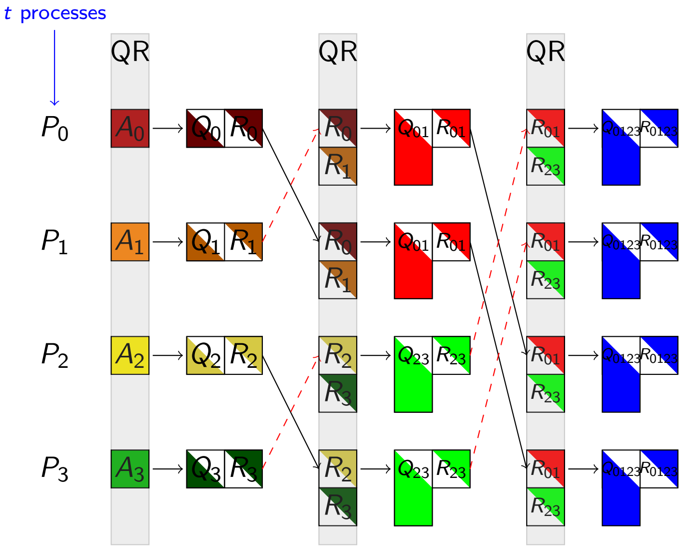
Fault-Tolerant TSQR



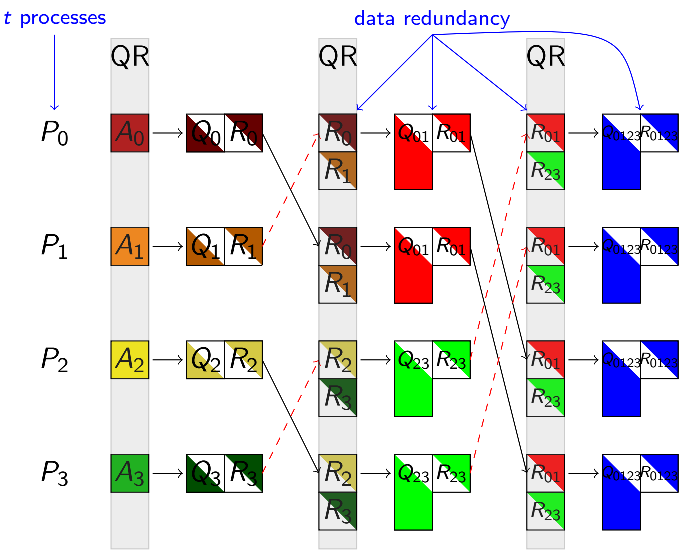
Fault-Tolerant TSQR



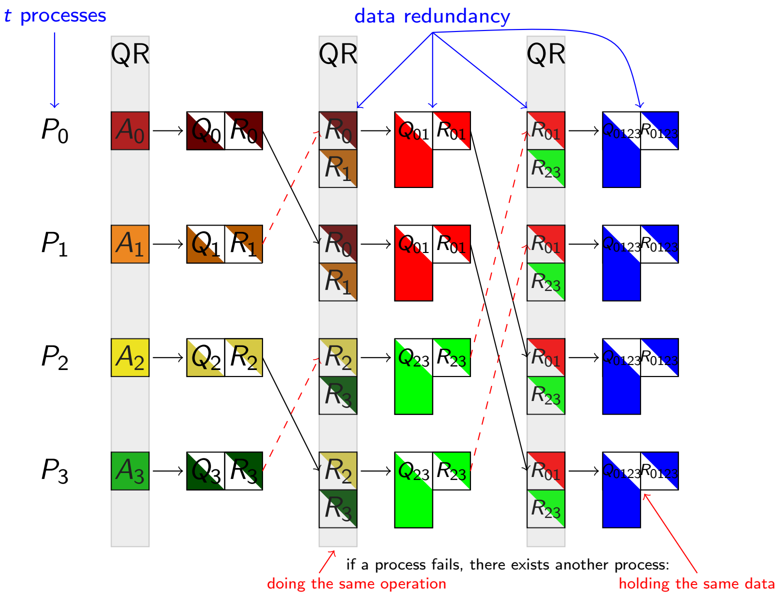
Fault-Tolerant TSQR



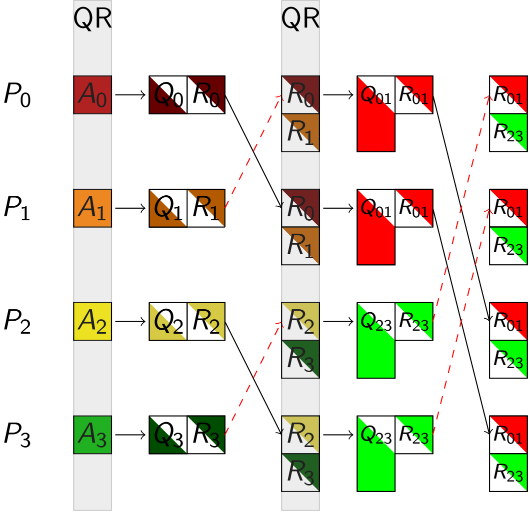
Fault-Tolerant TSQR



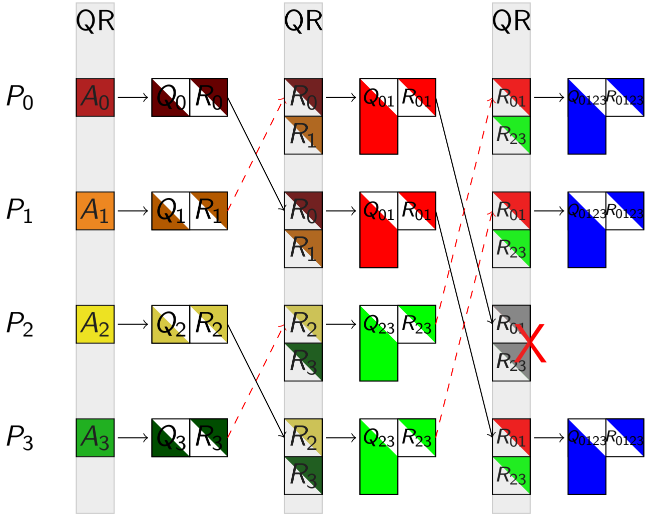
Fault-Tolerant TSQR



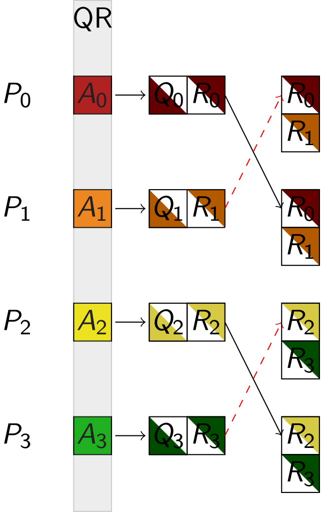
Fault-Tolerant TSQR Failure



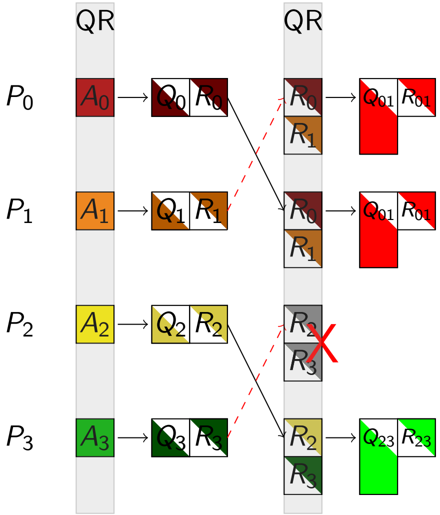
Fault-Tolerant TSQR Failure



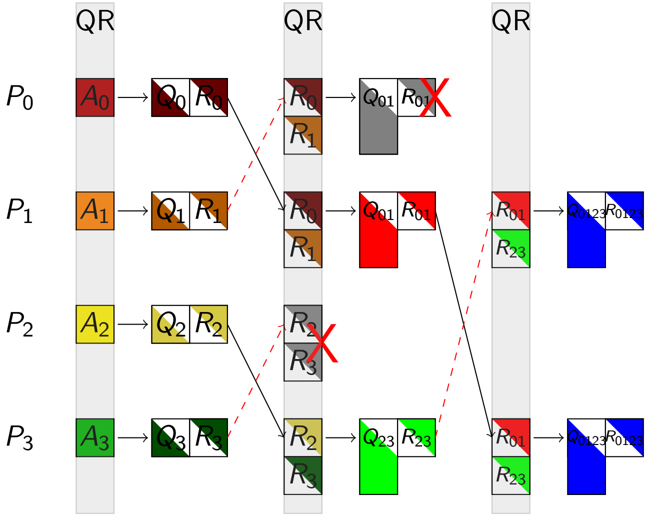
Fault-Tolerant TSQR Failure



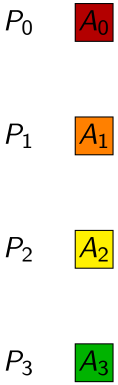
Fault-Tolerant TSQR Failure



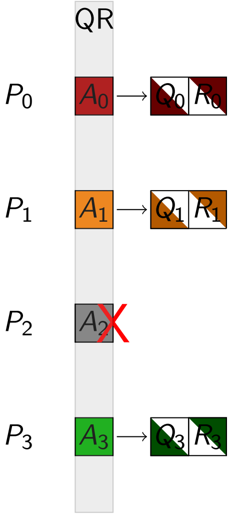
Fault-Tolerant TSQR Failure



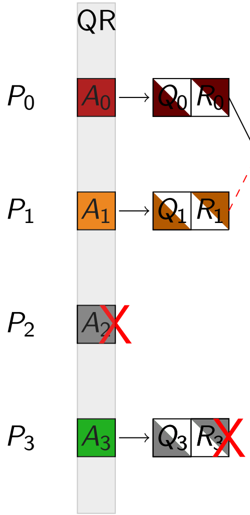
Fault-Tolerant TSQR Failure



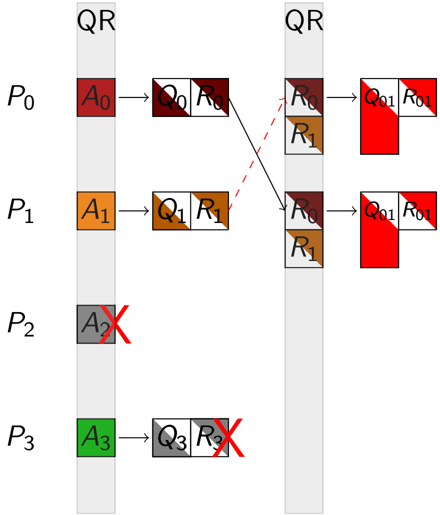
Fault-Tolerant TSQR Failure



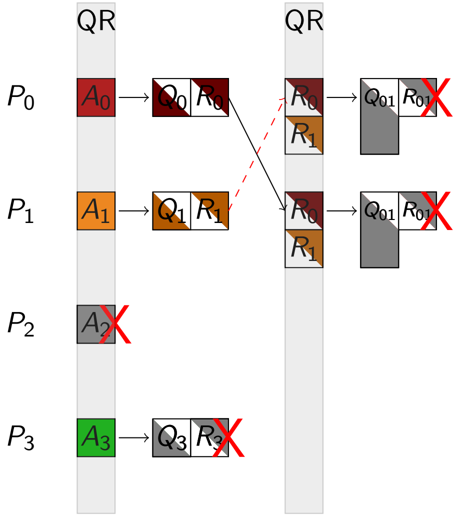
Fault-Tolerant TSQR Failure



Fault-Tolerant TSQR Failure



Fault-Tolerant TSQR Failure



The algorithm

Algorithm 1: FT-TSQR

Data: Submatrix A

$\tilde{Q}, \tilde{R} = \texttt{QR}(A)$;
s = 0 ;
while ! done() **do**
    $p_i = \texttt{myPartner}(s)$;
    $f = \texttt{sendRecv}(\tilde{R}, \tilde{R}', p_i)$;
    if FAIL == f **then**
        return;
    A = $\texttt{concatenate}(\tilde{R}, \tilde{R}')$;
    $\tilde{Q}, \tilde{R} = \texttt{QR}(A)$;
    $s = s + 1$;
/* All the surviving processes reach this point and own the final $\tilde{R}$ */
return $\tilde{R}$;

The model

$\sum_{0 \leq q < t} (q, 0, q)$

Processes

$PROC \times INT \times PROC$

contains triples (q, s, k)

q: a process number

s: the current step

k: index of the $\tilde{R}_i$ matrix

The model

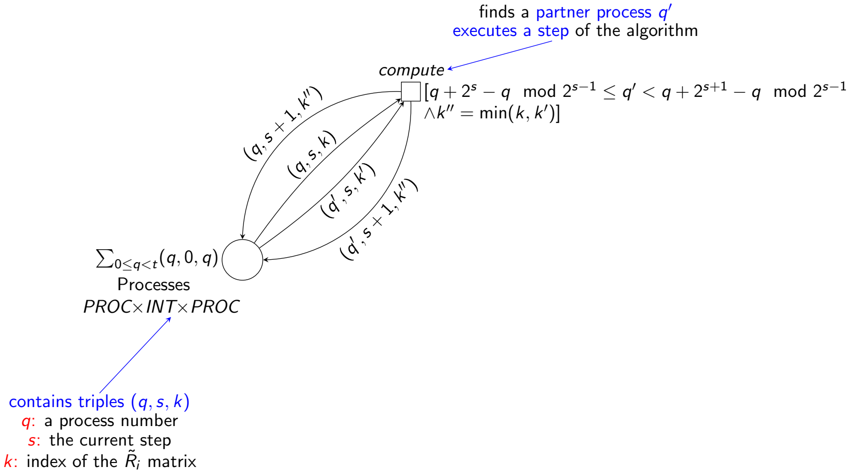
finds a partner process q'
executes a step of the algorithm

compute
$\square[q + 2^s - q \mod 2^{s-1} \leq q' < q + 2^{s+1} - q \mod 2^{s-1}$
$\wedge k'' = \min(k, k')]$

$\sum_{0 \leq q < t}(q, 0, q)$ ◯
Processes
$PROC \times INT \times PROC$

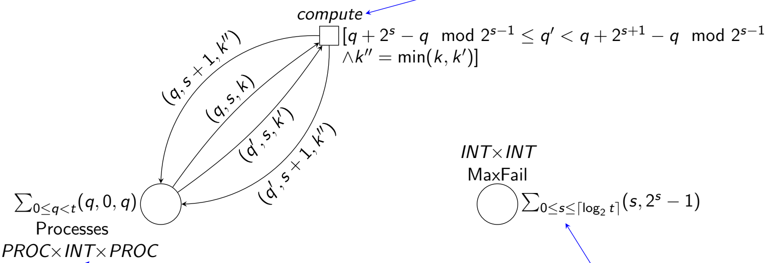
contains triples (q, s, k)
q: a process number
s: the current step
k: index of the $\tilde{R}_i$ matrix

The model


finds a partner process q'
executes a step of the algorithm
compute
$[q + 2^s - q \mod 2^{s-1} \leq q' < q + 2^{s+1} - q \mod 2^{s-1}$
$\wedge k'' = \min(k, k')]$
$(q, s+1, k'')$
(q, s, k)
(q', s, k')
$(q', s+1, k'')$
$\sum_{0 \leq q < t} (q, 0, q)$
Processes
$PROC \times INT \times PROC$
contains triples (q, s, k)
q: a process number
s: the current step
k: index of the $\tilde{R}_i$ matrix

The model

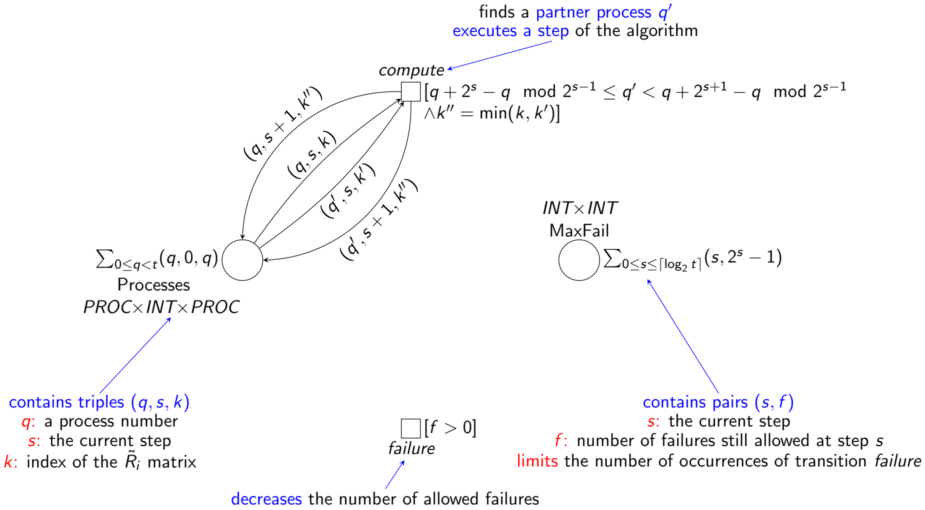
finds a partner process q'
executes a step of the algorithm



contains triples (q, s, k)
q: a process number
s: the current step
k: index of the $\tilde{R}_i$ matrix

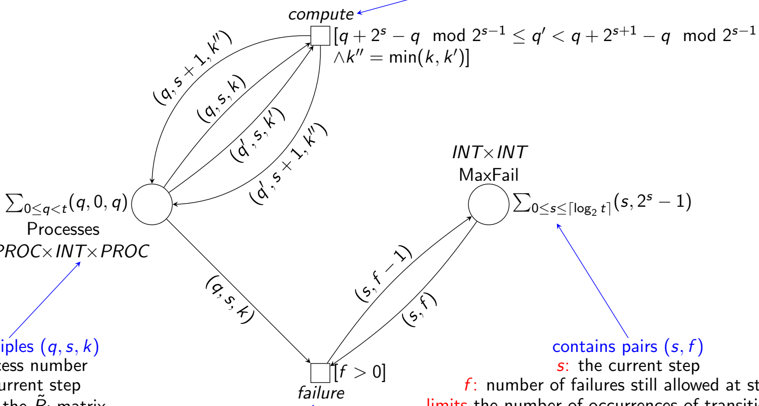
contains pairs (s, f)
s: the current step
f: number of failures still allowed at step s
limits the number of occurrences of transition *failure*

The model


finds a partner process q'
executes a step of the algorithm
compute
$[q + 2^s - q \mod 2^{s-1} \leq q' < q + 2^{s+1} - q \mod 2^{s-1}$
$\wedge k'' = \min(k, k')]$
$(q, s+1, k'')$
(q, s, k)
(q', s, k')
$(q', s+1, k'')$
$\sum_{0 \leq q < t} (q, 0, q)$
Processes
$PROC \times INT \times PROC$
$INT \times INT$
MaxFail
$\sum_{0 \leq s \leq \lceil \log_2 t \rceil} (s, 2^s - 1)$
contains triples (q, s, k)
q: a process number
s: the current step
k: index of the $\tilde{R}_i$ matrix
$[f > 0]$
failure
decreases the number of allowed failures
contains pairs (s, f)
s: the current step
f: number of failures still allowed at step s
limits the number of occurrences of transition *failure*

The model

finds a partner process q'
executes a step of the algorithm

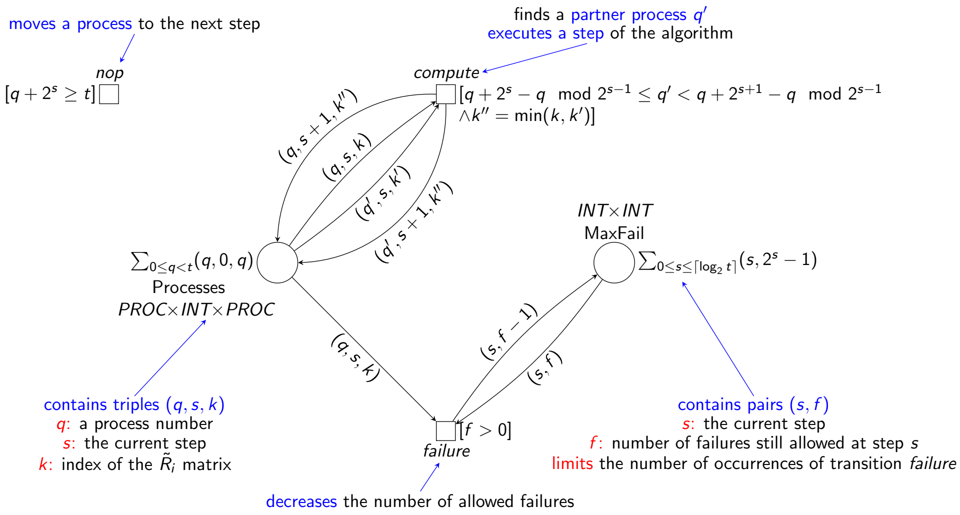


contains triples (q, s, k)
q: a process number
s: the current step
k: index of the $\tilde{R}_i$ matrix

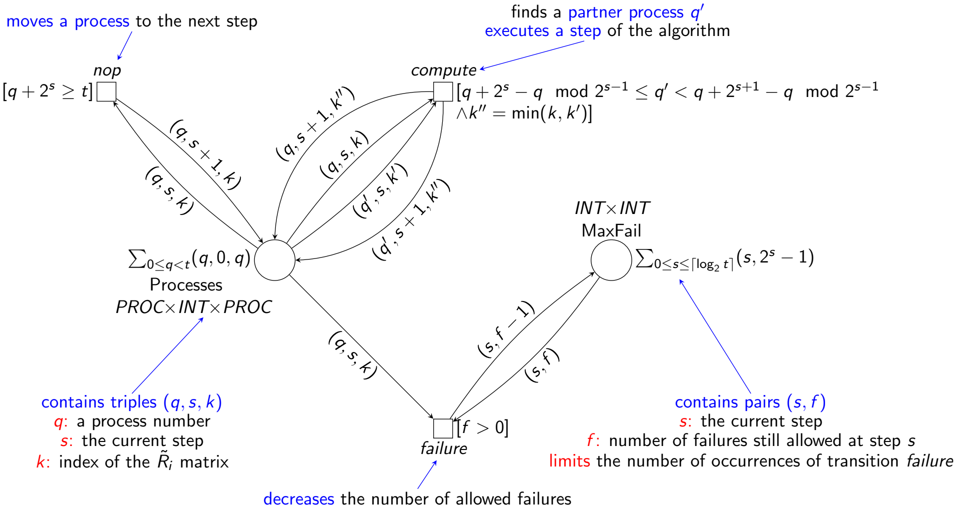
decreases the number of allowed failures

contains pairs (s, f)
s: the current step
f: number of failures still allowed at step s
limits the number of occurrences of transition *failure*

The model


moves a process to the next step
nop
$[q + 2^s \geq t]$
finds a partner process q'
executes a step of the algorithm
compute
$[q + 2^s - q \mod 2^{s-1} \leq q' < q + 2^{s+1} - q \mod 2^{s-1}$
$\wedge k'' = \min(k, k')]$
$(q, s+1, k'')$
(q, s, k)
(q', s, k')
$(q', s+1, k'')$
$\sum_{0 \leq q < t}(q, 0, q)$
Processes
$PROC \times INT \times PROC$
$INT \times INT$
MaxFail
$\sum_{0 \leq s \leq \lceil \log_2 t \rceil}(s, 2^s - 1)$
(q, s, k)
$(s, f-1)$
(s, f)
$[f > 0]$
failure
contains triples (q, s, k)
q: a process number
s: the current step
k: index of the $\tilde{R}_i$ matrix
contains pairs (s, f)
s: the current step
f: number of failures still allowed at step s
limits the number of occurrences of transition *failure*
decreases the number of allowed failures

The model


moves a process to the next step
nop
$[q + 2^s \geq t]$
finds a partner process q'
executes a step of the algorithm
compute
$[q + 2^s - q \mod 2^{s-1} \leq q' < q + 2^{s+1} - q \mod 2^{s-1}$
$\wedge k'' = \min(k, k')]$
$(q, s+1, k)$
(q, s, k)
$(q, s+1, k'')$
(q, s, k)
(q', s, k')
$(q', s+1, k'')$
$\sum_{0 \leq q < t}(q, 0, q)$
Processes
$PROC \times INT \times PROC$
$INT \times INT$
MaxFail
$\sum_{0 \leq s \leq \lceil \log_2 t \rceil}(s, 2^s - 1)$
(q, s, k)
$(s, f-1)$
(s, f)
$[f > 0]$
failure
contains triples (q, s, k)
q: a process number
s: the current step
k: index of the $\tilde{R}_i$ matrix
contains pairs (s, f)
s: the current step
f: number of failures still allowed at step s
limits the number of occurrences of transition *failure*
decreases the number of allowed failures

Properties

- The system can reach the end of the computation (prop. 1)
- The final result is unique and is the expected one (prop. 2)

- Projection functions
  - Π_x: select the xth element of a token which has a tuple value
  - $\Pi_{x,y}$: select the xth and yth elements to form a pair
- Π_x^s denotes the value of Π_x when the step number is s

Properties

Property 1

At every step $s > 0$, the system can tolerate at most $2^s - 1$ failures

Proof.

- When $s = 0$ each process performs a local computation
- When $s > 0$ transition *compute* takes the $\tilde{R}$ and $\tilde{R}'$ from two processes q and q' and produces $\tilde{R}''$ on both q and q' or transition *failure* consumes a process
  - By recursion, at each step $s > 0$, $\forall M \in [M_0 >$, it holds that:

$$|\Pi_3^s(M(Processes))| + \Pi_2^s(M(MaxFail)) = 2^s \leftarrow \text{invariant}$$

    At each step, the number of process with the same information increases by 2
  - The guard on transition *failure* ensures:

$$0 \leq \Pi_2^s(M(MaxFail)) \leq 2^s - 1$$

  - $1 \leq |\Pi_3^s(M(Processes))| \leq 2^s$: at least one process holds each intermediate $\tilde{R}$

Properties

Property 2

At the end of the computation, if the system did not suffer too many failures, at least one process holds the final R

Proof.

- From property 1, when $s > 0$: $|\Pi_3^s(M(Processes))| \geq 1$
  - For each $\tilde{R}$: $|\Pi_3^s(M(Processes))| + \Pi_2^s(M(MaxFail)) = 2^s$
  - $0 \leq \Pi_2^s(M(MaxFail)) \leq 2^s - 1$
- When $s = \log_2 t$: the algorithm reaches the final step
  - $|\Pi_3^s(M(Processes))| + \Pi_2^s(M(MaxFail)) = t$
  - Then, all non-failed processes hold the same $\tilde{R} \Rightarrow \tilde{R}$ is unique and is the final R

Conclusion and perspectives

Conclusions

- A formal model for a fault tolerant algorithm
  - How the failures are modelled
  - Design of proofs of fault tolerance properties
- Number of processes, size of the matrix: parameters of the model
  - The proof holds for any value of the parameters

Perspectives

- How to derive a general modelling and verification approach for:
  - fault tolerant algorithms?
  - square matrices?
  - the Trailing Matrix Update?
- Maximum number of errors allowed?
- Improvements
  - choosing among more partners?
  - recovery after failure: handled by a spawned node, an inactive one?

References I

Camille Coti.
Scalable, robust, fault-tolerant parallel QR factorization.
In Souheil Khaddaj, editor, *Distributed Computing and Applications to Business, Engineering and Science (DCABES), 2016 15th International Symposium on*. IEEE, 2016.

Camille Coti, Charles Lakos, and Laure Petrucci.
Formally proving and enhancing a self-stabilising algorithm.
In Lawrence Cabac, Lars Michael Kristensen, and Heiko Rölke, editors, *Petri Nets and Software Engineering. International Workshop, PNSE'16, Torun, Poland. Proceedings*, volume 1591 of *CEUR Workshop Proceedings*, pages 255–274. CEUR-WS.org, 2016.

James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou.
Communication-avoiding parallel and sequential QR factorizations.
CoRR, abs/0806.2159, 2008.

C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita, E. Rodriguez, and F. Cappello.
Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant mpi.
In *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 18–18, 2006.

C. Coti.
Exploiting redundant computation in communication-avoiding algorithms for algorithm-based fault tolerance.
In *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, pages 214–219, 2016.

C. Coti.
Scalable, robust, fault-tolerant parallel qr factorization.
In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pages 626–633, 2016.

References II

Kurt Jensen and Lars M. Kristensen.
Coloured Petri Nets: Modelling and Validation of Concurrent Systems.
Springer Publishing Company, Incorporated, 1st edition, 2009.

Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir.
Toward exascale resilience: 2014 update.
Supercomputing frontiers and innovations, 1(1):5–28, 2014.

Jack Dongarra et al.
The international exascale software project roadmap.
International Journal of High Performance Computing Applications, 2011.

Franck Cappello.
Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities.
International Journal of High Performance Computing Applications, 23(3):212–226, 2009.

William Gropp and Marc Snir.
Programming for exascale computers.
Computing in Science & Engineering, 15:27, 2013.

Rajeev Thakur, Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Torsten Hoefler, Sameer Kumar, Ewing Lusk, and Jesper Larsson Träff.
MPI at exascale.
In *Procceedings of SciDAC 2010*, Jun. 2010.

John Shalf, Sudip Dosanjh, and John Morrison.
Exascale computing technology challenges.
In *International Conference on High Performance Computing for Computational Science*, pages 1–25. Springer, 2010.

References III

Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer.
Lessons learned from the analysis of system failures at petascale: The case of Blue Waters.
In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 610–621. IEEE, 2014.

Reliability challenges in large systems.
Future Generation Computer Systems, 22(3):293 – 302, 2006.

Wesley Bland, Aurelien Bouteiller, Thomas Hérault, Joshua Hursey, George Bosilca, and Jack J. Dongarra.
An evaluation of user-level failure mitigation support in MPI.
Computing, 95(12):1171–1184, 2013.

Graham E Fagg and Jack J Dongarra.
FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world.
In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 346–353. Springer, 2000.