

Linear algebra kernels on petascale/exascale platforms: scheduling issues

Yves Robert

Ecole Normale Supérieure de Lyon
& Institut Universitaire de France
& University of Tennessee Knoxville

<http://graal.ens-lyon.fr/~yrobert/slides/inria-illinois.pdf.gz>

November 22, 2011

Co-authors

QR factorization

- Henc Bouwmeester, Julien Langou, Univ. Colorado Denver
- Jack Dongarra, Mathieu Faverge, Thomas Hérault,
Univ. Tennessee Knoxville
- Mathias Jacquelin, INRIA & ENS Lyon

Resilience/Checkpointing

- Marin Bougeret, Frédéric Vivien, Dounia Zaidouni,
INRIA & ENS Lyon
- Franck Cappello, UIUC-Inria joint lab
- Henri Casanova, Univ. Hawai'i

Exascale platforms

- Hierarchical with massively parallel nodes
 - 10^5 or 10^6 nodes
 - Each node equipped with 10^4 or 10^3 cores
 - Multi-level interconnect rather than flat layout?
- Failure-prone

MTTF of one node = 1 year
 \Rightarrow MTTF of platform with 10^6 nodes = 30sec

Better algorithms?

- Faster iron **and** better algorithms.
- Speedup(faster iron) \times Speedup(better algorithms)
- Linear programming:
 - 82 years to solve in 1988
 - roughly one minute in 2003
- Improvement by a factor of roughly 43 million:
 - 1,000 due to increased processor speed
 - 43,000 due to algorithms!

Linear algebra kernels

- No new matrix-product nor novel LU/QR factorization method
- Continuous effort to catch up with architectural changes
- Sometimes painful 😞 but often insightful 😊

Invent new versions of well-established algorithms
to squeeze the most out of the new big iron

Application scaling

- **Strong scaling:** apply more resources to same problem size
⇒ get results faster
- **Weak scaling:** solve larger problem

	Power rule	Increase in work	Increase in memory
2D scaling	$2/3$	$1000\times$	$100\times$
3D scaling	$3/4$	$1000\times$	$178\times$

Application scaling

- **Strong scaling:** apply more resources to same problem size
⇒ get results faster
- **Weak scaling:** solve larger problem

Exascale

\neq Petascale $\times 1000$

2D scaling
3D scaling

scaling memory

10x
178x

Platform scaling

Exascale \neq Petascale $\times 1000$

Algorithmic challenges

- Memory wall
- Difficult to maintain desired byte-to-flop ratio
- Locality: data motion carries highest energy cost
- Massive parallelism at node level

Platform scaling

Exascale \neq Petascale $\times 1000$

Hierarchical algorithmic model

- 1 Portable expression of parallelism and locality
- 2 Distribution and co-location of task and data
 \Rightarrow More than “just” mixing threads and MPI!
- 3 Asynchronous dynamic parallelism
- 4 Resilient algorithms

Outline

- 1 Towards an exascale QR factorization algorithm
 - Shared-memory algorithm (single multicore node)
 - Hierarchical algorithm (many multicore nodes)
- 2 Checkpointing
 - Steady-state throughout of large platforms
 - Exascale tightly-coupled job
- 3 Conclusion

Outline

- 1 Towards an exascale QR factorization algorithm
 - Shared-memory algorithm (single multicore node)
 - Hierarchical algorithm (many multicore nodes)
- 2 Checkpointing
 - Steady-state throughout of large platforms
 - Exascale tightly-coupled job
- 3 Conclusion

Outline

- 1 Towards an exascale QR factorization algorithm
 - Shared-memory algorithm (single multicore node)
 - Hierarchical algorithm (many multicore nodes)
- 2 Checkpointing
 - Steady-state throughout of large platforms
 - Exascale tightly-coupled job
- 3 Conclusion

Motivation

- Solving dense linear systems
- LU factorization not always stable enough
- QR factorization twice as costly but always robust
- Works for rectangular matrices (least squares)
- Use blocked algorithms (BLAS3 level)
- Use tile algorithms (multicores)

Generic QR algorithm

Algorithm 1: Generic QR algorithm for a tiled $p \times q$ matrix.

for $k = 1$ *to* $\min(p, q)$ **do**

for $i = k + 1$ *to* p **do**
 $\text{elim}(i, \text{piv}(i, k), k)$

- k : panel index
- **orthogonal transformation** to zero out tile (i, k)

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times & \times \\ & & & \times & \times & \times \end{bmatrix}$$

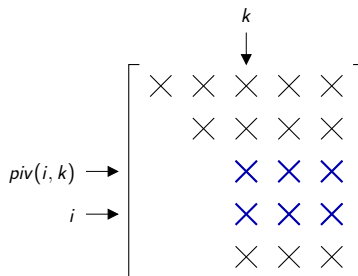
Generic QR algorithm

Algorithm 1: Generic QR algorithm for a tiled $p \times q$ matrix.

```

for  $k = 1$  to  $\min(p, q)$  do
  for  $i = k + 1$  to  $p$  do
     $\text{elim}(i, \text{piv}(i, k), k)$ 
  
```

- k : panel index
- **orthogonal transformation** to zero out tile (i, k)



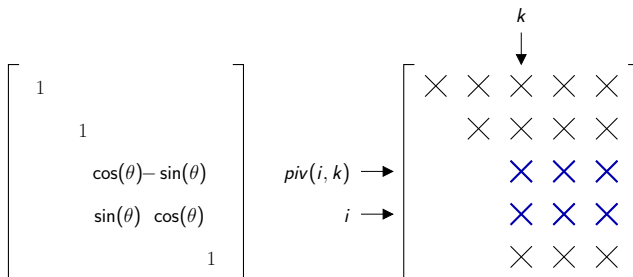
Generic QR algorithm

Algorithm 1: Generic QR algorithm for a tiled $p \times q$ matrix.

```

for  $k = 1$  to  $\min(p, q)$  do
  for  $i = k + 1$  to  $p$  do
     $\text{elim}(i, \text{piv}(i, k), k)$ 
  
```

- k : panel index
- **orthogonal transformation** to zero out tile (i, k)



Generic QR algorithm

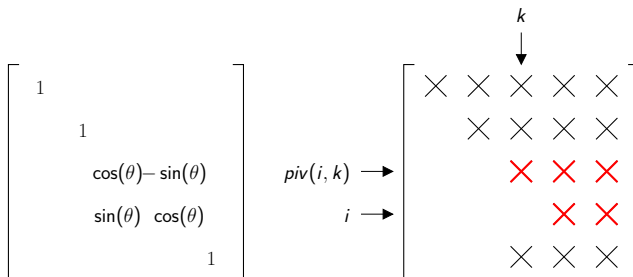
Algorithm 1: Generic QR algorithm for a tiled $p \times q$ matrix.

for $k = 1$ **to** $\min(p, q)$ **do**

for $i = k + 1$ **to** p **do**

$\text{elim}(i, \text{piv}(i, k), k)$

- k : panel index
- **orthogonal transformation** to zero out tile (i, k)



Generic QR algorithm

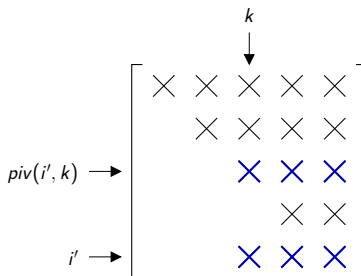
Algorithm 1: Generic QR algorithm for a tiled $p \times q$ matrix.

for $k = 1$ **to** $\min(p, q)$ **do**

for $i = k + 1$ **to** p **do**

$\text{elim}(i, \text{piv}(i, k), k)$

- k : panel index
- **orthogonal transformation** to zero out tile (i, k)



Generic QR algorithm

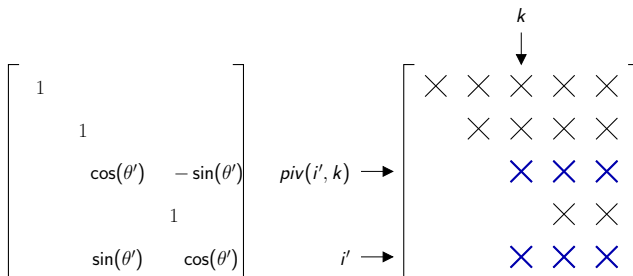
Algorithm 1: Generic QR algorithm for a tiled $p \times q$ matrix.

for $k = 1$ **to** $\min(p, q)$ **do**

for $i = k + 1$ **to** p **do**

$\text{elim}(i, \text{piv}(i, k), k)$

- k : panel index
- **orthogonal transformation** to zero out tile (i, k)



Generic QR algorithm

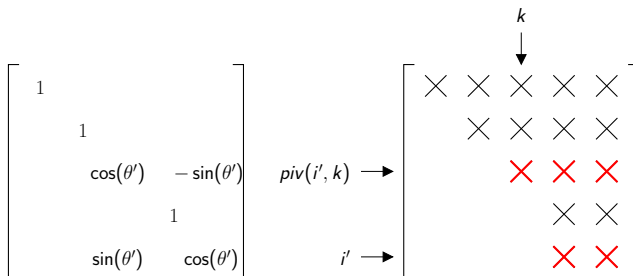
Algorithm 1: Generic QR algorithm for a tiled $p \times q$ matrix.

for $k = 1$ **to** $\min(p, q)$ **do**

for $i = k + 1$ **to** p **do**

$\text{elim}(i, \text{piv}(i, k), k)$

- k : panel index
- **orthogonal transformation** to zero out tile (i, k)



Kernels for orthogonal transformations

Algorithm 2: Elimination $\text{elim}(i, \text{piv}(i, k), k)$
via *TS* kernels.

```

GEQRT( $\text{piv}(i, k), k$ )
for  $j = k + 1$  to  $q$  do
  | UNMQR( $\text{piv}(i, k), k, j$ )
TSQRT( $i, \text{piv}(i, k), k$ )
for  $j = k + 1$  to  $q$  do
  | TSMQR( $i, \text{piv}(i, k), k, j$ )

```

TS– Triangle on top of square

Algorithm 3: Elimination $\text{elim}(i, \text{piv}(i, k), k)$
via *TT* kernels.

```

GEQRT( $\text{piv}(i, k), k$ )
GEQRT( $i, k$ )
for  $j = k + 1$  to  $q$  do
  | UNMQR( $\text{piv}(i, k), k, j$ )
  | UNMQR( $i, k, j$ )
TTQRT( $i, \text{piv}(i, k), k$ )
for  $j = k + 1$  to  $q$  do
  | TTMQR( $i, \text{piv}(i, k), k, j$ )

```

Triangle on top of triangle

Kernels for orthogonal transformations

Algorithm 2: Elimination $elim(i, piv(i, k), k)$
via TS kernels.

$$\overline{GEQRT}(\text{piv}(i, k), k)$$

```

for  $j = k + 1$  to  $q$  do

```

$$UNMQR(piv(i, k), k, j)$$
$$TSQRT(i, piv(i, k), k)$$

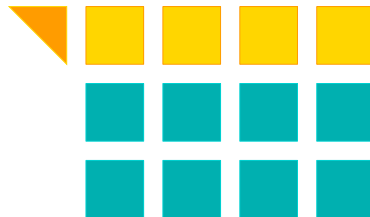
```

for j = k + 1 to q do

```

 $TSMQR(i, piv(i, k), k, j)$

TS– Triangle on top of square



Kernels for orthogonal transformations

Algorithm 2: Elimination $elim(i, piv(i, k), k)$
via TS kernels.

 $GEQRT(piv(i, k), k)$

```

for  $j = k + 1$  to  $q$  do

```

$$UNMQR(piv(i, k), k, j)$$
$$TSQRT(i, piv(i, k), k)$$

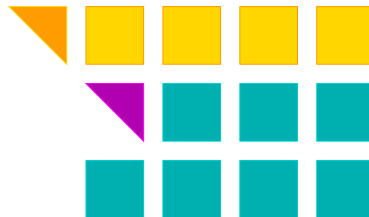
```

for j = k + 1 to q do

```

$$TSMQR(i, piv(i, k), k, j)$$

TS– Triangle on top of square



Kernels for orthogonal transformations

Algorithm 2: Elimination $elim(i, piv(i, k), k)$
via TS kernels.

$$GEQRT(piv(i, k), k)$$
for $j = k + 1$ **to** q **do** $UNMQR(piv(i, k), k, j)$
$$TSQRT(i, piv(i, k), k)$$

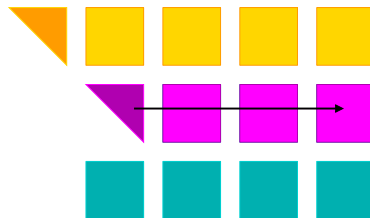
```

for j = k + 1 to q do

```

 $TSMQR(i, piv(i, k), k, j)$

TS– Triangle on top of square



Kernels for orthogonal transformations

Algorithm 2: Elimination $elim(i, piv(i, k), k)$
via TS kernels.

$$\overline{GEQRT(piv(i, k), k)}$$

```

for  $j = k + 1$  to  $q$  do

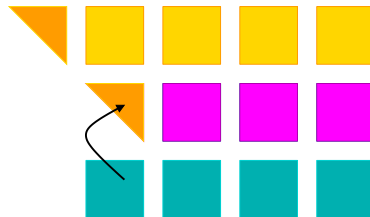
```

 $UNMQR(piv(i, k), k, j)$
$$TSQRT(i, piv(i, k), k)$$

```
for j = k + 1 to q do
```

$$TSMQR(i, piv(i, k), k, j)$$

TS– Triangle on top of square



Kernels for orthogonal transformations

Algorithm 2: Elimination $elim(i, piv(i, k), k)$
via TS kernels.

$$GEQRT(piv(i, k), k)$$

```

for  $j = k + 1$  to  $q$  do

```

$$UNMQR(piv(i, k), k, j)$$
$$TSQRT(i, piv(i, k), k)$$

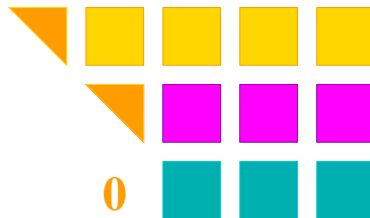
```

for j = k + 1 to q do

```

$$TSMQR(i, piv(i, k), k, j)$$

TS– Triangle on top of square



Kernels for orthogonal transformations

Algorithm 2: Elimination $elim(i, piv(i, k), k)$
via TS kernels.

$$\overline{GEQRT}(\text{piv}(i, k), k)$$

```

for  $j = k + 1$  to  $q$  do

```

$$UNMQR(piv(i, k), k, j)$$
$$TSQRT(i, piv(i, k), k)$$

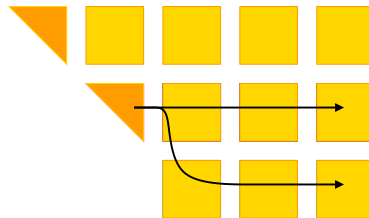
```

for j = k + 1 to q do

```

 $TSMQR(i, piv(i, k), k, j)$

TS– Triangle on top of square



Kernels for orthogonal transformations

Algorithm 3: Elimination $\text{elim}(i, \text{piv}(i, k), k)$
via TT kernels.

$\text{GEQRT}(\text{piv}(i, k), k)$

$\text{GEQRT}(i, k)$

for $j = k + 1$ **to** q **do**

$\text{UNMQR}(\text{piv}(i, k), k, j)$

$\text{UNMQR}(i, k, j)$

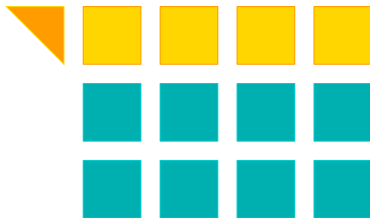
$\text{TTQRT}(i, \text{piv}(i, k), k)$

for $j = k + 1$ **to** q **do**

$\text{TTMQR}(i, \text{piv}(i, k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations



Algorithm 3: Elimination $\text{elim}(i, \text{piv}(i, k), k)$
via TT kernels.

$\text{GEQRT}(\text{piv}(i, k), k)$

$\text{GEQRT}(i, k)$

for $j = k + 1$ **to** q **do**

$\text{UNMQR}(\text{piv}(i, k), k, j)$

$\text{UNMQR}(i, k, j)$

$\text{TTQRT}(i, \text{piv}(i, k), k)$

for $j = k + 1$ **to** q **do**

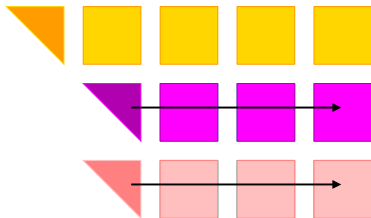
$\text{TTMQR}(i, \text{piv}(i, k), k, j)$

Triangle on top of triangle

A decorative graphic consisting of a grid of colored squares and triangles. The top row has five yellow squares, with the first one being a triangle. The second row has a purple triangle followed by three teal squares. The third row has a pink triangle followed by three teal squares.

Triangle on top of triangle

Kernels for orthogonal transformations



Algorithm 3: Elimination $\text{elim}(i, \text{piv}(i, k), k)$
via TT kernels.

$\text{GEQRT}(\text{piv}(i, k), k)$

$\text{GEQRT}(i, k)$

for $j = k + 1$ **to** q **do**

$\text{UNMQR}(\text{piv}(i, k), k, j)$

$\text{UNMQR}(i, k, j)$

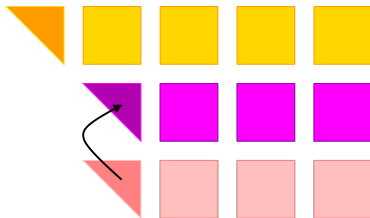
$\text{TTQRT}(i, \text{piv}(i, k), k)$

for $j = k + 1$ **to** q **do**

$\text{TTMQR}(i, \text{piv}(i, k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations



Algorithm 3: Elimination $\text{elim}(i, \text{piv}(i, k), k)$
via TT kernels.

$\text{GEQRT}(\text{piv}(i, k), k)$

$\text{GEQRT}(i, k)$

for $j = k + 1$ **to** q **do**

$\text{UNMQR}(\text{piv}(i, k), k, j)$

$\text{UNMQR}(i, k, j)$

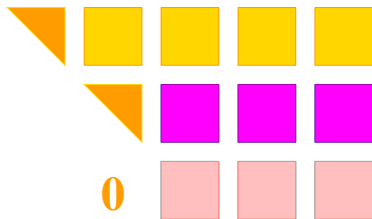
$\text{TTQRT}(i, \text{piv}(i, k), k)$

for $j = k + 1$ **to** q **do**

$\text{TTMQR}(i, \text{piv}(i, k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations



Algorithm 3: Elimination $\text{elim}(i, \text{piv}(i, k), k)$
via TT kernels.

$\text{GEQRT}(\text{piv}(i, k), k)$

$\text{GEQRT}(i, k)$

for $j = k + 1$ **to** q **do**

$\text{UNMQR}(\text{piv}(i, k), k, j)$

$\text{UNMQR}(i, k, j)$

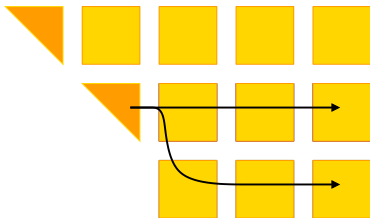
$\text{TTQRT}(i, \text{piv}(i, k), k)$

for $j = k + 1$ **to** q **do**

$\text{TTMQR}(i, \text{piv}(i, k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations



Algorithm 3: Elimination $\text{elim}(i, \text{piv}(i, k), k)$
via TT kernels.

$\text{GEQRT}(\text{piv}(i, k), k)$

$\text{GEQRT}(i, k)$

for $j = k + 1$ **to** q **do**

$\text{UNMQR}(\text{piv}(i, k), k, j)$

$\text{UNMQR}(i, k, j)$

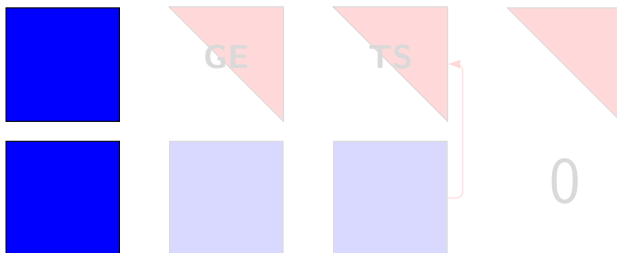
$\text{TTQRT}(i, \text{piv}(i, k), k)$

for $j = k + 1$ **to** q **do**

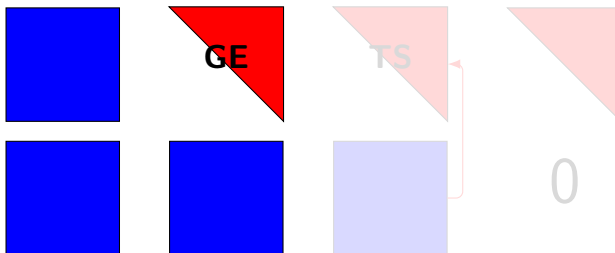
$\text{TTMQR}(i, \text{piv}(i, k), k, j)$

Triangle on top of triangle

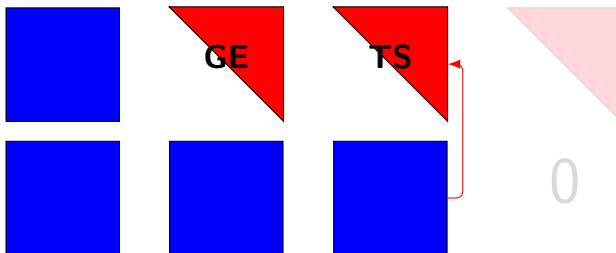
TS kernels



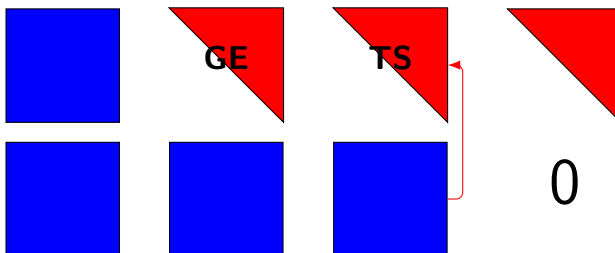
TS kernels



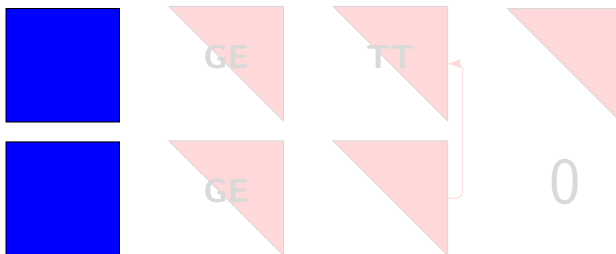
TS kernels



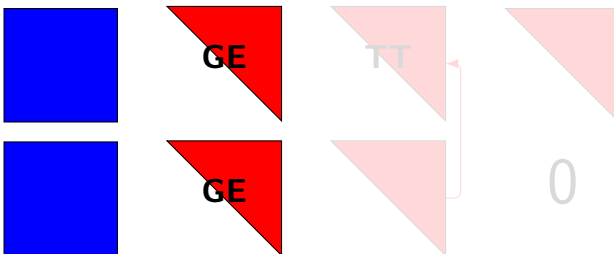
TS kernels



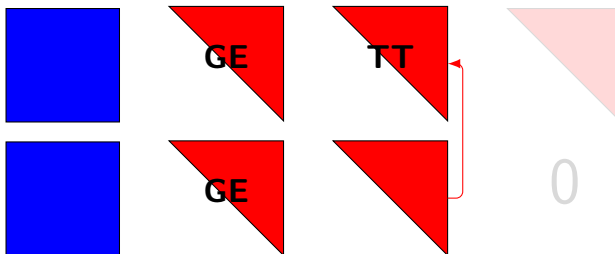
TT kernels



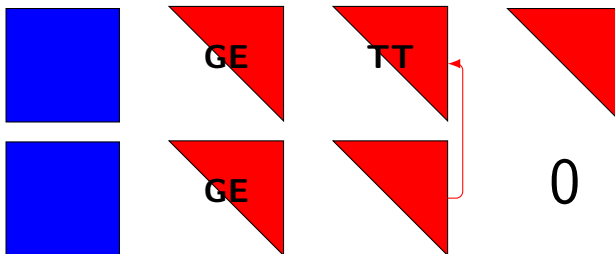
TT kernels



TT kernels



TT kernels



TS vs. TT Kernels

- **TS** *Triangle on top of square*
 - more data locality 😊
 - efficient PLASMA implementation 😊
 - only a single killer per panel 😞
- **TT** *Triangle on top of triangle*
 - several killers per panel 😊
 - more parallelism 😊
 - less efficient implementation 😞

TS Flat Tree in action (15×4 matrix)



TS Flat Tree in action (15×4 matrix)



○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



○○○○○○●○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



TS Flat Tree in action (15×4 matrix)



○○○○○○●○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



TS Flat Tree in action (15×4 matrix)



○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



TS Flat Tree in action (15×4 matrix)



○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



TS Flat Tree in action (15×4 matrix)



TS Flat Tree in action (15×4 matrix)



○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○

TS Flat Tree in action (15×4 matrix)



○○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○○○

TT Binary Tree in action (15×4 matrix)



○○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○○○

TT Binary Tree in action (15×4 matrix)



○○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○○○

TT Binary Tree in action (15×4 matrix)



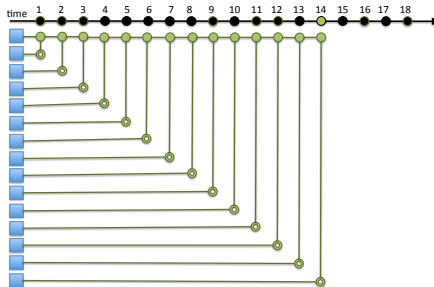
○○○○○○○○●○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○○○

TT Binary Tree in action (15×4 matrix)

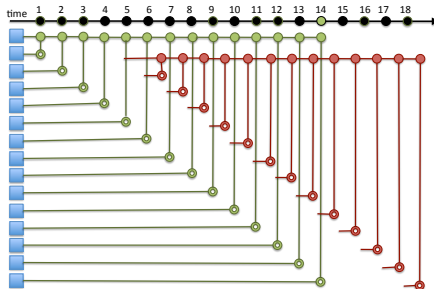


How well do you pipeline? **FlatTree**



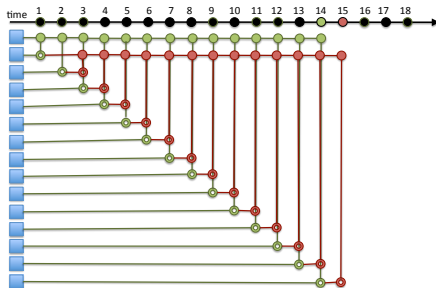
- **FLATTREE:**
1 panel: 14 steps
4 panels: 17 steps
- **BINARYTREE:**
1 panel: 4 steps
4 panels: 16 steps
- **FIBONACCI:**
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline?



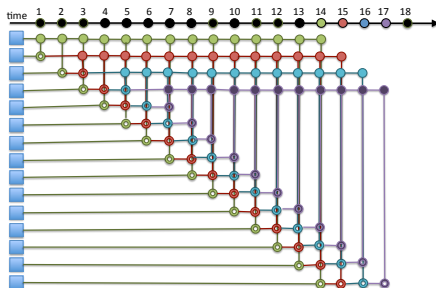
- **FLATTREE:**
1 panel: 14 steps
4 panels: 17 steps
- **BINARYTREE:**
1 panel: 4 steps
4 panels: 16 steps
- **FIBONACCI:**
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline?



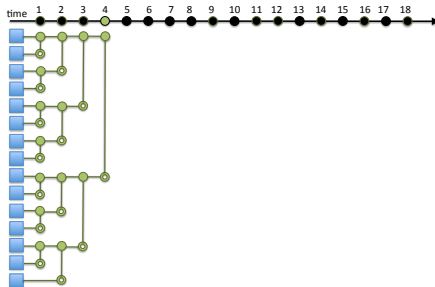
- FLATTREE:
 - 1 panel: 14 steps
 - 4 panels: 17 steps
- BINARYTREE:
 - 1 panel: 4 steps
 - 4 panels: 16 steps
- FIBONACCI:
 - 1 step: 5 steps
 - 4 panels: 16 steps

How well do you pipeline? **FlatTree**



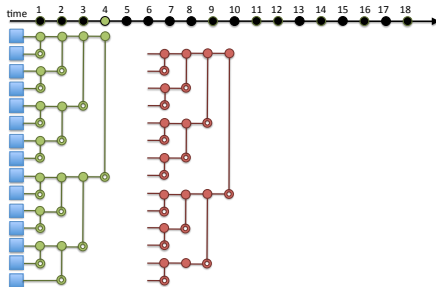
- **FLATTREE:**
1 panel: 14 steps
4 panels: 17 steps
- **BINARYTREE:**
1 panel: 4 steps
4 panels: 16 steps
- **FIBONACCI:**
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **BinaryTree**



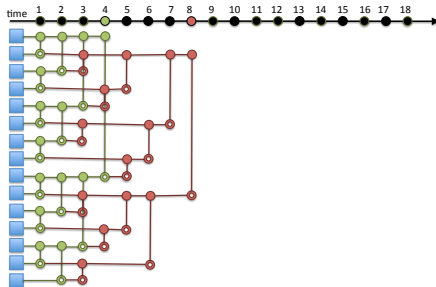
- **FLATTREE:**
1 panel: 14 steps
4 panels: 17 steps
- **BINARYTREE:**
1 panel: 4 steps
4 panels: 16 steps
- **FIBONACCI:**
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **BinaryTree**



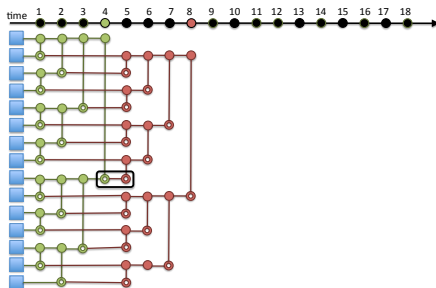
- **FLATTREE:**
1 panel: 14 steps
4 panels: 17 steps
- **BINARYTREE:**
1 panel: 4 steps
4 panels: 16 steps
- **FIBONACCI:**
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **BinaryTree**



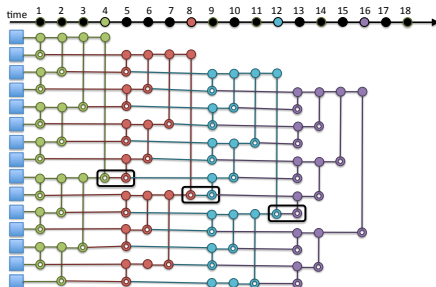
- **FLATTREE:**
1 panel: 14 steps
4 panels: 17 steps
- **BINARYTREE:**
1 panel: 4 steps
4 panels: 16 steps
- **FIBONACCI:**
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **BinaryTree**



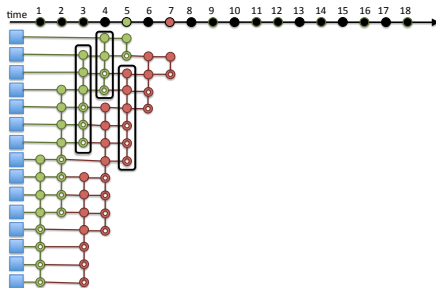
- **FLATTREE:**
1 panel: 14 steps
4 panels: 17 steps
- **BINARYTREE:**
1 panel: 4 steps
4 panels: 16 steps
- **FIBONACCI:**
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **BinaryTree**



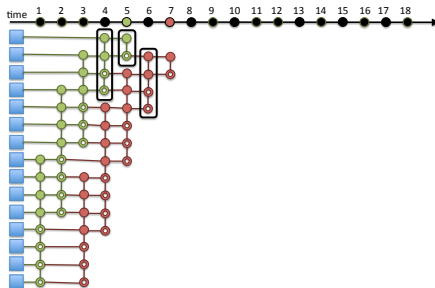
- **FLATTree:**
1 panel: 14 steps
4 panels: 17 steps
- **BINARYTree:**
1 panel: 4 steps
4 panels: 16 steps
- **FIBONACCI:**
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **Fibonacci**



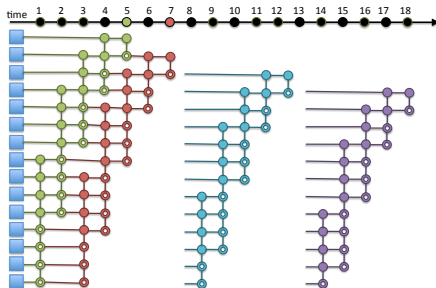
- FLATTREE:
1 panel: 14 steps
4 panels: 17 steps
- BINARYTREE:
1 panel: 4 steps
4 panels: 16 steps
- FIBONACCI:
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **Fibonacci**



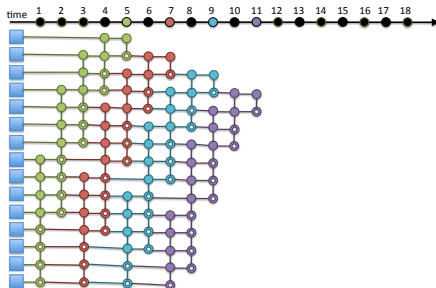
- FLATTREE:
1 panel: 14 steps
4 panels: 17 steps
- BINARYTREE:
1 panel: 4 steps
4 panels: 16 steps
- FIBONACCI:
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **Fibonacci**



- FLAT TREE:
1 panel: 14 steps
4 panels: 17 steps
- BINARY TREE:
1 panel: 4 steps
4 panels: 16 steps
- FIBONACCI:
1 step: 5 steps
4 panels: 16 steps

How well do you pipeline? **Fibonacci**



- FLAT TREE:
1 panel: 14 steps
4 panels: 17 steps
- BINARY TREE:
1 panel: 4 steps
4 panels: 16 steps
- FIBONACCI:
1 step: 5 steps
4 panels: 16 steps

Outline

- 1 Towards an exascale QR factorization algorithm
 - Shared-memory algorithm (single multicore node)
 - Hierarchical algorithm (many multicore nodes)
- 2 Checkpointing
 - Steady-state throughout of large platforms
 - Exascale tightly-coupled job
- 3 Conclusion

Communication-avoiding algorithms

Several existing solutions for tall & skinny matrices.

Main ideas

- Intra-nodes: FLAT TREE using TS kernels
⇒ efficiency of the tile algorithm locally
- Inter-nodes: BINARY TREE using TT kernels
⇒ *communication-avoiding*, introduces parallelism in communications

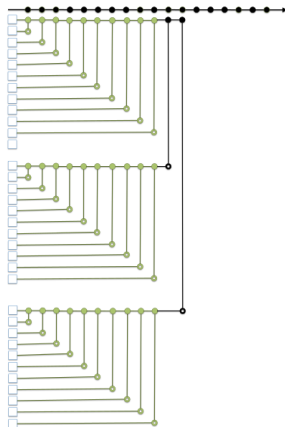
Problem

Not enough parallelism within each node

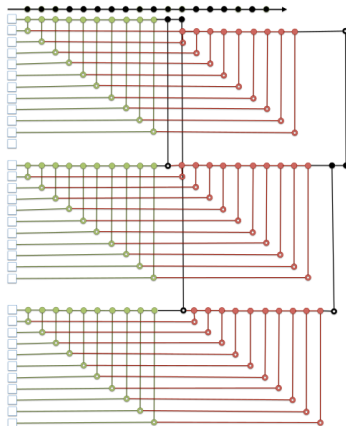
Hierarchical QR: design principles

- Domains of a tiles, and use TS kernels within domains
⇒ Arithmetic efficiency
- Intra-node reduction trees within nodes (TT kernels)
⇒ Communication avoiding

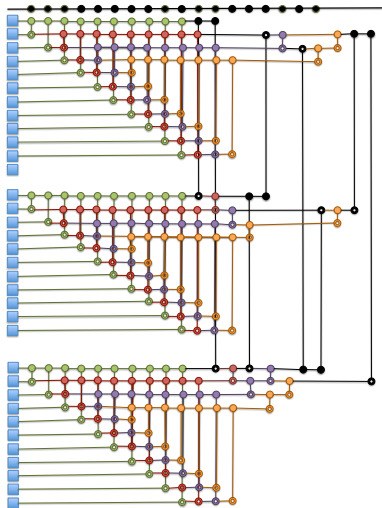
BINARYTREE pipelining



BINARYTREE pipelining



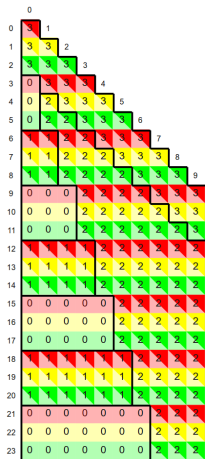
BINARYTREE pipelining **with domino**



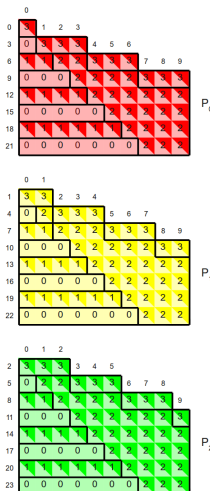
Hierarchical QR: design principles

- Domains of a tiles, and use TS kernels within domains
⇒ Arithmetic efficiency
- Intra-node reduction trees within nodes (TT kernels)
⇒ Communication avoiding
- Inter-node reduction trees of size p , across nodes (TT kernels)
⇒ Pipelining with domino-like effect
- 2D cyclic distribution of tiles along virtual $p \times q$ cluster grid
⇒ Load-balancing

Hierarchical QR: layout



Global view



Local view

Legend

 P_0 P_1 P_2

0 local TS

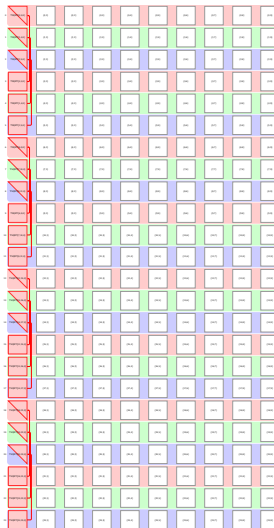
1 local TT

2 domino

3 global tree

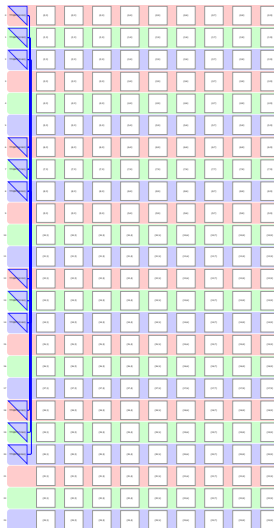
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



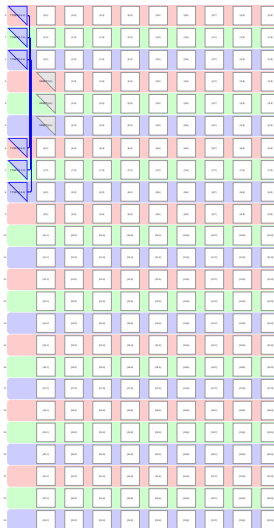
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



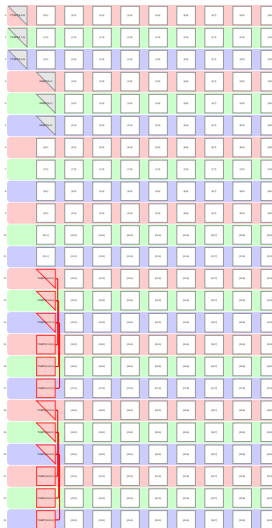
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



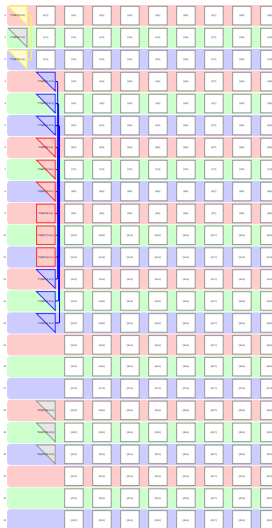
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



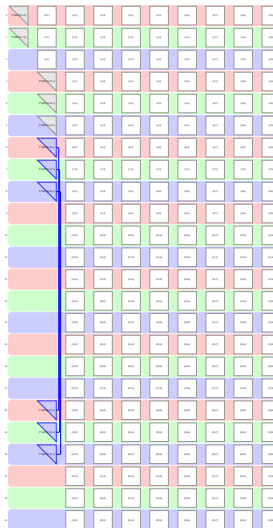
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



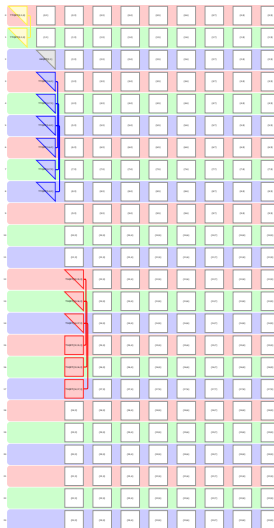
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



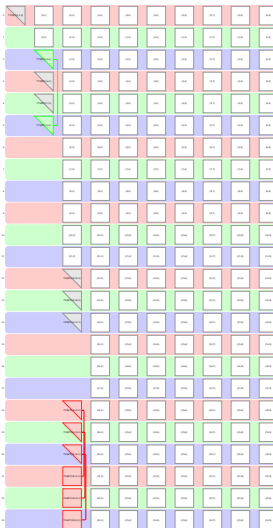
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



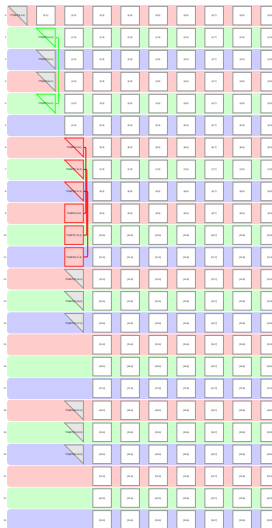
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



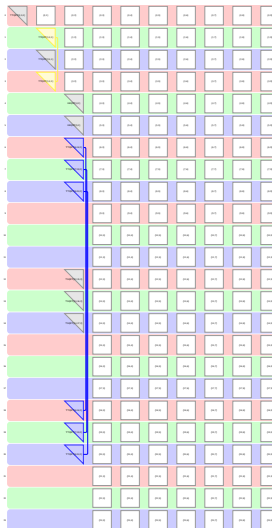
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



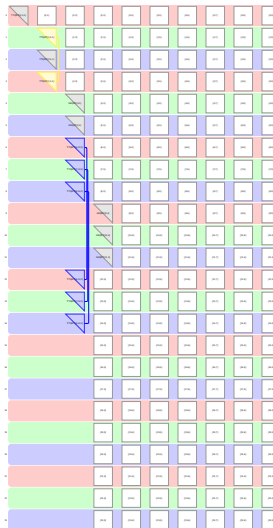
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



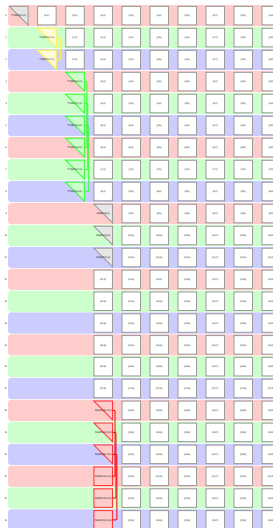
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



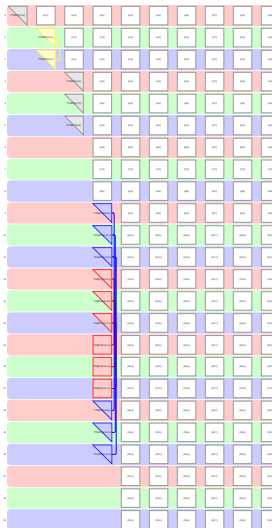
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



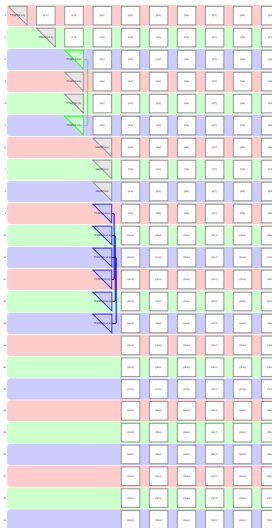
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



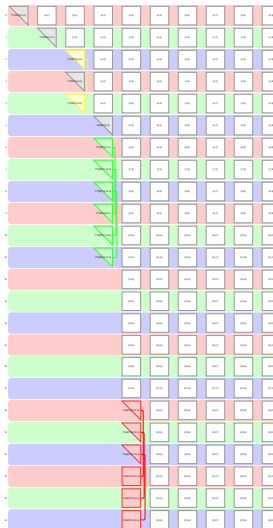
Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci



Hierarchical QR in action

- Matrix size:
 $m = 24, n = 10$
- Process grid:
 $p = 3, q = 1$
- TS parameter:
 $a = 2$
- Reduction trees:
Fibonacci/Fibonacci

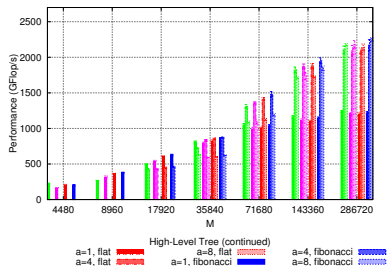
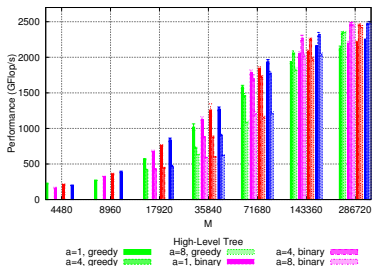


Experimental platform

Cluster Edel from Grid5000, Grenoble

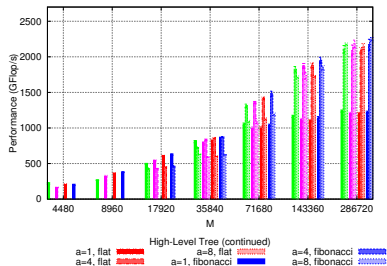
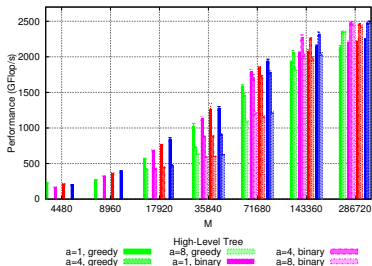
- 60 nodes
- 2 Nehalem Xeon E5520 at 2.27GHz per node (8 cores)
- 24 GB per node
- Infiniband 20G network
- Theoretical peak performance:
 - 9.08 GFlop/s per core
 - 72.64 GFlop/s per node
 - 4.358 TFlop/s for the whole machine

Influence of TS kernels and trees ($N=4480$)



Influence of the TS level size with fixed local reduction tree
(Left: Greedy, Right: Flat)

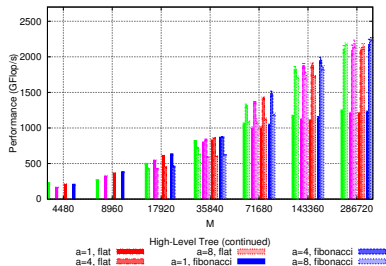
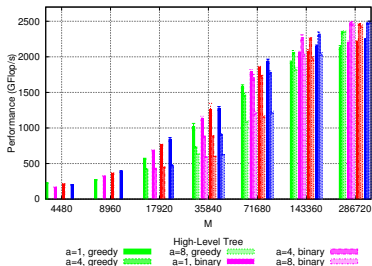
Influence of TS kernels and trees ($N=4480$)



Influence of the TS level size with fixed local reduction tree
(Left: Greedy, Right: Flat)

⇒ a must be tuned to matrix size

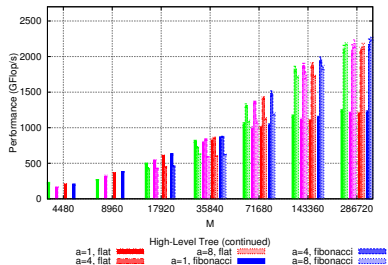
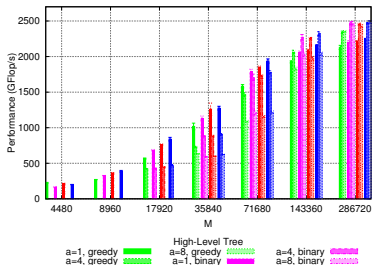
Influence of TS kernels and trees ($N=4480$)



Influence of the TS level size with fixed local reduction tree
(Left: Greedy, Right: Flat)

Low level tree: FLATTREE slower than others

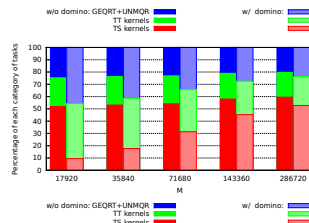
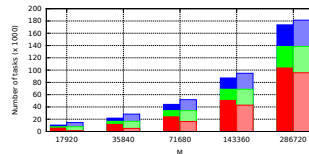
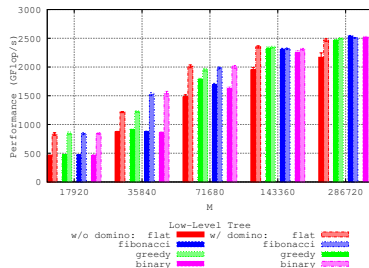
Influence of *TS* kernels and trees (N=4480)



Influence of the *TS* level size with fixed local reduction tree
(Left: Greedy, Right: Flat)

High level tree: FLATTREE slightly better than others

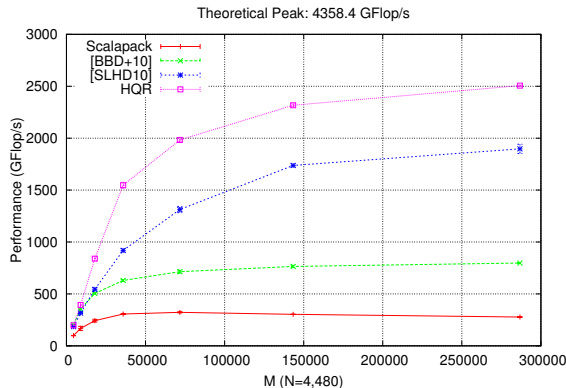
Influence of domino



Influence of low-level tree and domino optimization

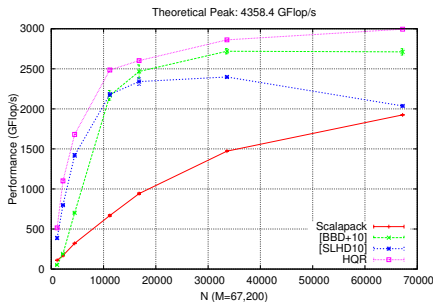
$N=4480$, $P=15$, $Q=4$, $MB=280$, $a=4$, High-level tree = Fibonacci

Scaling experiments (N=4480, M varies)



P=15, Q=4, MB=280, FIBONACCI/FIBONACCI,
a=4, domino enabled

Scaling experiments (M=67200, N varies)

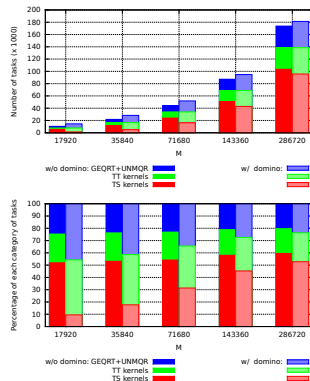


$P=15$, $Q=4$, $MB=280$,

FIBONACCI/FLAT TREE,

For $N \leq 16800$, $a=1$, domino enabled

For $N > 16800$, $a=4$, domino disabled



Lesson learnt

- Hierarchical algorithm quite complex 😞
- Outperforms existing solutions 😊
- Really flexible (architecture and matrix shape) 😊
- Used DAGUE scheduling software 😊
- Not ready for exascale yet 😞 😞 😞 😞 😞 😞 😞 😞
 - Add (yet another) level of hierarchy?
 - Extend ABFT techniques?
 - Use application checkpointing? replication? both?

References

Multicore: SC'2011

Petascale: LAWN 257

Lesson learnt

- Hierarchical algorithm quite complex 😞
- Outperforms existing solutions 😊
- Really flexible (architecture and matrix shape) 😊
- Used DAGUE scheduling software 😊
- Not ready for exascale yet 😞 😞 😞 😞 😞 😞 😞 😞
 - Add (yet another) level of hierarchy?
 - Extend ABFT techniques?
 - Use application checkpointing? replication? both?

References

Multicore: SC'2011

Petascale: LAWN 257

Outline

- 1 Towards an exascale QR factorization algorithm
 - Shared-memory algorithm (single multicore node)
 - Hierarchical algorithm (many multicore nodes)
- 2 Checkpointing
 - Steady-state throughout of large platforms
 - Exascale tightly-coupled job
- 3 Conclusion

Outline

- 1 Towards an exascale QR factorization algorithm
 - Shared-memory algorithm (single multicore node)
 - Hierarchical algorithm (many multicore nodes)
- 2 Checkpointing
 - Steady-state throughout of large platforms
 - Exascale tightly-coupled job
- 3 Conclusion

Notations

- C : checkpoint save time (in minutes)
- R : checkpoint recovery time (in minutes)
- D : down/reboot time (in minutes)
- μ : MTTF, mean time to failure
(e.g., $1/\lambda$ if failures are exponentially distributed)
- N : total number of multicore nodes

Scenario “2015”

- Phase-Change memory
 - read bandwidth 100GB/sec
 - write bandwidth 10GB/sec
- Checkpoint size 128GB
- $C = 0.21$, $R = 0.021$, $D = 0.25$ minutes
- $N = 2^8$ to $N = 2^{20}$
- $\mu = 1$ week, 1 month, 1|10|100|1000 years (per node)

Simulator

<http://navet.ics.hawaii.edu/~casanova/software/resilience.tgz>

Failures (1/2)

- Exponential: density $p(t) = \lambda e^{-\lambda t}$
 $\Rightarrow \mu = 1/\lambda$
- Weibull: density $p(t) = (a/\lambda)(t/\lambda)^{a-1} e^{-(t/\lambda)^a}$
 $\Rightarrow \mu = \lambda \Gamma(1 + 1/a)$
 \Rightarrow take $a = 0.5$ or $a = 0.78$ (values from literature)
- Values of MTTF
 - $\mu = 1$ year for ASCI Q machine
 - $\mu = 10$ -100 years for Jaguar

Failures (2/2)

- If a job uses two processors, what is the expected interval time between failures?
- μ_j mean of the minimum of 2^j i.i.d. variables
- If the variables are exponentially distributed, with scale parameter λ , then

$$\mu_j = 1/(\lambda 2^j) = \mu/2^j$$

- If the variables are Weibull, with scale parameter λ and shape parameter a , then

$$\mu_j = \lambda \Gamma(1 + 1/(a 2^j))$$

Distribution of parallel jobs (1/2)

Number of processors required by typical jobs: *two-stage log-uniform distribution biased to powers of two* (says Dr. Feitelson)

- Let $N = 2^Z$ for simplicity
- Probability that a job is sequential: $\alpha_0 = p_1 \approx 0.25$
- Otherwise, the job is parallel, and uses 2^j processors with **identical probability**

$$\alpha_j = \alpha = (1 - p_1) \times \frac{1}{Z}$$

for $1 \leq j \leq Z = \log_2 N$

Distribution of parallel jobs (2/2)

- **Steady-state** utilization of whole platform:
 - all processors always active
 - constant proportion of jobs using any processor number
- Expectation of the number of jobs:
 - K total number of jobs running
 - β_j jobs that use 2^j processors exactly
- Equations: $K = \sum_{j=0}^Z \beta_j$ with $\beta_j = \alpha_j K$, $\sum_{j=0}^Z 2^j \beta_j = N$

$$\frac{N}{K} = \sum_{j=0}^Z 2^j \alpha_j = p_1 + \frac{1-p_1}{Z} \sum_{j=1}^Z 2^j = p_1 + \frac{1-p_1}{Z} (2N-2)$$

hence the value of K and the β_j

Optimal checkpointing period T

W = expected percentage of time lost, or “wasted”:

$$W = \frac{C}{T} + \frac{\frac{T}{2} + R + D}{\mu}$$

- First term by definition:
 C time-steps devoted to checkpointing every T time-steps
- Every μ time-steps, a failure occurs
 \Rightarrow loss of $T/2$ time-steps in average, plus $D + R$

$$W_{min}^* = \min \left(1, \frac{R + D}{\mu} + \sqrt{\frac{2C}{\mu}} \right) \quad (1)$$

Platform throughput

Sequential jobs

$$\rho = (1 - W_{min}^*)N$$

Parallel jobs

$$\rho = \sum_{j=0}^Z (1 - W_{min}^*(j)) 2^j \beta_j$$

use μ_j instead of μ in (1) to derive $W_{min}^*(j)$

Numerical results: yield ρ/N for scenario “2015”

	N	Throughput
$\mu = 1 \text{ week}$	2^8	91.56%
	2^{11}	73.75%
	2^{14}	20.07%
	2^{17}	2.51%
	2^{20}	0.31%

	N	Throughput
$\mu = 1 \text{ month}$	2^8	96.04%
	2^{11}	88.23%
	2^{14}	62.28%
	2^{17}	10.66%
	2^{20}	1.33%

	N	Throughput
$\mu = 1 \text{ year}$	2^8	98.89%
	2^{11}	96.80%
	2^{14}	90.59%
	2^{17}	70.46%
	2^{20}	15.96%

ρ/N for unstable platforms

Numerical results: yield ρ/N for scenario “2015”

	N	Throughput
$\mu = 10$ years	2^8	99.65%
	2^{11}	99.00%
	2^{14}	97.15%
	2^{17}	91.63%
	2^{20}	74.01%

	N	Throughput
$\mu = 100$ years	2^8	99.89%
	2^{11}	99.69%
	2^{14}	99.11%
	2^{17}	97.45%
	2^{20}	92.56%

	N	Throughput
$\mu = 1000$ years	2^8	99.97%
	2^{11}	99.90%
	2^{14}	99.72%
	2^{17}	99.20%
	2^{20}	97.73%

ρ/N for stable platforms

Outline

- 1 Towards an exascale QR factorization algorithm
 - Shared-memory algorithm (single multicore node)
 - Hierarchical algorithm (many multicore nodes)
- 2 Checkpointing
 - Steady-state throughout of large platforms
 - Exascale tightly-coupled job
- 3 Conclusion

Motivation

Context

- A large divisible job
- A large number N of identical nodes (same processing speed, same failure distribution)

Questions

- How many processors ($\leq N$) to minimize expected makespan?
- Task duplication?

State-of-the-art

Sequential jobs

- Exponential: best strategy periodic; Young, Daly, optimal, ...
- Weibull: many heuristics

Parallel jobs

- No optimality result known
- Periodic heuristics available for Exponential and Weibull

Hypotheses

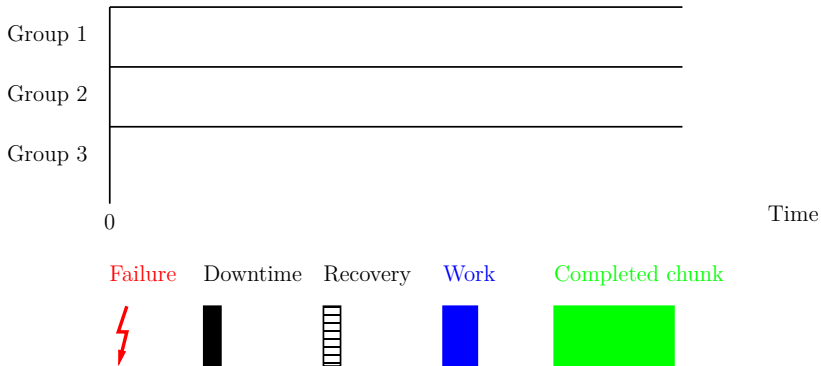
Parallelization

- Application is perfectly parallelizable $T_{\text{par}}(p) = \frac{T_{\text{seq}}}{p}$
(even more striking with Amdhal jobs or numerical kernels)

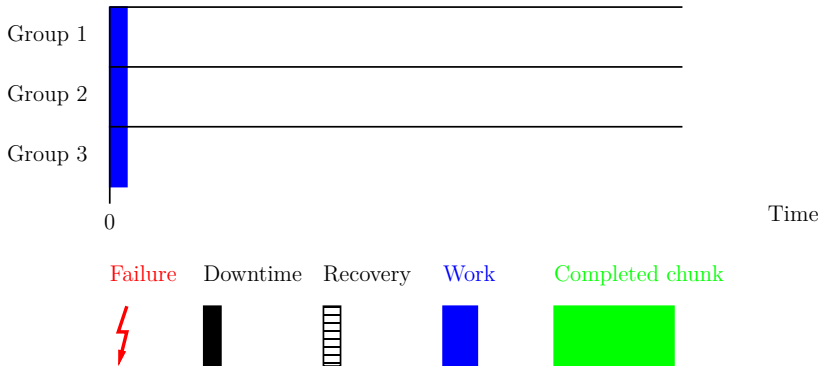
Duplication

- g groups of p processors ($g \times p \leq N$)

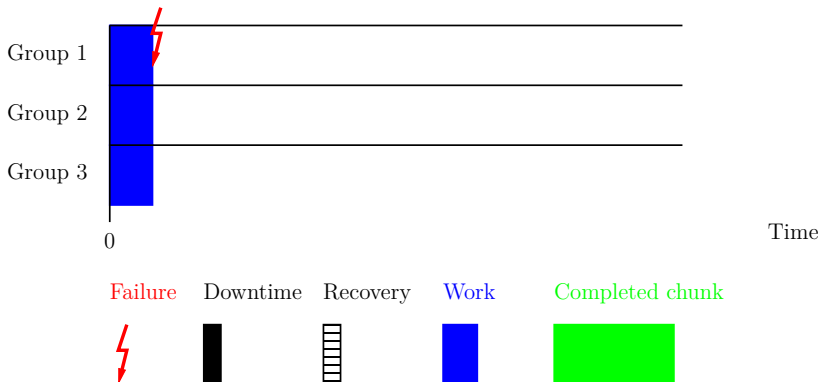
Group execution of a chunk



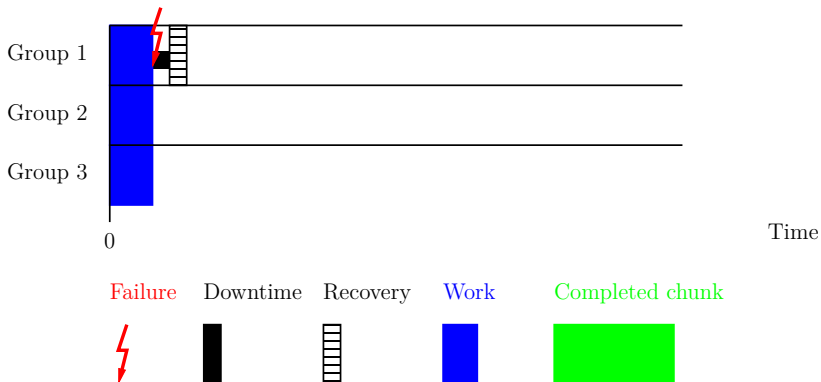
Group execution of a chunk



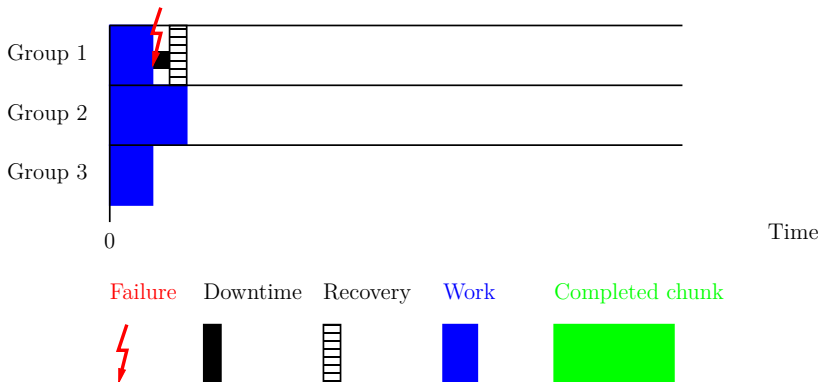
Group execution of a chunk



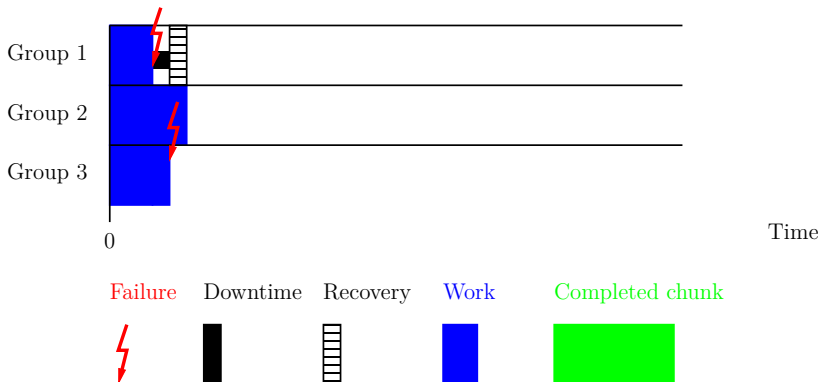
Group execution of a chunk



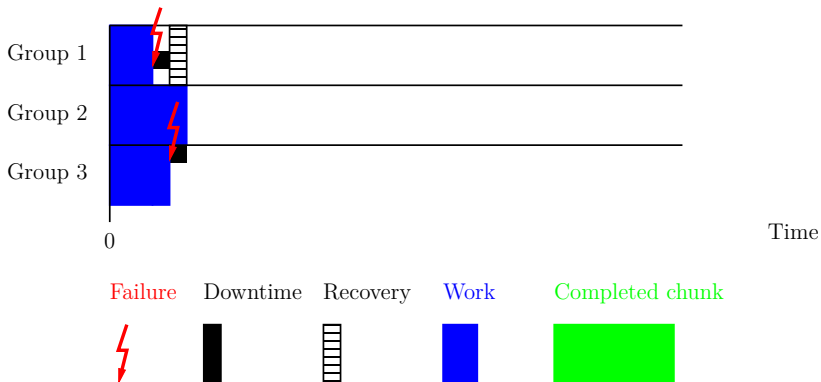
Group execution of a chunk



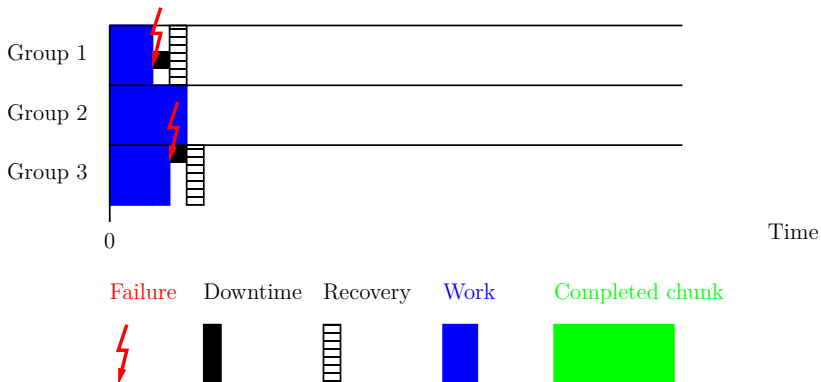
Group execution of a chunk



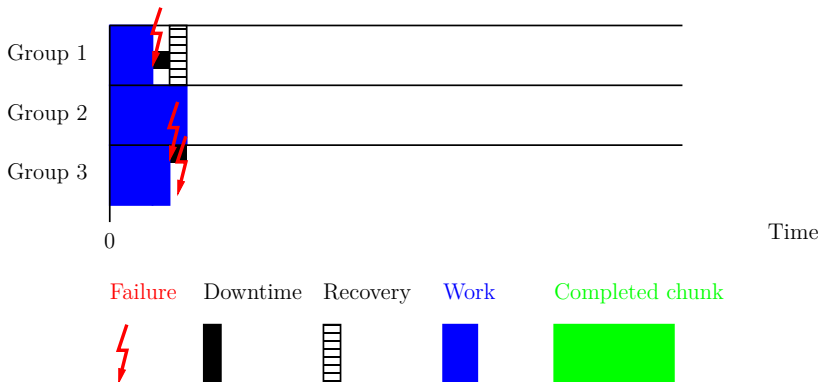
Group execution of a chunk



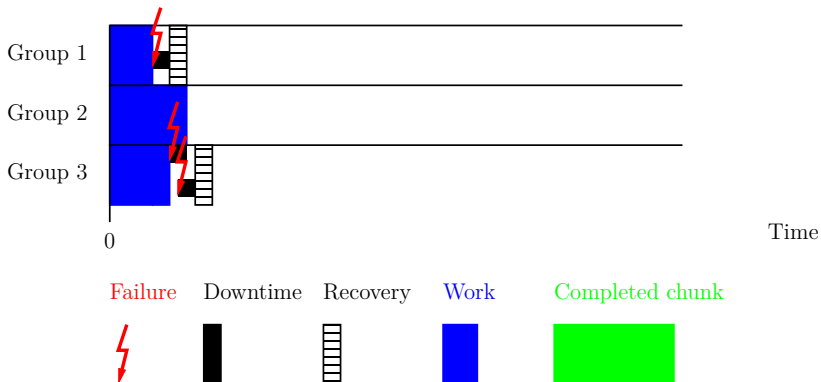
Group execution of a chunk



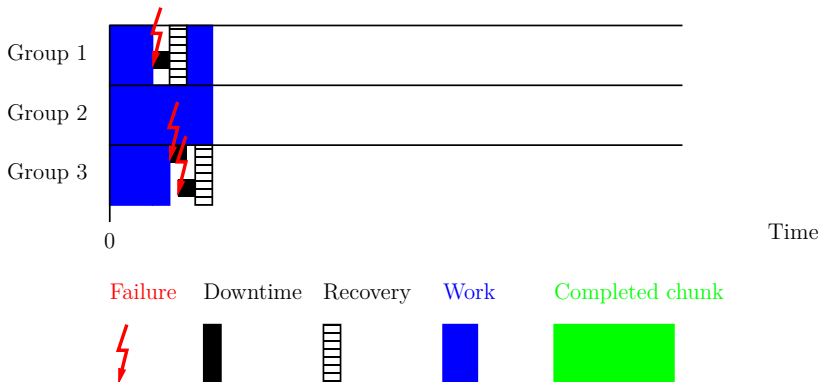
Group execution of a chunk



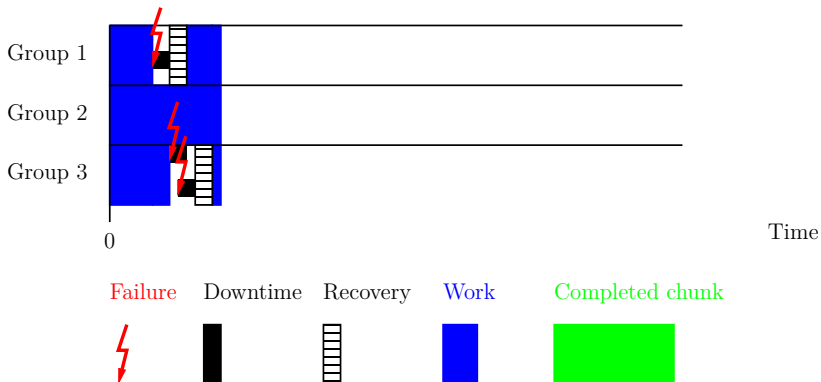
Group execution of a chunk



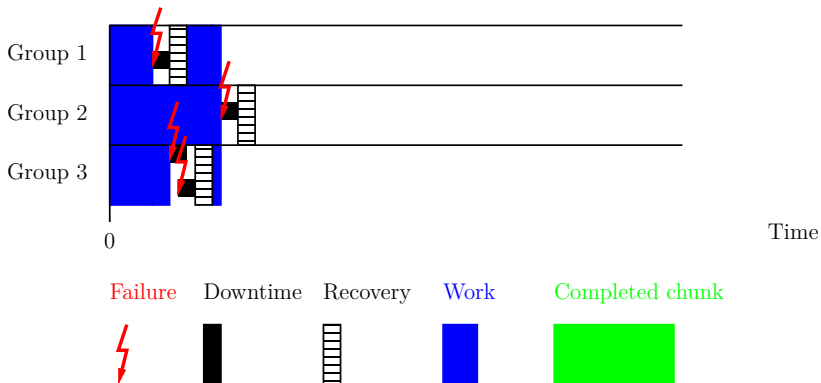
Group execution of a chunk



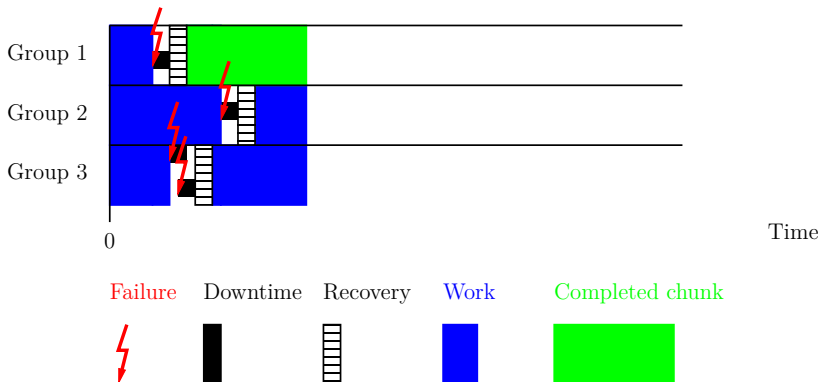
Group execution of a chunk



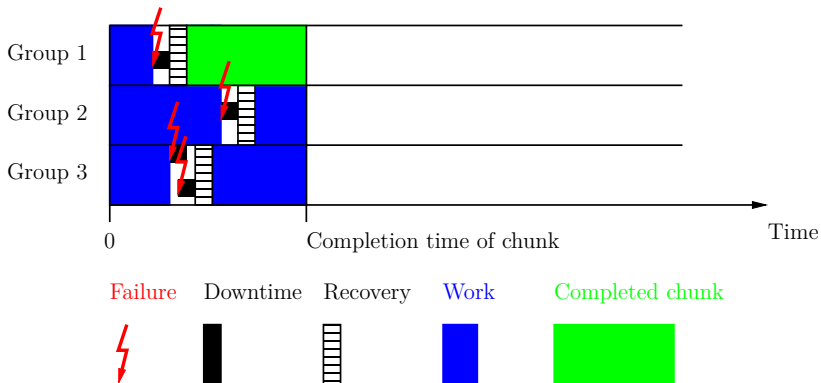
Group execution of a chunk



Group execution of a chunk



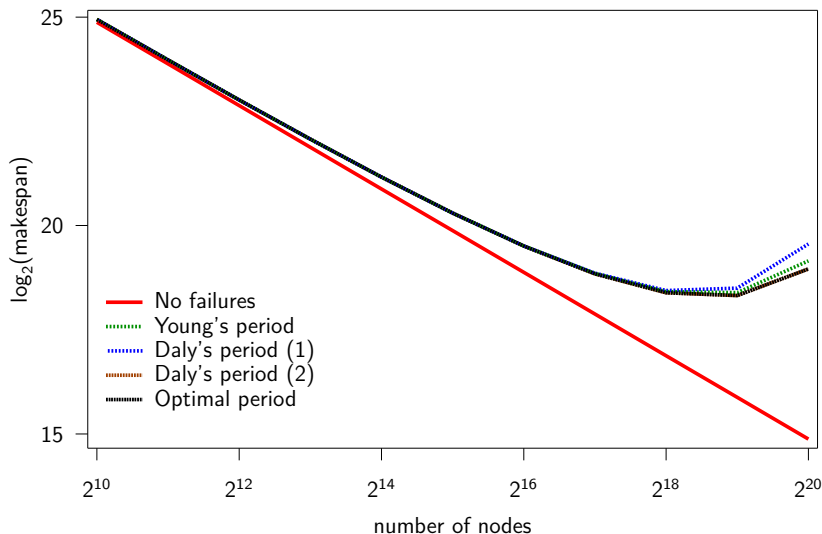
Group execution of a chunk



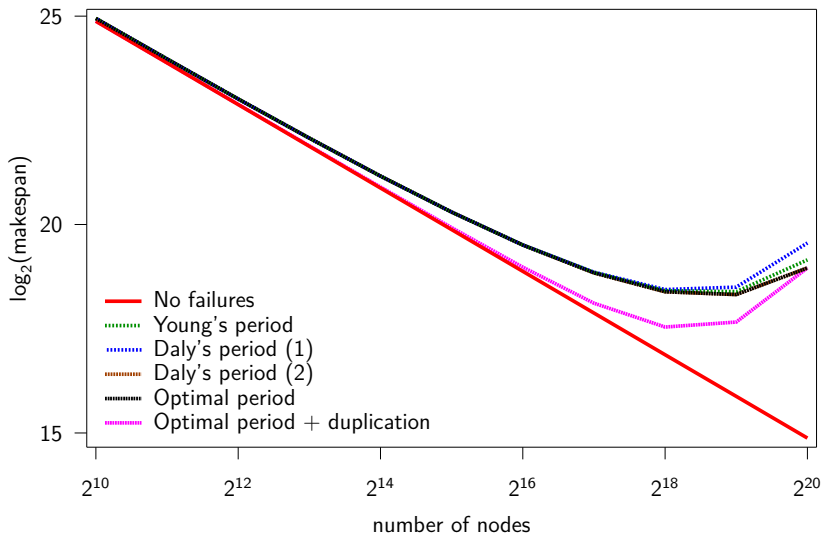
Simulation settings

- $C = R = 5mn$ & $D = 1mn$ (Local SDD, scenario “2012”)
- $N = 2^{20}$ nodes
- $\mathcal{W} = 1000$ years $\Rightarrow \approx 500mn$ on whole (failure-free) platform
- MTTF $\mu = 10$ years

Exponential distribution (MTTF $\mu = 10$ years)



Exponential distribution (MTTF $\mu = 10$ years)

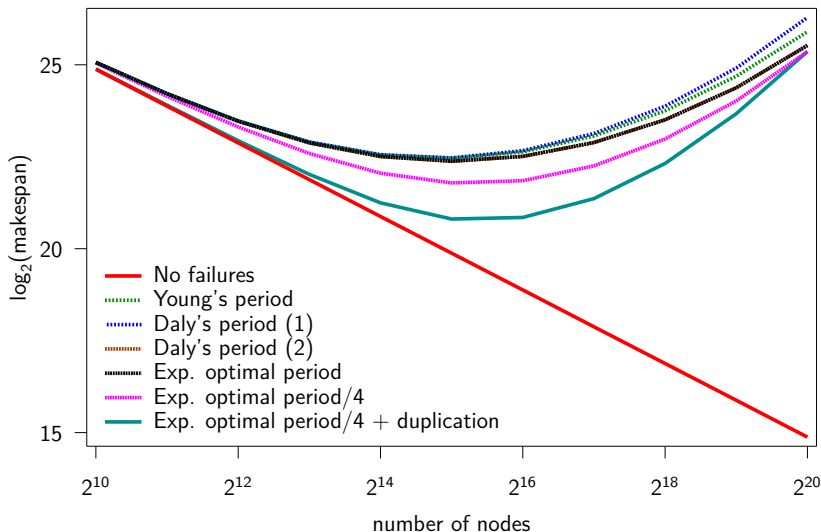


Best processor usage

Best makespan (without duplication) with 2^{19} cores

	Without Duplication	With Duplication
Number of nodes	2^{19}	2×2^{18}
Average makespan	344,493	206,718

Weibull distribution ($k=0.5$, MTTF $\mu = 10$ years)



Lesson learnt

- System-level checkpointing very costly
- Best resource usage = open problem
- Need energy-aware strategies too

References

Platforms: ICPP'2011 + Parallel Processing Letters

Parallel job: SC'2011 + ongoing work

Outline

- 1 Towards an exascale QR factorization algorithm
 - Shared-memory algorithm (single multicore node)
 - Hierarchical algorithm (many multicore nodes)
- 2 Checkpointing
 - Steady-state throughout of large platforms
 - Exascale tightly-coupled job
- 3 Conclusion

Conclusion

- Software/hardware techniques to reduce checkpoint, recovery, migration times and to improve failure prediction
- "Self-fault-tolerant" **algorithms** (e.g. asynchronous iterative)
- 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 😊