A Case Study of Designing Efficient Algorithm-based Fault Tolerant Application for Exascale Parallelism

Erlin Yao, Rui Wang, Mingyu Chen, Guangming Tan, Ninghui Sun

State Key Laboratory of Computer Architecture,

Institute of Computing Technology, Chinese Academy of Sciences

IPDPS 2012@Shanghai, China. 2012.5.22

INSTITUTE OF COMPUTING TECHNOLOGY 中科院计算所

Outline

- Hardware Resilience is a Prominent Issue to be Addressed
- Fault Tolerance Techniques for HPC
  - Classical Technique: **Checkpointing**
  - Conventional ABFT Technique: **ABFT Recovery**
- A New Efficient ABFT Scheme
- A Case Study of High Performance Linpack
- Theoretical Analysis of Fault Tolerance Overhead at Exascale
- Experimental Evaluation
- Discussion, Conclusion and Future Works

Hardware Resilience is a Prominent Issue to be Addressed

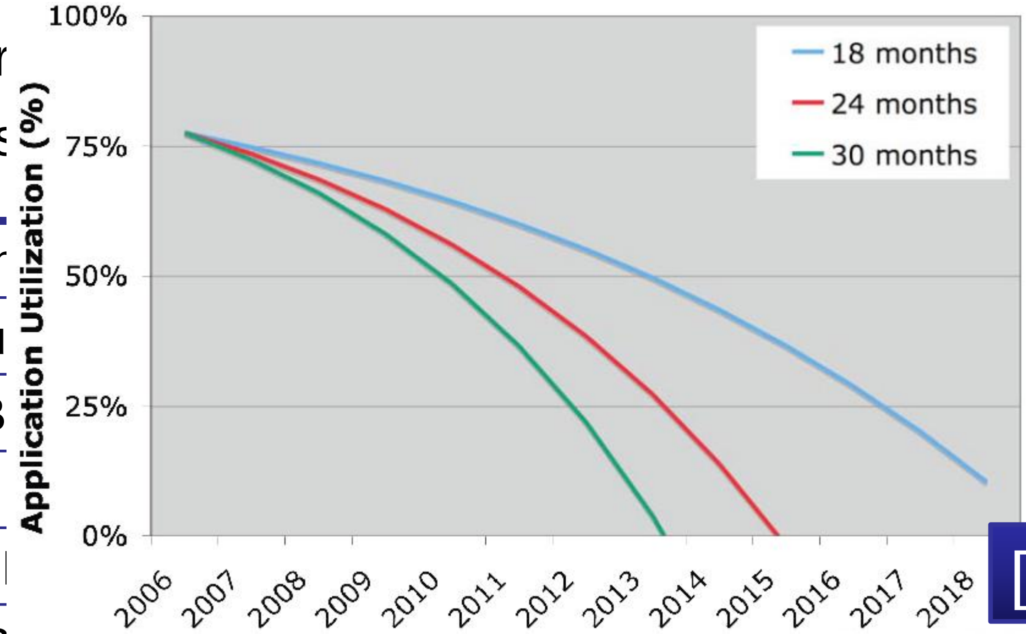
- With the growing scale of High Performance Computing (HPC) systems, faults are a norm rather than an exception
  - Exascale systems are projected to fail every 3~26 minutes[Schroeder and Gibson].
  - Even if each processing element fails only once every 10,000 years, a system that has a billion processing elements would have a fault once every 5 minutes. [IBM]

Outline

- Hardware Resilience is a Prominent Issue to be Addressed
- Fault Tolerance Techniques for HPC
  - Classical Technique: **Checkpointing**
  - Existing ABFT Technique: **ABFT Recovery**
- A New Efficient ABFT Scheme
- A Case Study of High Performance Linpack
- Theoretical Analysis of Fault Tolerance Overhead at Exascale
- Experimental Evaluation
- Discussion, Conclusion and Future Works

Classical Technique -Checkpointing

- Weakness
  - Need roll back
  - Per
  - Bas



System			
RoadRu			
LLNL B			
Argonne			
Total SGI			
IDRIS BG/P	100 TF	30 min.	IDRIS

[Gibson, ICPP2007]

Roadrunner

BG/L

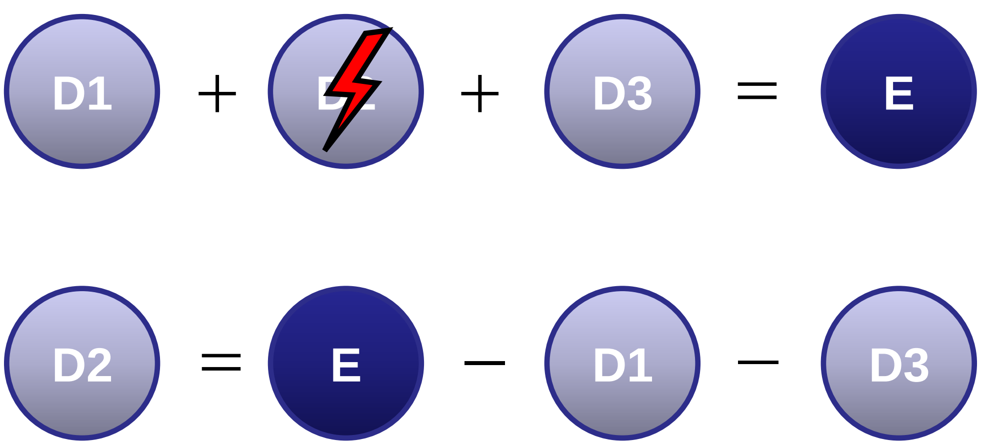
BG/P

[Cappello, 2009]

The application utilization of checkpointing will keep dropping to zero in the next decade under current technology trends!

Conventional Algorithm-Based Fault Tolerance Technique

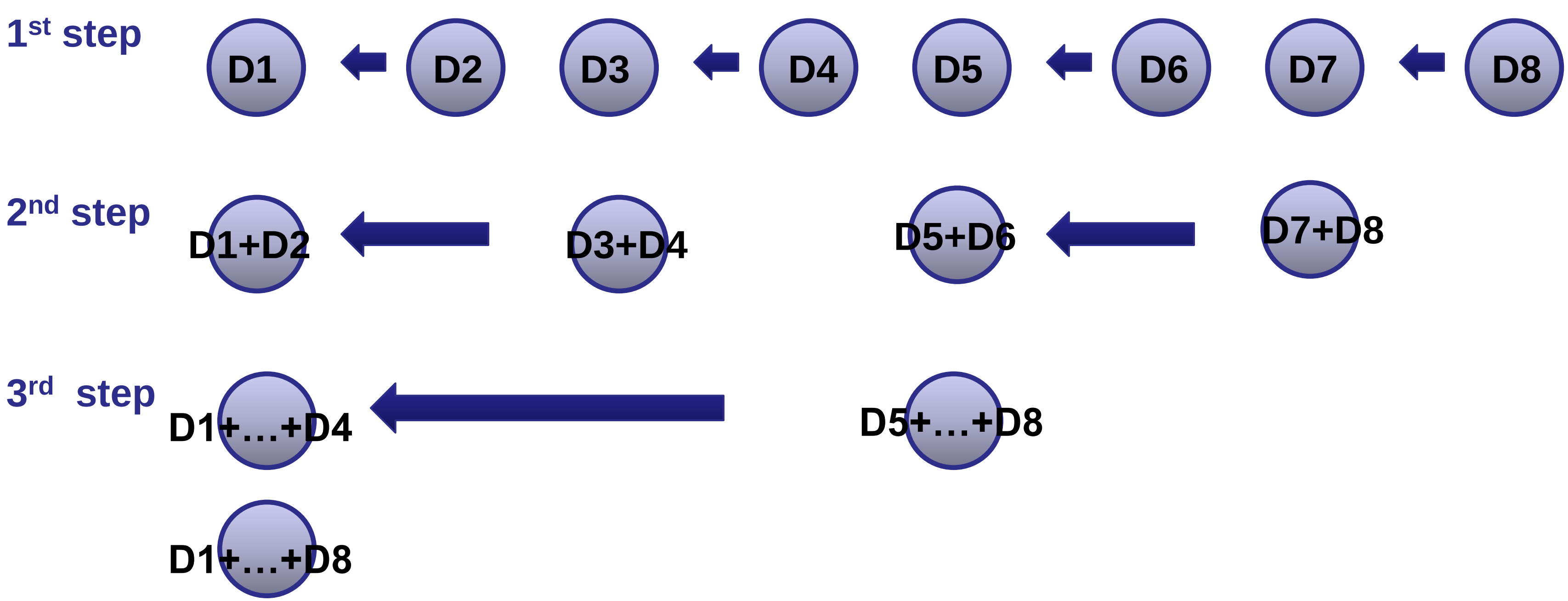
- ABFT Recovery [by Chen and Dongarra]
  - Add redundant node to store the encoded checksum of the original data
  - Re-design algorithm to compute the original data and the redundancy synchronously
  - Recover corrupted data upon failure based on checksum relationship



reduce operation

Conventional Algorithm-Based Fault Tolerance Technique

- Reduce Operation



  - a Q-nodes reduce operation, need $\lceil \log_2 Q \rceil$ steps communication and computation
- Recover one failed node

$$D_2 = E - D_1 - D_3$$

  - Q compute nodes -> $\lceil \log_2 Q \rceil$

Pros and Cons of ABFT Recovery

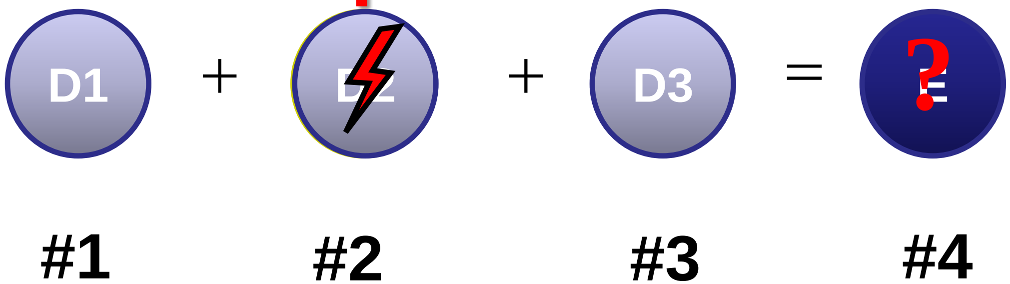
- Pros
  - No periodical stoppage to write checkpoints
    - Redundancy is computed with original data
  - No rollback
    - checksum relationship is maintained in the middle of computation
- Cons
  - Not transparent to application
  - Not as widely used as checkpointing
    - ScaLapack, HPL、PCG(preconditioned conjugate gradient) solver and iterative methods in solving equation and equation set
  - Limited by the bandwidth of network
  - Stop-and-wait scheme

Outline

- Hardware Resilience is a Prominent Issue to be Addressed
- Fault Tolerance Techniques for HPC
  - Classical Technique: **Checkpointing**
  - Existing ABFT Technique: **ABFT Recovery**
- A New Efficient ABFT Scheme
- A Case Study of High Performance Linpack
- Theoretical Analysis of Fault Tolerance Overhead at Exascale
- Experimental Evaluation
- Discussion, Conclusion and Future Works

A New Efficient ABFT Scheme

- Failure **Hot-Replacement**



For q compute nodes,

Before the replacement, $D = \left(D_1 \cdots D_{i-1} D_i D_{i+1} \cdots D_q\right)$

After the replacement, $D' = \left(D_1 \cdots D_{i-1} E D_{i+1} \cdots D_q\right)$

Assume $D' = D \times T$,

$$T = \begin{array}{c} \begin{array}{ccccc} & & i^{th} & & \end{array} \\ \begin{pmatrix} 1 & & 1 & & \\ & \ddots & \vdots & & \\ & & 1 & & \\ & & \vdots & \ddots & \\ & & 1 & & 1 \end{pmatrix} \end{array}$$

Transformation Matrix $T: q \times q$

Background Accelerated Recovery of Redundancy

- To tolerate multiple failures, we need to rebuild the redundancy.

D1 + E + D3 = E'

- The rebuilding process is a reduce operation
  - Use additional nodes as accelerating nodes and **do the the reduce operation in background**
    - so that compute nodes could continue normal execution immediately after sending data to accelerating nodes

faster network

- Redundant nodes fall behind compute nodes
  - Use additional nodes to **help redundant nodes to catch up with compute nodes**

faster nodes

Advantages of HRBR Technique

- Hot Replacement
  - do not need to stop-and-wait
  - Transformation Matrix T is very sparse, so recovery cost is low

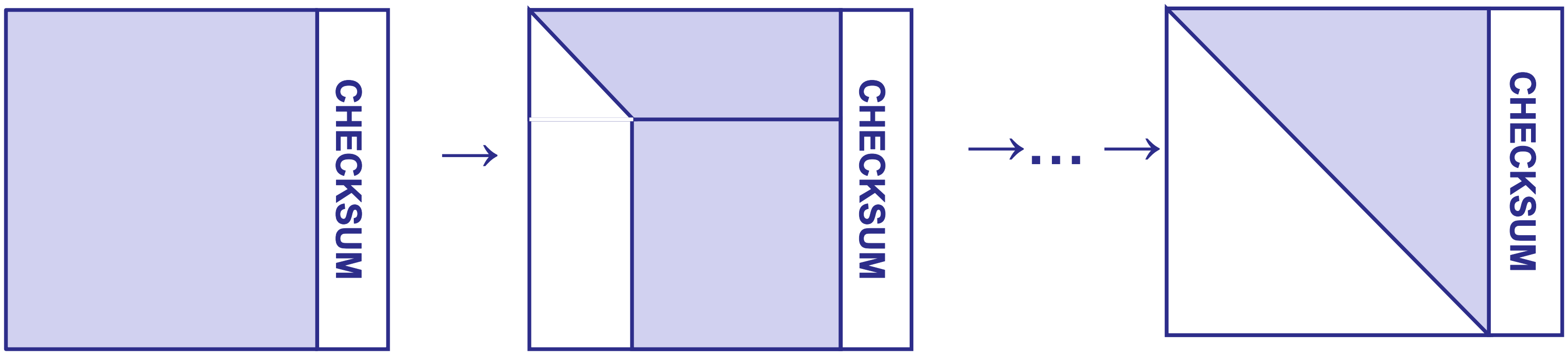
- Background Recovery
  - Rebuilding redundancy is in background and accelerated

Outline

- Hardware Resilience is a Prominent Issue to be Addressed
- Fault Tolerance Techniques for HPC
  - Classical Technique: **Checkpointing**
  - Existing ABFT Technique: **ABFT Recovery**
- A New Efficient ABFT Scheme
- A Case Study of High Performance Linpack
- Theoretical Analysis of Fault Tolerance Overhead at Exascale
- Experimental Evaluation
- Discussion, Conclusion and Future Works

A Case Study of High Performance Linpack

- High Performance Linpack (HPL) overview
  - benchmark for ranking supercomputers in top500
  - solve $Ax = b$ using GEPP(Gaussian Elimination with Partial Pivoting)



Each process generates its local random matrix *A*

for *i* = 0, 1, …

LU factorization $A_i = L_iU_i$; *computation*

Broadcast L_i right ; *communication*

Update the trailing sub-matrix *U* ; *computation*

solve upper-triangular $Ux = L^{-1}b$ to obtain *x* ; *back substitution phase*

checksum relationship maintained

A Case Study of High Performance Linpack

- Before hot-replacement, it solves

$$Ax = b$$

- After hot-replacement, it solves

$$A'y = b$$

- Assume $A' = A \times T$, the correct solution x could be obtained by

$$x = T \times y$$

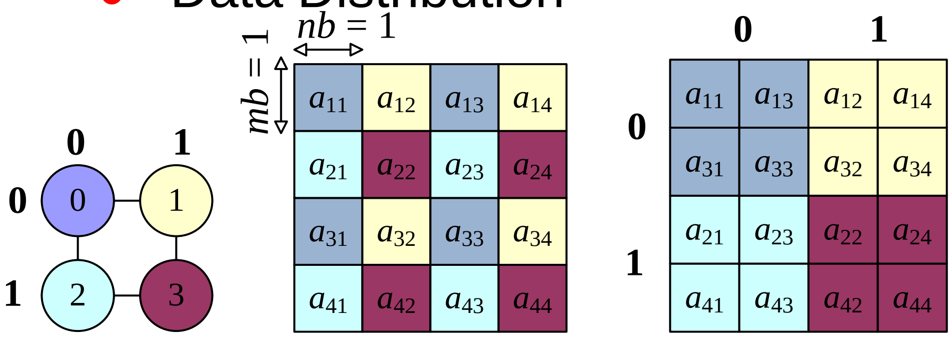
- If the transformation matrix is

$$T = \begin{array}{c} \begin{array}{ccccc} & & i^{\text{th}} & & \end{array} \\ \begin{pmatrix} 1 & & 1 & & \\ & \ddots & \vdots & & \\ & & 1 & & \\ & & \vdots & \ddots & \\ & & 1 & & 1 \end{pmatrix}_{q \times q} \end{array}$$

then $\begin{cases} X_j = Y_i + Y_j, \ 1 \le j \ne i < q \\ X_i = Y_i \end{cases}$

A Case Study of HPL

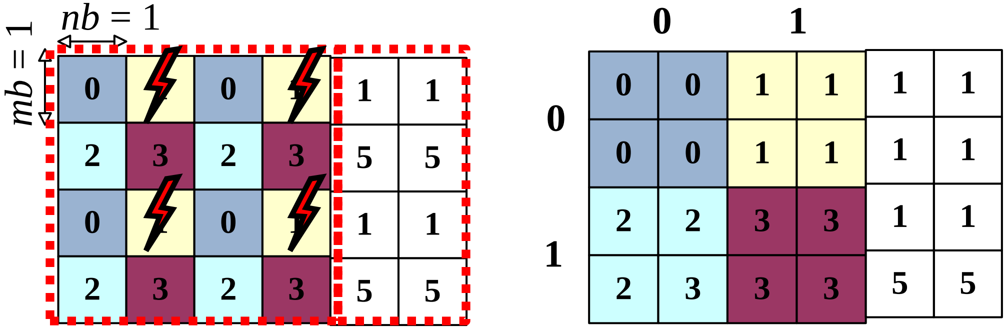
- Data Distribution



(a) (b) (c)

Fig. 1 2-dimensional block-cyclic data distribution with P=Q=2, mb=nb=1

- Example



(a) global view (b) distributed view

Fig. 3 Encoded Matrix

- Redundancy of matrix A can be denoted as: $A \times V$

- Encoding

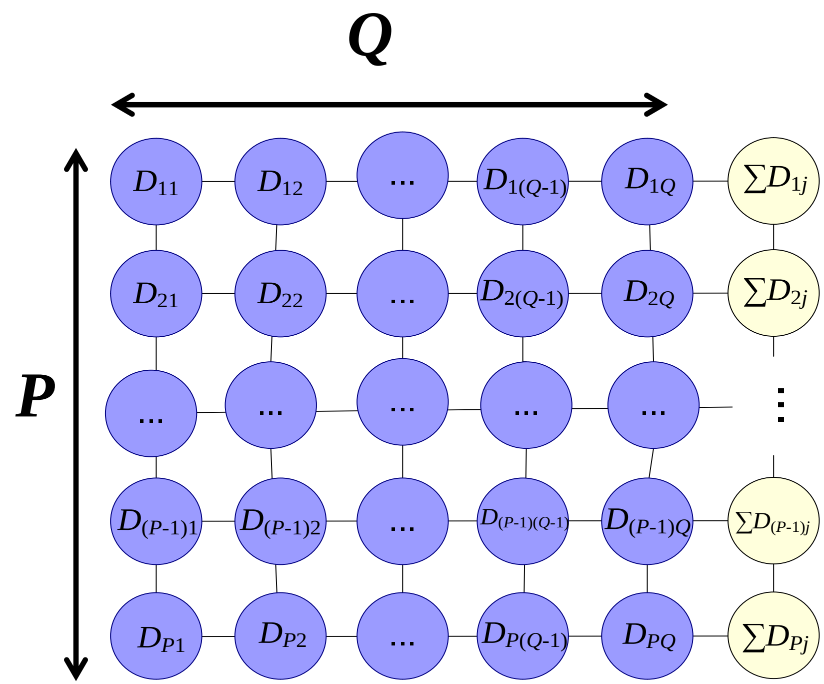


Fig. 2 New grid with redundancy

1	0
1	0
0	1
0	1

Fig. 4 Encoding matrix V

0	1	0	1
2	5	2	5
0	1	0	1
2	5	2	5

Fig. 5 After hot-replacement

A Case Study of High Performance Linpack

- **Lemma 1**: For an $\boldsymbol{n} \times \boldsymbol{n}$ matrix $\boldsymbol{A}$, suppose the redundancy is an $\boldsymbol{n} \times \boldsymbol{m}$ matrix: $(\boldsymbol{AV}_1|\boldsymbol{AV}_2| \cdots |\boldsymbol{AV}_m)$. If the $\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_m$ columns of $\boldsymbol{A}$ are replaced by these redundant columns respectively, then A becomes matrix $\boldsymbol{A'}$. There exists a matrix $\boldsymbol{T}$ such that $\boldsymbol{A'} = \boldsymbol{A}\times\boldsymbol{T}$, where T is an $\boldsymbol{n} \times \boldsymbol{n}$ matrix in the following form:

$$
\begin{array}{cccccc}
 & & \boldsymbol{i}_1{}^{\text{th}} & & \boldsymbol{i}_{\text{m}}{}^{\text{th}} & \\
\end{array}
$$

$$
\boldsymbol{T} = \begin{bmatrix}
1 & & \boldsymbol{V}_{1,1} & & \boldsymbol{V}_{\boldsymbol{m},1} & & \\
 & \ddots & \vdots & & \boldsymbol{V}_{\boldsymbol{m},2} & & \\
 & & \boldsymbol{V}_{1,\boldsymbol{i}_1} & & \vdots & & \\
 & & \vdots & \ddots & \vdots & & \\
 & & \vdots & & \boldsymbol{V}_{\boldsymbol{m},\boldsymbol{i}_m} & & \\
 & & \boldsymbol{V}_{1,\boldsymbol{n}-1} & & \vdots & \ddots & \\
 & & \boldsymbol{V}_{1,\boldsymbol{n}} & & \boldsymbol{V}_{\boldsymbol{m},\boldsymbol{n}} & & 1
\end{bmatrix}
$$

0	1	0	1
2	3	2	3
0	1	0	1
2	3	2	3

1	1
5	5
1	1
5	5

(a) Before hot-replacement

0	1	0	1
2	5	2	5
0	1	0	1
2	5	2	5

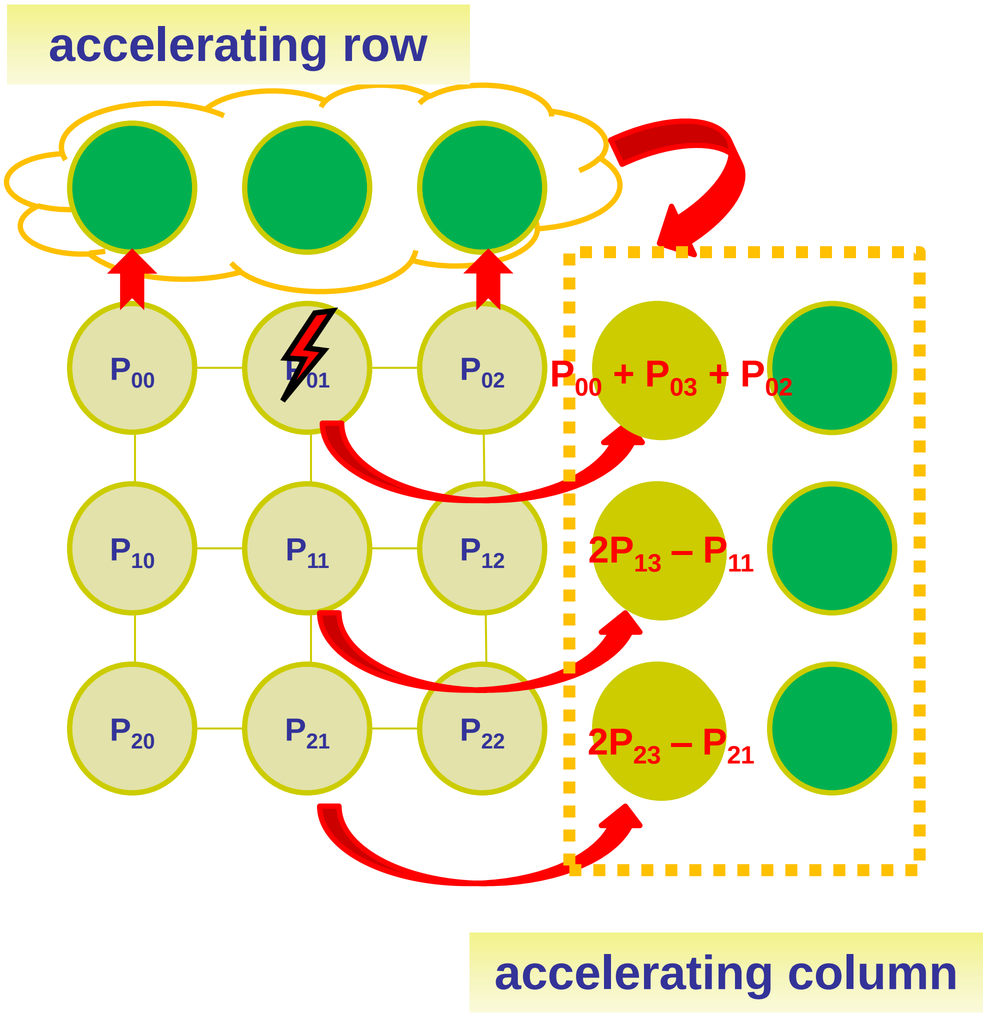
(b) After hot-replacement

$$
\boldsymbol{T}_1 = \begin{bmatrix}
1 & 1 & & 0 \\
 & 1 & & 0 \\
 & 0 & 1 & 1 \\
 & 0 & & 1
\end{bmatrix}
$$

$$
\begin{cases}
\boldsymbol{x}_1 = \boldsymbol{y}_1 + \boldsymbol{y}_2 \\
\boldsymbol{x}_2 = \boldsymbol{y}_2 \\
\boldsymbol{x}_3 = \boldsymbol{y}_3 + \boldsymbol{y}_4 \\
\boldsymbol{x}_4 = \boldsymbol{y}_4
\end{cases}
$$

Background Accelerated Recovery of Redundancy for HPL

1. Node failure
2. Hot-Replacement
3. Compute nodes send data to redundant column and accelerating row nodes
4. Compute nodes continue normal execution, while **accelerating row nodes reduce redundancy in background**
5. Redundant nodes receive data and rebuild redundancy
6. Redundant nodes and accelerating column nodes **catch up with compute nodes** in parallel
7. Synchronization of compute nodes and redundant nodes



Outline

- Hardware Resilience is a Prominent Issue to be Addressed
- Fault Tolerance Techniques for HPC
  - Classical Technique: **Checkpointing**
  - Existing ABFT Technique: **ABFT Recovery**
- A New Efficient ABFT Scheme
- A Case Study of High Performance Linpack
- Theoretical Analysis of Fault Tolerance Overhead at Exascale
- Experimental Evaluation
- Discussion, Conclusion and Future Works

Theoretical Analysis of Fault Tolerance Overhead at Exascale

- Assumptions:
  - The MTTF of Exascale system is 20 minutes, and there could be hundreds of faults during the execution of HPL at Exascale.

    3~26 minutes

  - Exascale means that $pf = O(10^{18})$ flops.

    p: num. of processes
    f: floating point computing power of each process

  - In Exascale HPL $n = O(10^8)$.

    Supercomputer K: 8×10^{15} flops, $n \approx 10^7$.

- Then the total execution time of HPL at Exascale can be estimated as:

$$T = \frac{\frac{2}{3}n^3}{pf} = \frac{\frac{2}{3}O(10^{24})}{O(10^{18})} \text{ seconds} \approx 200 \text{ hours}$$

- So there could be 600 faults during the execution of Exascale HPL.

Fault Tolerance Overhead Analysis of ABFT Methods

- The fault tolerance overhead mainly consists of two parts:
  - Encoding of data matrix at the very beginning of execution
  - Fault recovery after each failure
- During the execution of Exscale HPL, there could be hundreds of faults recovery.
- Each fault recovery corresponds to one fault, and so to the period of one system MTTF
- Overhead = (one fault recovery cost) / (system MTTF)

Overhead of ABFT Recovery

- (floating point computing power) / (communication bandwidth) = c
- The time to recover one floating point number is

$$\frac{\log_2 Q}{f} + \frac{\log_2 Q}{\frac{f}{c}} = \frac{(c+1)\log_2 Q}{f}$$

- The time to recover all the n^2/p floating point numbers on one failed process is

$$\frac{n^2}{p} \times \frac{(c+1)\log_2 Q}{f} = \frac{(c+1)n^2 \log_2 Q}{pf}$$

p: num. of processes
n: matrix order

- At Exascale(pf=10^{18}), if c =4000, n = 10^8, p = Q^2 =10^6,

$$\frac{4001 \times 10^{16} \times \log_2 10^3}{10^{18}} \text{ seconds} \approx 7 \text{ minutes}$$

- The system MTTF = 20 minutes, then the overhead is

$$\frac{7\text{minutes}}{\text{System MTTF}} = \frac{7}{20} = 35\%$$

Overhead = (one fault recovery cost) / (system MTTF)

Overhead Analysis of HRBR

- The time of one fault recovery at Exascale using HRBR could be

$$\frac{cn^2}{pf} = \frac{4000 \times 10^{16}}{10^{18}} = 40 \ \text{seconds} \approx 0.7 \ \text{minutes}$$

$$\text{ABFT Recovery Overhead} : \frac{(c+1)n^2 \log_2 Q}{pf}$$

- So the overhead of HRBR at Exascale is about

$$\frac{0.7 \ \text{minutes}}{\text{System MTTF}} = \frac{0.7}{20} = 3.5\%$$

- HRBR scheme could still be efficient for the fault tolerance of HPL at Exascale and beyond.

Outline

- Hardware Resilience is a Prominent Issue to be Addressed
- Fault Tolerance Techniques for HPC
  - Classical Technique: **Checkpointing**
  - Existing ABFT Technique: **ABFT Recovery**
- A New Efficient ABFT Scheme
- A Case Study of High Performance Linpack
- Theoretical Analysis of Fault Tolerance Overhead at Exascale
- Experimental Evaluation
- Discussion, Conclusion and Future Works

Experimental Evaluation

- Three sets of experiments performed:
  - Overhead for constructing checksum matrix
  - Performance: HRBR vs. ABFT recovery
  - Numerical correctness: HRBR vs. ABFT recovery
- Keep the amount of data in each process fixed and increase the size of test matrices
  - 5120×5120 floating point numbers per process
- 10 Simulated failures occurred during computation
- One failure at a time
- Same mappings between computing processes and physical cores

Experimental Evaluation

Process grid w/out redt.	Process grid w/ redt.	Num. of acc. processes	Size of original matrix
10 by 10	10 by 11	20	51,200
14 by 14	14 by 15	28	71,680
18 by 18	18 by 19	36	92,160
22 by 22	22 by 23	44	112,640
26 by 26	26 by 27	52	133,120
30 by 30	30 by 31	60	153,600
34 by 34	34 by 35	68	174,080
38 by 38	38 by 39	76	194,650
42 by 42	42 by 43	84	215,040

Overhead for constructing checksum matrix

- The result is rather close to the $\lceil \log Q \rceil$ fitting curve, which is in accordance with our theoretical analysis.

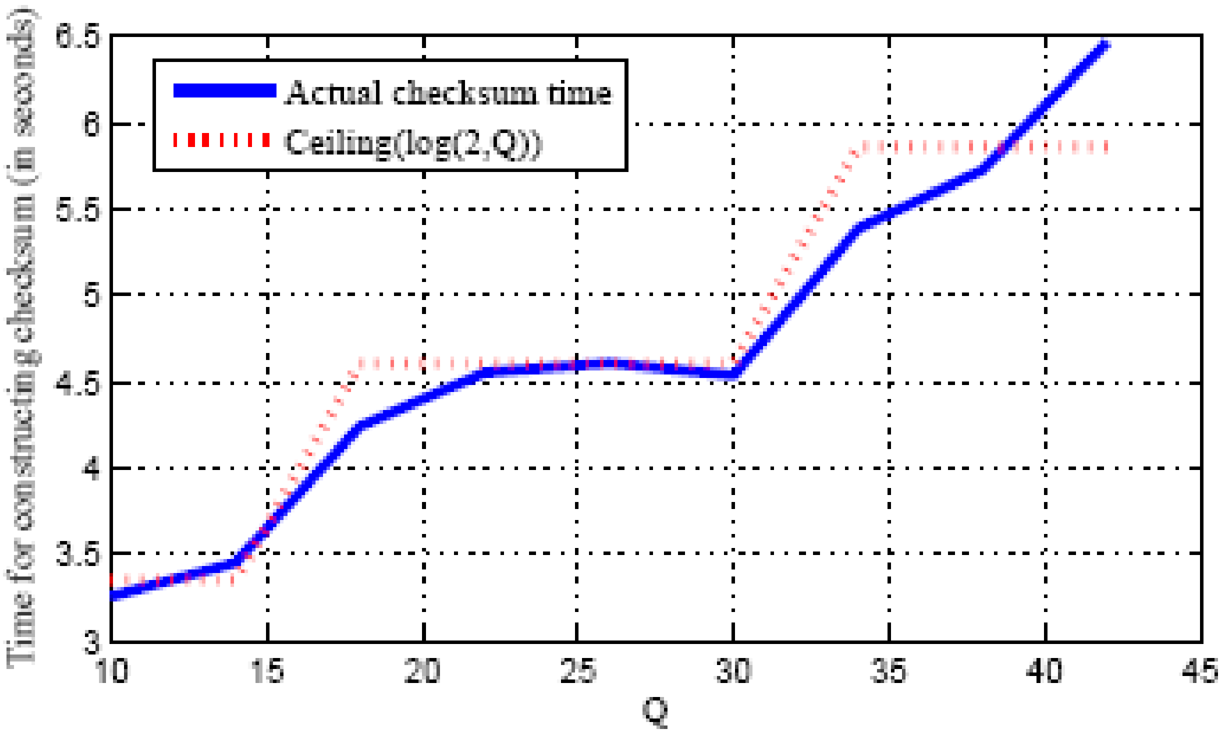


Fig. 6. Time for Constructing Checksum

Performance: HRB recovery

- The total execution time of HPL

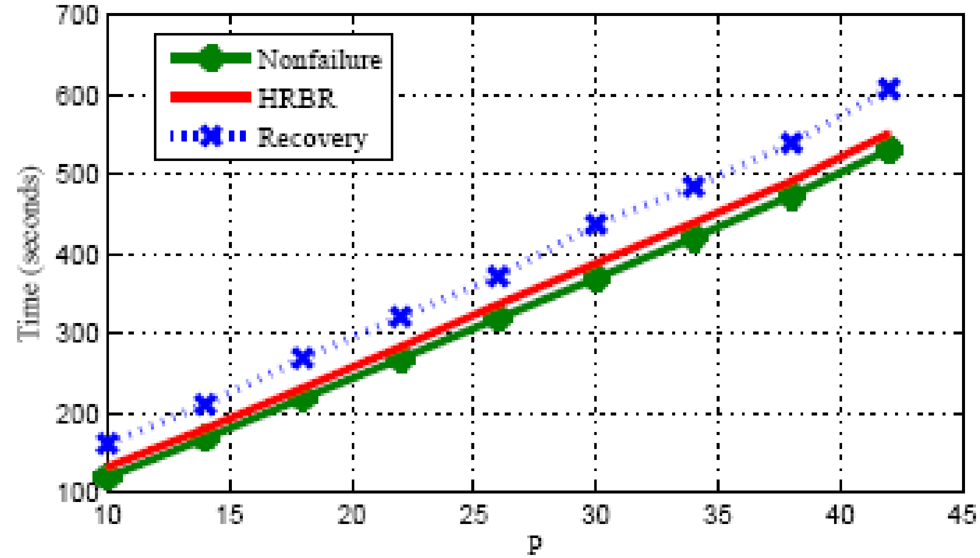


Fig. 8. Total execution time of HPL(in seconds)

- Fault tolerance Overhead of the two ABFT technique

- **HRBR has an obvious advantage over the ABFT recovery technique**

75% decrease

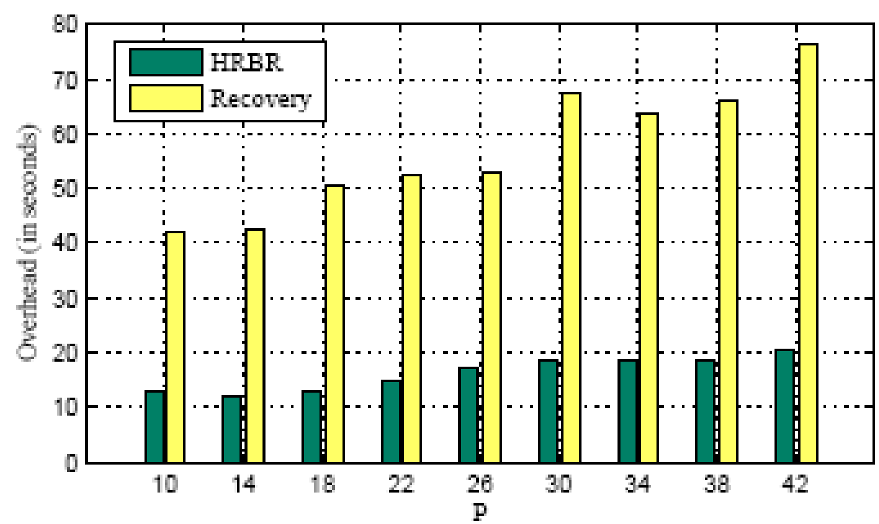


Fig. 7. Overhead of different fault tolerance methods

Numerical correctness: HRBR vs. ABFT recovery

- Norm of residuals (vertical axis is binary logarithm)

$$\frac{\|A}{\varepsilon \cdot (\|A\|_\infty}$$

HRBR introduces more round-off errors but still be acceptable.

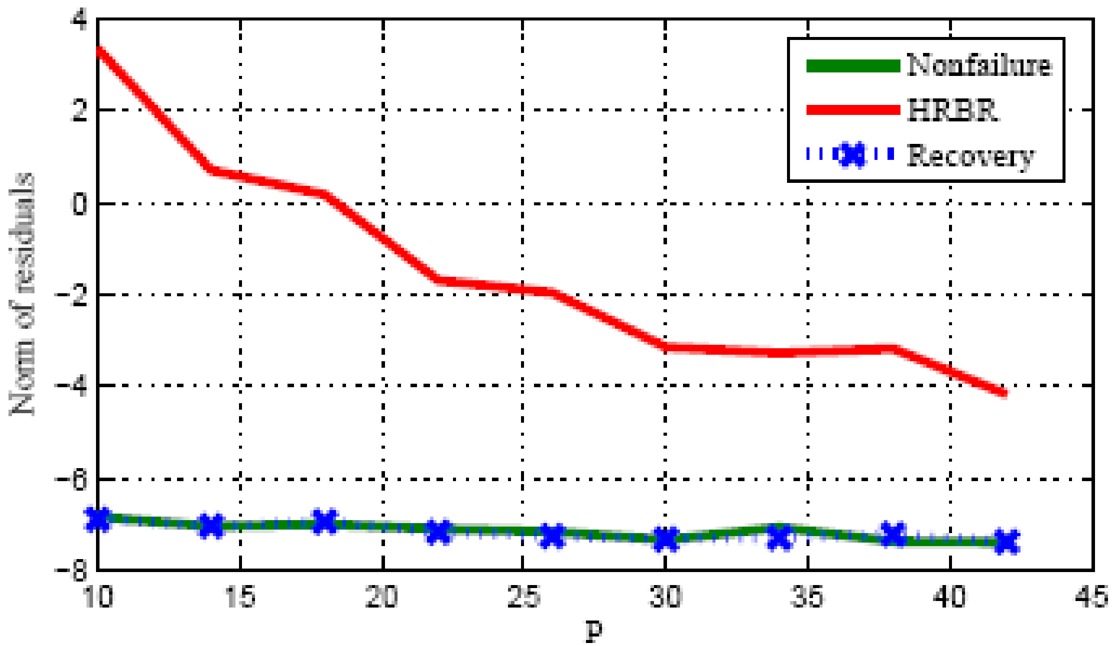


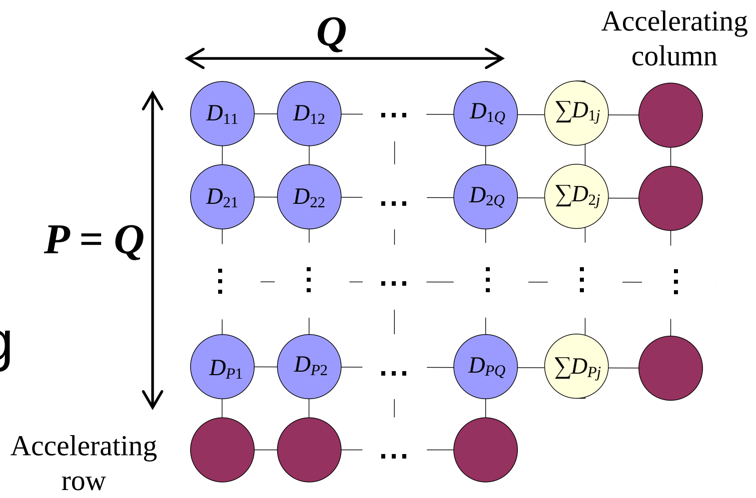
Fig. 9. Norm of residuals using different fault tolerant methods

Outline

- Hardware Resilience is a Prominent Issue to be Addressed
- Fault Tolerance Techniques for HPC
  - Classical Technique: **Checkpointing**
  - Existing ABFT Technique: **ABFT Recovery**
- A New Efficient ABFT Scheme
- A Case Study of High Performance Linpack
- Theoretical Analysis of Fault Tolerance Overhead at Exascale
- Experimental Evaluation
- Discussion, Conclusion and Future Works

Disccussion

- HRBR vs. ABFT Recovery
  - overhead: 2 / $\boldsymbol{Q}$
    - dedicate 2Q nodes as accelerating nodes
  - speedup: at Exascale ($\boldsymbol{Q}$ = 1000), $\mathbf{log}\boldsymbol{Q}$ = 10



- If MTTF keeps decreasing as system scales up
  - HRBR scheme will stand on its own
  - As $\boldsymbol{Q}$ increases, the overhead decreases but the speedup ratio increases

Conclusion

- A non-stop ABFT scheme—HRBR
  - Hot Replacement with Background Recovery
- A Case study of High Performance Linpack
- Theoretical analysis indicates
  - HRBR could be efficient at **Exascale**
- Experimental evaluation verifies
  - HRBR is more efficient than ABFT recovery

Future Works

- Extend HRBR to more HPC applications
- A more robust and efficient implementation
  - at large scale
  - under real circumstances
- Accuracy and stability at large scale

Thanks!