Improving Performance of Iterative Methods by Lossy Checkpointing

Dingwen Tao (University of California, Riverside)

Sheng Di (Argonne National Laboratory)

Xin Liang (University of California, Riverside)

Zizhong Chen (University of California, Riverside)

Franck Cappello (Argonne National Laboratory)

June 2018

UNIVERSITY OF CALIFORNIA UCRIVERSIDE

Argonne NATIONAL LABORATORY

Outline

- Introduction
  - Why we need to checkpoint iterative methods?

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead

Outline

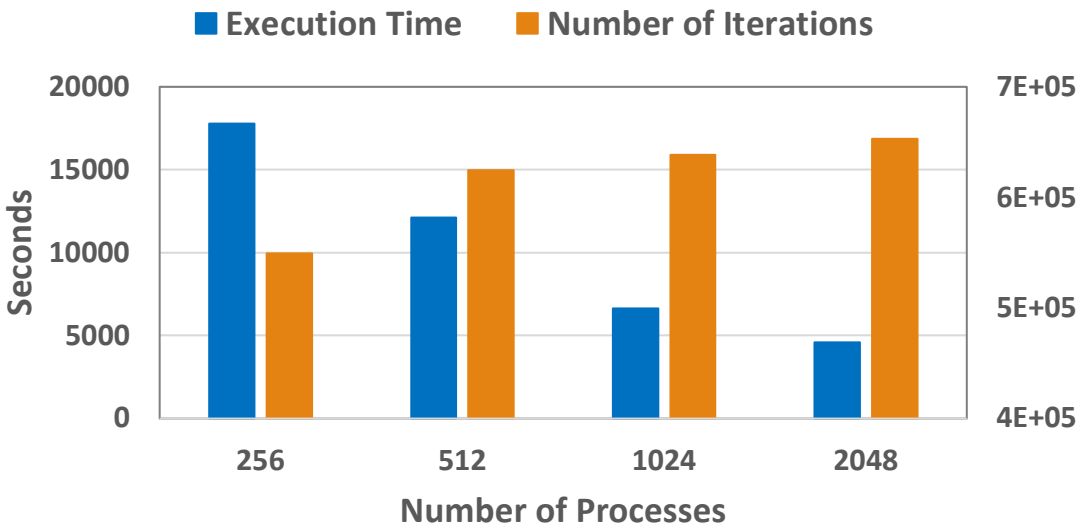
- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Why Need to Checkpoint Iterative Methods?

- Iterative methods used for solving large, sparse linear system
  - "Gaia" mission by European Space Agency (ESA)
    - Producing 5-parameter astrometric catalogue at the microarcsecond for 1 billion stars in Galaxy
    - Resulting a very large, sparse linear system of 72 billion equations
    - Scientists use LSQR iterative algorithm
    - Takes more than 54 hours on 2,048 BlueGene/Q nodes

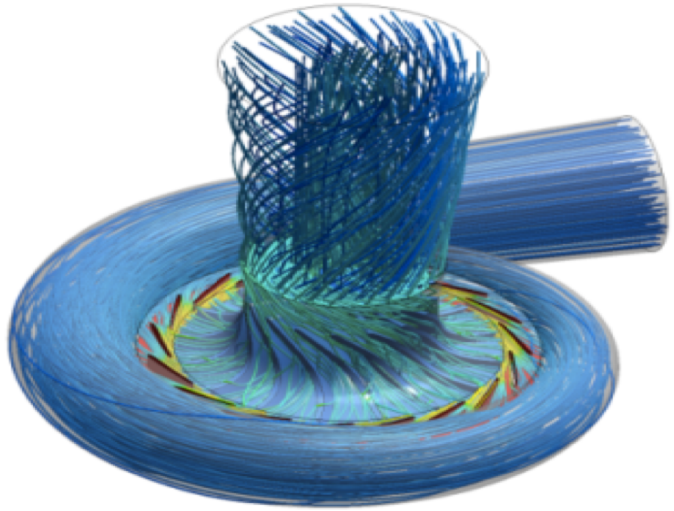


- Largest symmetric indefinite sparse matrix from UFL sparse matrix collection (KKT240 with 28 million linear equations)
- 2,048 cores / 64 nodes on Bebop cluster at Argonne
- GMRES solver implemented in PETSc
- Relative convergence tolerance of 10^{-6}, execution time > 1 hour
- MTBF of Sunway TaihuLight supercomputer can be hourly or less than 1 hour

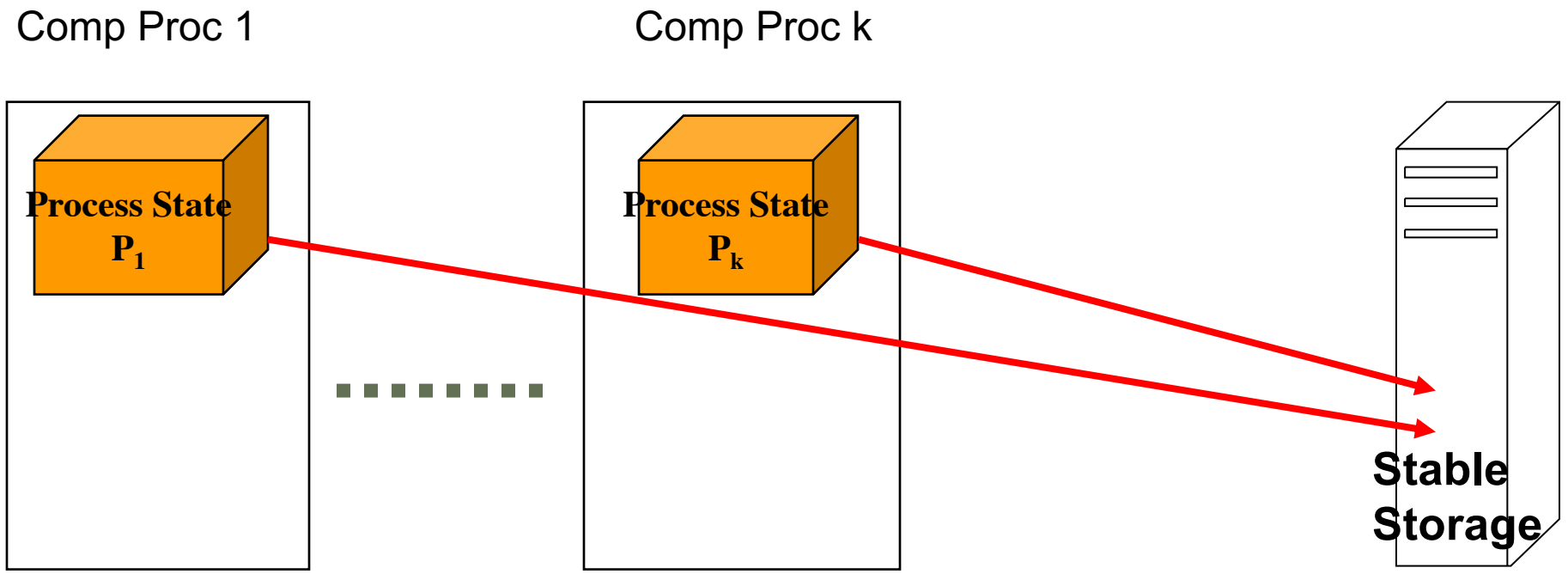
Importance of Improving Checkpointing Performance of Iterative Methods

- Scientific simulations involving PDEs
  - Solve linear systems within each timestep
  - Sparse linear systems include most of the variables
  - E.g., 3D CFD problems from Navier-Stokes equations
    - *Semi-Implicit Method for Pressure-Linked Equations* (SIMPLE) algorithm
    - 5 out of 9 fluid-flow variables need to be checkpointed in iterative method
- Significantly Improve Checkpointing Performance of Iterative methods

  → Significantly Improve Application Performance



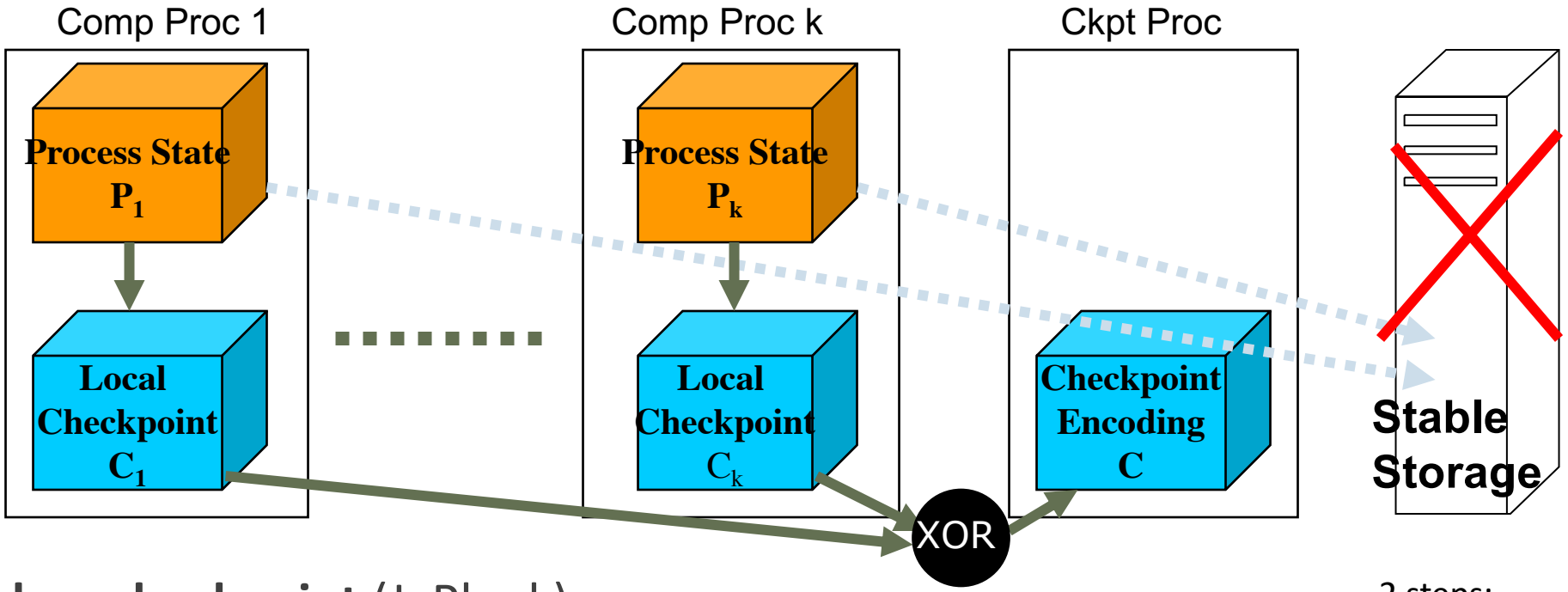
State-of-the-Art: Failure-Stop Failure



Checkpoint/Restart Model

- Periodical checkpoint to file system is **expensive**
- Difficult to **scale up** due to bottleneck of I/O bandwidth

State-of-the-Art: Failure-Stop Failure



Diskless checkpoint (J. Plank)

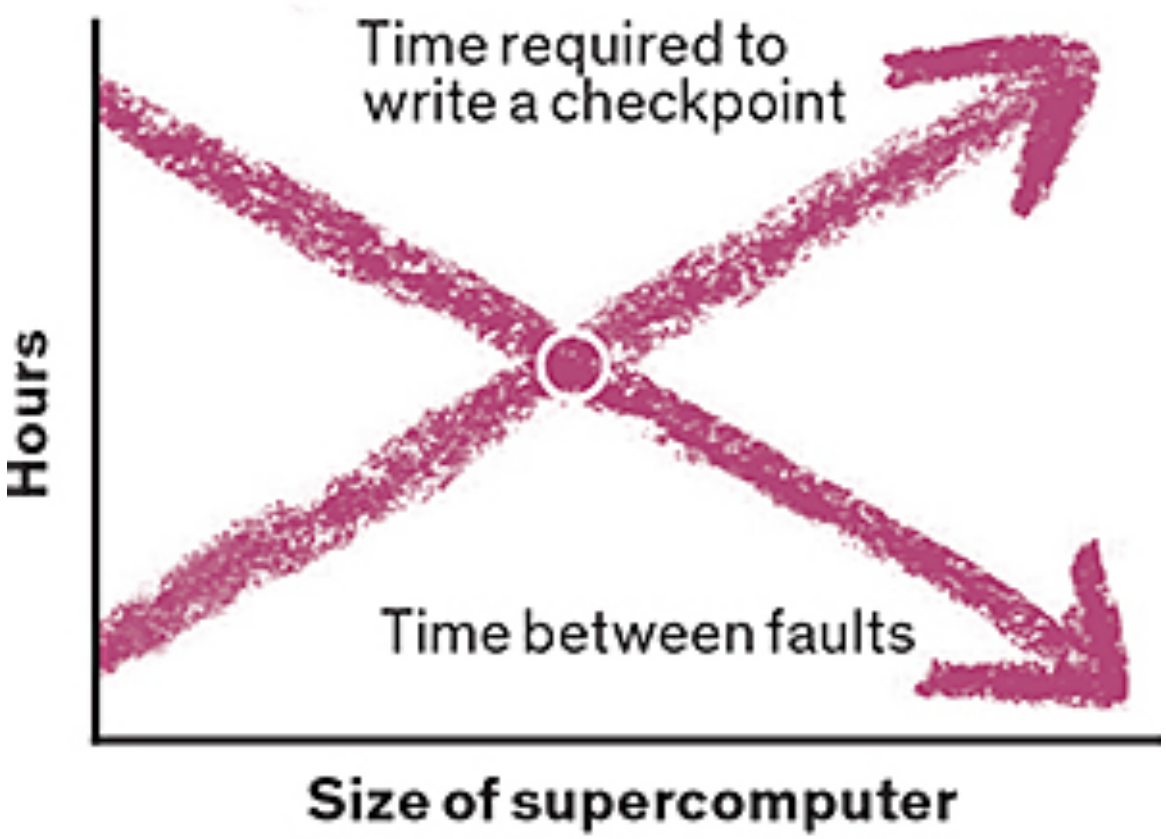
- More scalable (pros)
- **2X** or more memory overhead (cons) → Reduce **usable memory** and **problem size**
- Only able to tolerate with partial failures , not for a whole system failure (cons)
- Requires spare nodes and dedicates processors (cons)

$C_1 + \ldots + C_n = C$

2 steps:

1. Checkpointing state of each application processor in memory
2. Encoding these in-memory checkpoints and storing the encodings in checkpointing processors

Failures and Checkpointing

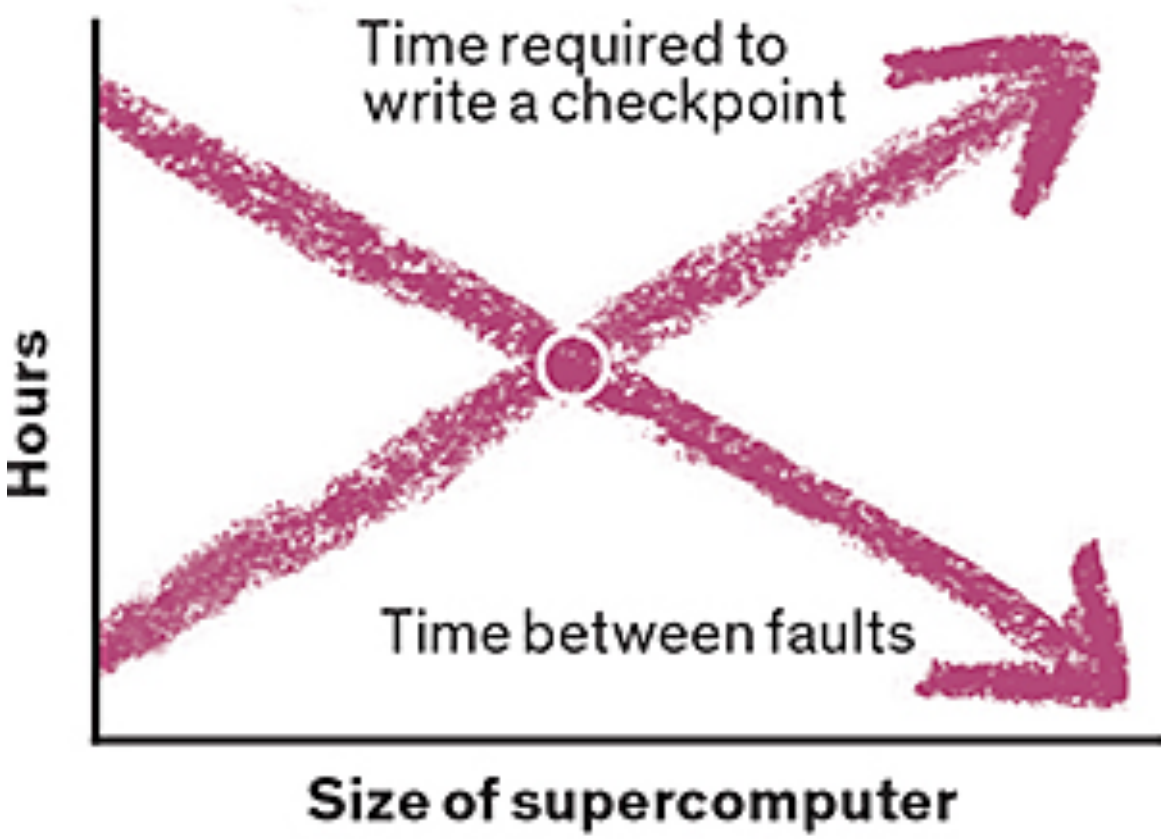


Optimized Techniques to Improve *Scalability* of Checkpoint

- Diskless checkpoint
- Multi-level checkpoint
- Asynchronized checkpoint
- Lossless-compressed checkpoint
- ……

Question: *Can we use lossy compression to (1) reduce checkpointing size and overhead and (2) improve the performance and scalability?*

Failures and Checkpointing



Question: *Can we use lossy compression to (1) reduce checkpointing size and overhead and (2) improve the performance and scalability?*

→ **Lossy checkpointing**

Two important questions:
(1) What is the *impact* of the lossy checkpointing data on the execution performance?
(2) Can lossy checkpointing actually *improve* the overall performance (including C/R and lossy compression) in the context of restarting with alternated data?

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Traditional Checkpointing for Iterative Methods

- Checkpoint
  1. Checkpoint static variables (e.g., **A**, **M**) at the beginning
  2. Checkpoint dynamic variables (e.g., i, ρ, ***p***, ***x***) every several iterations

Algorithm 1 Fault-tolerant preconditioned conjugate gradient (PCG) algorithm with traditional checkpointing.

Input: linear system matrix A, preconditioner M, and right-hand side vector b
Output: approximate solution x

1: Compute $r^{(0)} = b - Ax^{(0)}$, $z^{(0)} = M^{-1}r^{(0)}$, $p^{(0)} = z^{(0)}$, $\rho_0 = {r^{(0)}}^T z^{(0)}$ for some initial guess $x^{(0)}$
2: **for** $i = 0, 1, \cdots$ **do**
3: **if** $((i > 0)$ and $(i\%ckpt\_intvl = 0))$ **then**
4: *Checkpoint:* i, ρ_i *and* $p^{(i)}, x^{(i)}$
5: **end if**
6: **if** $((i > 0)$ and (recover)) **then**
7: *Recover:* $A, M, i, \rho_i, p^{(i)}, x^{(i)}$
8: Compute $r^{(i)} = b - Ax^{(i)}$
9: **end if**
10: $q^{(i)} = Ap^{(i)}$
11: $\alpha_i = \rho_i / {p^{(i)}}^T q^{(i)}$
12: $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$
13: $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$
14: solve $Mz^{(i+1)} = r^{(i+1)}$
15: $\rho_{i+1} = {r^{(i+1)}}^T z^{(i+1)}$
16: $\beta_i = \rho_{i+1} / \rho_i$
17: $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
18: check convergence; continue if necessary
19: **end for**

Traditional Checkpointing for Iterative Methods

- Checkpoint
  1. Checkpoint static variables (e.g., **A**, **M**) at the beginning
  2. Checkpoint dynamic variables (e.g., i, ρ, ***p***, ***x***) every several iterations
- Recovery
  1. Recover a correct computational environment
  2. Recover static variables
  3. Recover dynamic variables
  4. Recover recomputed variables (e.g., ***r***)

Algorithm 1 Fault-tolerant preconditioned conjugate gradient (PCG) algorithm with traditional checkpointing.

Input: linear system matrix A, preconditioner M, and right-hand side vector b
Output: approximate solution x

1: Compute $r^{(0)} = b - Ax^{(0)}$, $z^{(0)} = M^{-1}r^{(0)}$, $p^{(0)} = z^{(0)}$, $\rho_0 = r^{(0)^T} z^{(0)}$ for some initial guess $x^{(0)}$
2: **for** $i = 0, 1, \cdots$ **do**
3: **if** $((i > 0)$ and $(i\%ckpt\_intvl = 0))$ **then**
4: *Checkpoint:* i, ρ_i *and* $p^{(i)}, x^{(i)}$
5: ~~**end if**~~
6: **if** $((i > 0)$ and (recover)) **then**
7: *Recover:* $A, M, i, \rho_i, p^{(i)}, x^{(i)}$
8: Compute $r^{(i)} = b - Ax^{(i)}$
9: ~~**end if**~~
10: $q^{(i)} = Ap^{(i)}$
11: $\alpha_i = \rho_i / p^{(i)^T} q^{(i)}$
12: $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$
13: $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$
14: solve $Mz^{(i+1)} = r^{(i+1)}$
15: $\rho_{i+1} = r^{(i+1)T} z^{(i+1)}$
16: $\beta_i = \rho_{i+1} / \rho_i$
17: $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
18: check convergence; continue if necessary
19: **end for**

Traditional Checkpointing for Iterative Methods

- Checkpoint
  1. Checkpoint static variables (e.g., **A**, **M**) at the beginning
  2. Checkpoint dynamic variables (e.g., i, ρ, ***p***, ***x***) every several iterations
- Recovery
  1. Recover a correct computational environment
  2. Recover static variables
  3. Recover dynamic variables
  4. Recover recomputed variables (e.g., ***r***)
- C/R cost dominated by dynamic variables
  - Static variables not checkpointed along iterations (at most once)
  - Static variables: linear system matrix **A** and preconditioner **M**
    - **A** usually has 1x ~ 10x nnz than dynamic variables' size (i.e., vector size)
    - **M** is much sparse than A, e.g., block Jacobi, ILU
  - Checkpoint frequency is usually much higher than failure rate
    - MTTI = 4 hrs., $\text{Time}_{\text{ckpt}}$ = 18 s ➔ Checkpoint interval (*Young' formula*) = 12 mins
    - Checkpoint frequency is **30x** higher than recovery frequency

Algorithm 1 Fault-tolerant preconditioned conjugate gradient (PCG) algorithm with traditional checkpointing.

Input: linear system matrix A, preconditioner M, and right-hand side vector b
Output: approximate solution x

1: Compute $r^{(0)} = b - Ax^{(0)}$, $z^{(0)} = M^{-1}r^{(0)}$, $p^{(0)} = z^{(0)}$, $\rho_0 = r^{(0)^T} z^{(0)}$ for some initial guess $x^{(0)}$
2: **for** $i = 0, 1, \cdots$ **do**
3: **if** $((i > 0)$ and $(i\%ckpt\_intvl = 0))$ **then**
4: *Checkpoint:* i, ρ_i *and* $p^{(i)}, x^{(i)}$
5: **end if**
6: **if** $((i > 0)$ and (recover)) **then**
7: *Recover:* $A, M, i, \rho_i, p^{(i)}, x^{(i)}$
8: Compute $r^{(i)} = b - Ax^{(i)}$
9: **end if**
10: $q^{(i)} = Ap^{(i)}$
11: $\alpha_i = \rho_i / p^{(i)^T} q^{(i)}$
12: $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$
13: $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$
14: solve $Mz^{(i+1)} = r^{(i+1)}$
15: $\rho_{i+1} = r^{(i+1)T} z^{(i+1)}$
16: $\beta_i = \rho_{i+1} / \rho_i$
17: $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
18: check convergence; continue if necessary
19: **end for**

Traditional Checkpointing for Iterative Methods

➢Checkpoint

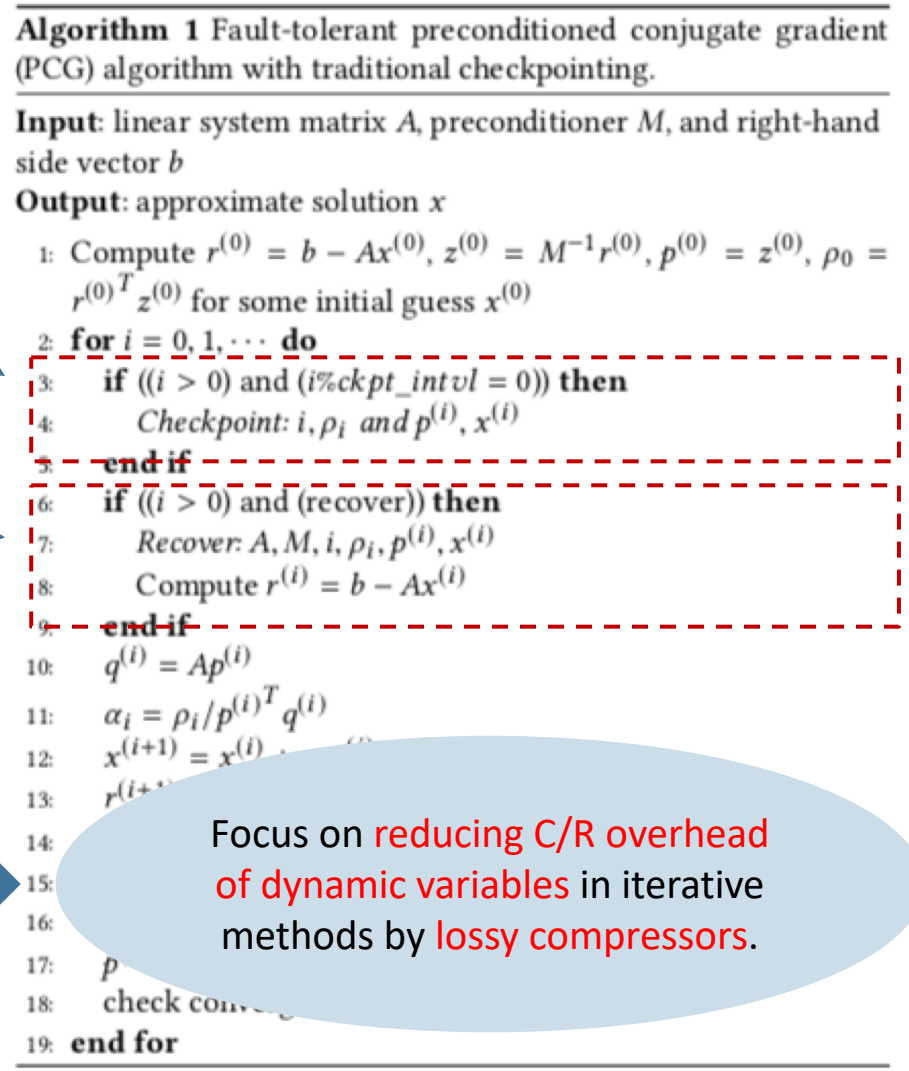
1. Checkpoint static variables (e.g., **A**, **M**) at the beginning
2. Checkpoint dynamic variables (e.g., i, ρ, ***p***, ***x***) every several iterations

➢Recovery

1. Recover a correct computational environment
2. Recover static variables
3. Recover dynamic variables
4. Recover recomputed variables (e.g., ***r***)

➢C/R cost dominated by dynamic variables

- Static variables not checkpointed along iterations (at most once)
- Static variables: linear system matrix **A** and preconditioner **M**
  - **A** usually has 1x ~ 10x nnz than dynamic variables' size (i.e., vector size)
  - **M** is much sparse than A, e.g., block Jacobi, ILU
- Checkpoint frequency is usually much higher than failure rate
  - MTTI = 4 hrs., $Time_{ckpt}$ = 18 s ➔ Checkpoint interval (*Young' formula*) = 12 mins
  - Checkpoint frequency is **30x** higher than recovery frequency



Algorithm 1 Fault-tolerant preconditioned conjugate gradient (PCG) algorithm with traditional checkpointing.

Input: linear system matrix A, preconditioner M, and right-hand side vector b
Output: approximate solution x

1: Compute $r^{(0)} = b - Ax^{(0)}$, $z^{(0)} = M^{-1}r^{(0)}$, $p^{(0)} = z^{(0)}$, $\rho_0 = r^{(0)^T} z^{(0)}$ for some initial guess $x^{(0)}$
2: **for** $i = 0, 1, \cdots$ **do**
3: **if** $((i > 0)$ and $(i\%ckpt\_intvl = 0))$ **then**
4: *Checkpoint:* i, ρ_i *and* $p^{(i)}, x^{(i)}$
5: **end if**
6: **if** $((i > 0)$ and (recover)) **then**
7: *Recover:* $A, M, i, \rho_i, p^{(i)}, x^{(i)}$
8: Compute $r^{(i)} = b - Ax^{(i)}$
9: **end if**
10: $q^{(i)} = Ap^{(i)}$
11: $\alpha_i = \rho_i / p^{(i)^T} q^{(i)}$
12: $x^{(i+1)} = x^{(i)}$
13:
14:
15:
16:
17: p
18: check con
19: **end for**

Focus on reducing C/R overhead of dynamic variables in iterative methods by lossy compressors.

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Theoretical Analysis of Checkpointing Overhead for Iterative Methods

- **Overall execution time** $T_t = NT_{it} + T_{ckp}\frac{N}{k} + \frac{T_t}{T_f}(T_{rc} + T_{rb})$

Iteration time — Checkpoint time — Recover/rollback time

T_{it}	Mean time of an iteration
T_{ckp}	Mean time to perform a checkpoint
T_{rc}	Mean time to recover the application with the correct environment and data from the last checkpoint
T_{rb}	Mean time to perform a rollback of some redundant computations
T_f	Mean time to interruption
$T_{overhead}^{CR}$	Mean time overhead of checkpoint/recovery
λ	Failure rate, i.e, $1/T_f$
k	Checkpoint frequency - a checkpoint is performed every k iterations
N	Number of iterations to converge without failures

Theoretical Analysis of Checkpointing Overhead for Iterative Methods

- **Overall execution time** $T_t = NT_{it} + T_{ckp}\frac{N}{k} + \frac{T_t}{T_f}(T_{rc} + T_{rb})$

  Iteration time — Checkpoint time — Recover/rollback time

- Based on Young's formula $k \cdot T_{it} = \sqrt{2T_f \cdot T_{ckp}}$ and Expected mean time of a rollback $T_{rb} = kT_{it}/2$

- Overall time can be simplified to $T_t = \frac{NT_{it}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{rc}}$

T_{it}	Mean time of an iteration
T_{ckp}	Mean time to perform a checkpoint
T_{rc}	Mean time to recover the application with the correct environment and data from the last checkpoint
T_{rb}	Mean time to perform a rollback of some redundant computations
T_f	Mean time to interruption
$T_{overhead}^{CR}$	Mean time overhead of checkpoint/recovery
λ	Failure rate, i.e, $1/T_f$
k	Checkpoint frequency - a checkpoint is performed every k iterations
N	Number of iterations to converge without failures

Theoretical Analysis of Checkpointing Overhead for Iterative Methods

- **Overall execution time** $T_t = NT_{it} + T_{ckp}\frac{N}{k} + \frac{T_t}{T_f}(T_{rc} + T_{rb})$

  Iteration time — Checkpoint time — Recover/rollback time

- Based on Young's formula $k \cdot T_{it} = \sqrt{2T_f \cdot T_{ckp}}$ and Expected mean time of a rollback $T_{rb} = kT_{it}/2$

- Overall time can be simplified to $T_t = \frac{NT_{it}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{rc}}$

- Fault tolerance overhead $T_{overhead}^{CR} = T_t - NT_{it} = NT_{it} \cdot \frac{\sqrt{2\lambda T_{ckp}} + \lambda T_{rc}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{rc}}$

- Fault tolerance overhead (%) $\frac{T_{overhead}^{CR}}{NT_{it}} = \frac{\sqrt{2\lambda T_{ckp}} + \lambda T_{ckp}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{ckp}}$ (assume $T_{ckp} \sim T_{rc}$)

T_{it}	Mean time of an iteration
T_{ckp}	Mean time to perform a checkpoint
T_{rc}	Mean time to recover the application with the correct environment and data from the last checkpoint
T_{rb}	Mean time to perform a rollback of some redundant computations
T_f	Mean time to interruption
$T_{overhead}^{CR}$	Mean time overhead of checkpoint/recovery
λ	Failure rate, i.e, $1/T_f$
k	Checkpoint frequency - a checkpoint is performed every k iterations
N	Number of iterations to converge without failures

Theoretical Analysis of Checkpointing Overhead for Iterative Methods

T_{it}	Mean time of an iteration
T_{ckp}	Mean time to perform a checkpoint
T_{rc}	Mean time to recover the application with the correct environment and data from the last checkpoint
T_{rb}	Mean time to perform a rollback of some redundant computations
T_f	Mean time to interruption
$T_{overhead}^{CR}$	Mean time overhead of checkpoint/recovery
λ	Failure rate, i.e, $1/T_f$
k	Checkpoint frequency - a checkpoint is performed every k iterations
N	Number of iterations to converge without failures

- **Overall execution time** $T_t = NT_{it} + T_{ckp}\frac{N}{k} + \frac{T_t}{T_f}(T_{rc} + T_{rb})$

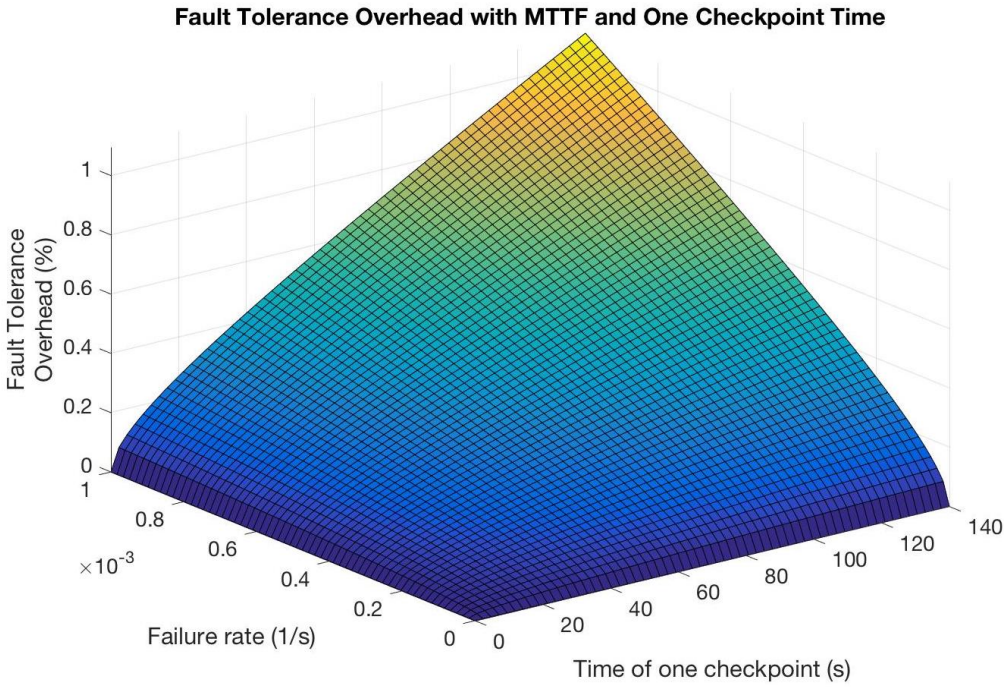
  Iteration time — Checkpoint time — Recover/rollback time

- Based on Young's formula $k \cdot T_{it} = \sqrt{2T_f \cdot T_{ckp}}$ and Expected mean time of a rollback $T_{rb} = kT_{it}/2$

- Overall time can be simplified to $T_t = \frac{NT_{it}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{rc}}$

- Fault tolerance overhead $T_{overhead}^{CR} = T_t - NT_{it} = NT_{it} \cdot \frac{\sqrt{2\lambda T_{ckp}} + \lambda T_{rc}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{rc}}$

- Fault tolerance overhead (%) $\frac{T_{overhead}^{CR}}{NT_{it}} = \frac{\sqrt{2\lambda T_{ckp}} + \lambda T_{ckp}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{ckp}}$



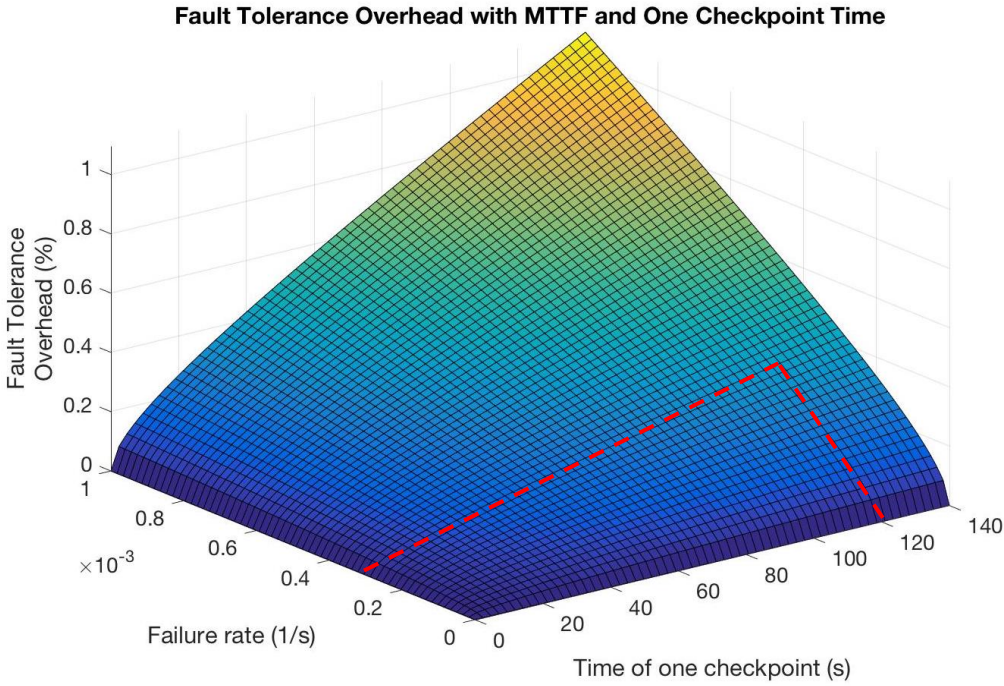
Theoretical Analysis of Checkpointing Overhead for Iterative Methods

T_{it}	Mean time of an iteration
T_{ckp}	Mean time to perform a checkpoint
T_{rc}	Mean time to recover the application with the correct environment and data from the last checkpoint
T_{rb}	Mean time to perform a rollback of some redundant computations
T_f	Mean time to interruption
$T_{overhead}^{CR}$	Mean time overhead of checkpoint/recovery
λ	Failure rate, i.e, $1/T_f$
k	Checkpoint frequency - a checkpoint is performed every k iterations
N	Number of iterations to converge without failures

- **Overall execution time** $T_t = NT_{it} + T_{ckp}\frac{N}{k} + \frac{T_t}{T_f}(T_{rc} + T_{rb})$

  Iteration time | Checkpoint time | Recover/rollback time

- Based on Young's formula $k \cdot T_{it} = \sqrt{2T_f \cdot T_{ckp}}$ and Expected mean time of a rollback $T_{rb} = kT_{it}/2$
- Overall time can be simplified to $T_t = \frac{NT_{it}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{rc}}$
- Fault tolerance overhead (%) $\frac{T_{overhead}^{CR}}{NT_{it}} = \frac{\sqrt{2\lambda T_{ckp}} + \lambda T_{ckp}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{ckp}}$
- For example, MTTI is 1 hour ($\lambda = 2.7 \times 10^{-4}$)
  - T_{ckpt} = 120 s, expected FT overhead ~ 40%
    - Checkpoint ***x*** (GMRES) on 64 nodes (2,048 cores) on Bebop at ANL



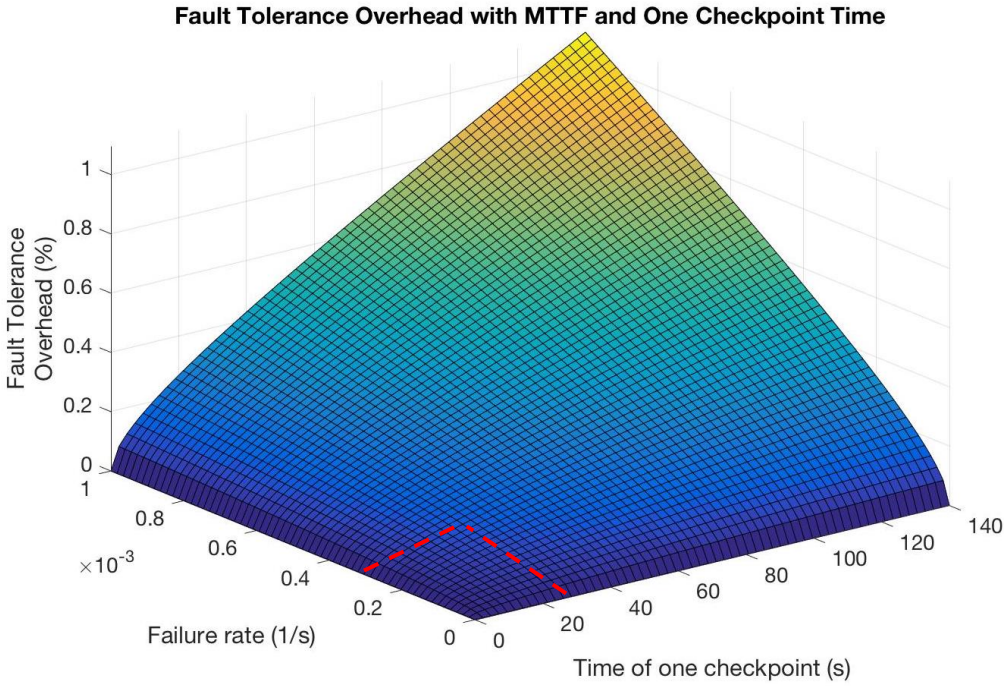
Theoretical Analysis of Checkpointing Overhead for Iterative Methods

T_{it}	Mean time of an iteration
T_{ckp}	Mean time to perform a checkpoint
T_{rc}	Mean time to recover the application with the correct environment and data from the last checkpoint
T_{rb}	Mean time to perform a rollback of some redundant computations
T_f	Mean time to interruption
$T_{overhead}^{CR}$	Mean time overhead of checkpoint/recovery
λ	Failure rate, i.e, $1/T_f$
k	Checkpoint frequency - a checkpoint is performed every k iterations
N	Number of iterations to converge without failures

- **Overall execution time** $T_t = NT_{it} + T_{ckp}\frac{N}{k} + \frac{T_t}{T_f}(T_{rc} + T_{rb})$

  Iteration time | Checkpoint time | Recover/rollback time

- Based on Young's formula $k \cdot T_{it} = \sqrt{2T_f \cdot T_{ckp}}$ and Expected mean time of a rollback $T_{rb} = kT_{it}/2$
- Overall time can be simplified to $T_t = \frac{NT_{it}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{rc}}$
- Fault tolerance overhead (%) $\frac{T_{overhead}^{CR}}{NT_{it}} = \frac{\sqrt{2\lambda T_{ckp}} + \lambda T_{ckp}}{1 - \sqrt{2\lambda T_{ckp}} - \lambda T_{ckp}}$
- For example, MTTI is 1 hour ($\lambda = 2.7 \times 10^{-4}$)
  - T_{ckpt} = 120 s, expected FT overhead ~ 40%
    - Checkpoint ***x*** (GMRES) on 64 nodes (2,048 cores) on Bebop at ANL
  - T_{ckpt} = 25 s, expected FT overhead ~ 14% (significantly reduced!)



Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Lossy Checkpointing Scheme for Iterative Methods

- Lossy checkpointing scheme for iterative methods has two steps
  - Compress dynamic variables with lossy compressor before each checkpointing
  - Decompress compressed dynamic variables after each recovering

Algorithm 2 Fault-tolerant preconditioned conjugate gradient algorithm with lossy checkpointing technique

Input: linear system matrix A, preconditioner M, and right-hand side vector b

Output: approximate solution x

1: Initialization: same as line 1 in Algorithm 1
2: **for** $i = 0, 1, \cdots$ **do**
3: **if** $((i > 0)$ and $(i\%ckpt\_intvl = 0))$ **then**
4: **Compress:** $x^{(i)}$ **with lossy compressor**
5: *Checkpoint:* i *and compressed* $x^{(i)}$
6: **end if**
7: **if** $((i > 0)$ and (recover)) **then**
8: *Recover:* A, M, i *and compressed* $x^{(i)}$
9: **Decompress:** $x^{(i)}$ **with lossy compressor**
10: Compute $r^{(i)} = b - Ax^{(i)}$
11: Solve $Mz^{(i)} = r^{(i)}$
12: $p^{(i)} = z^{(i)}$
13: $\rho_i = {r^{(i)}}^T z^{(i)}$
14: **end if**
15: Computation: same as lines 10–17 in Algorithm 1
16: **end for**

Lossy Checkpointing Scheme for Iterative Methods

- Lossy checkpointing scheme for iterative methods has two steps
  - Compress dynamic variables with lossy compressor before each checkpointing
  - Decompress compressed dynamic variables after each recovering
- Orthogonality dependent iterative methods
  - For example, CG maintains a series of orthogonality relations
    - $\mathbf{p}^{(k)}$ and $\mathbf{Aq}^{(j)}$, $\mathbf{r}^{(k)}$ and $\mathbf{p}^{(j)}$, $\mathbf{r}^{(k)}$ and $\mathbf{r}^{(j)}$ for any $j < k$
  - CG's superlinear convergence relies on these orthogonality
  - CG after lossy checkpointing may lose superlinear convergence

Algorithm 2 Fault-tolerant preconditioned conjugate gradient algorithm with lossy checkpointing technique

Input: linear system matrix A, preconditioner M, and right-hand side vector b

Output: approximate solution x

1: Initialization: same as line 1 in Algorithm 1
2: **for** $i = 0, 1, \cdots$ **do**
3: **if** $((i > 0)$ and $(i\%ckpt\_intvl = 0))$ **then**
4: **Compress:** $x^{(i)}$ **with lossy compressor**
5: *Checkpoint:* i *and compressed* $x^{(i)}$
6: **end if**
7: **if** $((i > 0)$ and (recover)) **then**
8: *Recover:* A, M, i *and compressed* $x^{(i)}$
9: **Decompress:** $x^{(i)}$ **with lossy compressor**
10: Compute $r^{(i)} = b - Ax^{(i)}$
11: Solve $Mz^{(i)} = r^{(i)}$
12: $p^{(i)} = z^{(i)}$
13: $\rho_i = {r^{(i)}}^T z^{(i)}$
14: **end if**
15: Computation: same as lines 10–17 in Algorithm 1
16: **end for**

Lossy Checkpointing Scheme for Iterative Methods

- Lossy checkpointing scheme for iterative methods has two steps
  - Compress dynamic variables with lossy compressor before each checkpointing
  - Decompress compressed dynamic variables after each recovering
- Orthogonality dependent iterative methods
  - For example, CG maintains a series of orthogonality relations
    - $\mathbf{p}^{(k)}$ and $\mathbf{Aq}^{(j)}$, $\mathbf{r}^{(k)}$ and $\mathbf{p}^{(j)}$, $\mathbf{r}^{(k)}$ and $\mathbf{r}^{(j)}$ for any $j < k$
  - CG's superlinear convergence relies on these orthogonality
  - CG after lossy checkpointing may lose superlinear convergence
- Restarted scheme
  - Periodically treat current approximate solution as new initial guess
  - **Advantages**
    - Less time and space complexity, such as GMRES ~ $O(N^2)$, where N is time step
    - Restarted scheme may not delay but even accelerate the convergence (jump out of local search)

Algorithm 2 Fault-tolerant preconditioned conjugate gradient algorithm with lossy checkpointing technique

Input: linear system matrix A, preconditioner M, and right-hand side vector b

Output: approximate solution x

1: Initialization: same as line 1 in Algorithm 1
2: **for** $i = 0, 1, \cdots$ **do**
3: **if** $((i > 0)$ and $(i\%ckpt\_intvl = 0))$ **then**
4: **Compress:** $x^{(i)}$ **with lossy compressor**
5: *Checkpoint:* i *and compressed* $x^{(i)}$
6: **end if**
7: **if** $((i > 0)$ and (recover)) **then**
8: *Recover:* A, M, i *and compressed* $x^{(i)}$
9: **Decompress:** $x^{(i)}$ **with lossy compressor**
10: Compute $r^{(i)} = b - Ax^{(i)}$
11: Solve $Mz^{(i)} = r^{(i)}$
12: $p^{(i)} = z^{(i)}$
13: $\rho_i = {r^{(i)}}^T z^{(i)}$
14: **end if**
15: Computation: same as lines 10–17 in Algorithm 1
16: **end for**

Lossy Checkpointing Scheme for Iterative Methods

- Lossy checkpointing scheme for iterative methods has two steps
  - Compress dynamic variables with lossy compressor before each checkpointing
  - Decompress compressed dynamic variables after each recovering
- Restarted scheme
  - Periodically treat current approximate solution as new initial guess
  - Advantages
    - Less time and space complexity, such as GMRES ~ $O(N^2)$, where N is time step
    - Restarted scheme may not delay but even accelerate the convergence (jump out of local search)
- Lossy checkpointing with restarted scheme
  - Checkpoint only approximate solution $\boldsymbol{x_i}$
  - Lossy decompressed $\boldsymbol{x_i}$ as new initial guess
  - Reconstruct orthogonal relations and superlinear convergence

Algorithm 2 Fault-tolerant preconditioned conjugate gradient algorithm with lossy checkpointing technique

Input: linear system matrix A, preconditioner M, and right-hand side vector b

Output: approximate solution x

1: Initialization: same as line 1 in Algorithm 1
2: **for** $i = 0, 1, \cdots$ **do**
3: **if** $((i > 0)$ and $(i\%ckpt\_intvl = 0))$ **then**
4: **Compress:** $x^{(i)}$ **with lossy compressor**
5: *Checkpoint:* i *and compressed* $x^{(i)}$
6: **end if**
7: **if** $((i > 0)$ and (recover)) **then**
8: *Recover:* A, M, i *and compressed* $x^{(i)}$
9: **Decompress:** $x^{(i)}$ **with lossy compressor**
10: Compute $r^{(i)} = b - Ax^{(i)}$
11: Solve $Mz^{(i)} = r^{(i)}$
12: $p^{(i)} = z^{(i)}$
13: $\rho_i = {r^{(i)}}^T z^{(i)}$
14: **end if**
15: Computation: same as lines 10–17 in Algorithm 1
16: **end for**

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Performance Model of Lossy Checkpointing

- **Overall execution time** $T_t = NT_{it} + T_{ckp}^{lossy}\frac{N}{k} + \frac{T_t}{T_f}(N'T_{it} + T_{rc}^{lossy} + T_{rb})$

Mean extra iterations to convergence caused by one lossy recovery

Iteration time

Lossy checkpoint time

Recover/rollback time

T_{comp}	Mean time of performing lossy compression
T_{decomp}	Mean time of performing lossy decompression
T_{ckp}^{trad}	Mean time of performing one traditional checkpoint
T_{ckp}^{lossy}	Mean time of performing a lossy checkpointing
$T_{overhead}^{lossyCR}$	Time overhead of performing lossy checkpoint/recovery
N'	Mean number of extra iterations caused by per lossy recovery

Performance Model of Lossy Checkpointing

- **Overall execution time** $T_t = NT_{it} + T_{ckp}^{lossy}\frac{N}{k} + \frac{T_t}{T_f}(N'T_{it} + T_{rc}^{lossy} + T_{rb})$

Mean extra iterations to convergence caused by one lossy recovery

Iteration time

Lossy checkpoint time

Restart/rollback time

T_{comp}	Mean time of performing lossy compression
T_{decomp}	Mean time of performing lossy decompression
T_{ckp}^{trad}	Mean time of performing one traditional checkpoint
T_{ckp}^{lossy}	Mean time of performing a lossy checkpointing
$T_{overhead}^{lossyCR}$	Time overhead of performing lossy checkpoint/recovery
N'	Mean number of extra iterations caused by per lossy recovery

- Similarly, overall time can be simplified to $T_t = \frac{NT_{it}}{1 - \sqrt{2\lambda T_{ckp}^{lossy}} - \lambda T_{rc}^{lossy} - \lambda N'T_{it}}$

- Fault tolerance overhead of lossy checkpointing

$$T_{overhead}^{lossyCR} = NT_{it} \cdot \frac{\sqrt{2\lambda T_{ckp}^{lossy}} + \lambda T_{rc}^{lossy} + \lambda N'T_{it}}{1 - \sqrt{2\lambda T_{ckp}^{lossy}} - \lambda T_{rc}^{lossy} - \lambda N'T_{it}}$$

Theoretical Analysis of N' for Performance Gain

To have the lossy checkpointing overhead lower than that of traditional checkpointing: $T_{overhead}^{lossyCR} < T_{overhead}^{CR}$

$$\frac{\sqrt{2\lambda T_{ckp}^{lossy}}+\lambda T_{ckp}^{lossy}+\lambda N' T_{it}}{1-\sqrt{2\lambda T_{ckp}^{lossy}}-\lambda T_{ckp}^{lossy}-\lambda N' T_{it}} \leq \frac{\sqrt{2\lambda T_{ckp}^{trad}}+\lambda T_{ckp}^{trad}}{1-\sqrt{2\lambda T_{ckp}^{trad}}-\lambda T_{ckp}^{trad}}$$

$$N' \leq \frac{(\sqrt{2\lambda T_{ckp}^{trad}}+\lambda T_{ckp}^{trad})-(\sqrt{2\lambda T_{ckp}^{lossy}}+\lambda T_{ckp}^{lossy})}{\lambda T_{it}}$$

THEOREM 1. *Denote λ and T_{it} by the expected failure rate and expected execution time of an iteration, respectively. The lossy checkpointing scheme will improve the execution performance for an iterative method as long as the following inequality holds.*

$$N' \leq (f(T_{ckp}^{trad}, \lambda) - f(T_{ckp}^{trad}, \lambda))/(\lambda T_{it}), \quad (9)$$
$$\text{where } f(t, \lambda) = \sqrt{2\lambda t} + \lambda t$$

How to use Theorem 1?

- For example, MTTI is 1 hour ($\lambda = 2.7 \times 10^{-4}$)
- Lossy compression reduces T_{ckp} from 120 seconds to 25 seconds
- T_{it} = 1.2 s for GMRES (7160 s with 5875 itrs)
- Based on Theorem 1, lossy checkpointing is worthwhile if N' <= 500
- If one lossy recovery causes 500 (~ 9% of total itrs) or fewer extra itrs to converge, lossy checkpointing can improve overall performance

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Impact Analysis of Lossy Checkpointing on Iterative Methods

- Stationary Iterative Methods
- Conjugate Gradient (CG) Method
- Generalized Minimum Residual (GMRES) Method

Impact Analysis of Lossy Checkpointing on Iterative Methods

- Stationary Iterative Methods
  - Most classic
- Conjugate Gradient (CG) Method
  - Most popular for SPD systems
- Generalized Minimum Residual (GMRES) Method
  - Most general (asymmetric, indefinite, …), robust

Impact Analysis of Lossy Checkpointing on Iterative Methods — Stationary Iteration

- Stationary iterative methods: $x^{(i)} = Gx^{(i-1)} + c$
- $||x^{(i)} - x^*|| \approx R^i \cdot ||x^{(0)} - x^*||$
  - R is the spectral radius of matrix G (the largest eigenvalue of G, R < 1)
  - x^* is the exact solution, $x^{(0)}$ is the initial guess

Impact Analysis of Lossy Checkpointing on Iterative Methods — Stationary Iteration

- Stationary iterative methods: $x^{(i)} = Gx^{(i-1)} + c$
- $||x^{(i)} - x^*|| \approx R^i \cdot ||x^{(0)} - x^*||$
  - R is the spectral radius of matrix G (the largest eigenvalue of G, R < 1)
- If stationary methods encounter a failure and restarts at t^{th} iteration
- Lossy compression introduces an error vector ***e*** with a relative error bound eb
  - $\left|x_i^{(t)} - x'^{(t)}_i\right| \leq eb \cdot |x_i^{(t)}|$ for $1 \leq i \leq n$
- Computation restarts from alternated vector $x'^{(t)} = x^{(i)} + e$

Impact Analysis of Lossy Checkpointing on Iterative Methods — Stationary Iteration

- Stationary iterative methods: $x^{(i)} = Gx^{(i-1)} + c$
- $||x^{(i)} - x^*|| \approx R^i \cdot ||x^{(0)} - x^*||$
  - R is the spectral radius of matrix G (the largest eigenvalue of G, R < 1)
- If stationary methods encounter a failure and restarts at t^{th} iteration
- Lossy compression introduces an error vector ***e*** with a relative error bound eb
  - $\left|x_i^{(t)} - x'^{(t)}_i\right| \leq eb \cdot |x_i^{(t)}|$ for $1 \leq i \leq n$
- Computation restarts from alternated vector $x'^{(t)} = x^{(i)} + e$
- After a series of derivations, upper bound of N' is $t - log_R(R^t + eb) \coloneqq ub(t)$

Impact Analysis of Lossy Checkpointing on Iterative Methods — Stationary Iteration

- Stationary iterative methods: $x^{(i)} = Gx^{(i-1)} + c$
- $||x^{(i)} - x^*|| \approx R^i \cdot ||x^{(0)} - x^*||$
  - R is the spectral radius of matrix G (the largest eigenvalue of G, R < 1)
- If stationary methods encounter a failure and restarts at t^{th} iteration
- Lossy compression introduces an error vector ***e*** with a relative error bound eb
  - $\left|x_i^{(t)} - x'^{(t)}_i\right| \leq eb \cdot |x_i^{(t)}|$ for $1 \leq i \leq n$
- Computation restarts from alternated vector $x'^{(t)} = x^{(i)} + e$
- After a series of derivations, upper bound of N' is $t - log_R(R^t + eb) \coloneqq ub(t)$
- Expected upper bound of N' falls into $[\frac{N+1}{2} - log_R\left(R^{\frac{N+1}{2}} + eb\right),\ N - log_R(R^N + eb)]$
  - Due to $ub(t)$ is monotonic function, $\mathrm{E}[ub(t)] \leq ub(T)$
  - Due to $ub(t)$ is convex function, $\mathrm{E}[ub(t)] \geq ub(E[t])$ (based on Jensen inequality)

THEOREM 2. *Based on the convergence rate (Equation (10)), the expected upper bound of the number of extra iterations for the stationary iterative methods falls into the interval* $[\frac{N+1}{2} - \log_R(R^{\frac{N+1}{2}} + eb),\ N - \log_R(R^N + eb)]$, *where eb is a constant relative error bound and R and N remain the same definitions as in the earlier discussion.*

Impact Analysis of Lossy Checkpointing on Iterative Methods — Stationary Iteration

- Stationary iterative methods: $x^{(i)} = Gx^{(i-1)} + c$
- $||x^{(i)} - x^*|| \approx R^i \cdot ||x^{(0)} - x^*||$
  - R is the spectral radius of matrix G (the largest eigenvalue of G, R < 1)
- If stationary methods encounter a failure and restarts at t^{th} iteration
- Lossy compression introduces an error vector ***e*** with a relative error bound eb
  - $\left|x_i^{(t)} - x'^{(t)}_i\right| \leq eb \cdot |x_i^{(t)}|$ for $1 \leq i \leq n$
- Computation restarts from alternated vector $x'^{(t)} = x^{(i)} + e$
- After a series of derivations, upper bound of N' is $t - log_R(R^t + eb) \coloneqq ub(t)$
- Expected upper bound of N' falls into $[\frac{N+1}{2} - log_R\left(R^{\frac{N+1}{2}} + eb\right), N - log_R(R^N + eb)]$
  - Due to $ub(t)$ is monotonic function, $\mathrm{E}[ub(t)] \leq ub(T)$
  - Due to $ub(t)$ is convex function, $\mathrm{E}[ub(t)] \geq ub(E[t])$ (based on Jensen inequality)

THEOREM 2. *Based on the convergence rate (Equation (10)), the expected upper bound of the number of extra iterations for the stationary iterative methods falls into the interval* $[\frac{N+1}{2} - \log_R(R^{\frac{N+1}{2}} + eb), N - \log_R(R^N + eb)]$, *where eb is a constant relative error bound and R and N remain the same definitions as in the earlier discussion.*

Impact Analysis of Lossy Checkpointing on Iterative Methods — GMRES

- Not easy to analyze N’ for nonstationary methods (unlike stationary methods)

Impact Analysis of Lossy Checkpointing on Iterative Methods — GMRES

- Not easy to analyze N' for nonstationary methods (unlike stationary methods)
- GMRES can converge to the same accuracy with no delay or even exhibit an acceleration sometimes if restarted residual is close to previous residual

J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. *Recovery patterns for iterative methods in a parallel unstable environment.* SIAM Journal on Scientific Computing, 30(1):102–116, 2007.

Impact Analysis of Lossy Checkpointing on Iterative Methods — GMRES

- Not easy to analyze N' for nonstationary methods (unlike stationary methods)
- GMRES can converge to the same accuracy with no delay or even exhibit an acceleration sometimes if restarted residual is close to previous residual

J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. *Recovery patterns for iterative methods in a parallel unstable environment.* SIAM Journal on Scientific Computing, 30(1):102–116, 2007.

1. GMRES is easy to stagnate in practice
2. Lossy recovered data can form a new approximate solution with different spectral properties
3. A failure happened during stagnation may help GMRES jump out of stagnation

Impact Analysis of Lossy Checkpointing on Iterative Methods — GMRES

- Not easy to analyze N' for nonstationary methods (unlike stationary methods)
- GMRES can converge to the same accuracy with no delay or even exhibit an acceleration sometimes if restarted residual is close to previous residual

J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. *Recovery patterns for iterative methods in a parallel unstable environment.* SIAM Journal on Scientific Computing, 30(1):102–116, 2007.

1. GMRES is easy to stagnate in practice
2. Lossy recovered data can form a new approximate solution with different spectral properties
3. A failure happened during stagnation may help GMRES jump out of stagnation

- An adaptive error bound scheme for GMRES
  - Based on Theorem 3: if *eb* is set to $||r^{(t)}||/||b||$, new residual norm is close to the previous residual
  - Error-bound lossy compressors (such as SZ and ZFP) can control the distortion of data within $eb \cdot ||x^{(t)}||$

THEOREM 3. *For the GMRES method, after a restart with lossy checkpointing, the new residual norm is controlled close to or at least on the same order as the previous residual if the relative error bound eb is set to* $O(||r^{(t)}||/||b||)$.

PROOF. Similar to Equation (11), we have the following.

$$\begin{aligned} ||r'^{(t)}|| &= ||b - Ax'^{(t)}|| = ||b - Ax^{(t)} + A(x^{(t)} - x'^{(t)})|| \\ &\le ||r^{(t)}|| + ||Ae|| \le ||r^{(t)}|| + eb \cdot ||Ax^{(t)}|| \\ &= ||r^{(t)}|| + eb \cdot ||b - r^{(t)}|| \le (1 + eb)||r^{(t)}|| + eb \cdot ||b|| \\ &\approx ||r^{(t)}|| + eb \cdot ||b|| \end{aligned} \tag{14}$$

If *eb* is set to $O(||r^{(t)}||/||b||)$, then $eb \cdot ||b||$ is $O(||r^{(t)}||)$; hence, $||r^{(t)}|| + eb \cdot ||b||$ is $O(||r^{(t)}||)$, which means that the new residual norm $||r'^{(t)}||$ will be of the same order as the previous residual norm $||r^{(t)}||$ based on Equation (14). □

Impact Analysis of Lossy Checkpointing on Iterative Methods — GMRES

- Not easy to analyze N' for nonstationary methods (unlike stationary methods)
- GMRES can converge to the same accuracy with no delay or even exhibit an acceleration sometimes if restarted residual is close to previous residual

J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. *Recovery patterns for iterative methods in a parallel unstable environment.* SIAM Journal on Scientific Computing, 30(1):102–116, 2007.

1. GMRES is easy to stagnate in practice
2. Lossy recovered data can form a new approximate solution with different spectral properties
3. A failure happened during stagnation may help GMRES jump out of stagnation

- An adaptive error bound scheme for GMRES
  - Based on Theorem 3: if *eb* is set to $||r^{(t)}||/||b||$, new residual norm is close to the previous residual
  - Error-bound lossy compressors (such as SZ and ZFP) can control the distortion of data within $eb \cdot ||x^{(t)}||$

THEOREM 3. *For the GMRES method, after a restart with lossy checkpointing, the new residual norm is controlled close to or at least on the same order as the previous residual if the relative error bound eb is set to* $O(||r^{(t)}||/||b||)$.

PROOF. Similar to Equation (11), we have the following.

$$||b - Ax'^{(t)}|| = ||b - Ax^{(t)} + A(x^{(t)} - x'^{(t)})||$$
$$\leq ||r^{(t)}|| + ||Ae|| \leq ||r^{(t)}|| + eb \cdot ||Ax^{(t)}||$$
$$\leq ||r^{(t)}|| + eb \cdot ||b - r^{(t)}|| \leq (1 + eb)||r^{(t)}|| + eb \cdot ||b||$$
$$\approx ||r^{(t)}|| + eb \cdot ||b|| \quad (14)$$

If *eb* is set to $O(||r^{(t)}||/||b||)$, then $eb \cdot ||b||$ is $O(||r^{(t)}||)$; hence, $||r^{(t)}|| + eb \cdot ||b||$ is $O(||r^{(t)}||)$, which means that the new residual norm $||r'^{(t)}||$ will be of the same order as the previous residual norm $||r^{(t)}||$ based on Equation (14). □

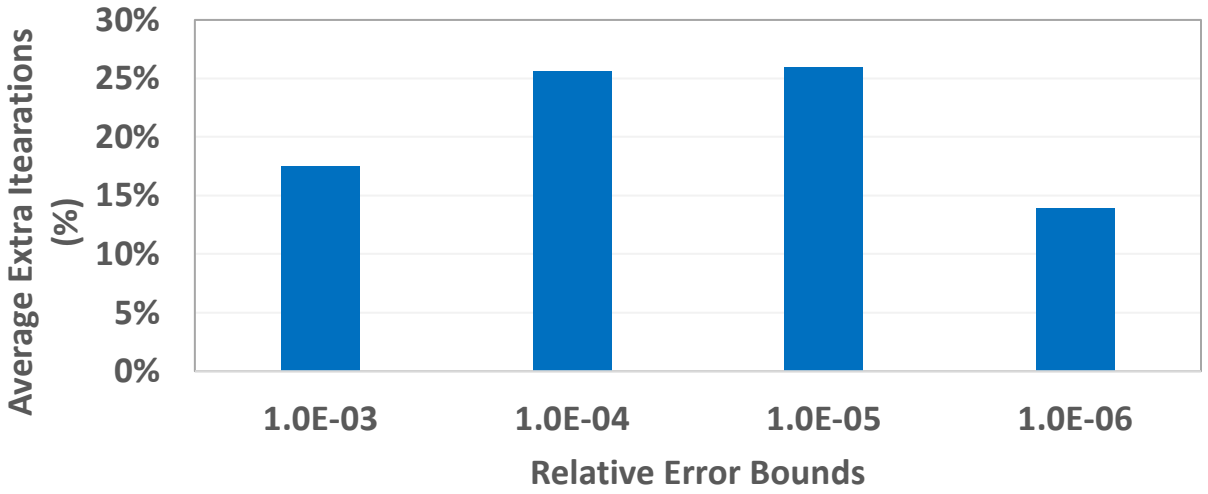
N' = 0 for GMRES

Impact Analysis of Lossy Checkpointing on Iterative Methods — CG

- Extra convergence steps N' for CG exhibit randomness (even if ensure close restarted residual)
- We adopt empirical evaluation for N'
  - Randomly select an iteration to compress and decompress ***x*** in each execution

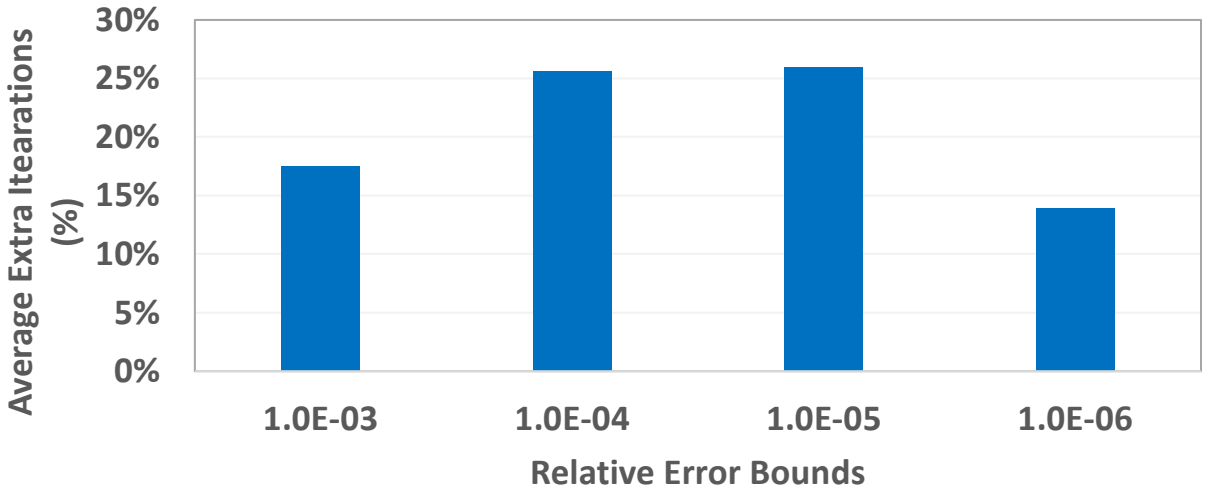
Impact Analysis of Lossy Checkpointing on Iterative Methods — CG

- Extra convergence steps N' for CG exhibit randomness (even if ensure close restarted residual)
- We adopt empirical evaluation for N'
  - Randomly select an iteration to compress and decompress ***x*** in each execution
  - Average N' varies from 10% to 25% with different *eb*



Impact Analysis of Lossy Checkpointing on Iterative Methods — CG

- Extra convergence steps N' for CG exhibit randomness (even if ensure close restarted residual)
- We adopt empirical evaluation for N'
  - Randomly select an iteration to compress and decompress ***x*** in each execution
  - Average N' varies from 10% to 25% with different *eb*



N' = 25%·N for CG if *eb* = 10^{-4}

Performance Evaluation

- Experimental platform
  - 2048 cores from 64 nodes (each node with 2 Intel Xeon E5-2695 v4 processors + 128 GB memory) in Bebop cluster at Argonne
  - I/O and storage are typical high-end supercomputer facilities

- Implementation
  - FTI checkpointing library (v0.9.5)
    - MPI-IO mode to write checkpoint data to PFS
  - SZ lossy compression library (v1.4.12)
    - SZ has better compression performance on 1D data
  - Iterative methods implemented in PETSc (v3.8)
- Experimental Setup
  - Jacobi for stationary methods, CG, and GMRES(30)
  - Default preconditioner (block Jacobi with ILU/IC)
  - *eb* = 10^{-4} for Jacobi and CG, adaptive *eb* for GMRES
  - Relative convergence tolerance of 10^{-4}, 10^{-6}, 10^{-7} for Jacobi, GMRES, CG

Linear System Configuration

- Linear system (arising from 3D Poisson)

$$A_{n^3\times n^3}x_{n^3\times 1} = b_{n^3\times 1},$$

where

$$A_{n^3\times n^3} = \begin{pmatrix} M_{n^2\times n^2} & I_{n^2\times n^2} & & & & \\ I_{n^2\times n^2} & M_{n^2\times n^2} & I_{n^2\times n^2} & & & \\ & \ddots & \ddots & \ddots & & \\ & & I_{n^2\times n^2} & M_{n^2\times n^2} & I_{n^2\times n^2} & \\ & & & I_{n^2\times n^2} & M_{n^2\times n^2} \end{pmatrix},$$

$$M_{n^2\times n^2} = \begin{pmatrix} T_{n\times n} & I_{n\times n} & & & & \\ I_{n\times n} & T_{n\times n} & I_{n\times n} & & & \\ & \ddots & \ddots & \ddots & & \\ & & I_{n\times n} & T_{n\times n} & I_{n\times n} & \\ & & & I_{n\times n} & T_{n\times n} \end{pmatrix},$$

$$T_{n\times n} = \begin{pmatrix} -6 & 1 & & & & \\ 1 & -6 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & -6 & 1 & \\ & & & 1 & -6 \end{pmatrix},$$

- 3D Poisson matrix can increase the problem size as the scale increases

- Weak-scaling study
- Choose largest problem size that can be held in memory by using 64 nodes for GMRES(30)

Num. of Proc.	Problem Size	Checkpoint Size Per Proc (M					
		Traditional Checkpointing			Lossless Checkpointing		
		Jacobi	GMRES	CG	Jacobi	GMRES	CG
256	1088^3	38.4	38.4	76.8	5.99	34.6	69.5
512	1368^3	38.2	38.2	76.4	5.96	34.0	71.2
768	1568^3	38.3	38.3	76.6	5.98	34.1	73.6
1024	1728^3	38.4	38.4	76.8	5.99	34.0	69.4
1280	1856^3	39.9	39.9	79.8	6.24	33.6	69.1
1536	1968^3	39.7	39.7	79.4	6.20	33.1	69.2
1792	2064^3	39.3	39.3	78.6	6.13	32.8	70.7
2048	2160^3	39.4	39.4	78.8	6.15	32.7	67.9

One vector (double precision) of size 2160^3 (~10^{10}) ~ **80 GB**

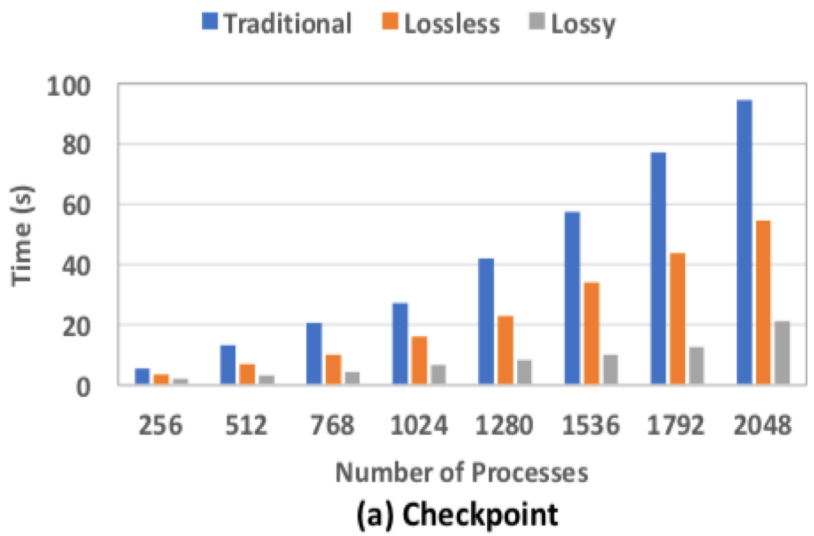
Lossy Checkpointing Performance

Num. of Proc.	Problem Size	Checkpoint Size Per Proc (MB)								
		Traditional Checkpointing			Lossless Checkpointing			Lossy Checkpointing		
		Jacobi	GMRES	CG	Jacobi	GMRES	CG	Jacobi	GMRES	CG
256	1088^3	38.4	38.4	76.8	5.99	34.6	69.5	1.33	1.23	1.69
512	1368^3	38.2	38.2	76.4	5.96	34.0	71.2	1.35	1.13	1.58
768	1568^3	38.3	38.3	76.6	5.98	34.1	73.6	1.37	1.21	1.47
1024	1728^3	38.4	38.4	76.8	5.99	34.0	69.4	1.28	1.18	1.49
1280	1856^3	39.9	39.9	79.8	6.24	33.6	69.1	1.33	1.19	1.46
1536	1968^3	39.7	39.7	79.4	6.20	33.1	69.2	1.23	1.17	1.42
1792	2064^3	39.3	39.3	78.6	6.13	32.8	70.7	1.30	1.17	1.35
2048	2160^3	39.4	39.4	78.8	6.15	32.7	67.9	1.16	1.16	1.33

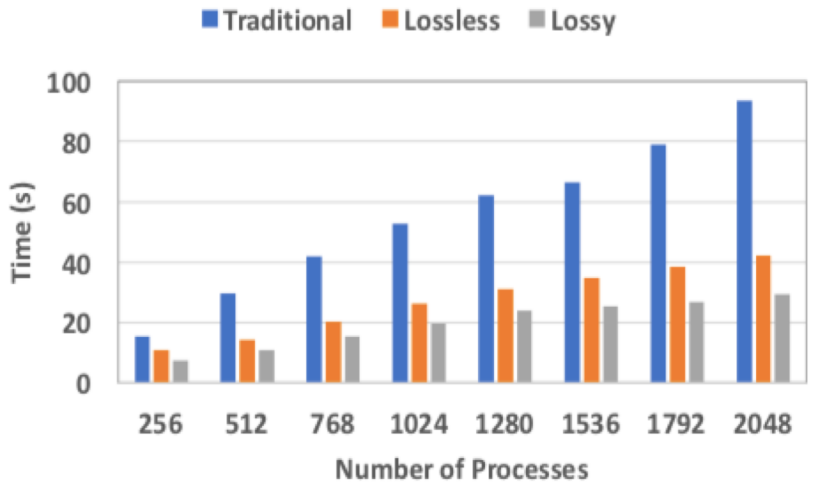
- Experiment for one checkpoint/recovery performance
  - Fixed C/R frequency
  - Average time and size over the entire execution
- Average checkpointing size
  - Lossless compression reduces checkpoint size up to 1/6
  - Lossy compression reduces checkpoint size to 1/20 ~ 1/60

Lossy Checkpointing Performance

Jacobi

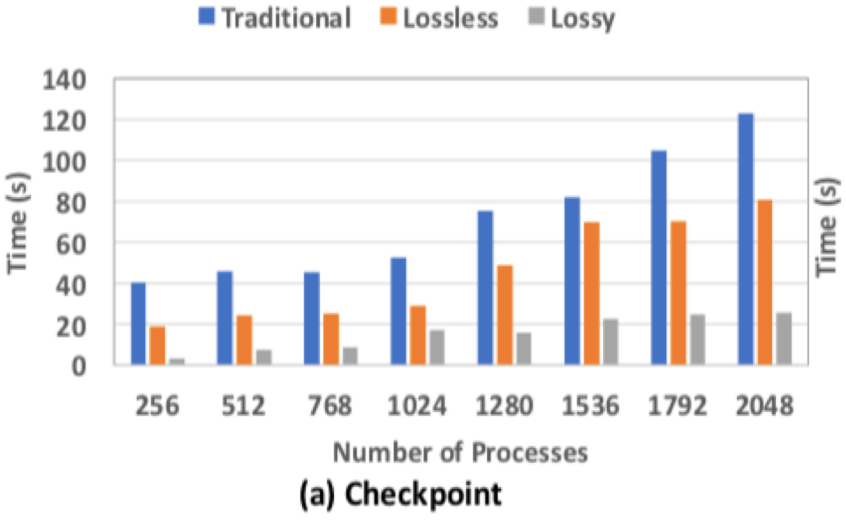


(a) Checkpoint

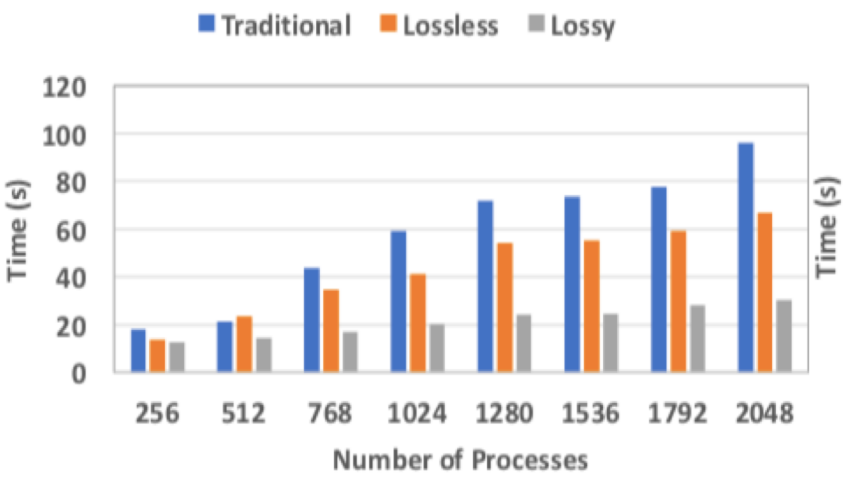


(b) Recovery

GMRES

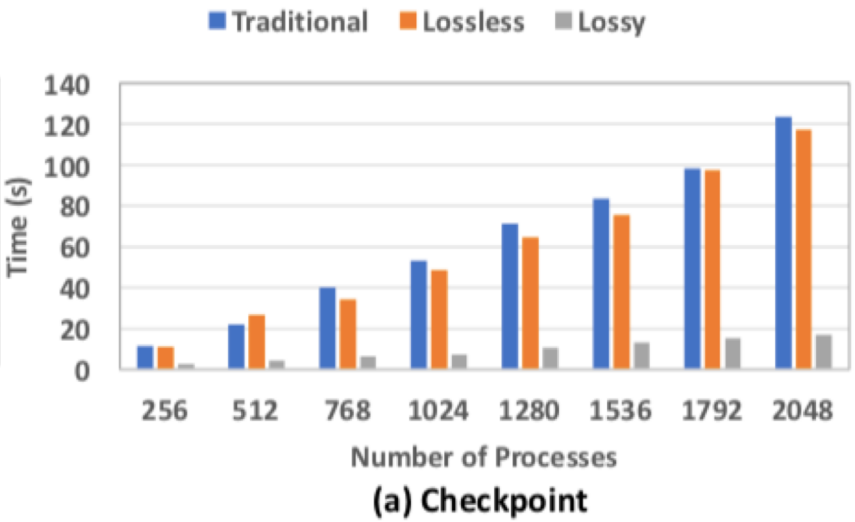


(a) Checkpoint

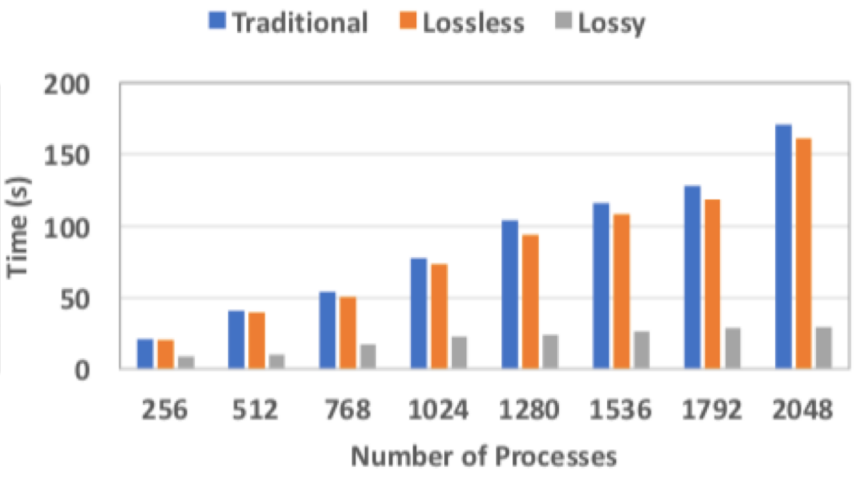


(b) Recovery

CG



(a) Checkpoint

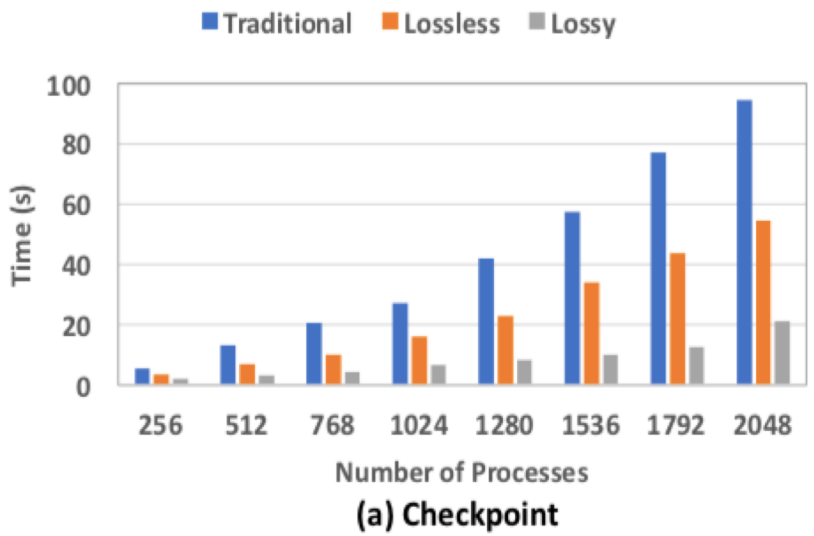


(b) Recovery

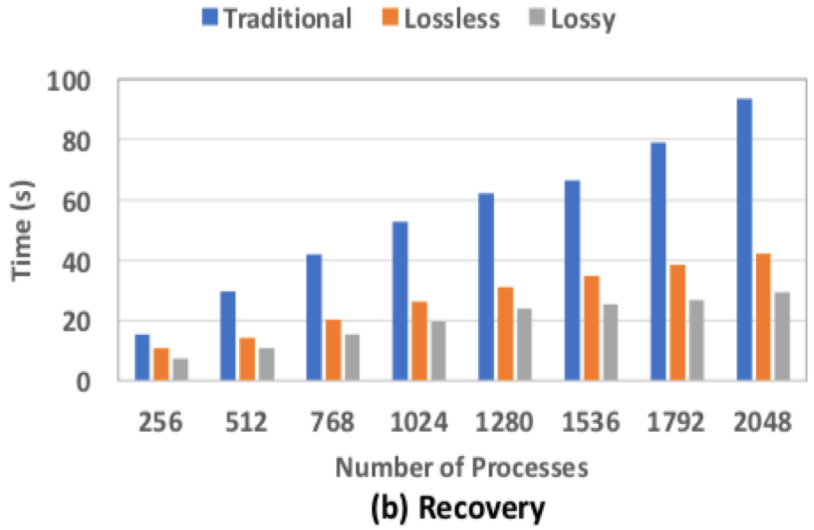
Lossy Checkpointing Performance

Lossy checkpointing can significantly reduce C/R time!

Jacobi

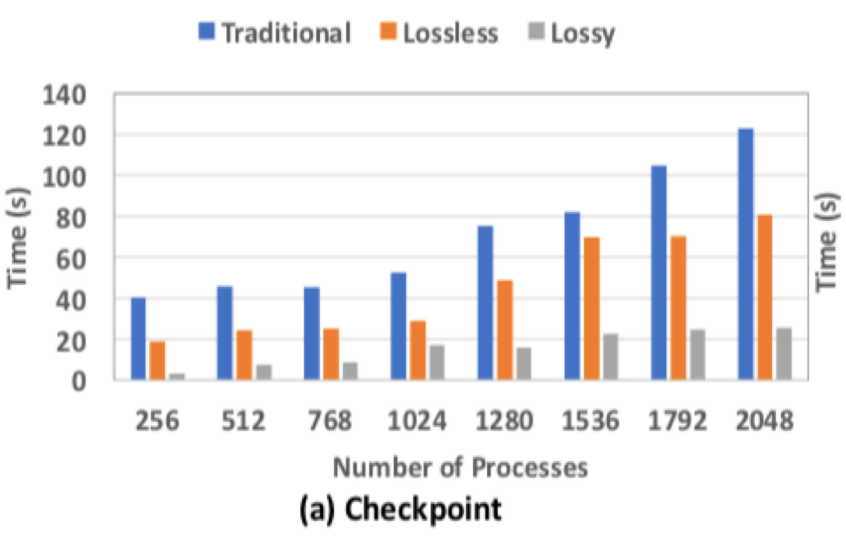


(a) Checkpoint

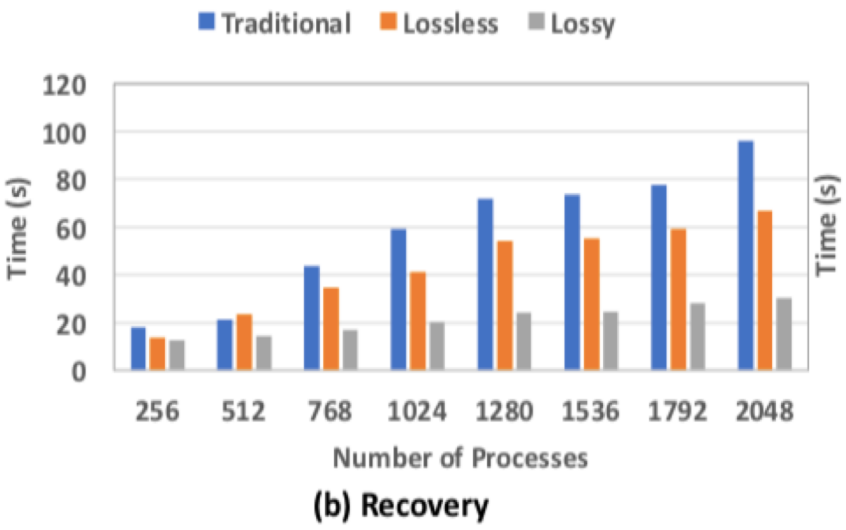


(b) Recovery

GMRES

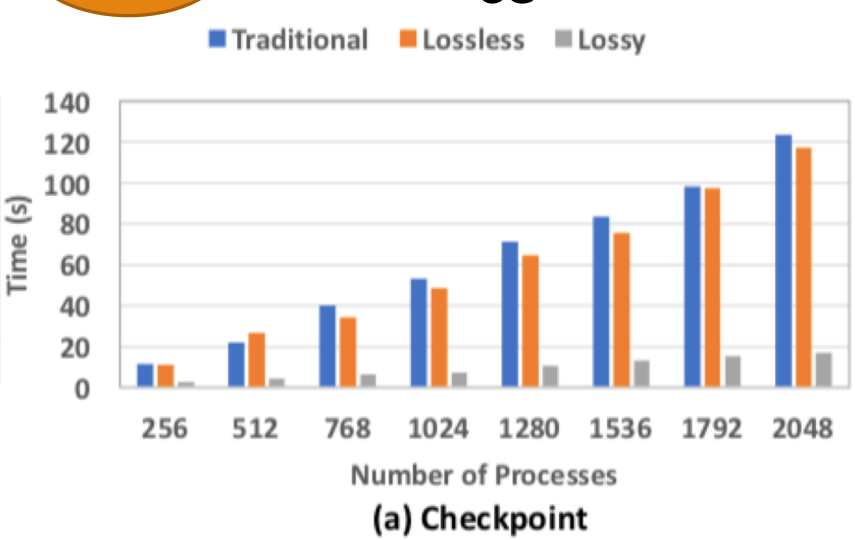


(a) Checkpoint

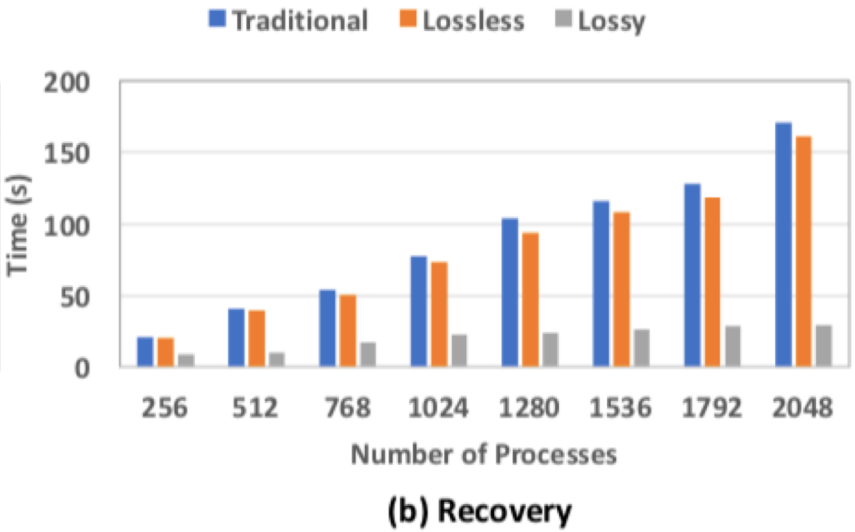


(b) Recovery

CG



(a) Checkpoint



(b) Recovery

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Theoretical Performance Analysis

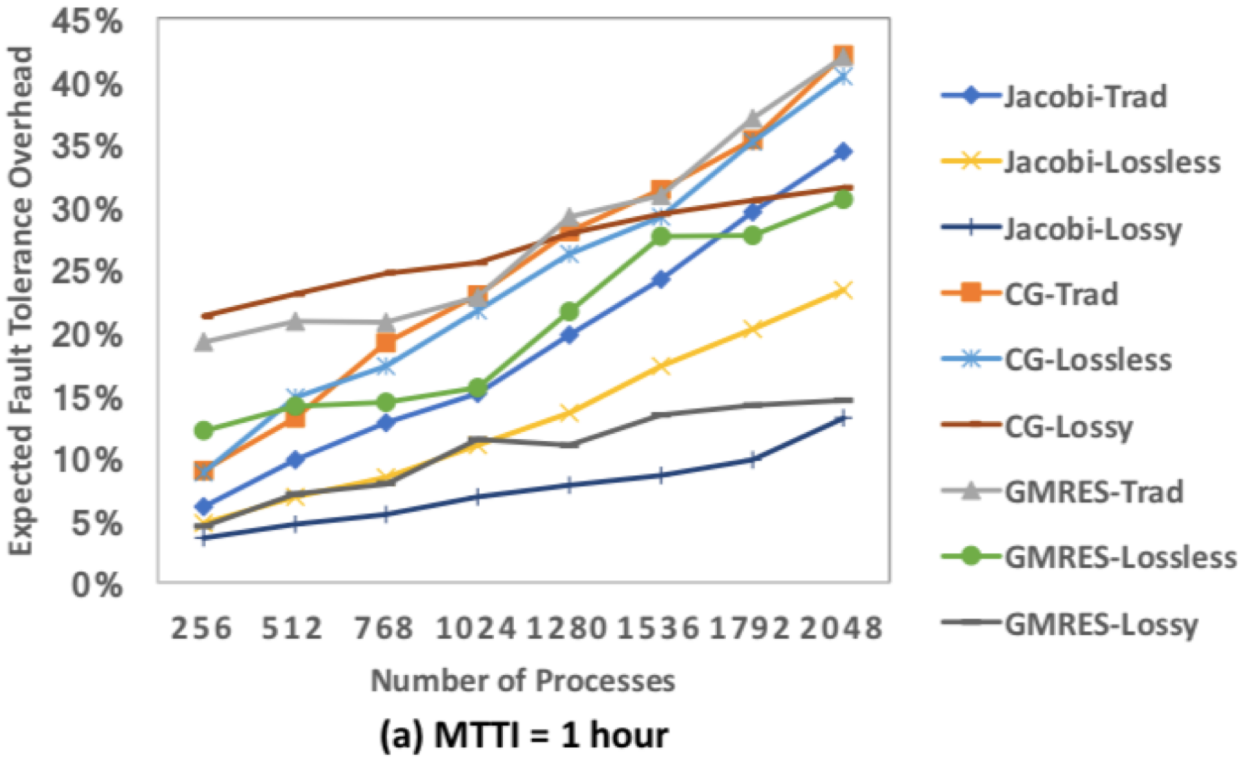
We can analyze expected fault tolerance overhead based on our lossy checkpointing performance model

- For Jacobi, based on Theorem 2, $5.2 \leq N' \leq 5.5$ → N' = 6
- For GMRES, N' = 0
- For CG, N' = 594 (25% of total iterations) based on empirical evaluation

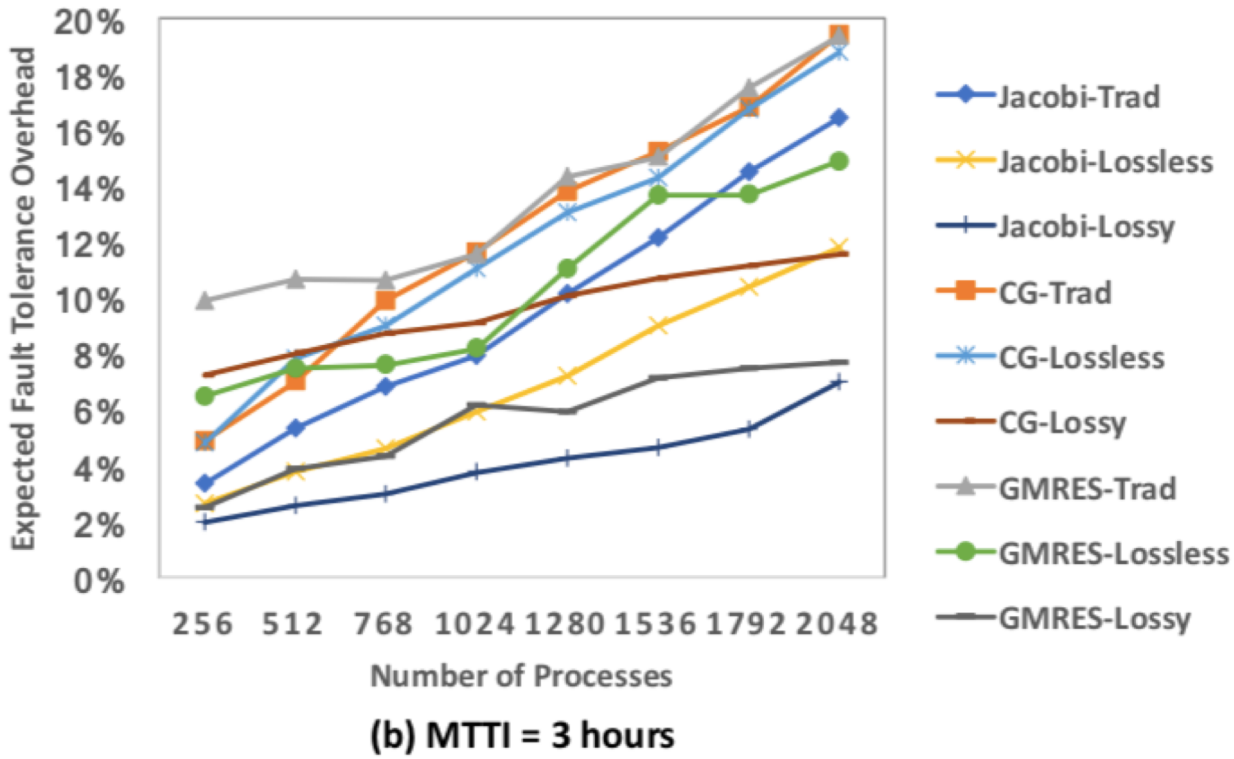
Theoretical Performance Analysis

We can analyze expected fault tolerance overhead based on our lossy checkpointing performance model

- For Jacobi, based on Theorem 2, $5.2 \leq N' \leq 5.5$ → N' = 6
- For GMRES, N' = 0
- For CG, N' = 594 (25% of total iterations) based on empirical evaluation



(a) MTTI = 1 hour

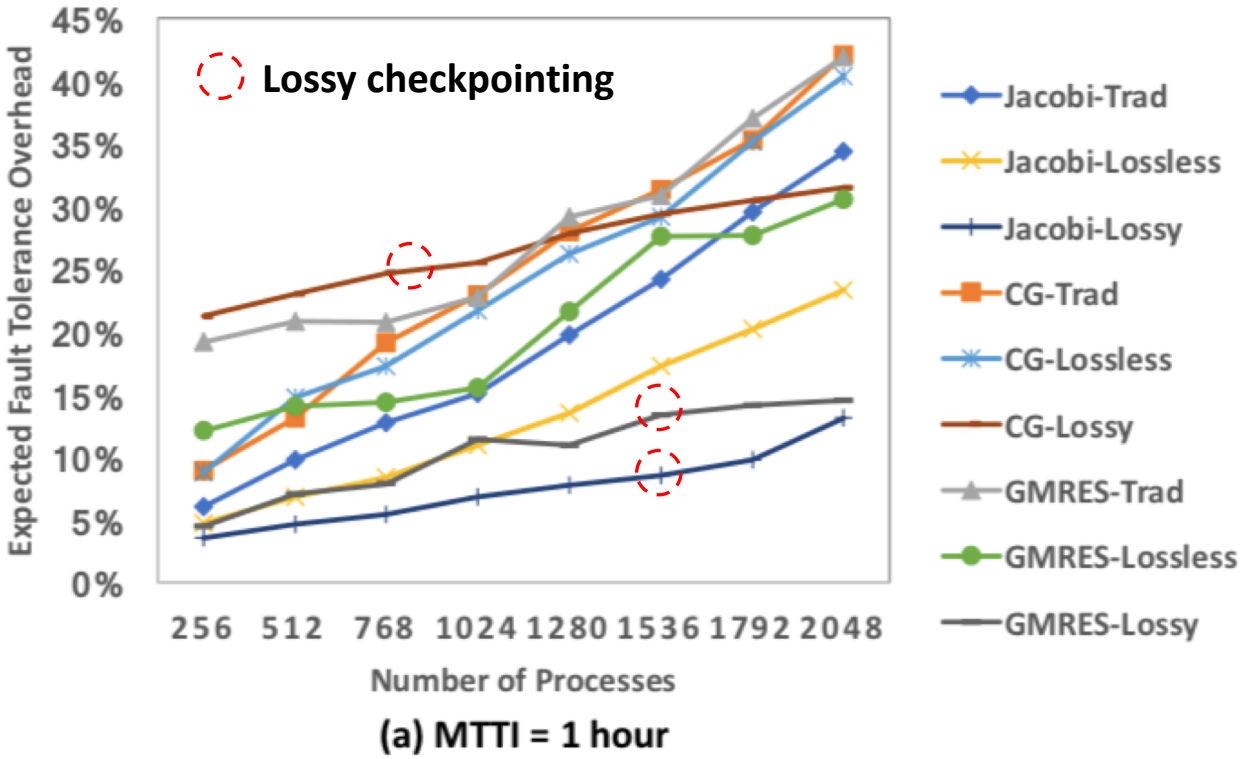


(b) MTTI = 3 hours

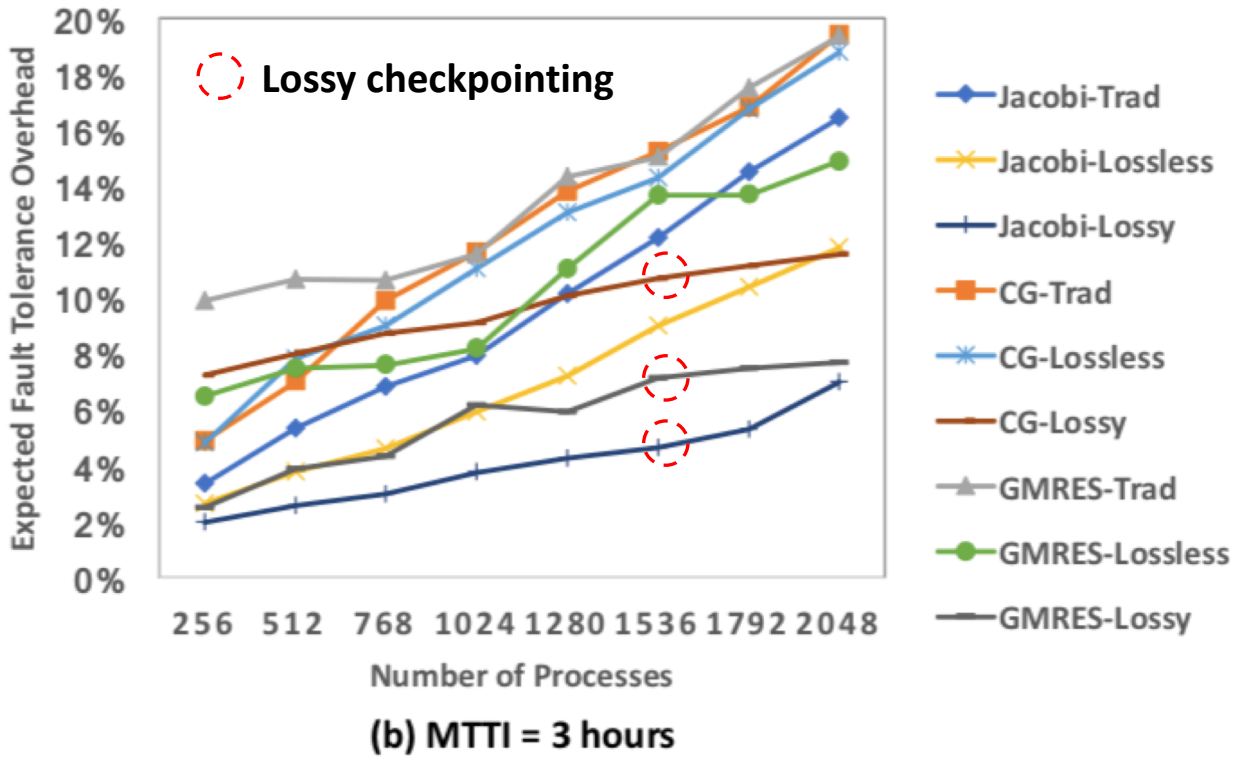
Theoretical Performance Analysis

We can analyze expected fault tolerance overhead based on our lossy checkpointing performance model

- For Jacobi, based on Theorem 2, $5.2 \leq N' \leq 5.5$ → N' = 6
- For GMRES, N' = 0
- For CG, N' = 594 (25% of total iterations) based on empirical evaluation



(a) MTTI = 1 hour

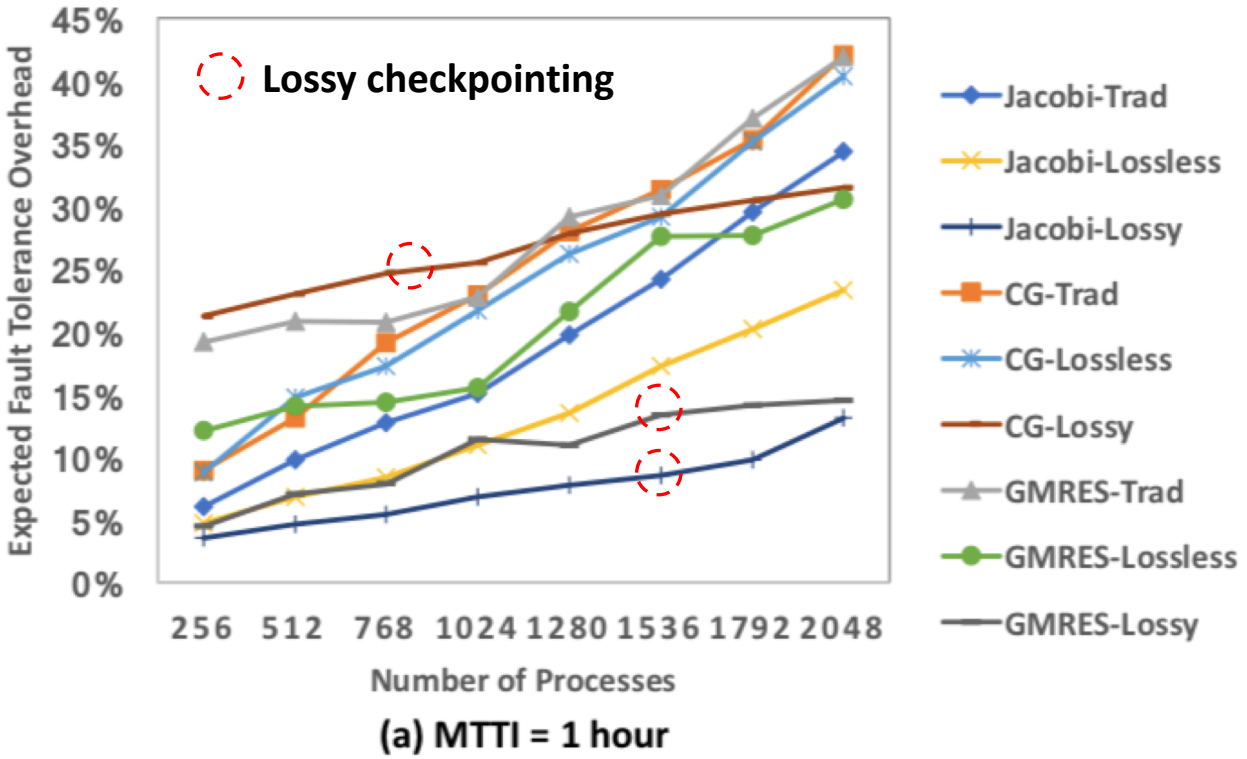


(b) MTTI = 3 hours

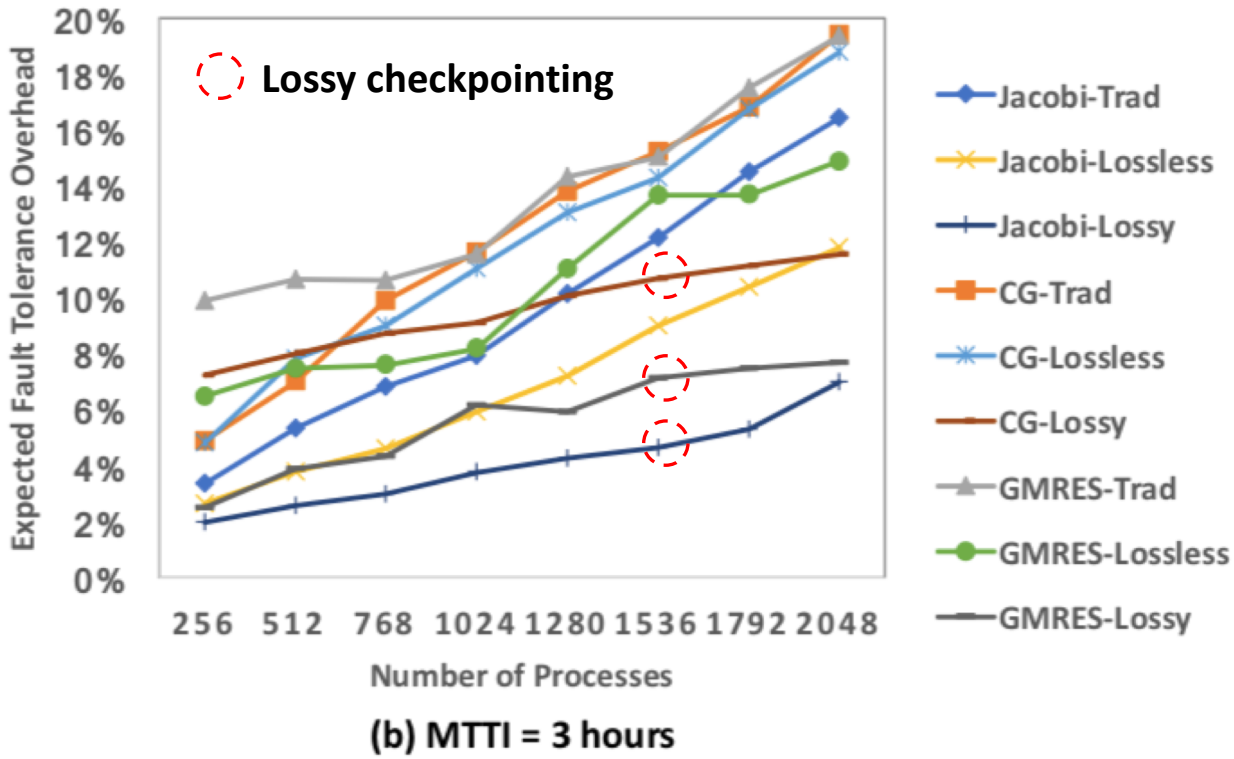
Theoretical Performance Analysis

Observations

- GMRES and Jacobi: lossy checkpoint is always better than lossless and traditional checkpoint
- CG: lossy checkpoint is better than lossless and traditional checkpoint when # processes > 1536 / 768
- Curves of lossy checkpoint increase much slowly than curves of other two solutions → Our proposed lossy checkpoint is expected to achieve **more performance gain as scale increases**



(a) MTTI = 1 hour



(b) MTTI = 3 hours

Outline

- Introduction
  - Why we need to checkpoint iterative methods?
- Background
  - Traditional checkpointing for iterative methods
  - Performance model of traditional checkpointing
- Our Designs
  - Lossy checkpointing for iterative methods
  - Performance model of our new checkpointing
- Theoretical Analysis
  - Impact of lossy checkpointing for different methods
  - Expected fault tolerance overhead
- Experimental Evaluation

Experimental Evaluation with Failures

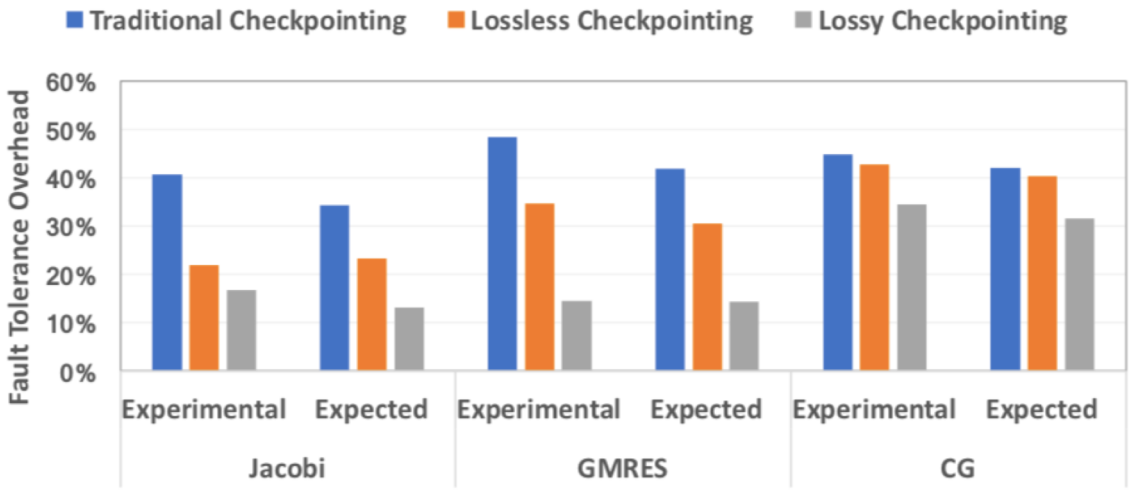
- Failure Injection
  - MTTI = 1 hour
  - Failure intervals follow an exponential distribution

Experimental Evaluation with Failures

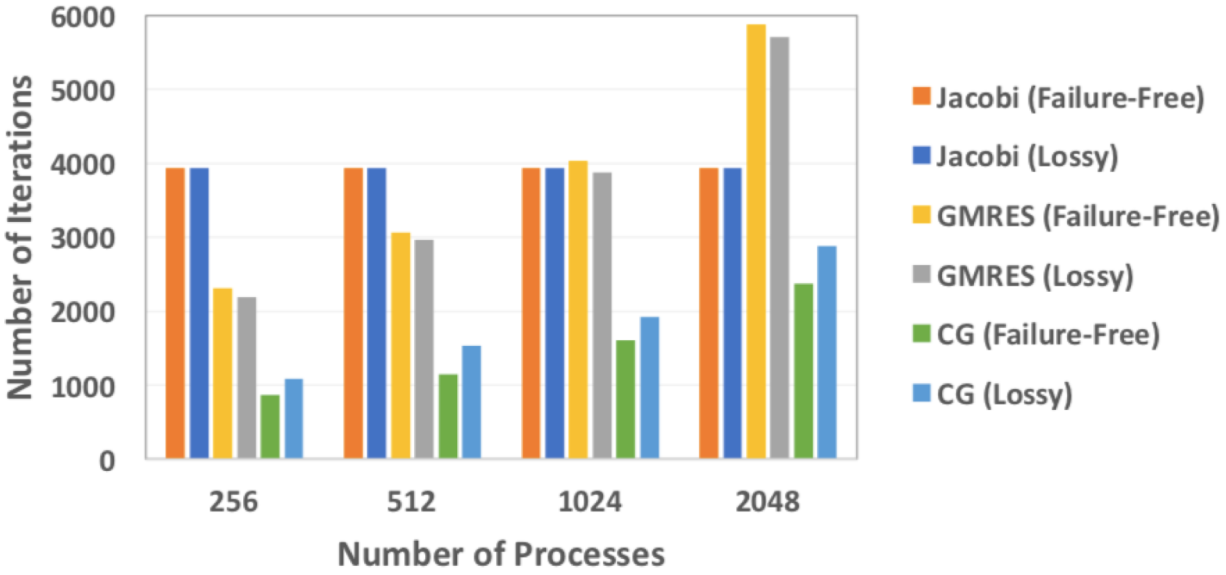
- Failure Injection
  - MTTI = 1 hour
  - Failure intervals follow an exponential distribution
- Checkpoint Interval
  - $Time_{ckpt}^{Trad} \sim 120\ s, Time_{ckpt}^{Lossless} \sim 70\ s, Time_{ckpt}^{Lossy} \sim 20s$
  - Based on checkpointing time and Young's formula
    - $Intvl_{ckpt}^{Trad} = 16\ mins, Intvl_{ckpt}^{Trad} = 12\ mins, Intvl_{ckpt}^{Trad} = 7\ mins$

Experimental Evaluation with Failures

- Failure Injection
  - MTTI = 1 hour
  - Failure intervals follow an exponential distribution
- Checkpoint Interval
  - $Time_{ckpt}^{Trad} \sim 120\ s, Time_{ckpt}^{Lossless} \sim 70\ s, Time_{ckpt}^{Lossy} \sim 20s$
  - Based on checkpointing time and Young's formula
    - $Intvl_{ckpt}^{Trad} = 16\ mins, Intvl_{ckpt}^{Trad} = 12\ mins, Intvl_{ckpt}^{Trad} = 7\ mins$

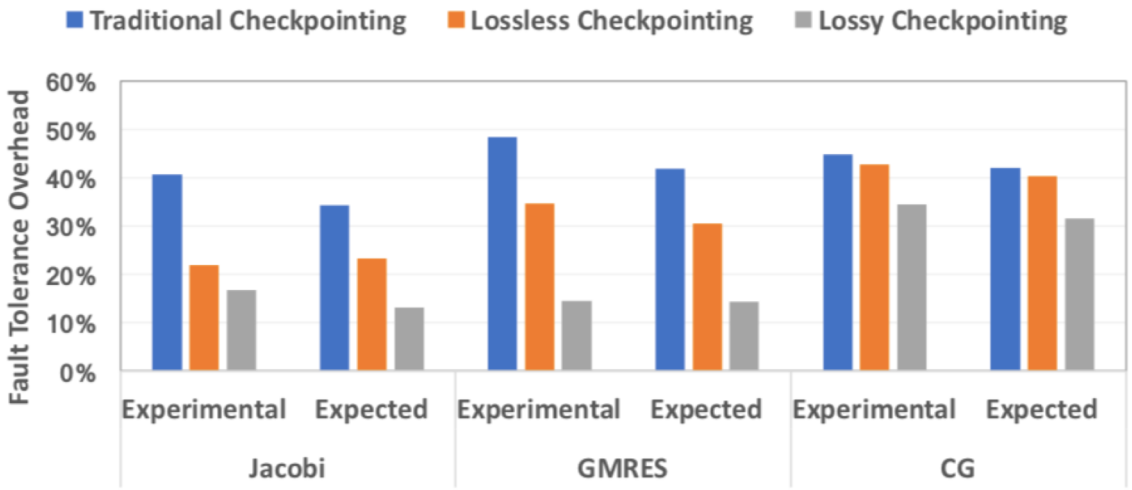


Number of convergence iterations with lossy checkpointing for Jacobi, GMRES, and CG



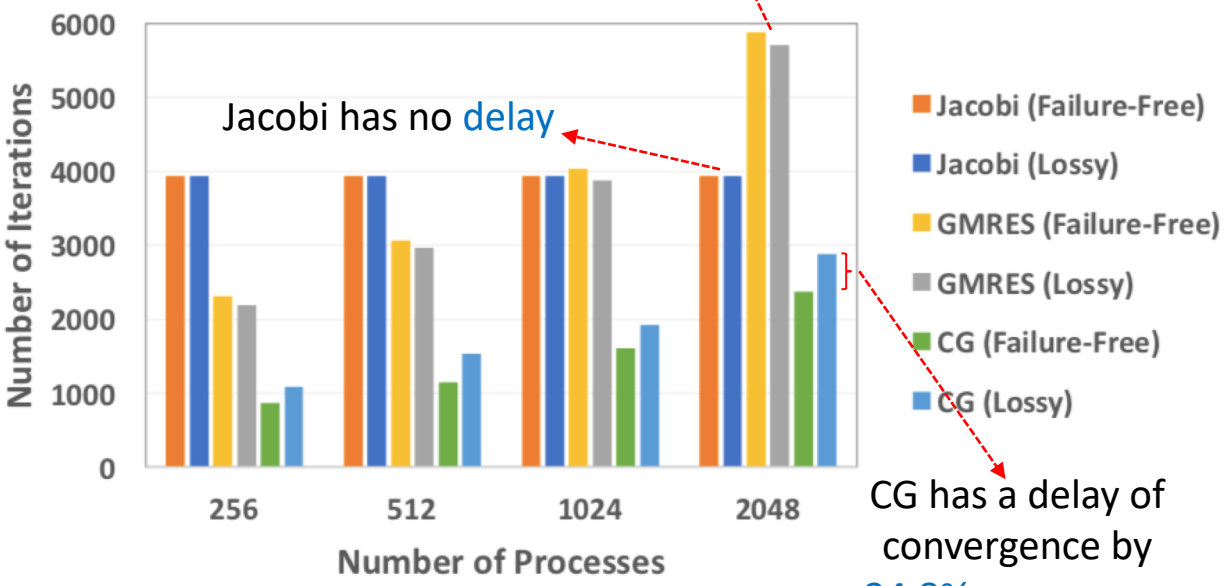
Experimental Evaluation with Failures

- Failure Injection
  - MTTI = 1 hour
  - Failure intervals follow an exponential distribution
- Checkpoint Interval
  - $Time_{ckpt}^{Trad} \sim 120\ s, Time_{ckpt}^{Lossless} \sim 70\ s, Time_{ckpt}^{Lossy} \sim 20s$
  - Based on checkpointing time and Young's formula
    - $Intvl_{ckpt}^{Trad} = 16\ mins, Intvl_{ckpt}^{Trad} = 12\ mins, Intvl_{ckpt}^{Trad} = 7\ mins$



GMRES has an acceleration

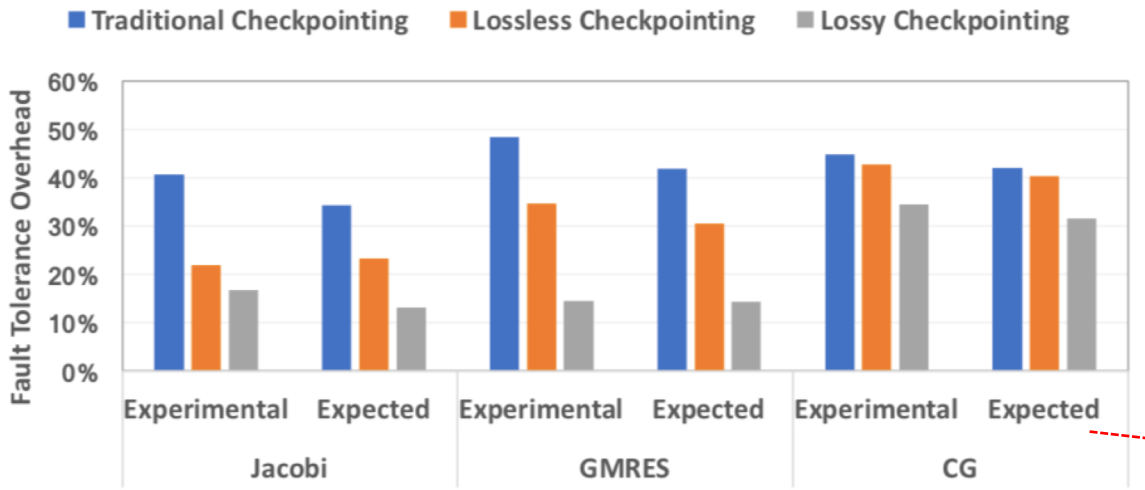
Number of convergence iterations with lossy checkpointing for Jacobi, GMRES, and CG



CG has a delay of convergence by 24.8% on average

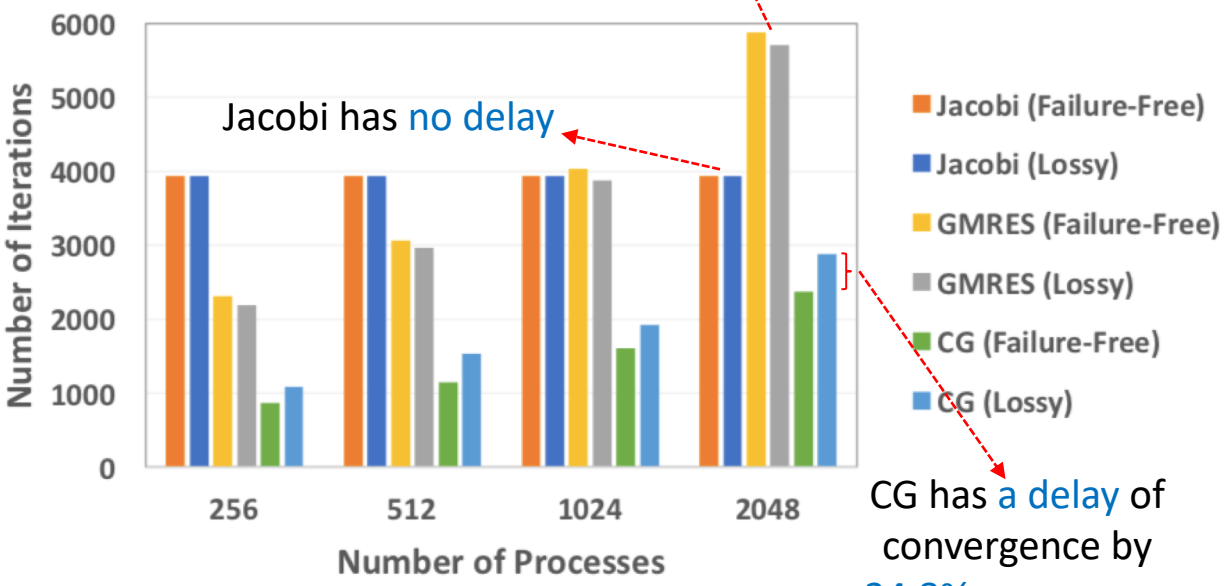
Experimental Evaluation with Failures

- Failure Injection
  - MTTI = 1 hour
  - Failure intervals follow an exponential distribution
- Checkpoint Interval
  - $Time_{ckpt}^{Trad} \sim 120\ s, Time_{ckpt}^{Lossless} \sim 70\ s, Time_{ckpt}^{Lossy} \sim 20s$
  - Based on checkpointing time and Young's formula
    - $Intvl_{ckpt}^{Trad} = 16\ mins, Intvl_{ckpt}^{Trad} = 12\ mins, Intvl_{ckpt}^{Trad} = 7\ mins$



GMRES has an acceleration

Number of convergence iterations with lossy checkpointing for Jacobi, GMRES, and CG



CG has a delay of convergence by 24.8% on average

- Jacobi: FT overhead reduced by 59% compared with traditional ckpt and 24% compared with lossless ckpt
- GMRES: FT overhead reduced by 70% and 58%
- CG: FT overhead reduced by 23% and 20%

Experimental results are very close to theoretical analysis!

Conclusion

- Propose an efficient lossy checkpointing scheme to improve C/R performance for iterative methods
- Formulate a lossy checkpointing performance model
- Quantify the tradeoff between reduced overhead and extra # of iterations
- Analyze the impact of lossy checkpointing on multiple iterative methods (stationary, GMRES, CG)
- Evaluate lossy checkpointing on a HPC environment with 2,048 cores
- Experiments show our lossy checkpointing can significantly reduce the fault tolerance overhead in the presence of failures
  - Reduced by 23%~70% compared with traditional checkpoint and by 20%~58% with lossless checkpoint
- Future work
  - Explore lossy checkpointing in other scientific computational components (such as AMG, AMR, FFT)
  - Evaluate lossy checkpointing in real HPC simulations
  - Evaluate lossy checkpointing in other I/O intensive and error resilient applications

Acknowledge

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative. The material was also supported by and supported by the National Science Foundation under Grant No. 1305624, No. 1513201, and No. 1619253.

National Science Foundation
Directorate for Computer & Information Science & Engineering (CISE)

Thank you!

Any questions are welcome!

Contact:
Dingwen Tao (dingwen.tao@ieee.org)