Pacific Northwest NATIONAL LABORATORY

UC RIVERSIDE UNIVERSITY OF CALIFORNIA

New-Sum: A Novel Online ABFT Scheme For General Iterative Methods

Dingwen Tao (University of California, Riverside)
Shuaiwen Leon Song (Pacific Northwest National Laboratory)
Sriram Krishnamoorthy (Pacific Northwest National Laboratory)
Panruo Wu (University of California, Riverside)
Xin Liang (University of California, Riverside)
Eddy Z. Zhang (Rutgers University)
Darren Kerbyson (Pacific Northwest National Laboratory)
Zizhong Chen (University of California, Riverside)

HPDC'16 The 25th International Symposium on High Performance Parallel and Distributed Computing
Kyoto, JAPAN, May 31 - June 4, 2016

Introduction

- Supercomputers are increasingly susceptible to Silent Data Corruption (SDC)
  - Large number of complex architecture components (e.g., novel memory designs)
  - Each component has growing on-chip transistor density
  - Limited power and energy consumption

- This phenomena has been already observed on several real-world leadership-class supercomputers

- Titan/Jaguar case study†
  - Constant stream of single bit flips
  - Double-bit flip every 24 hrs.
  - 20 faults per hour
    - Heartbeat fault every 3 minutes
    - 12 kernel panics in 3 days

†Al Geist, How To Kill A Supercomputer, 2016

Outline

- Algorithm-Based Fault Tolerance (ABFT)
- Limitations of Traditional ABFT for matrix-vector multiplication (MVM)
- Limitations of Existing Techniques for FT-Iterative Methods

Outline

- Algorithm-Based Fault Tolerance (ABFT)
- Limitations of Traditional ABFT for matrix-vector multiplication (MVM)
- Limitations of Existing Techniques for FT-Iterative Methods
- Our Designs
  - Novel Error-Preserving Checksum for MVM
  - New Online ABFT Schemes

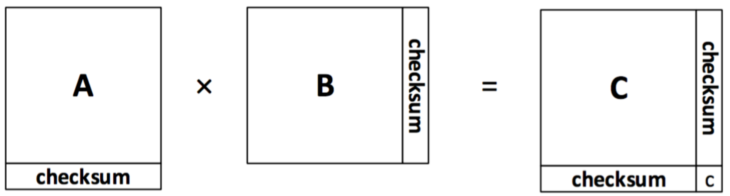
Outline

- Algorithm-Based Fault Tolerance (ABFT)
- Limitations of Traditional ABFT for matrix-vector multiplication (MVM)
- Limitations of Existing Techniques for FT-Iterative Methods
- Our Designs
  - Novel Error-Preserving Checksum for MVM
  - New Online ABFT Schemes
- Theoretical Comparison
- Empirical Evaluation
- Conclusions

Algorithm-Based Fault Tolerance (ABFT)

- Algorithm-Based Fault Tolerance (ABFT) is a checksum-based approach to detect and correct/locate SDCs at a low cost

- Traditional checksum encoding scheme
  - **Encoding**: augment the input matrices with a checksum computed from the rows or columns
  - **Computation**: perform a matrix operation on the augmented matrices and compute a checksum for the output matrix automatically
  - **Verification**: any error in the computation will break the encoding relationship between the output matrix and its checksum

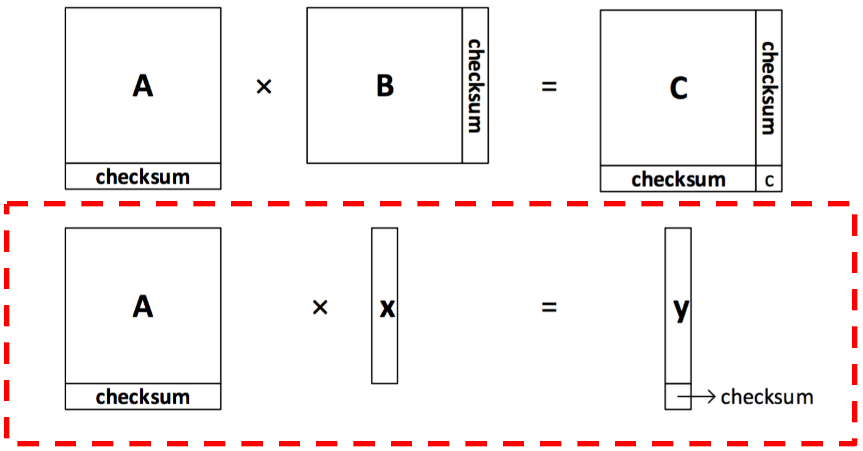


Algorithm-Based Fault Tolerance (ABFT)

- Algorithm-Based Fault Tolerance (ABFT) is a checksum-based approach to detect and correct/locate SDCs at a low cost

- Traditional checksum encoding scheme
  - **Encoding**: augment the input matrices with a checksum computed from the rows or columns
  - **Computation**: perform a matrix operation on the augmented matrices and compute a checksum for the output matrix automatically
  - **Verification**: any error in the computation will break the encoding relationship between the output matrix and its checksum

- Example: applying traditional checksum encoding scheme [Huang and Abraham] to ***matrix-vector multiplication*** (**MVM**)



Algorithm-Based Fault Tolerance for MVM

Pacific Northwest NATIONAL LABORATORY

Proudly Operated by Battelle Since 1965

UCR

- **Encoding**: $A \rightarrow A^* = \begin{pmatrix} A \\ \vec{c}^{\,T} A \end{pmatrix}$
  - ***c*** is a predefined vector, normally, ***c*** = $(1, 1, \ldots, 1)^T$
- **Computation on encoded matrix and vectors**:

$$\vec{y}^* = A^* \vec{x} = \begin{pmatrix} A \\ \vec{c}^{\,T} A \end{pmatrix} \vec{x} = \begin{pmatrix} A\vec{x} \\ \vec{c}^{\,T} A\vec{x} \end{pmatrix} \begin{matrix} \longrightarrow \text{computed vector } \boldsymbol{y} \\ \longrightarrow \text{checksum}(\boldsymbol{y}) \end{matrix}$$

- **Verification**: $checksum(y) = \vec{c}^{\,T} \vec{y}$
  - Checksum relationship
  - Satisfied: ***no error*** in the computation
  - Not Satisfied: ***error(s) occurred*** in the computation

Limitations of Traditional ABFT for MVM

- Consider an error in ***x*** before the MVM, resulting in an erroneous vector ***x'***

- **Computation**:

$$\vec{y}'^{*} = A * \vec{x}' = \begin{pmatrix} A \\ \vec{c}^{T} A \end{pmatrix} \vec{x}' = \begin{pmatrix} A\vec{x}' \\ \vec{c}^{T} A \vec{x}' \end{pmatrix} \begin{matrix} \longrightarrow \text{erroneous computed } \boldsymbol{y} \\ \longrightarrow \text{erroneous checksum}(\boldsymbol{y}) \end{matrix}$$

- **Verification**:
  - Checksum relationship for erroneous computed vector ***y*** still holds
  - Checking the output vector's checksum relationship cannot identify all SDCs

Problems With the Traditional ABFT Scheme:

(1) Traditional encoding scheme may **FAIL** for **MVM**

(2) Traditional encoding scheme can **NOT** address **cache/register bit-flips** and protect **preconditioners**

ABFT for Iterative Methods

- Iterative Methods are widely used for solving systems of equations or computing eigenvalues of large sparse matrices

- Existing fault tolerant techniques for iterative methods

  - [Shantharam et al., ICS'12]: requires matrix A to be ***strictly diagonally dominant*** – **NOT GENERAL**
  - [Chen, PPoPP'13]: only covers a subset of Krylov methods that can offer ***orthogonality*** – **NOT GENERAL**
  - [Sloan et al., DSN'13]: uses the ***traditional checksum***-encoding mechanism – **MAY FAIL**

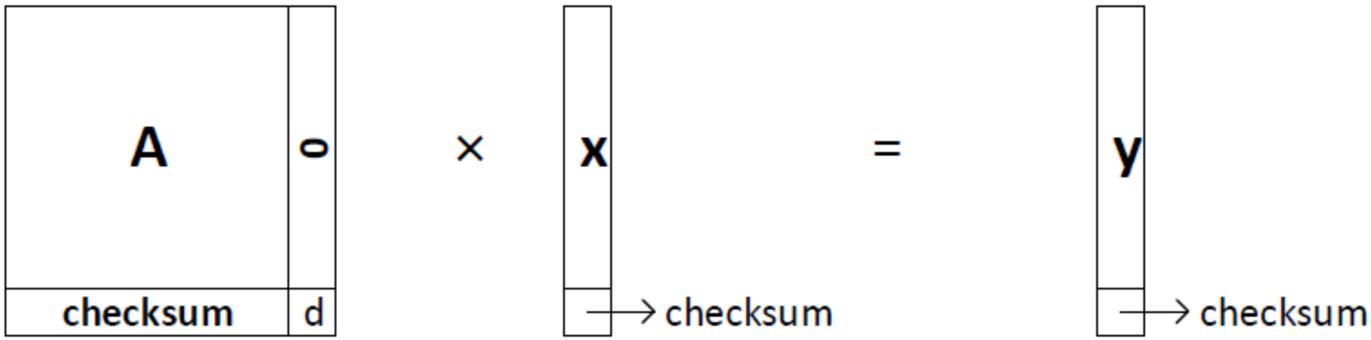
- Developing a ***general*** and ***low-cost*** ABFT scheme with ***good coverage*** for iterative methods is in high demand

Outline

- Algorithm-Based Fault Tolerance (ABFT)
- Limitations of Traditional ABFT for matrix-vector multiplication (MVM)
- Limitations of Existing Techniques for FT-Iterative Methods
- Our Designs
  - **Novel Error-Preserving Checksum for MVM**
  - New Online ABFT Schemes
- Theoretical Comparison
- Empirical Evaluation
- Conclusions

Novel Error-Preserving Checksum for MVM

- **New Encoding Mechanism**



$$A \to A^* = \boxed{\begin{pmatrix} A & \vec{0} \\ \vec{c}^{\,T} A - d\vec{c}^{\,T} & d \end{pmatrix}} \qquad \vec{x} \to \vec{x}^* = \begin{pmatrix} \vec{x} \\ \vec{c}^{\,T} \vec{x} \end{pmatrix}$$

$$\vec{y}^* = A^* \vec{x}^*$$

$$= \begin{pmatrix} A & \vec{0} \\ \vec{c}^{\,T} A - d\vec{c}^{\,T} & d \end{pmatrix} \begin{pmatrix} \vec{x} \\ \vec{c}^{\,T} \vec{x} \end{pmatrix} = \begin{pmatrix} A\vec{x} \\ \vec{c}^{\,T} A\vec{x} \end{pmatrix} \longrightarrow checksum(y) = \vec{c}^{\,T} \vec{y}$$

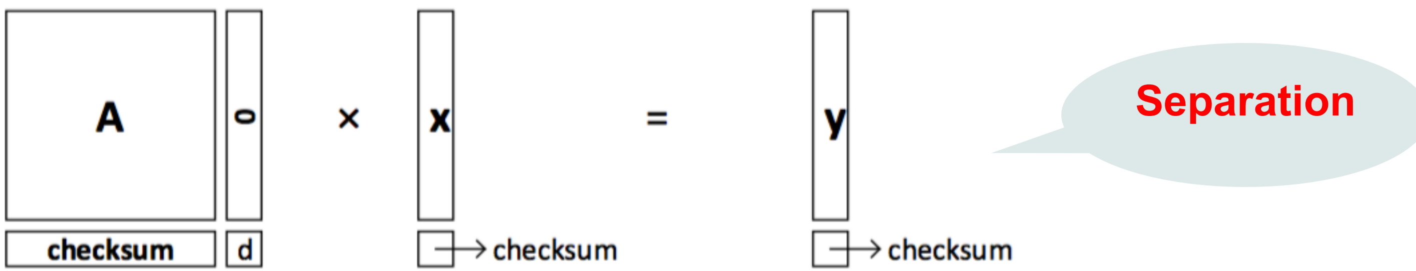
Checksum Relationship Maintained

Novel Error-Preserving Checksum for MVM (cont.)

- **Computation**

$$\vec{y}* = A * \vec{x} *$$

$$= \begin{pmatrix} A & \vec{0} \\ checksum(A) & d \end{pmatrix} \begin{pmatrix} \vec{x} \\ checksum(\vec{x}) \end{pmatrix} = \begin{pmatrix} A\vec{x} \\ checksum(A)\vec{x} + d \bullet checksum(\vec{x}) \end{pmatrix}$$



- Checksum can be updated separately after **MVM** ***y* = *Ax***

$$checksum(\vec{y}) = checksum(A)\vec{x} + d \bullet checksum(\vec{x})$$

Novel Error-Preserving Checksum for MVM (cont.)

Jacobi Method

for $k = 1, 2, \cdots$ **do**
  for $i = 1, 2, \cdots, n$ **do**
    $x_i = 0$
    for $j = 1, 2, \cdots, i-1, i+1, \cdots, n$ **do**
      $x_i = x_i + a_{i,j} x_j^{(k-1)}$ ○
    end for
    $x_i = (b_i - x_i)/a_{i,j}$ ○
  end for
  $x^{(k)} = x$
  check convergence; continue if necessary
end for

Preconditioned CG

for $i = 0, 1, \cdots$ **do**
  $q^{(i)} = Ap^{(i)}$ ●
  $\alpha_i = \rho_i / p^{(i)T} q^{(i)}$ ■
  $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$ ○
  $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$ ○
  solve $Mz^{(i+1)} = r^{(i+1)}$ ◀
  $\rho_{i+1} = r^{(i+1)T} z^{(i+1)}$ ■
  $\beta_i = \rho_{i+1}/\rho_i$ ■
  $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$ ○
  check convergence; continue if necessary
end for

Preconditioned Chebyshev

for $i = 1, 2, \cdots$ **do**
  solve $Mz^{(i-1)} = r^{(i)}$ ◀
  if $(i = 1)$ **then**
    $p^{(1)} = z^{(0)}$
    $\alpha_1 = 2/\theta$
  else
    $\beta_{i-1} = \alpha_{i-1}(\varepsilon/2)^2$ ■
    $\alpha_i = 1/(\theta - \beta_{i-1})$ ■
    $p^{(i)} = z^{(i)} + \beta_{i-1} p^{(i-1)}$ ○
  end if
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ ○
  $r^{(i)} = b - Ax^{(i)} (= r^{(i-1)} - \alpha_i A p^{(i)})$ ●
  check convergence; continue if necessary
end for

● Represents MVM

○ Represents VLO

◀ Represents PCO

■ Represents Vector Inner-Product (VDP) or scalar computation

- Vector Linear Operation (**VLO**): $\boldsymbol{z} = \alpha \boldsymbol{x} + \beta \boldsymbol{y}$ and Solving Preconditioned System (**PCO**): $\boldsymbol{Mz} = \boldsymbol{r}$

$$checksum(\vec{z}) = \alpha \bullet checksum(\vec{x}) + \beta \bullet checksum(\vec{y})$$

$$checksum(\vec{z}) = (checksum(M)\vec{z} - checksum(\vec{r}))/d$$

Novel Error-Preserving Checksum for MVM (cont.)

- **Theorem**: For any matrix-vector multiplication (MVM), vector linear operation (VLO), preconditioning operation (PCO), the **checksum relationship** of the output vector is ***preserved*** if and only if there are ***no soft errors*** **before or during** the operation. (key theorem)

  Proof: see our paper for the details.

- **Verification**:

  - ***No error*** occurred **before or during** the computation <=> checksum($\boldsymbol{y}$) = $\boldsymbol{c}^T\boldsymbol{y}$

  - Any errors => Checksum relationship will ***preserve to be BROKEN*** in the **subset iterations**

Outline

- Algorithm-Based Fault Tolerance (ABFT)
- Limitations of Traditional ABFT for matrix-vector multiplication (MVM)
- Limitations of Existing Techniques for FT-Iterative Methods
- Our Designs
  - Novel Error-Preserving Checksum for MVM
  - **New Online ABFT Schemes**
- Theoretical Comparison
- Empirical Evaluation
- Conclusions

New Online ABFT Schemes

- Based on our new designed checksum encoding scheme, we design efficient online ABFT solutions for iterative methods

  - **"Lazy" Detection**: Low-Cost Online ABFT Algorithm
  - **"Eager" Recovery for MVM**: Triple Checksums
  - **"Hybrid" Detection**: Two-Level Online ABFT Algorithm

“Lazy” Detection: Low-Cost Online ABFT Algorithm

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

UCR

- Step 1: update checksum after each MVM, VLO, PCO (red in algorithm 1)

- Step 2: low-cost error detection
  - Verify checksum relationship after **every MVM, VLO, PCO – high detection cost**
  - *Do we need to verify the checksum relationship every iteration?*
    - <u>*Observation 1*</u>: Soft errors in ***p***, ***x***, ***r*** will propagate to the subsequent iterations
    - **Verify checksum relationship every several iterations** (blue in algorithm 1)
  - *Do we need to verify all the operations?*
    - <u>*Observation 2*</u>: Soft errors in ***z***, ***p***, ***q*** will eventually propagate to ***x*** and ***r***
    - **Only verify 2 checksum relationships**, ***x*** and ***r*** (blue in algorithm 1)

Algorithm 1 Online-ABFT Preconditioned Conjugate Gradient Algorithm Based on New Checksum with checkpoint/restart Technique

Compute $r^{(0)} = b - Ax^{(0)}, z^{(0)} = M^{-1}r^{(0)}, p^{(0)} = z^{(0)}$ and $\rho_0 = {r^{(0)}}^T z^{(0)}$ for some initial guess $x^{(0)}$
Compute $checksum(A) = c^T A - dc^T$, $checksum(M) = c^T M - dc^T$, $checksum(b) = c^T b^{(0)}$, $checksum(x) = c^T x^{(0)}$, $checksum(r) = c^T r^{(0)}$, $checksum(z) = [(checksum(r) - checksum(M)z^{(0)})]/d$, $checksum(p) = checksum(z)$
Checkpoint A, M
for $i = 0, 1, \cdots$ **do**
  if $((i > 0)$ and $(i\%d = 0))$ **then**
    if $(||checksum(r) - c^T r^{(i)}||/n > \theta)$ or $(||checksum(x) - c^T x^{(i)}||/n > \theta)$ **then**
      Rollback: recover $A, M, i, \rho_i, p^{(i)}, x^{(i)}, r^{(i)}$
    else if $(i\%(cd) = 0)$ **then**
      Checkpoint: $i, \rho_i, p^{(i)}, x^{(i)}$
    end if
  end if
  $q^{(i)} = Ap^{(i)}$
  $checksum(q) = checksum(A)p^{(i)} + d \cdot checksum(p)$
  $\alpha_i = \rho_i / {p^{(i)}}^T q^{(i)}$
  $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$
  $checksum(x) = checksum(x) + \alpha_i \cdot checksum(p)$
  $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$
  $checksum(r) = checksum(r) - \alpha_i \cdot checksum(q)$
  solve $Mz^{(i+1)} = r^{(i+1)}$
  $checksum(z) = \frac{(checksum(r) - checksum(M)z^{(i+1)})}{d}$
  $\rho_{i+1} = {r^{(i+1)}}^T z^{(i+1)}$
  $\beta_i = \rho_{i+1}/\rho_i$
  $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
  $checksum(p) = checksum(z) + \beta_i \cdot checksum(p)$
  check convergence; continue if necessary
end for

"Lazy" Detection: Low-Cost Online ABFT Algorithm (cont.)

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

UCR

Algorithm 1 Online-ABFT Preconditioned Conjugate Gradient Algorithm Based on New Checksum with checkpoint/restart Technique

Compute $r^{(0)} = b - Ax^{(0)}, z^{(0)} = M^{-1}r^{(0)}, p^{(0)} = z^{(0)}$ and $\rho_0 = {r^{(0)}}^T z^{(0)}$ for some initial guess $x^{(0)}$
Compute $checksum(A) = c^T A - dc^T$, $checksum(M) = c^T M - dc^T$, $checksum(b) = c^T b^{(0)}$, $checksum(x) = c^T x^{(0)}$, $checksum(r) = c^T r^{(0)}$, $checksum(z) = [(checksum(r) - checksum(M) z^{(0)})]/d$, $checksum(p) = checksum(z)$
Checkpoint A, M
for $i = 0, 1, \cdots$ **do**
  if $((i > 0)$ and $(i\%d = 0))$ **then**
    if $(||checksum(r) - c^T r^{(i)}||/n > \theta)$ or $(||checksum(x) - c^T x^{(i)}||/n > \theta)$ **then** — 2 VDP (4n FLOPS)
      Rollback: recover $A, M, i, \rho_i, p^{(i)}, x^{(i)}, r^{(i)}$ — 2 Mcpy + 4 Vcpy
    else if $(i\%(cd) = 0)$ **then**
      Checkpoint: $i, \rho_i, p^{(i)}, x^{(i)}$ — 2 Vcpy
    end if
  end if
  $q^{(i)} = Ap^{(i)}$ — 1 MVM (nnz FLOPS)
  $checksum(q) = checksum(A) p^{(i)} + d \cdot checksum(p)$ — 1 VDP (2n FLOPS)
  $\alpha_i = \rho_i / {p^{(i)}}^T q^{(i)}$ — 1 VDP (2n FLOPS)
  $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$ — 1 VLO (2n FLOPS)
  $checksum(x) = checksum(x) + \alpha_i \cdot checksum(p)$ — 2 FLOPS
  $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$ — 1 VLO (2n FLOPS)
  $checksum(r) = checksum(r) - \alpha_i \cdot checksum(q)$ — 2 FLOPS
  solve $Mz^{(i+1)} = r^{(i+1)}$ — 1 PCO
  $checksum(z) = \frac{(checksum(r) - checksum(M) z^{(i+1)})}{d}$ — 1 VDP (2n FLOPS)
  $\rho_{i+1} = {r^{(i+1)}}^T z^{(i+1)}$ — 1 VDP (2n FLOPS)
  $\beta_i = \rho_{i+1}/\rho_i$ — 1 FLOPS
  $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$ — 1 VLO (2n FLOPS)
  $checksum(p) = checksum(z) + \beta_i \cdot checksum(p)$ — 2 FLOPS
  check convergence; continue if necessary
end for

- Step 3: low-cost *error recovery*
  - Error detect every several iterations => Checkpoint/Rollback
  - Do we need to checkpoint every vector?
    - *Observation 3*: Using ***p*** and ***x*** can compute the other 3 vectors
    - **Only checkpoint 2 vectors**, ***p*** and ***x*** (in purple)
- Overhead Summary
  - Checksum update: 2 vector dot-products every iteration
  - Error detection: 2 vector dot-products every *d* iterations
  - Rollback recovery: 2 matrix copies every *cd* iterations
  - Checkpoint: 2 vector copies every *cd* iterations

"Lazy" Detection: Low-Cost Online ABFT Algorithm (cont.)

Algorithm 1 Online-ABFT Preconditioned Conjugate Gradient Algorithm Based on New Checksum with checkpoint/restart Technique

```
Compute r^(0) = b − Ax^(0), z^(0) = M^{-1} r^(0), p^(0) = z^(0)
    and ρ_0 = r^(0)T z^(0) for some initial guess x^(0)
Compute checksum(A) = c^T A − dc^T, checksum(M) = c^T M − dc^T, checksum(b) = c^T b^(0), checksum(x) = c^T x^(0), checksum(r) = c^T r^(0), checksum(z) = [(checksum(r) − checksum(M) z^(0))]/d, checksum(p) = checksum(z)
Checkpoint A, M
for i = 0, 1, ··· do
  if ((i > 0) and (i%d = 0)) then
    if (||checksum(r) − c^T r^(i)||/n > θ) or (||checksum(x) − c^T x^(i)||/n > θ) then      [2 VDP (4n FLOPS)]
      Rollback: recover A, M, i, ρ_i, p^(i), x^(i), r^(i)      [2 Mcpy + 4 Vcpy]
    else if (i%(cd) = 0) then
      Checkpoint: i, ρ_i, p^(i), x^(i)      [2 Vcpy]
    end if
  end if
  q^(i) = A p^(i)      [1 MVM (nnz FLOPS)]
  checksum(q) = checksum(A) p^(i) + d · checksum(p)      [1 VDP (2n FLOPS)]
  α_i = ρ_i / p^(i)T q^(i)      [1 VDP (2n FLOPS)]
  x^(i+1) = x^(i) + α_i p^(i)      [1 VLO (2n FLOPS)]
  checksum(x) = checksum(x) + α_i · checksum(p)      [2 FLOPS]
  r^(i+1) = r^(i) − α_i q^(i)      [1 VLO (2n FLOPS)]
  checksum(r) = checksum(r) − α_i · checksum(q)      [2 FLOPS]
  solve M z^(i+1) = r^(i+1)      [1 PCO]
  checksum(z) = (checksum(r) − checksum(M) z^(i+1)) / d      [1 VDP (2n FLOPS)]
  ρ_{i+1} = r^(i+1)T z^(i+1)      [1 VDP (2n FLOPS)]
  β_i = ρ_{i+1} / ρ_i      [1 FLOPS]
  p^(i+1) = z^(i+1) + β_i p^(i)      [1 VLO (2n FLOPS)]
  checksum(p) = checksum(z) + β_i · checksum(p)      [2 FLOPS]
  check convergence; continue if necessary
end for
```

- Step 3: low-cost *error recovery*
  - Error detect every several iterations => Checkpoint/Rollback
  - Do we need to checkpoint every vector?
    - *Observation 3*: Using ***p*** and ***x*** can compute the other 3 vectors
    - **Only checkpoint 2 vectors**, ***p*** and ***x*** (in purple)
- Overhead Summary
  - Checksum update: 2 vector dot-products every iteration
  - Error detection: 2 vector dot-products every *d* iterations
  - Rollback recovery: 2 matrix copies every *cd* iterations
  - Checkpoint: 2 vector copies every *cd* iterations
  - Original algorithm: 1 MVM and 1 PCO
  - MVM and PCO >> vector dot-product, matrix and vector copy

"Eager" Recovery for MVM: Triple Checksums

Pacific Northwest NATIONAL LABORATORY Proudly Operated by Battelle Since 1965

UCR

- **MVM**: computation-intensive, vulnerable => faster recovery under a high error rate can be beneficial

- **Encoding**:

$$A \rightarrow A^* = \begin{pmatrix} A & \vec{0} & \vec{0} & \vec{0} \\ \vec{c_1}^T A - d_1\vec{c_1}^T - d_2\vec{c_2}^T - d_3\vec{c_3}^T & d_1 & d_2 & d_3 \\ \vec{c_2}^T A - d_2\vec{c_1}^T - d_3\vec{c_2}^T - d_1\vec{c_3}^T & d_2 & d_3 & d_1 \\ \vec{c_3}^T A - d_3\vec{c_1}^T - d_1\vec{c_2}^T - d_2\vec{c_3}^T & d_3 & d_1 & d_2 \end{pmatrix} \quad \vec{x} \rightarrow \vec{x}^* = \begin{pmatrix} \vec{x} \\ \vec{c_1}^T \vec{x} \\ \vec{c_2}^T \vec{x} \\ \vec{c_3}^T \vec{x} \end{pmatrix} \begin{array}{l} \\ \rightarrow \text{checksum}_1(\boldsymbol{x}) \\ \rightarrow \text{checksum}_2(\boldsymbol{x}) \\ \rightarrow \text{checksum}_3(\boldsymbol{x}) \end{array}$$

- **Checksum update:** similar to 1 checksum (see in the paper)

- **Verification**:
  - Detect if there is ***any error***: $checksum_1(y) ?= \vec{c_1}^T \vec{y}$
  - Identify whether there is ***more than one error***:
    - Choose $\boldsymbol{c_1} = (1, 1, \ldots, 1)^T$, $\boldsymbol{c_2} = (1, 2, \ldots, n)^T$, $\boldsymbol{c_3} = (1, 1/2, \ldots, 1/n)^T$
      $(checksum_1(y) - \vec{c_1}^T \vec{y})^2 ?= (checksum_2(y) - \vec{c_2}^T \vec{y})(checksum_3(y) - \vec{c_3}^T \vec{y})$
  - If there is one error, ***locate and correct***:
    - Erroneous locate: $(checksum_2(y) - \vec{c_2}^T \vec{y}) / (checksum_1(y) - \vec{c_1}^T \vec{y})$

"Hybrid" Detection: Two-Level Online ABFT Algorithm

Algorithm 2 Two-Level Online-ABFT Preconditioned Conjugate Gradient Algorithm

Compute $r^{(0)} = b - Ax^{(0)}$, $z^{(0)} = M^{-1}r^{(0)}$, $p^{(0)} = z^{(0)}$ and $\rho_0 = {r^{(0)}}^T z^{(0)}$ for some initial guess $x^{(0)}$
Compute $checksum_1$, $checksum_2$, $checksum_3$ of A, M, x, r, z, p
Checkpoint A, M
for $i = 0, 1, \cdots$ **do**
  if $((i > 0)$ and $(i\%d = 0))$ **then**
    if $(||checksum_1(r) - c_1^T r^{(i)}||/n^2 > \theta)$ or $(||checksum_1(x) - c_1^T x^{(i)}||/n^2 > \theta)$ **then**
      Rollback: recover $A, M, i, \rho_i, p^{(i)}, x^{(i)}, r^{(i)}$
    else if $(i\%(cd) = 0)$ **then**
      Checkpoint: $i, \rho_i, p^{(i)}, x^{(i)}$
    end if
  end if
  $q^{(i)} = Ap^{(i)}$
  $checksum_1(q) = checksum_1(A)p^{(i)} + d_1 \cdot checksum_1(p) + d_2 \cdot checksum_2(p) + d_3 \cdot checksum_3(p)$
  $checksum_2(q) = checksum_2(A)p^{(i)} + d_2 \cdot checksum_1(p) + d_3 \cdot checksum_2(p) + d_1 \cdot checksum_3(p)$
  $checksum_3(q) = checksum_3(A)p^{(i)} + d_3 \cdot checksum_1(p) + d_1 \cdot checksum_2(p) + d_2 \cdot checksum_3(p)$
  $\delta_1 = c_1^T q^{(i)} - checksum_1(q)$
  $\delta_2 = c_2^T q^{(i)} - checksum_2(q)$
  $\delta_3 = c_3^T q^{(i)} - checksum_3(q)$
  if $(|\delta_1| > \theta)$ **then**
    if $(|1 - \delta_1^2/\delta_2\delta_3| < \theta)$ **then**
      Locate and correct now:
      $j = \delta_2/\delta_1$
      $q_j^{(i)} = q_j^{(i)} - \delta_1$
    else
      Rollback: recover $A, M, b, i, \rho_i, p^{(i)}, x^{(i)}, r^{(i)}$
    end if
  end if
  $\alpha_i = \rho_i / {p^{(i)}}^T q^{(i)}$
  $x^{(i+1)} = x^{(i)} + \alpha p^{(i)}$
  $checksum_1(x) = checksum_1(x) + \alpha_i \cdot checksum_1(p)$
  $checksum_2(x) = checksum_2(x) + \alpha_i \cdot checksum_2(p)$
  $checksum_3(x) = checksum_3(x) + \alpha_i \cdot checksum_3(p)$
  $r^{(i+1)} = r^{(i)} - \alpha q^{(i)}$
  $checksum_1(r) = checksum_1(r) - \alpha_i \cdot checksum_1(q)$
  $checksum_2(r) = checksum_2(r) - \alpha_i \cdot checksum_2(q)$
  $checksum_3(r) = checksum_3(r) - \alpha_i \cdot checksum_3(q)$
  solve $Mz^{(i+1)} = r^{(i+1)}$
  Compute $checksum_1(z)$, $checksum_2(z)$, $checksum_3(z)$
  $\rho_{i+1} = {r^{(i+1)}}^T z^{(i+1)}$
  $\beta_i = \rho_{i+1}/\rho_i$
  $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
  $checksum_1(p) = checksum_1(z) + \beta_i \cdot checksum_1(p)$
  $checksum_2(p) = checksum_2(z) + \beta_i \cdot checksum_2(p)$
  $checksum_3(p) = checksum_3(z) + \beta_i \cdot checksum_3(p)$
  check convergence; continue if necessary
end for

Outer Protection

Inner Protection

- **Inner protection**
  - Detect a single error in MVM: locate and correct
  - Detect multiple errors in MVM: rollback immediately
- **Outer Protection**
  - Detect error(s) in VLOs and multiple errors can not be recovered in inner protection

“Hybrid” Detection: Two-Level Online ABFT Algorithm (cont.)

General procedure to construct a Two-Level ABFT:

1. Encode matrices and vectors
2. Compute checksum updates after MVM, VLO, PCO
3. Analyze dependency relationship between vectors
4. Every *d* iterations, invoke the outer-level protection
5. Every *cd* iterations, checkpoint the minimum number of vectors
6. After each MVM, add the inner-level protection

Outline

- Algorithm-Based Fault Tolerance (ABFT)
- Limitations of Traditional ABFT for matrix-vector multiplication (MVM)
- Limitations of Existing Techniques for FT-Iterative Methods
- Our Designs
  - Novel Error-Preserving Checksum for MVM
  - New Online ABFT Schemes
- Theoretical Comparison
- Empirical Evaluation
- Conclusions

Theoretical Comparison - Error Coverage and Feature

	Offline residual	Online MV	Online orthogonality	Basic/two-level online ABFT method
Can protect arithmetic error	Yes	Yes	Yes	Yes
Can protect memory bit flips	Yes	Yes	Yes	Yes
Can protect cache or register bit flips	Yes	No	No	Yes
Can be applied to all iterative methods	Yes	Yes	No	Yes
Not necessary to check every iteration	Yes	No	Yes	Yes
Not necessary to check every operation	Yes	No	Yes	Yes

Offline residual: verify residual at the end of computation and recompute

Online MV: online MVM scheme using the traditional checksum proposed by [Sloan et al., DSN'13]

Online Orthogonality: online orthogonally checking proposed by [Chen, PPOPP'13]

Basic Online ABFT: our proposed "lazy" online ABFT using checksum updates and C/R

Two-Level ABFT: our proposed two-level online ABFT using triple-checksum mechanism

Theoretical Comparison - Performance

- 3 designed scenarios
  - One error in MVM during the **entire execution** – **low error rate**
  - One error in MVM **every *cd* iterations** – **medium/high error rate**
  - One error in MVM **every iteration** – **extremely high error rate**

	Basic online ABFT (O_1)	Two-level online ABFT (O_2)	Online MV (O_3)
Scenario 1	(2/d+2)VDP+2VLO/cd	(2/d+9)VDP+2VLO/cd	1PCO+2VDP+3VLO
Scenario 2	0.5MVM+(2/d+5)VDP+0.5PCO+(6(1+c_0)/cd + 1.5)VLO	(2/d+9)VDP+2VLO/cd	1PCO+(5/cd+2)VDP+3VLO
Scenario 3	$+\infty$	(2/d+9)VDP+2VLO/cd	1PCO+7VDP+3VLO

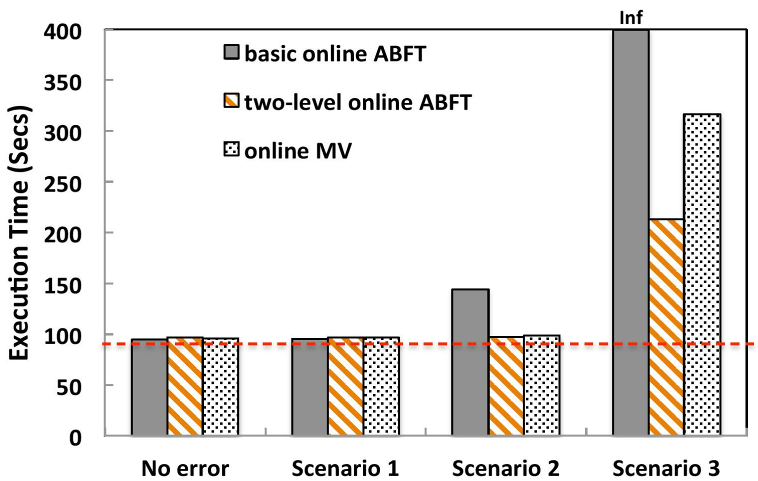
- Theoretical Comparison on PCG
  - Low error rate: **basic online ABFT** has the lowest overhead
  - Medium/high error rate: **two-level online ABFT** has the lowest overhead
  - Extremely high error rate: **two-level online ABFT** has the lowest overhead

- Overall, no matter the error rate, one of our approaches will outperform the online MV for PCG.

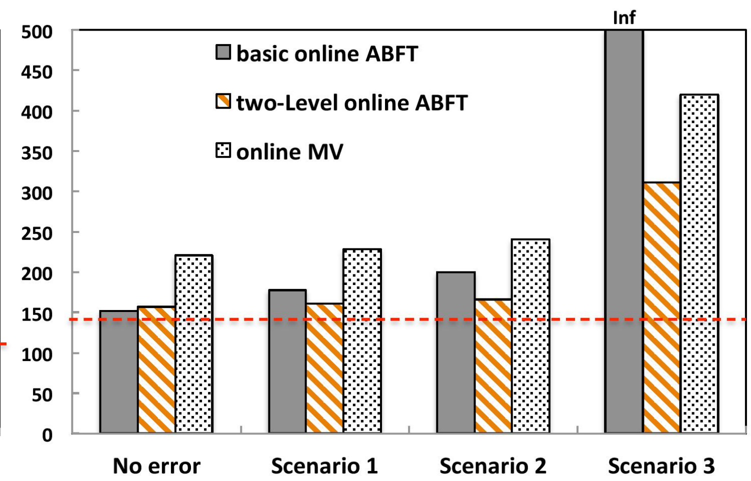
Empirical Evaluation – Configurations

- Platforms
  - **Stampede** supercomputer at TACC, each node with 2 Intel Xeon E5-2680 processors
  - **Tianhe-2** supercomputer (No.1 in Top 500), each node with 2 Intel Xeon E5-2692 processors
- Implemented our proposed online ABFT schemes in **PETSc**
- Evaluated solvers
  - Preconditioned Conjugate Gradient (**PCG**): has orthogonality relation
  - Preconditioned Biconjugate Gradient Stabilized (**PBiCGSTAB**): no orthogonality relation (against [Chen, PPOPP'13])
- Input Matrix: G3_circuit
  - The **largest** SPD matrix from the University of Florida Sparse Matrix Collection
  - 1,585,478 rows and columns with 7,660,826 nonzero elements

Empirical Evaluation – Results



(a) Comparison with **PCG** on ***Stampede***

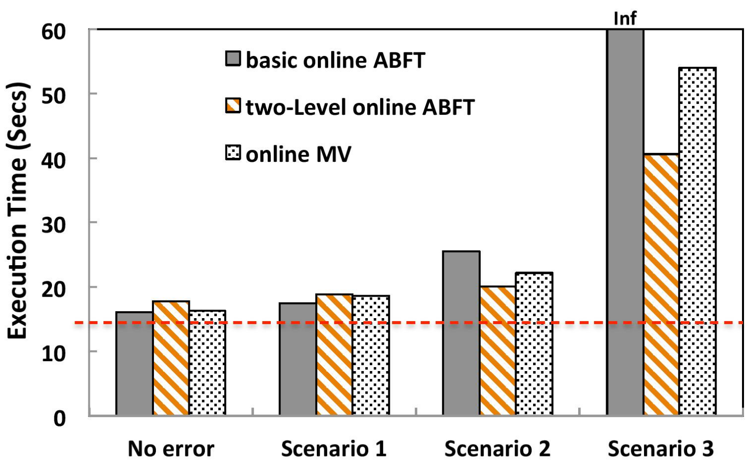


(b) Comparison with **PBiCGSTAB** on **Stampede**

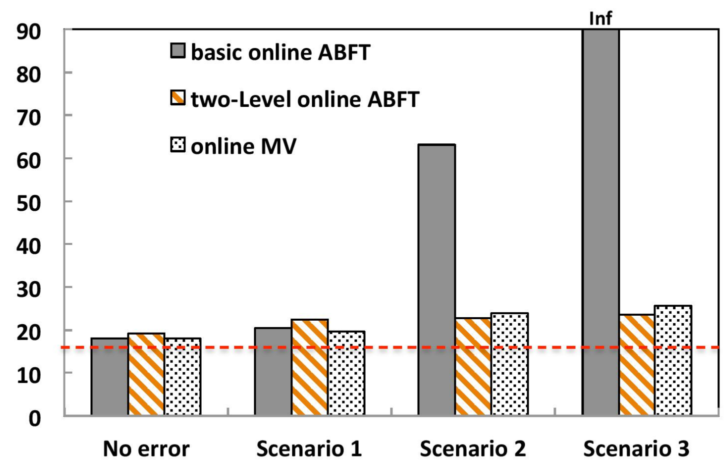
- Empirical comparison
  - Failure-free: overhead is low for both proposed online ABFT (***0.4%*** and ***1.3%*** for PCG, ***1.0%*** and ***4.0%*** for PBiCGSTAB)
  - Low error rate: **basic online ABFT** has the lowest overhead
  - Medium/high error rate: **two-level online ABFT** has the lowest overhead
  - Extremely high error rate: **two-level online ABFT** has the lowest overhead (online MV is ***48%*** higher than two-level)

Consistent with theoretical analysis

Empirical Evaluation – Results (cont.)



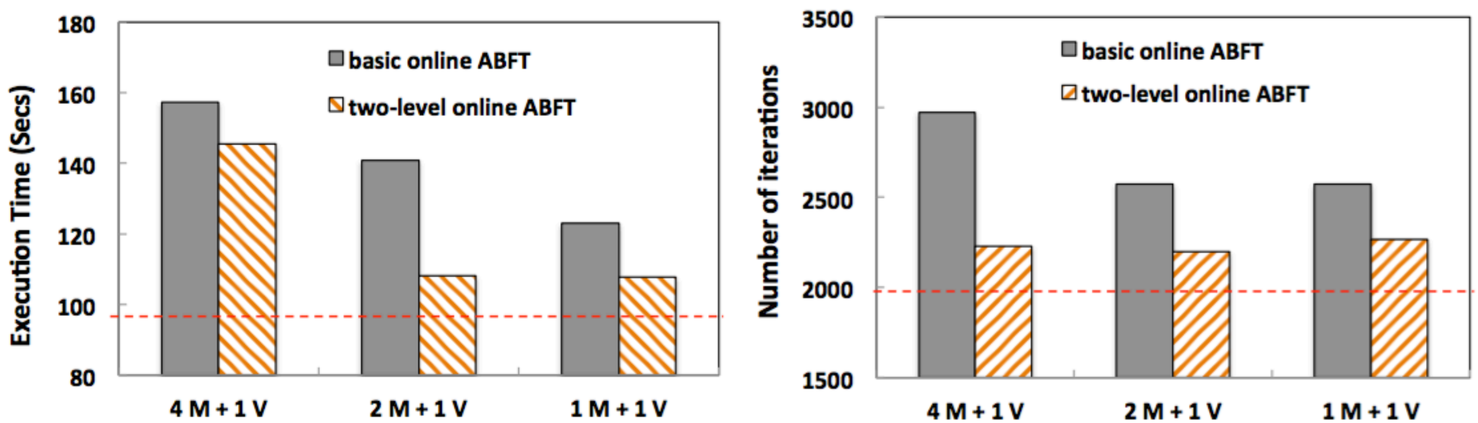
(c) Comparison with **PCG** on ***Tianhe-2***



(d) Comparison with **PBiCGSTAB** on ***Tianhe-2***

Similar conclusions as on ***Stampede***.

Empirical Evaluation – Results (cont.)



(e) Comparison with **PCG** under a **high error-rate** scenario on ***Stampede***

Two-level online ABFT outperforms **basic** online ABFT by *32.1%* on average under the high error rate scenario.

Conclusions

- HPC platforms are anticipated to be more susceptible to soft errors in both logic circuits and memory subsystems
- Proposed a new checksum encoding mechanism
- Developed two online ABFT algorithms for general iterative methods – “basic” and “two-level” – that allow errors to be detected eagerly and lazily
- Experimental results demonstrate our designs are efficient and effective to detect and recover soft errors for general iterative methods

Thank you !

Contact:

Dingwen Tao (dtao001@cs.ucr.edu)

Shuaiwen Leon Song (Shuaiwen.Song@pnnl.gov)

Acknowledgement: DOE CESAR Project, DOE ADEM Project and NSF