University of Basel

Resilient N-Body Tree Computations with Algorithm-Based Focused Recovery: Model and Performance Analysis

Aurélien Cavelan[1,2] Aiman Fang[3] Andrew A. Chien[3,4] Yves Robert[2,5]

<aurelien.cavelan@unibas.ch>

[1]University of Basel, Switzerland. [2]Laboratoire LIP, ENS Lyon & Inria, France.

[3]University of Chicago, USA. [4]Argonne National Laboratory, USA.

[5]University of Tennessee Knoxville, USA.

Resilience: Top 10 Exascale Challenges

In February 2014, the Advanced Scientific Computing Advisory Committee identified resilience as one of the top 10 challenges for Exascale:

Ensuring **correct scientific computation** in face of faults, **reproducibility**, and **algorithm verification challenges**.

Computing at Exascale

To achieve 1000x increase in performance:

- Larger node count: 10^5 or 10^6 nodes, each with 10^2 or 10^3 cores
- Shorter Mean Time Between Failures (MTBF)

Theorem: $MTBF_p = \frac{MTBF_{\text{ind}}}{p}$ for arbitrary distributions.

MTBF (individual node)	1 year	10 years	100 years
MTBF (platform of 10^6 nodes)	30 secs	5 mins	50 mins

We need more resilient techniques!

Silent Data Corruptions (SDCs)

In this paper, we focus on **Silent Data Corruptions**:

- Caused by bit-flips in the memory
- RAM, CPU, GPU, cache, disk, ...
- Errors are only detected when corrupted data is activated
- Long **detection latency**: thousands (10^3) to billions (10^9) cycles
- Undetected errors **propagate** and corrupt application data

Existing hardware protections are not enough:

- ECC/chipkill codes only target specific parts of the memory
- We focus on errors that escape such simple system level detection

Errors can be exposed via sophisticated application checks.

Silent Error Detectors

General-purpose approaches

- Replication [**Fiala et al. 2012**] or triple modular redundancy and voting [**Lyons and Vanderkulk 1962**]

Application-specific approaches

- Algorithm-based fault tolerance (ABFT): checksums in dense matrices limited to one error detection and/or correction in practice [**Huang and Abraham 1984**]
- Partial differential equations (PDE): use lower-order scheme as verification mechanism [**Benson, Schmit and Schreiber 2014**]
- Preconditioned conjugate gradients (PCG): orthogonalization check every k iterations, re-orthogonalization if problem detected [**Sao and Vuduc 2013, Chen 2013**]

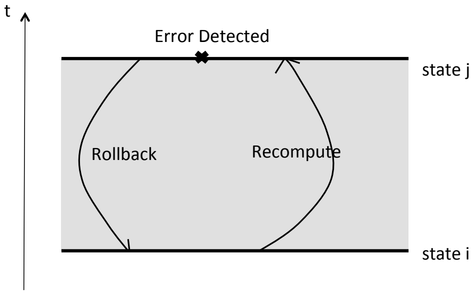
Data-analytics approaches

- Dynamic monitoring of HPC datasets based on physical laws (e.g., temperature limit, speed limit) and **space or temporal proximity** [**Bautista-Gomez and Cappello 2014**]
- Time-series prediction, **spatial multivariate interpolation** [**Di et al. 2014**]

Standard Checkpoint and Recovery (CR)

We must ensure that checkpoints are correct:

- Check application data before checkpointing with SDC detector
- Upon error, rollback to the last correct checkpoint and re-execute



There are downsides:

- Only a small fraction of the data might be corrupted
- **But classic CR incurs considerable overhead in case of error**

This Paper

Application-Based Focused Recovery (ABFR):

ABFR takes a different approach:

- Exploits application dataflow to bound the impact of the error
- **Only recompute potentially corrupted data**

ABFR works in three steps:

1. **Inverse propagation**: identify potential root causes (PRC)
2. **Diagnosis**: locate the root cause of the error (prune PRCs)
3. **Recomputation**: recompute potentially corrupted data

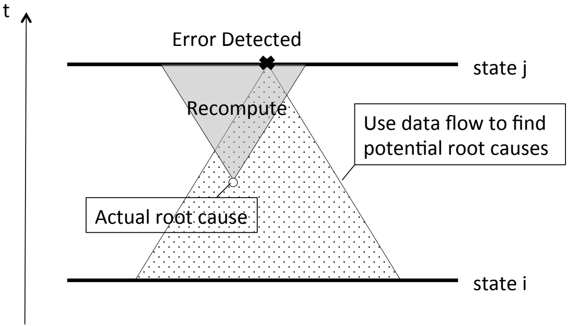
We take advantage of frequent versioning:

- Exploits DRAM and high bandwidth and capacity burst buffers
- This allows frequent versioning of application data

ABFR for Stencil Computations (previous work)

ABFR works in three steps:

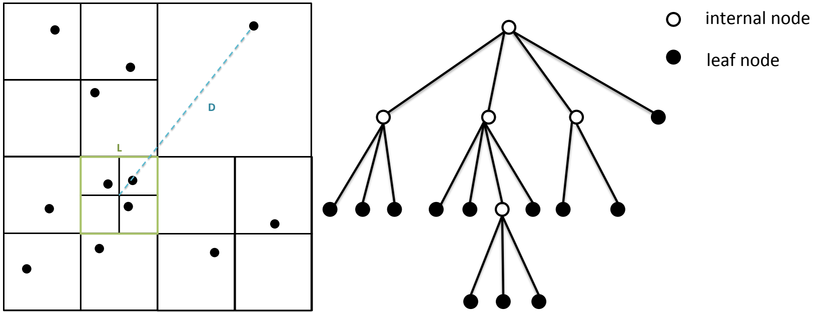
1. **Inverse propagation**: identify potential root causes (PRC)
2. **Diagnosis**: locate the root cause of the error (prune PRCs)
3. **Recomputation**: recompute potentially corrupted data



ABFR only recomputes a small fraction of the data.

Quad-Tree for N-Body Computations

We investigate the impact of ABFR on N-Body computations.



N-Body computations are much more challenging:

- Information is exchanged along the tree
- Nodes are updated at different time intervals

Contributions

In this paper we focus on **perfect binary trees**:

- It encompasses the intrisic comlexity of the model
- The model remains valid for arbitrary N-Body trees
- It is amenable to an exact analytical evaluation

Two major contributions:

1. A detailed analytical model to compare ABFR with CR
2. A comprehensive performance study for perfect binary trees

The goal is obtain a better understanding of the potential impact of ABFR on N-body computations.

Application Model

We consider a perfect binary tree $\mathcal{T}_n$ of depth n.
We assume the following execution model (example with $n = 3$):

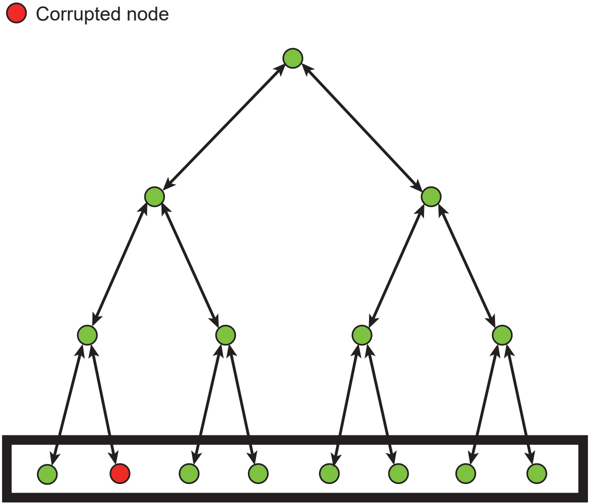
Level	Iterations							
	1	2	3	4	5	6	7	8
0								2^0
1				2^1				2^1
2		2^2		2^2		2^2		2^2
3	$K \cdot 2^3$	$K \cdot 2^3$	$K \cdot 2^3$	$K \cdot 2^3$	$K \cdot 2^3$	$K \cdot 2^3$	$K \cdot 2^3$	$K \cdot 2^3$

We version every step.
We assume exponentially distributed errors.

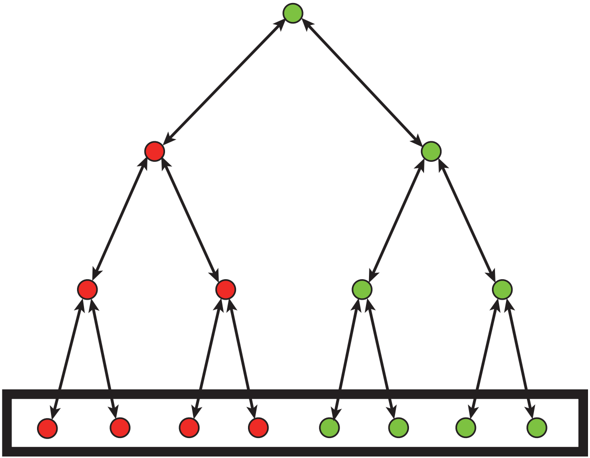
We make the following assumptions on the detector

- The detector is computationally expensive
- The SDC detector has 100% coverage
- The detector signals the manifestation of the error

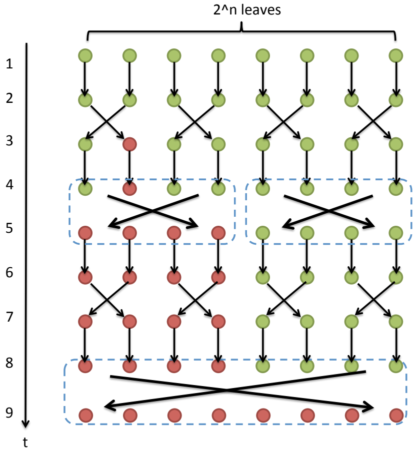
Error Propagation ($t = 3$)



Error Propagation ($t = 5$)



Error Propagation ($t = 9$)



- After one period, all nodes are corrupted
- Detection should be done after iteration 2, 4, or 8

Model Parameters

Definitions	
n	Height of tree
K	Number of updates performed at level n (tree leaves)
Error Rate	
λ	Errors per second per leaf
Time	
c	Time to compute one leaf
d	Time to detect errors on one leaf
v	Time to version one leaf
r	Time to recover one leaf
Tree-wise	
T_c	Time to compute the tree without errors
T_d	Time for detection the tree without errors
T_v	Time for versioning the tree without errors
Frequency	
D	**Detection interval** of the form $2^x \cdot K$

Analytical Model

Expected execution time for one period with **CR**:

$$\mathbb{E}(T_{CR}) = T_c + 2^n \cdot (d + v) + (1 - e^{-\lambda T_c})T_c \ .$$

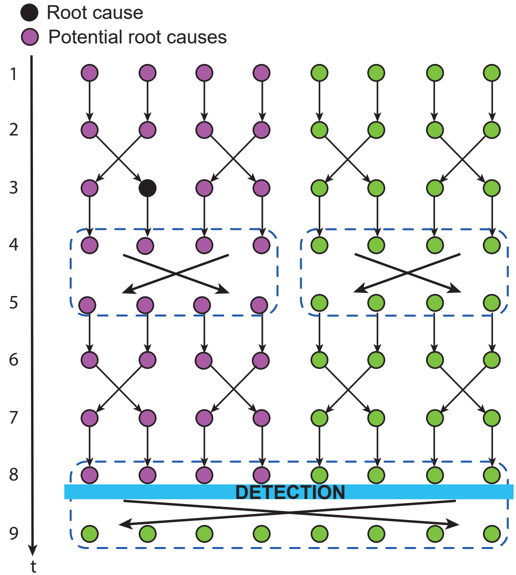
Expected execution time for one period with **ABFR**:

$$\mathbb{E}(T_{ABFR}) = T_c + T_d + T_v + (1 - e^{-\lambda T_c}) \left(\mathbb{E}(T_{diag}) + \mathbb{E}(T_{recomp})\right)$$

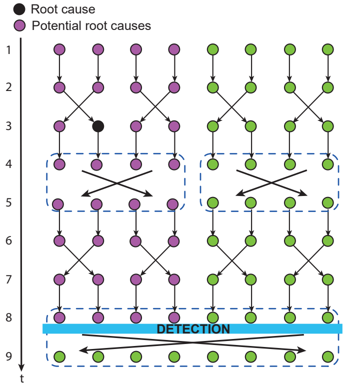
ABFR works in three steps:

1. **Inverse propagation**: identify potential root causes (PRC)
2. **Diagnosis**: bound error impact (prune PRC)
3. **Recomputation**: recompute potentially corrupted data

1. Inverse Propagation



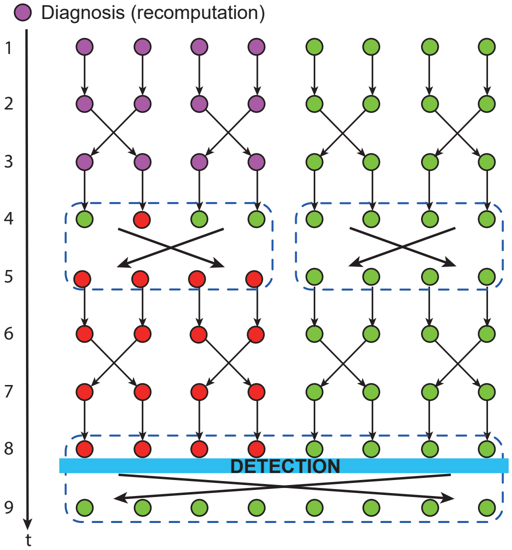
1. Inverse Propagation



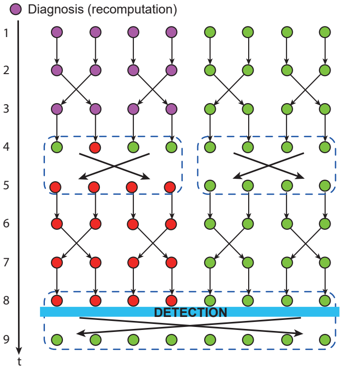
- The number of PRCs depends on the detection interval $D = 2^x K$
- The error can only be located in the 2^{x-1} nodes
- In total, there are $2^x \cdot 2^{x-1}$ PRCs

Number of PRCs strongly depends on the detection interval.

2. Diagnosis



2. Diagnosis

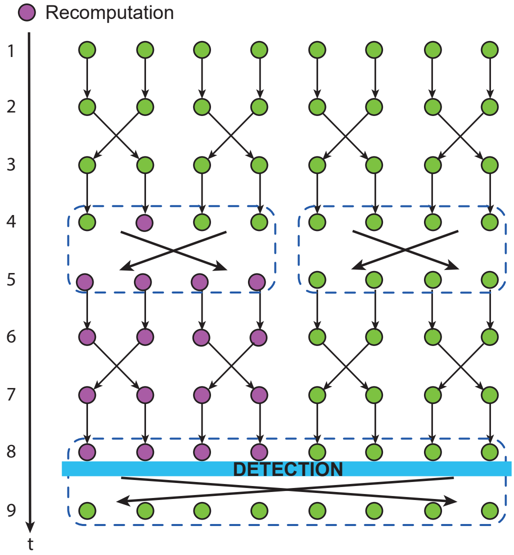


Assuming that faults are uniformly distributed among nodes:

$$\mathbb{E}(T_{diag}) = \sum_{i=1}^{2^x} \frac{1}{2^x} \cdot i \cdot 2^{x-1}(Kc + r)$$

Cost of diagnosis strongly depends on the detection interval.

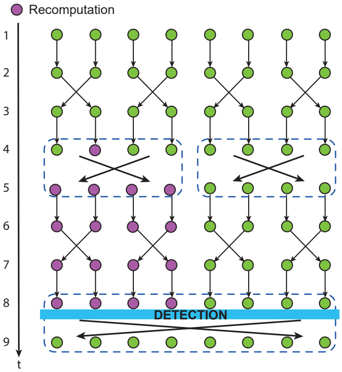
3. Recomputation



3. Recomputation

$$\mathbb{E}(T_{recomp}) = \frac{1}{2} \begin{cases} 2^{n-1}2^{n-1}(K \cdot c + v) + \frac{1}{2} \begin{cases} 2^{n-2}2^{n-2}(K \cdot c + v) + \frac{1}{2} \begin{cases} (K \cdot c + \\ 0 \end{cases} \\ \frac{1}{2} \begin{cases} (K \cdot c + v) \\ 0 \end{cases} \end{cases} \\ \frac{1}{2} \begin{cases} 2^{n-2}2^{n-2}(K \cdot c + v) + \frac{1}{2} \begin{cases} (K \cdot c + v) \\ 0 \end{cases} \\ \frac{1}{2} \begin{cases} (K \cdot c + v) \\ 0 \end{cases} \end{cases} \end{cases}$$

3. Recomputation



$$\mathbb{E}(T_{recomp}) = \frac{1}{6}(4^x - 1)(Kc + v)$$

Again, recomputation cost depends on the detection interval.

Expected Execution Time

Finally, **we obtain an exact analytical formula:**

$$\mathbb{E}(T_{ABFR}) = T_c + T_d + T_v + \left(1 - e^{-\lambda T_c}\right)\left(\mathbb{E}(T_{diag}) + \mathbb{E}(T_{recomp})\right)$$

$$\mathbb{E}(T_{diag}) = \sum_{i=1}^{2^x} \frac{1}{2^x} \cdot i \cdot 2^{x-1}(Kc + r)$$

$$\mathbb{E}(T_{recomp}) = \frac{1}{6}(4^x - 1)(Kc + v)$$

What is the optimal detection interval? i.e. optimal x?
Find tradeoff between **detection cost**, **diagnosis** and **recomputation**.

Expected Overhead

Expected overhead for **CR**:

$$\mathbb{E}(H_{CR}) = \frac{\mathbb{E}(T_{CR})}{T_c} - 1 \ .$$

Expected overhead for **ABFR**:

$$\mathbb{E}(H_{ABFR}) = \frac{\mathbb{E}(T_{ABFR})}{T_c} - 1 \ .$$

We can analytically compare ABFR and CR. Finally, in order to get the optimal value for x, we need to solve (numerically):

$$\frac{\partial \mathbb{E}(H_{ABFR})}{\partial x} = 0$$

Simulations

The goal is twofold:

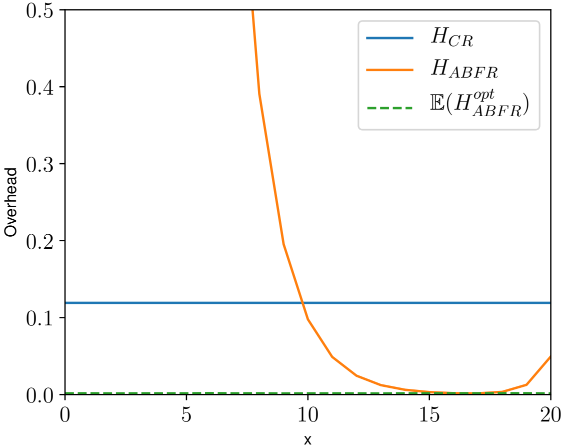
- Show the accuracy of the theoretical analysis
- Assess the performance of ABFR against CR

Simulation settings:

- Number of nodes in the tree: $n = \lceil log_2(10^6) \rceil = 20$
- Computing one leaf: $c = 10^{-5}$s
- Repetition at leaf level: $K = 100$
- Reload?version cost: $r = v = \frac{c}{100}$
- Detection cost: $d = 100 \cdot c$
- Error rate: $\lambda = 1.15 \cdot 10^{-10}$

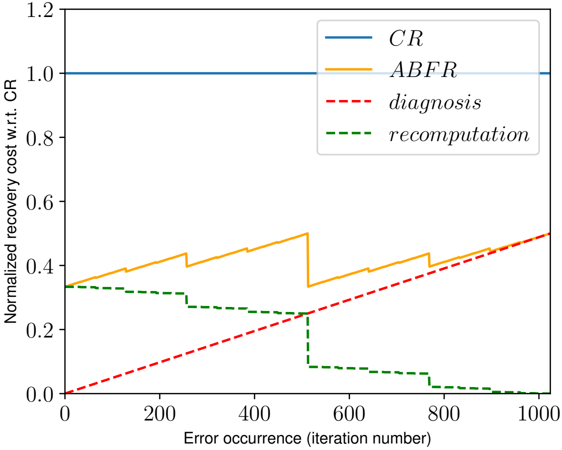
The overhead of the simulation is obtained by averaging the results of 1000 runs.

Optimal Detection Interval



$$D^{opt} = 2^{17}K.$$

Normalized Recovery Cost



Conclusion

- We have applied **ABFR** for N-Body tree computations to efficiently recover from latent errors
- We have proposed an **analytical model** to compare the performace of **ABFR** against **CR**
- Simulation results show that **ABFR reduces recovery overhead by** 60% compared to the standard **CR** approach
- While the model is built for binary trees, it can be generalized for arbitrary higher dimensions

Future directions include applying ABFR to production N-Body tree codes

Questions?

aurelien.cavelan@unibas.ch