Argonne NATIONAL LABORATORY

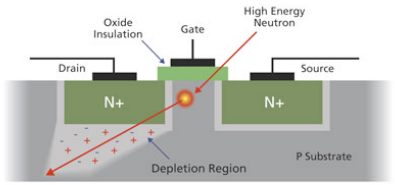
Toward Approximate Detection of Silent Data Corruptions

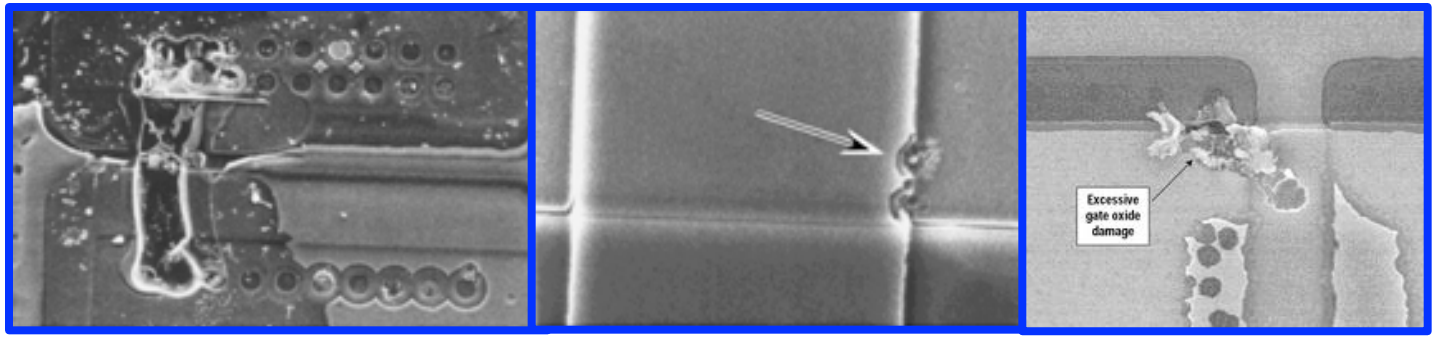
Franck Cappello
ANL&UIUC

Clusters, Clouds, and Data for Scientific Computing, CCDSC 2014

Type of errors

- Hard error: An irreversible change in operation that is typically associated with permanent damage to one or more elements of a device or circuit (e.g., gate oxide rupture, etc.).





metal melt, gate oxide damage

- Soft error (transient errors): An erroneous output signal from a latch or memory cell that can be corrected by performing one or more normal functions of the device containing the latch or memory cell:
  - Cause: Alpha particles from package decay, Cosmic rays creating energetic neutrons
  - Soft errors can occur on transmission lines, in digital logic, processor pipeline, etc.
  - Soft error rate (SER).
  - **BW (1.5PB of memory): 1M memory errors in 261 days → 1 every 4s**

Silent Data Corruptions?

Errors affecting data become Silent Data Corruptions if they are not detected. Some errors in control flow also lead to SDCs.

Silent Data Corruptions may be absorbed (no effect), may lead to process crashes (noticeable effect) or may lead to wrong results (silent effect).

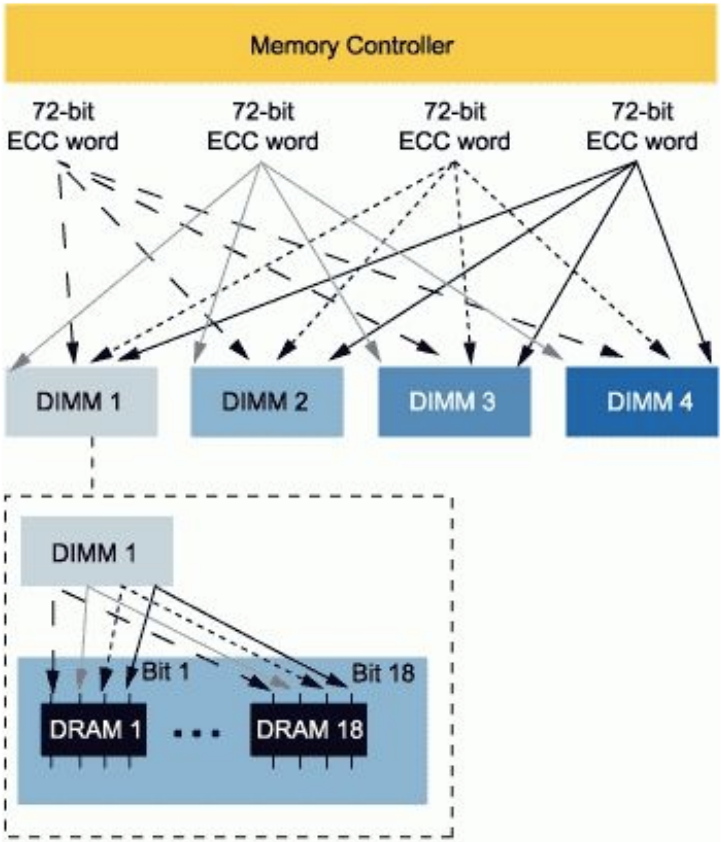
Main source of Errors:

-Memory (but Chipkill and ECC reduce the probability of SDCs to negligible)

BW: 28 uncorrectable errors in 261 days

-Toward exascale (Latchs, Flip flops in processor pipelines and control structures)

Classic Hardware techniques



Exascale Projections (from ICIS report 2014)

FIT: number of failures that can be expected in one billion (10^9) device-hours of operation

Array interleaving and SECDED (Baseline)

	DCE [FIT]		DUE [FIT]		SE [FIT]	
	45 nm	11 nm	45 nm	11 nm	45 nm	11 nm
Arrays	5000	100,000	50	20,000	1	1000
Scattered latches	200	4000	N/A	N/A	20	400
Combinational logic	20	400	N/A	N/A	0	4
DRAM	50	50	0.5	0.5	0.005	0.005
Total (per processor)	1000–5000	100,000	10–100	5000–20,000	10–50	500–5000

~1000 FIT per processor → 1M processors → 1B FIT → 1 SE per hour

Array interleaving and SECDED + latch parity

(45 nm overhead ~10%; 11 nm overhead: ~20% area and ~25% power)

May not happen due to lack of market

	DCE [FIT]		DUE [FIT]		SE [FIT]	
	45 nm	11 nm	45 nm	11 nm	45 nm	11 nm
Arrays	5000	100,000	50	1000	1	5
Scattered latches	200	4000	20	400	0.01	0.5
Combinational logic	20	400	N/A	N/A	0.2	5
DRAM	0	0	0.1	0.0	0.1	0.001
Total	1500–6500	100,000	25–100	2000–10,000	1	5–20

This is not only for exascale: case 1

Los Alamos experiment (Credit: Nathan DeBardeleben)

- NVIDIA Tesla M2090 GPGPUs (Fermi)
- CUDA 5.0
- 40,171 single GPU runs of HPL : 16,738 total hours
- 1624 bad residual (test at the end of HPL): **4% of the runs**
- **Only 1.6%** of those bad residuals reported in event log

$$\frac{\|Ax-b\|_\infty}{\varepsilon\|A\|_1 n}, \quad \frac{\|Ax-b\|_\infty}{\varepsilon\|A\|_1\|x\|_1}, \quad \frac{\|Ax-b\|_\infty}{\varepsilon\|A\|_\infty\|x\|_\infty n},$$

- 11,000 runs of HPL on 2 K20 nodes (Kepler)
- **6 Bad residuals…**

- **And…HPL residual check is a bad test for detecting data corruptions (corruptions on the right part of the mantissa will be unnoticed!)**

Argonne experiment (Credit: Leonardo Bautista Gomez)

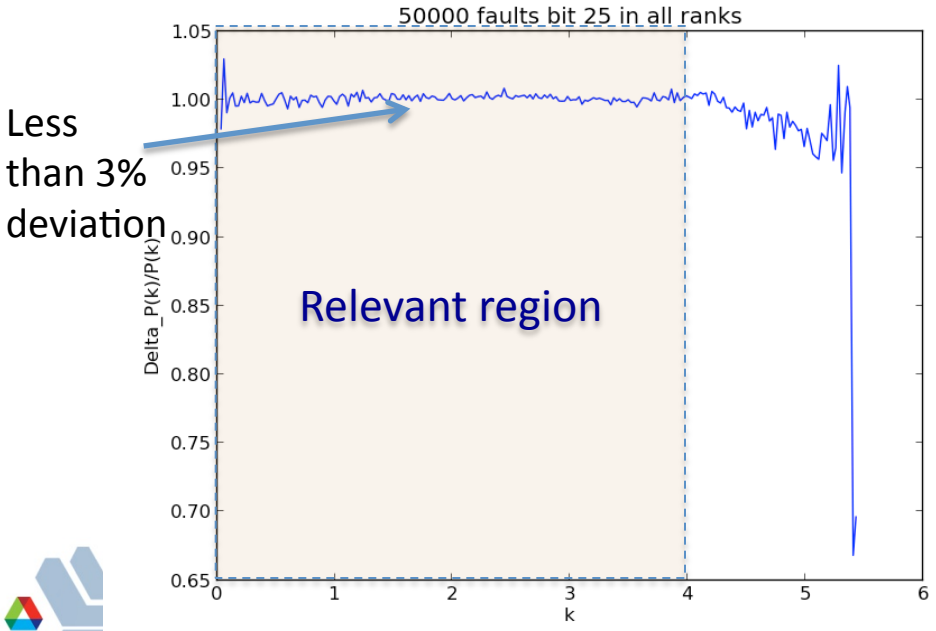
- No corruption on 9000 runs of 20K HPL on M2050 (Fermi)

Sanity checks may not suffice: case 2

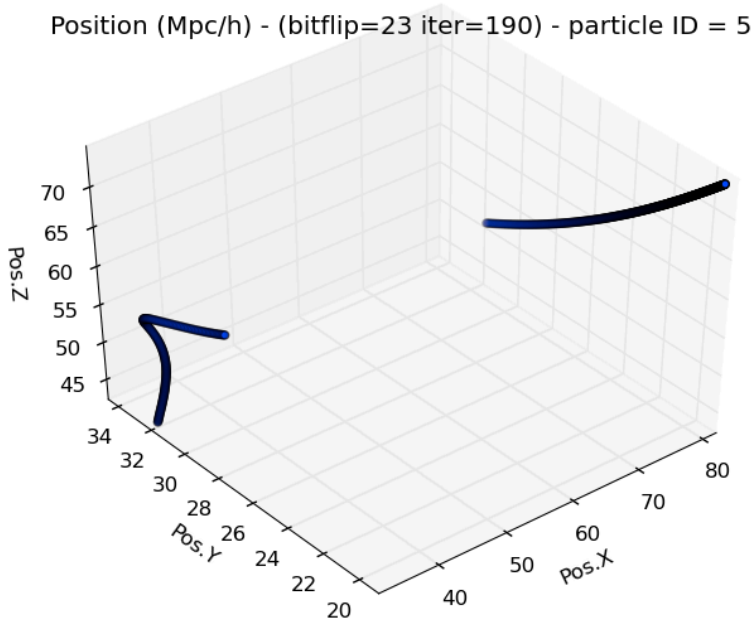
HACC (Cosmology: Particle position from gravity, FFTs, density fields)

- Issues with Memory DIMM, Issues with I/O, Bugs
- The power spectrum represents the amplitude of variations on different mass scales. It is also used as high level checking of the computation.
- Effect of bit flips in particle position are not observable on the power spectrum BUT the result may be corrupted.
- Bit flips in FFTs or density field may lead to more noticeable effects

Power spectrum



Particle trajectory



Optimization algorithms sensitivity: case 3

Optimization algorithms iteratively reduce the error in the initial guess to reach the intended solution.

→They seem to be resilient to SDC: Yes, but to a certain extend!

Checked the Hartree−Fock method (optimization algorithms).

- Inject a single bit flip in floating point #, in bit (**20, 40, 46, 51, 52**, **57, 61, 62, 63**)
- Randomly corrupted data elements
- At least one SDC in each execution
- Obtained statistical results

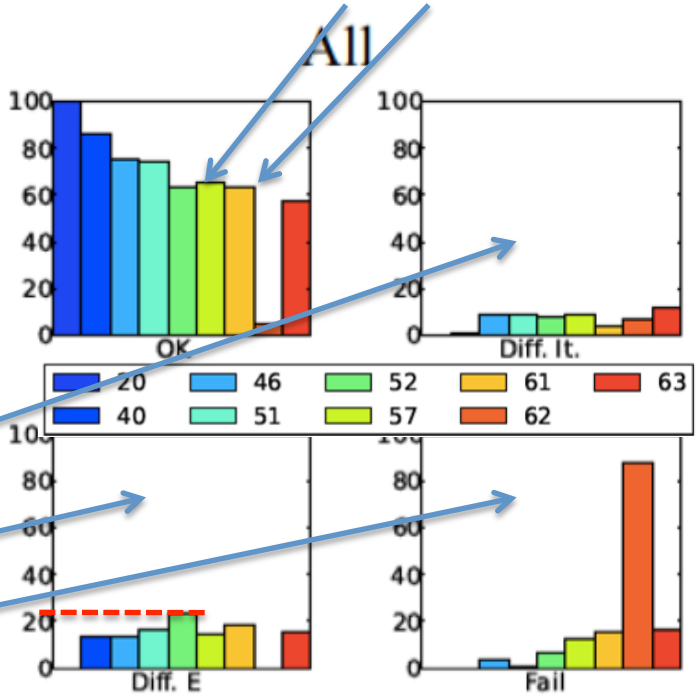
Percentage of executions that converge to

Correct solution or suffer from SDCs

Converge but need more iterations

Converge to different energy

Fail to converge (stop abruptly)

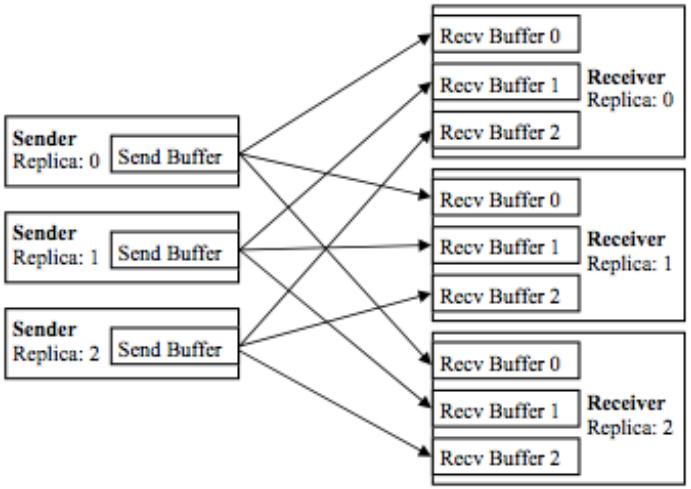


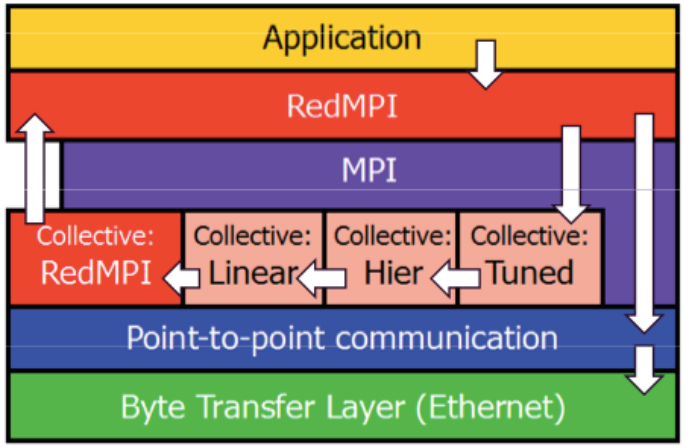
Van Dam, H. J. J., Vishnu, A., and de Jong, W. A. (2013). A case for soft error detection and correction in computational chemistry. Journal of Chemical Theory and Computation, 9(9):3995{4005.

Detection from classic replication

Replicate execution and compare execution state

Ex: RedMPI: compares content of messages coming from MPI process replicas





LAMMPS INPUT CHUTE.SCALED

Size	1x [sec]	2x [sec]	3x [sec]	2x OV	3x OV
128	137.50	138.38	139.01	0.6%	1.1%
256	138.26	140.43	140.00	1.6%	1.3%
512	139.19	140.22	140.67	0.7%	1.1%

- Low performance overhead
- A nice approach if you can afford 2x or 3x the resources and energy.

D. Fiala, F. Mueller, C. Engelmann, K. Ferreira, R. Brightwell, and R. Riesen. Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing, SC12.

Detection from Numerical Algorithms: ABFT

Huang and Abraham, proposed the ABFT (1984) to detect and correct errors in some matrix operations. Use row and column checksums and property of linear algebra: Operation on checksumed inputs produce checksumed resultt hat is checked for detection

To support f failures, f checksums yi are computed

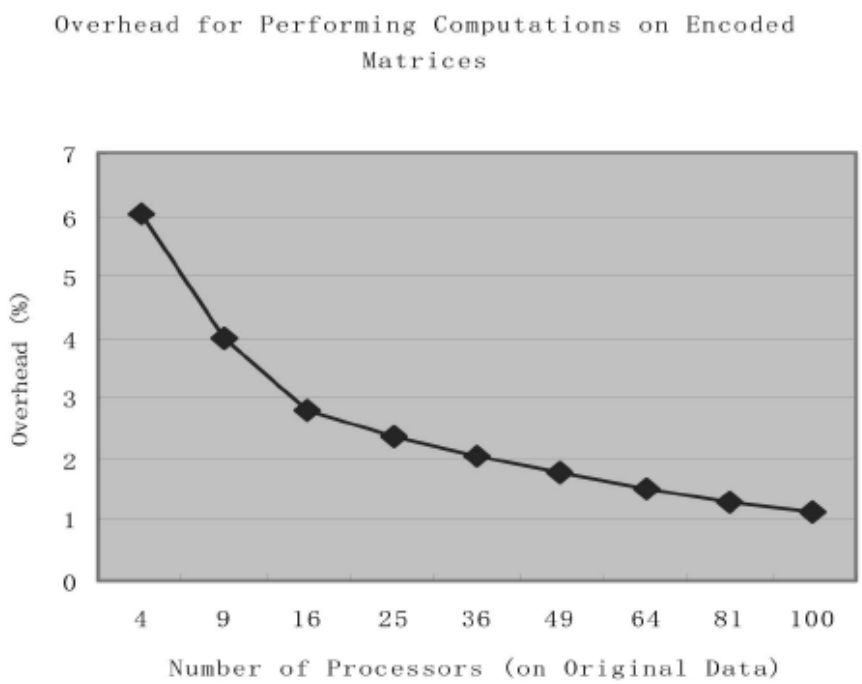
$$\begin{cases} y_1 = a_{11}x_1 + \ldots + a_{1p}x_p, \\ \vdots \\ y_f = a_{f1}x_1 + \ldots + a_{fp}x_p. \end{cases}$$

Works for many Linear Algebra operations:

Matrix Multiplication: A * B = C -> Ac * Br = Cf
LU Decomposition: C = L * U -> Cf = Lc * Ur
Addition: A + B = C -> Af + Bf = Cf
Scalar Multiplication: c * Af = (c * A)f
Transpose: AfT = (AT)f
Cholesky factorization & QR factorization

Performance is quite good

Figure presents the overhead for computing the matrix product on encoded matrices.
The time to compute the checksum should be added as well.



Zizhong Chen, and Jack Dongarra, Algorithm-Based Fault Tolerancefor Fail-Stop Failures, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 19, NO. 12, DEC. 2008

Generalization of ABFT

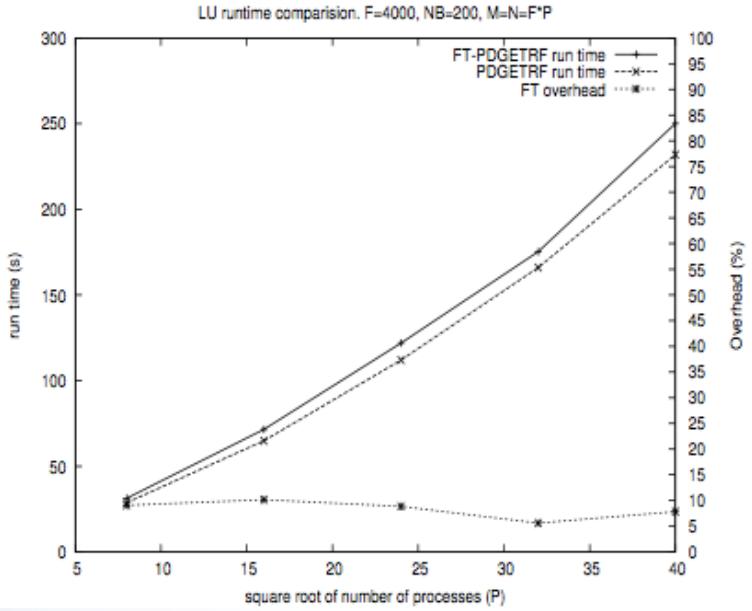
Latest results in that domain

Online ABFT for Krylov subspace iterative methods
simple verification of orthogonality and residual
→ tested on BiCG, and Lanczos biorthogonalization, PCG
→ Works also for GM-RES, Arnoldi (eigenvalue problems), etc.

Zizhong Chen. 2013. Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proceedings ACM* PPoPP '13

1 : 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 $x^{(0)}$
2 : checkpoint: A, M, and b
3 : for $i = 0, 1, \ldots$
4 : if ((i>0) and ($i\%d = 0$))
5 : if ($\frac{{p^{(i+1)}}^T q^{(i)}}{||p^{(i+1)}||.||q^{(i)}||} > 10^{-10}$
or $\frac{||r^{(i+1)} + Ax^{(i+1)} - b||}{||b||.||A||} > 10^{-10}$)
6 : recover: A, M, b, i, ρ_i, $p^{(i)}$, $x^{(i)}$, and $r^{(i)}$.
7 : else if ($i\%(cd) = 0$)
8 : checkpoint: i, ρ_i, $p^{(i)}$, and $x^{(i)}$
9: endif
10: endif
11: $q^{(i)} = Ap^{(i)}$
12: $\alpha_i = \rho_i / {p^{(i)}}^T q^{(i)}$
13: $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$
14: $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$
15: solve $Mz^{(i+1)} = r^{(i+1)}$, where $M = M^T$
16: $\rho_{i+1} = {r^{(i+1)}}^T z^{(i+1)}$
17: $\beta_i = \rho_{i+1} / \rho_i$
10: $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
19: check convergence; continue if necessary
20: end

Online ABFT for LU, QR and Cholesky
→Overhead stays <10% (fault free situation)
(Stampede, weak scaling)



Panruo Wu , Zizhong Chen , FT-ScaLAPACK: Correcting Soft Errors On-Line for ScaLAPACK Cholesky, QR, and LU Factorization Routines, Proceedings of ACM HPDC 2014

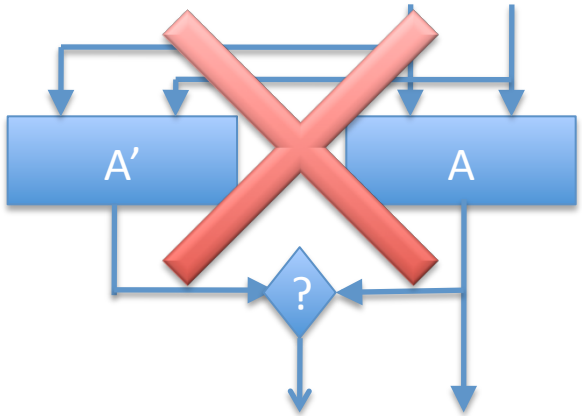
Low cost techniques

Previous techniques try to detect exhaustively errors (up to what the round-off errors would permit)

→ There is no (or little) optimization of the trade-off between coverage, cost (overhead in time, resource, energy) and generality

What about low cost techniques that would detect most of the errors but cannot guarantee detecting all errors?

→ **Approximate detection**



→ The important notions are:

→**Coverage:** % of detected SDCsx

→**Overhead:** % more time, resource, energy (checker+false alerts)

→**Generality:** from algorithm specific to application agnostic

Low Cost Hardware Detection

Example: M. Maniatakos et al., Low-Cost Concurrent Error Detection for Floating-Point Unit (FPU) Controllers (2013)

Conjecture: Error in the control logic will lead to extensive data path corruption, which will propagate to the exponent part of the floating- point representation.

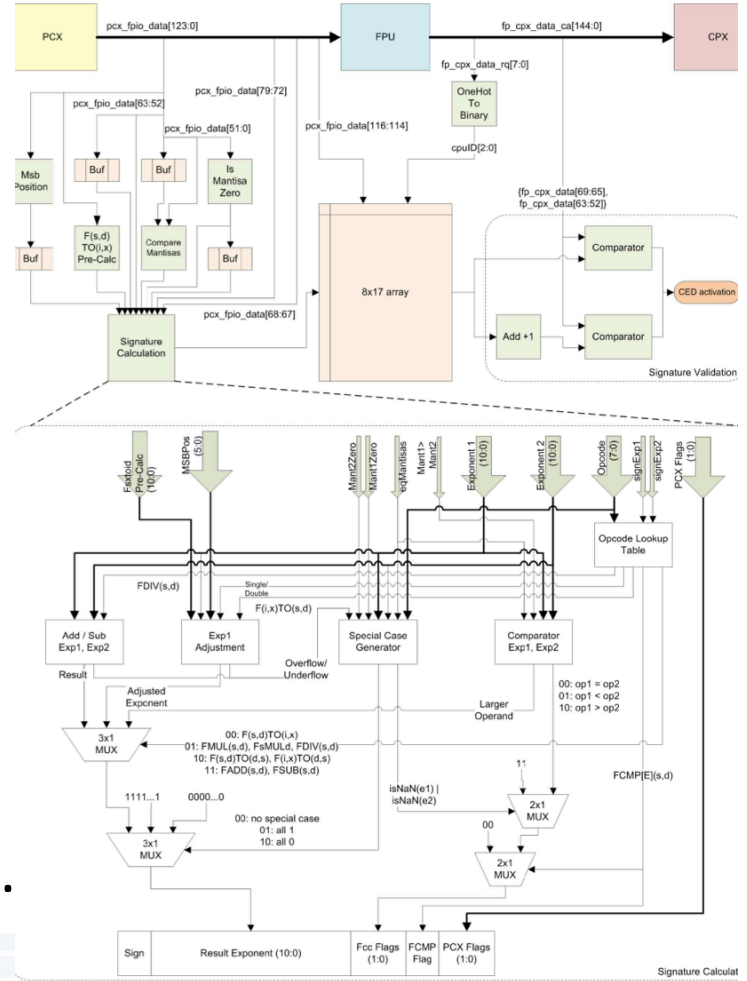
→ Check the exponent

+ base-15 residue code for the fraction

SDC detection solution for the entire FPU:

-**Coverage of 94.1** percent of all errors

-Area cost of 16.32 percent of the FPU size.



Low Cost Algorithm Specific Detection

A first approach is to develop ad-hoc detection approaches

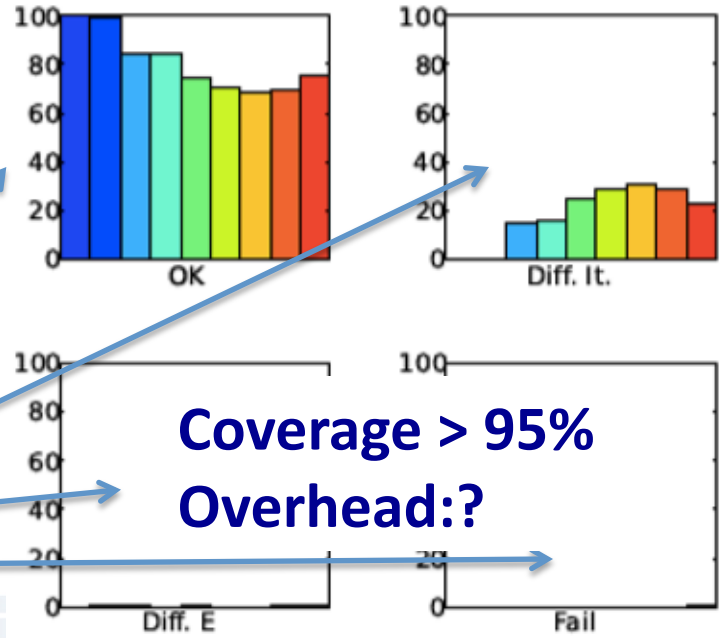
Case 3: Hartree–Fock optimization algorithm:

- Mechanisms for detecting and correcting soft errors in various data structures
- Replicated and distributed data structures (geometry and basis set objects)
- Checksum (MD5) for replicated data structures compared during execution
- For distributed data structures:

  Detection from numerical assertions:
  - Conditions or bounds on the valid values

  → Exact Condition for Detecting Orbital Orthonormality Violations, etc.

  Correction by replacing value.



Converge as in SDC free situation

Converge but need more iterations

Converge to different energy

Fail to converge (stop abruptly)

Coverage > 95%

Overhead:?

Low Cost Detection from Auxiliary Scheme

Scope: time dependent, initial value problems.

•Compare a primary time stepping scheme and an auxiliary checking scheme at every time step (same initial conditions)

•Open loop: the auxiliary scheme « restarts » from the primary intermediate solution at each time step.

•Detection from a sequence of the norm of the difference between prim. and aux. solutions

Example:

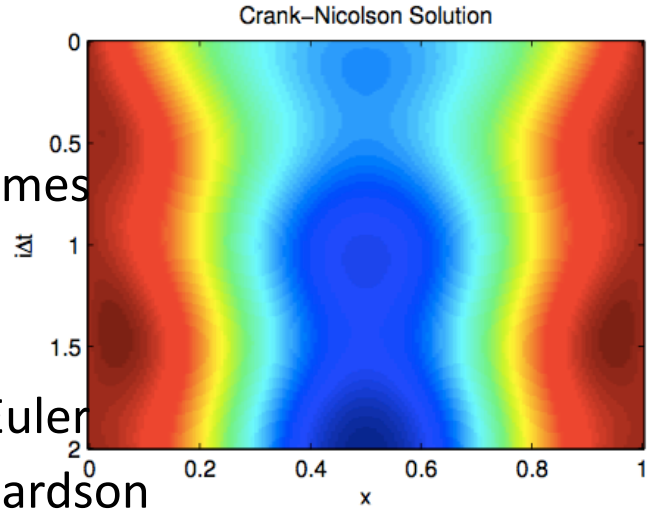
-finite difference schemes

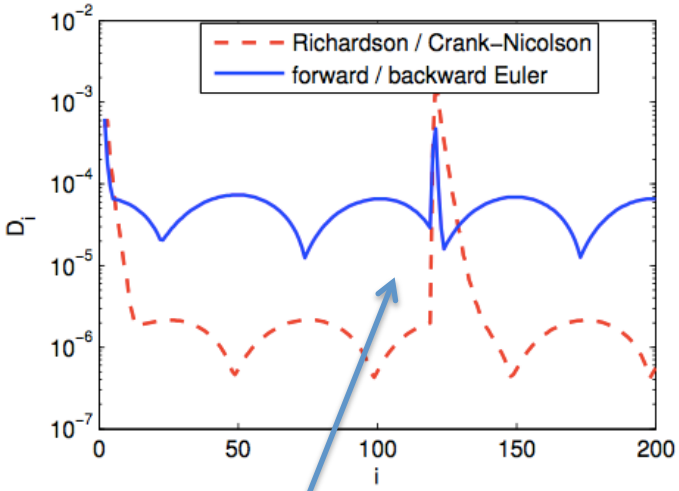
for PDEs

-Heat equation:

--Backward/forward Euler

--Crank-Nicolson/Richardson





Detection of the norm of the difference between the two solutions

- For now, no localization of the SDC

Benson, Austin R., Sven Schmit, and Robert Schreiber. "Silent error detection in numerical time-stepping schemes." International Journal of High Performance Computing Applications (2014).

Another Potential Low Cost Approach

Running the same code at a smaller scales, extrapolate and compare.

Case2: HACC Cosmology code.

Objective: Gravitational *N*-body simulations producing power spectra accurate to 1%

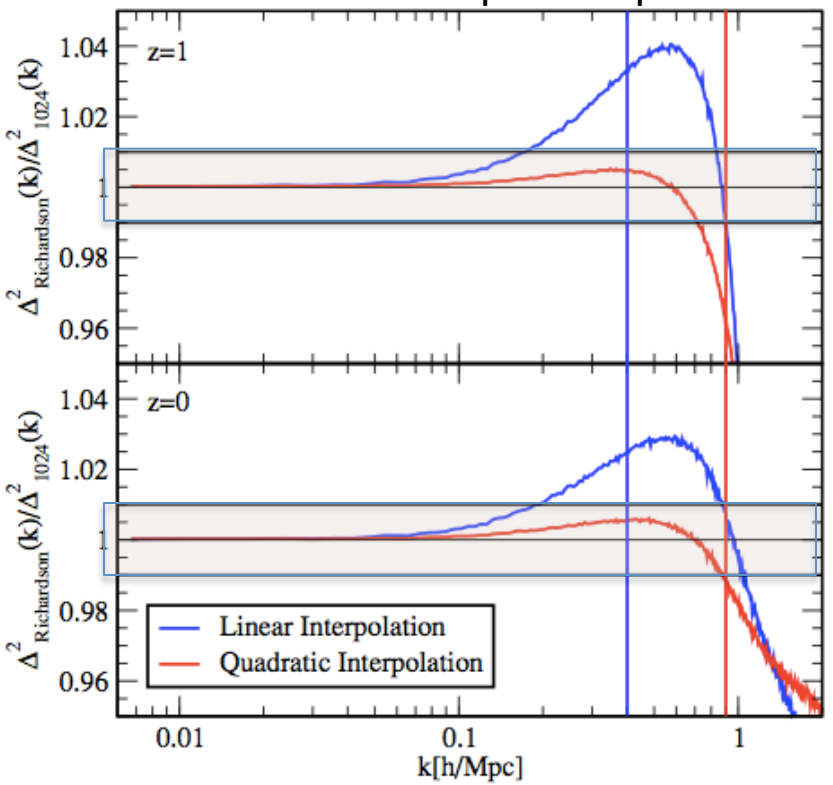
Run many tests to establish the accuracy: studies of the initial conditions (ICs), convergence to linear theory at very large length scales, the mass resolution requirement, etc.

One test is comparison of runs at 1024^3 with a Richardson extrapolation (linear or quadratic interpolation) from runs at 256^3 and 512^3.

For quadratic, the difference in the power spectrums is <1%

This suggests that extrapolations from smaller runs could be used as checker for larger runs.

nonlinear matter power spectrum

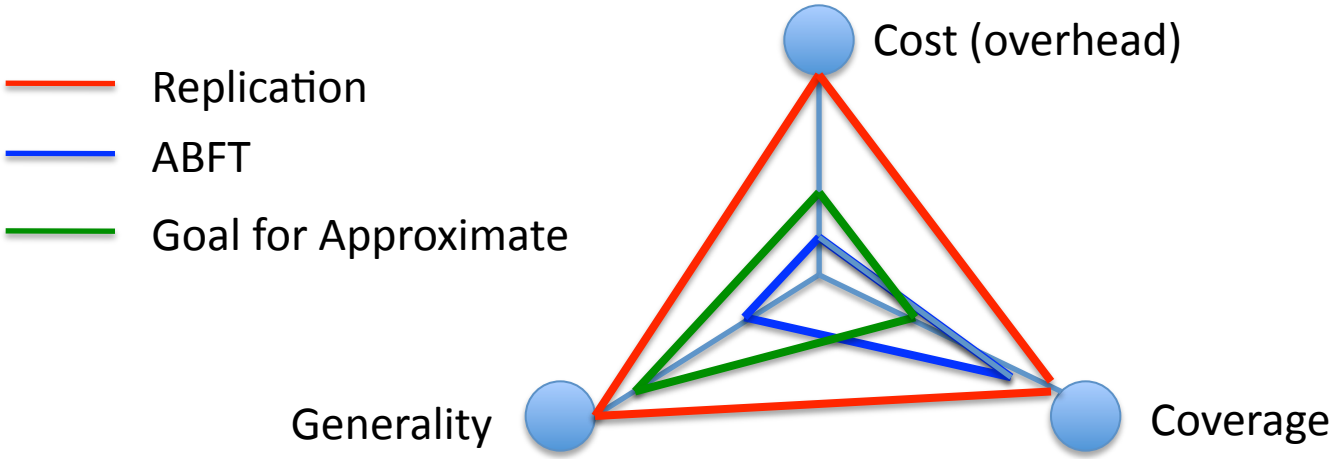


Katrin Heitmann et al., The Coyote Universe. I. Precision determination of the nonlinear matter power spectrum, The Astrophysical Journal, 715:104–121, 2010 May 20

Conclusion

The community will have to deal with SDCs (with a high probability)

The main problem is detection (numerous recovery approaches)



The precise understanding of the applicability of the approximate detection techniques is an open problem

Questions?