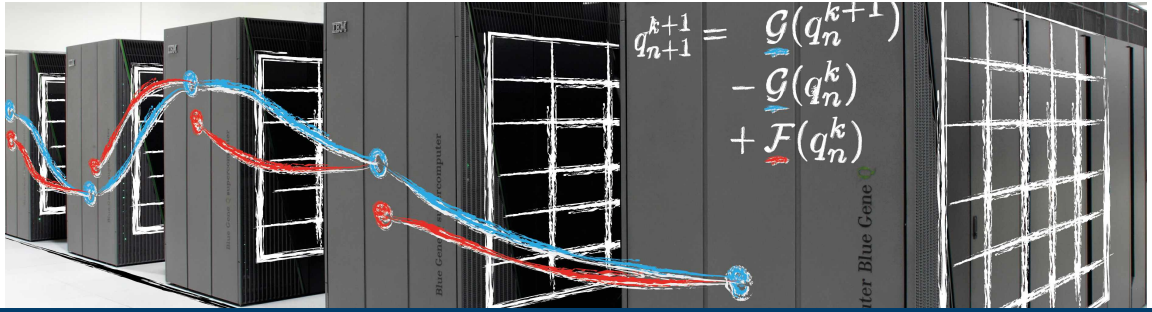
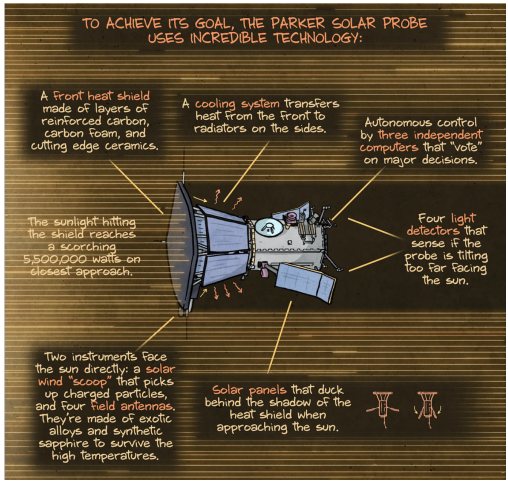


Resilience in (Time-Parallel) Spectral Deferred Corrections

April 26, 2022 | Thomas Baumann | Jülich Supercomputing Centre

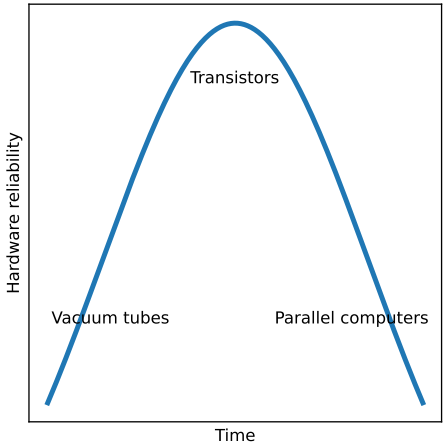
Faults: Same Computation, Same Result? Not in Space!



PHD Comics: To Touch the Sun [2]

- Space craft uses three computers that perform the same operations to make sure the computation is correct
- Frequent radiation induced bit flips this close to the sun
- Replication is simple and effective, but expensive resilience strategy

Faults Will Come After You on Earth as Well!



Sketch (!) of hardware reliability evolution

- Vacuum tube computers fail all the time
- More reliable transistor based computing from the 1960s
- Transistors shrink to gain efficiency at the cost of reliability
- Memory is protected by error correction codes, but processing units and their caches remain exposed
- Modern HPC is based on parallelism and the rate of failure scales with the core count

Fault Rates in the Wild

- Google in 2008: 25 to 70 faults per thousand device hours and Gbit [6]
- Lawrence Livermore National Laboratory in 2007: Uncorrectable faults in L1 cache occur on average every eight hours across the largest computer at the time [4]
- Disastrous implications for exascale, unless we figure out a way to recover from faults!
- Need more expensive ECCs or algorithm-based fault tolerance (ABFT)

PinT algorithms target large machines and need to be protected against faults
Iterative time marching schemes allow for cheap resilience against soft faults

Spectral Deferred Corrections (SDC) [3]: Serial for Now

- Write ODE in Picard form:

$$u(t) = u(t_0) + \int_{t_0}^{t} f(u(\tau))\, d\tau$$

- Discretize using quadrature: (vector components correspond to quadrature nodes)

$$(I - \Delta t Q F)(\vec{u}) = \vec{u}_0$$

- Use preconditioner:

$$(I - \Delta t Q_\Delta F)(\vec{u}) = \vec{u}_0 + \Delta t (Q - Q_\Delta) F(\vec{u})$$

- Iterate:

$$(I - \Delta t Q_\Delta F)(\vec{u}^{k+1}) = \vec{u}_0 + \Delta t (Q - Q_\Delta) F(\vec{u}^k)$$

Strategies for Fault Correction in SDC

Error Oblivious Algorithms

- It's iterative, it'll fix itself!
- Adaptivity

→ Fault correction without explicit detection

Error Aware Algorithms

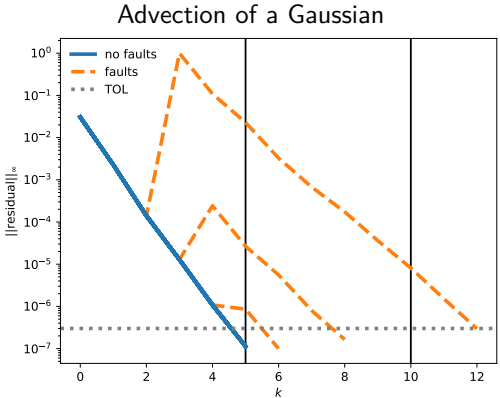
- Check contraction factor
- Hot Rod [5]

→ Algorithms detect faults

Four strategies for now:

1. Iterate to nirvana
2. Sweep it under the rug: Combine iterating with contraction factor estimates
3. Adaptivity
4. Hot Rod

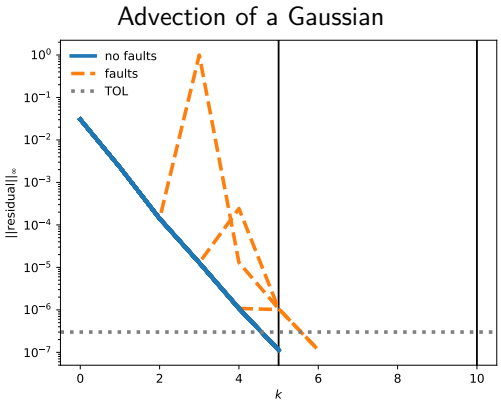
1 Iterate to Nirvana



Iterate until reaching residual threshold

- Sometimes less efficient than restart
- Faults to initial conditions can not be fixed without restart

2 “Sweep it Under the Rug”[1]

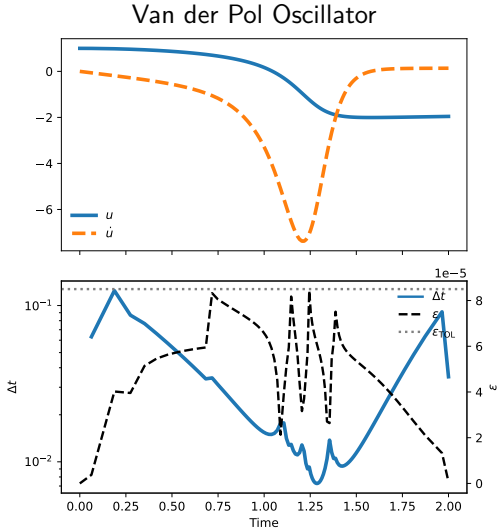


When is it more efficient to keep iterating than to restart?

- Estimate the contraction factor
- Predict sweeps required for convergence
- Restart or continue iterating
- Bonus: If the contraction factor exceeds one, restart only the last sweep
- Here: Fix all faults with only one extra iteration

[1] This is funny because people call SDC iterations “sweeps”

3 Adaptivity



Dynamically select the step size

- Idea: $\frac{e^{(n+1)}}{e^{(n)}} = \left(\frac{h^{(n+1)}}{h^{(n)}}\right)^{k+1}$
- Estimate local error ϵ
- Plug in $e^{(n+1)} = \epsilon_{\mathrm{TOL}}$ and safety factor β
- Next step size: $h^{(n+1)} = \beta h^{(n)} \left(\frac{\epsilon_{\mathrm{TOL}}}{\epsilon}\right)^{\frac{1}{k}}$
- Recompute if $\epsilon > \epsilon_{\mathrm{TOL}}$

Estimating the Local Error: Embedded Method

- Compute two solutions $u^{(k)}$ and $u^{(k-1)}$ of orders k and $k-1$
- Act as if $u^{(k)}$ was the exact solution:

$$\epsilon = \|u^{(k)} - u^{(k-1)}\| = \| \left(u^{(k)} - u^*\right) - \left(u^{(k-1)} - u^*\right) \| = \|e^{(k)} - e^{(k-1)}\| = e^{(k-1)} + \mathcal{O}\left(\Delta t^{k+1}\right)$$

- With SDC: Simply subtract two consecutive sweeps (with the right preconditioner)
- Estimate the error of the second to last sweep with virtually no overhead
- Estimate order $k-1$ error, but advance with order k solution

Estimating the Local Error: Extrapolation based [7, 1]

- Do Taylor expansions and find coefficients a_j and b_j finite difference style such that:

$$u_{\text{extrapolation}}(t) = \sum_{j=1}^{n} a_j u(t - j\Delta t) - b_j j\Delta t\, f(u(t - j\Delta t)) + \mathcal{O}\left(\Delta t^{2n+1}\right)$$

- Estimate error:

$$\begin{aligned}
\epsilon_{\text{extrapolation}} &= 1/\mathcal{P} \| u - u_{\text{extrapolation}} \| \\
&= 1/\mathcal{P} \| u^* + \mathcal{O}\left(\Delta t^{k+1}\right) - \left(u^* + \mathcal{O}\left(\Delta t^{2n+1}\right) + (\mathcal{P}-1)\,\mathcal{O}\left(\Delta t^{k+1}\right)\right) \| \\
&= \mathcal{O}\left(\Delta t^{k+1}\right) + \mathcal{O}\left(\Delta t^{2n+1}\right)
\end{aligned}$$

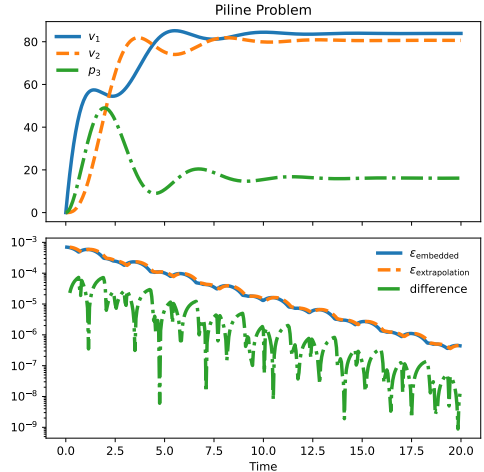
- Need to store both u and f from $\frac{k+2}{2}$ previous steps $\rightarrow$ large memory overhead

Estimating The Local Error

	Embedded method	Extrapolation method
Order estimate	k	$k+1$
Computational overhead	insignificant	insignificant
Memory overhead	insignificant	large
Use with adaptivity	simple	tricky

$\rightarrow$ Prefer embedded method, but what if we use both?

4 Hot Rod [5]

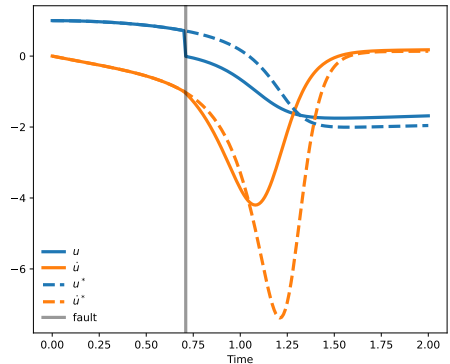


- Estimate local error of sweep $k-1$ with both methods: $\epsilon_{\text{embedded}}$ and $\epsilon_{\text{extrapolation}}$
- Compute difference:

$$\begin{aligned}\Delta =& \| \epsilon_{\text{embedded}} + \mathcal{O}\left(\Delta t^{k+1}\right) - \\ & \left(\epsilon_{\text{extrapolation}} + \mathcal{O}\left(\Delta t^{k+1}\right)\right) \| \\ =& \mathcal{O}\left(\Delta t^{k+1}\right)\end{aligned}$$

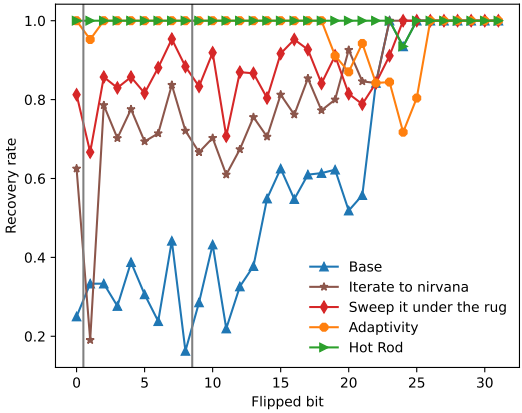
- Restart if $\Delta > \text{TOL}$
- Need to advance with second to last sweep for extrapolation estimate $\implies$ significant computational and memory overhead

Experiment: Faults in Van der Pol Oscillator



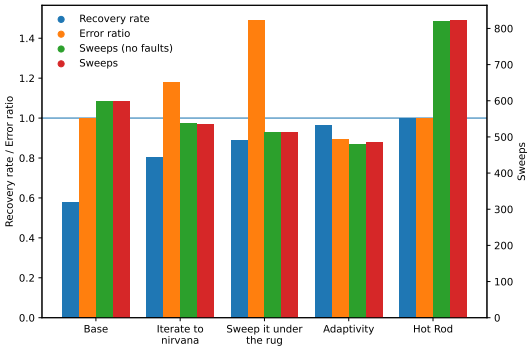
- Manually flip random single bit in a random iteration, collocation node and "problem position" of the solution
- Accept fault as recovered if $e < 2e_{\text{fault-free}}$
- Compare to base scheme with fixed Δt and k

Recovery Rates



- Low impact of faults in insignificant mantissa bits
- Schemes without restarts cannot fix faults to initial conditions
- Smaller faults might fly under the radar of adaptivity but appear in the final solution

Overhead



- Difficult to match final error due to timescale changes
- Hot Rod is very resilient, but adds significant cost
- Adaptive schemes are resilient, efficient and easy to implement (in serial SDC)
- Negligible overhead from fault correction

How to PinT with SDC

Move from collocation problem for single step

$$(I - \Delta t Q F)(\vec{u}) = \vec{u}_0$$

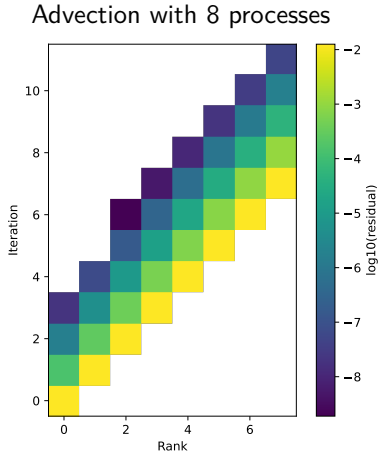
to composite collocation problem containing L steps

$$\begin{pmatrix} I - \Delta t Q F & & & \\ -N & I - \Delta t Q F & & \\ & \ddots & \ddots & \\ & & -N & I - \Delta t Q F \end{pmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vdots \\ \vec{u}_L \end{pmatrix} = \begin{pmatrix} \vec{u}_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

No need to "fully" solve the collocation problems before sending solutions forward!

Solving the composite collocation problem iteratively allows for parallelization in time

Block Gauß-Seidel SDC



- Eight steps need 24 iterations to converge in serial SDC
- A block of eight steps converges after 11 “composite iterations”
- Simple communication structure
- Expanding on this idea (a lot) $\rightsquigarrow$ PFASST

Next Steps

- Implement the resilience strategies in pySDC
- Try the same strategies with PinT (Block-Gauß-Seidel SDC, eventually PFASST)
- Think of new resilience strategies
- Try out more realistic fault injection
- Make an attempt at dealing with dying processes

Summary

- PinT is targeting huge machines, which are susceptible to faults $\rightarrow$ need ABFT
- Iterative PinT schemes give ample opportunity for ABFT
- Adaptivity+SDC for the win: Very efficient + good resilience
- Hot Rod is too expensive: Need to sacrifice one iteration

Thank You for Your Attention

Sources I

J. Butcher and P. Johnston.

Estimating local truncation errors for runge-kutta methods.
Journal of Computational and Applied Mathematics, 45(1):203–212, 1993.

J. Cham.

To touch the sun.
https://physics.aps.org/articles/v14/178.
Accessed: 2-March-2022.

A. Dutt, L. Greengard, and V. Rokhlin.

Spectral deferred correction methods for ordinary differential equations.
BIT Numerical Mathematics, 40(2):241–266, 2000.

J. N. Glosli, D. F. Richards, K. J. Caspersen, R. E. Rudd, J. A. Gunnels, and F. H. Streitz.

Extending stability beyond cpu millennium: a micron-scale atomistic simulation of kelvin-helmholtz instability.
In *SC'07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–11. IEEE, 2007.

P.-L. Guhur, H. Zhang, T. Peterka, E. Constantinescu, and F. Cappello.

Lightweight and accurate silent data corruption detection in ordinary differential equation solvers.
In P.-F. Dutot and D. Trystram, editors, *Euro-Par 2016: Parallel Processing*, pages 644–656, Cham, 2016. Springer International Publishing.

B. Schroeder, E. Pinheiro, and W.-D. Weber.

Dram errors in the wild: a large-scale field study.
ACM SIGMETRICS Performance Evaluation Review, 37(1):193–204, 2009.

Sources II

L. Stoller and D. Morrison.

A method for the numerical integration of ordinary differential equations.
Mathematical Tables and Other Aids to Computation, 12(64):269–272, 1958.