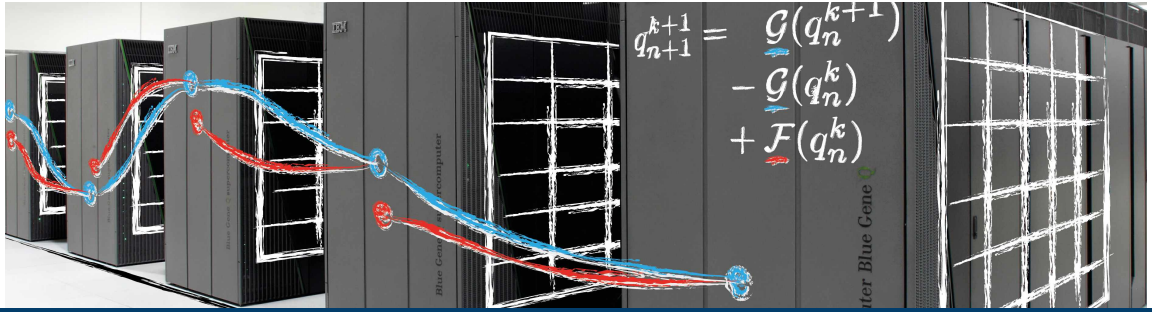
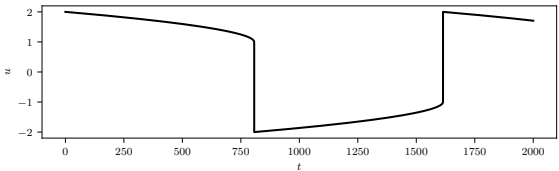


Efficient Massively Space-Time-Parallel Simulations with Adaptive Spectral Deferred Correction

3.12.25 | Thomas Saupe | Jülich Supercomputing Centre

Efficient Massively Space-Time-Parallel Simulations with Adaptive Spectral Deferred Correction



Motivation

- Use **adaptivity** to **efficiently simulate** problems with non-constant global timescale
- Use **spectral deferred correction (SDC)** because of its capabilities for stiff PDEs and parallelisation opportunities
- Use **space-time-parallel** implementations to cater to modern HPC machines

Spectral deferred correction (SDC)

[Dutt et al., 2000]

- SDC numerically approximates solutions to initial value problems (IVP):

$$\partial_t u = f(u,t), \quad u(t=0) = u_0, \quad t \in [0,T]$$

- Integral formulation:

$$u(T) = u(0) + \int_0^T f(u,t)dt$$

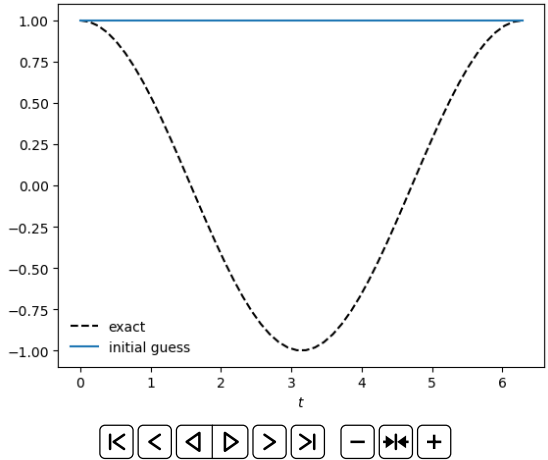
- Discretize the interval with M collocation nodes $\tau_m \in [0,T]$ and define $u_m \approx u(\tau_m)$
- Discretize the integral with quadrature rule Q to arrive at the collocation problem:

$$u_m = u_0 + \sum_{j=1}^{M} Q_{mj} f(u_j,\tau_j), \quad m = 1,\ldots,M$$

SDC iteratively solves collocation problem

- Q is dense, coupling all u_m to each other
- Use lower triangular preconditioner Q_Δ to remove forward coupling between u_m
- "Sweep" through collocation nodes to update u_m^k one by one (k: iteration index)

Example: $\partial_t u = iu,\ u(0) = 1,\ Re(u(t)) = \cos(t)$



Adaptive Algorithm 0: k-adaptivity

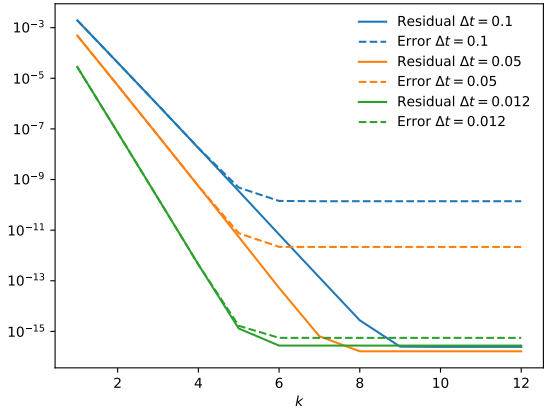
Residual

Subtract both sides of collocation problem

$$r_m^k = u_0 + \sum_{j=1}^{M} Q_{mj} f(u_j^k, \tau_j) - u_m^k$$

Issues with choosing only k adaptively

- Residual indicates how close u_m^k is to the solution of the collocation problem
- Accuracy of the collocation problem itself depends on Δt!



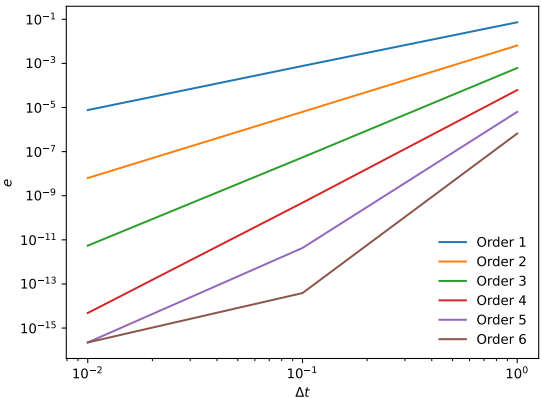
Adaptive selection of Δt in embedded Runge-Kutta

[Hairer et al., 1993, Chapter II.4]

- Order of accuracy p: $e \propto \Delta t^{p+1}$
- Measured error ϵ of order p with Δt, but wanted ϵ_{TOL}? Use

$$\Delta t_{\mathrm{opt}} = \beta \Delta t \left(\frac{\epsilon_{\mathrm{TOL}}}{\epsilon} \right)^{1/(p+1)}$$

- If $\epsilon > \epsilon_{\mathrm{TOL}}$, recompute current step with Δt_{opt}, otherwise move on to next step
- Use $\beta = 0.9$ to prevent restarts



JÜLICH Forschungszentrum

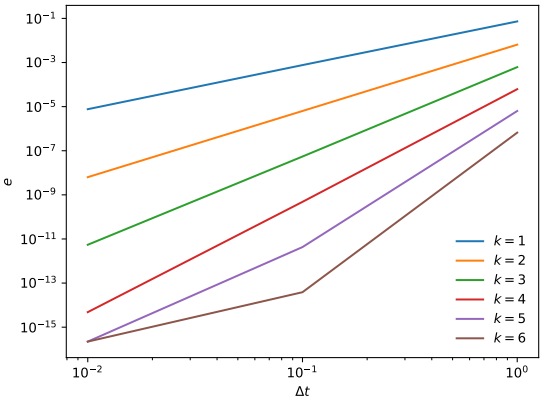
Adaptive Algorithm I: Δt-adaptivity

[Saupe et al., 2024a]

- Order $p = \min(k, 2M - x)$, x depends on Q
- Increment is error estimate of order $k - 1$:

$$\begin{aligned}\epsilon &= \|u_M^k - u_M^{k-1}\|_\infty \\ &= \|u_M^k - u^*(\tau_M) + u^*(\tau_M) - u_M^{k-1}\|_\infty \\ &= \|e_M^k - e_M^{k-1}\|_\infty \\ &\approx \|e_M^{k-1}\|_\infty \propto \Delta t^k\end{aligned}$$

  (embedded Runge-Kutta works very similarly)
- Use fixed number of iterations k
- Plug ϵ and $p = k - 1$ into step size equation from embedded Runge-Kutta

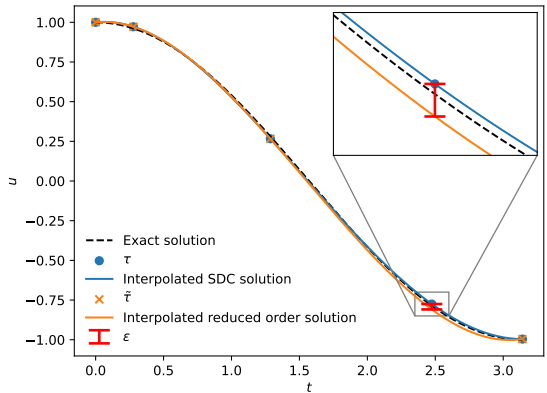


JÜLICH Forschungszentrum

Adaptive Algorithm II: Δt-k-adaptivity

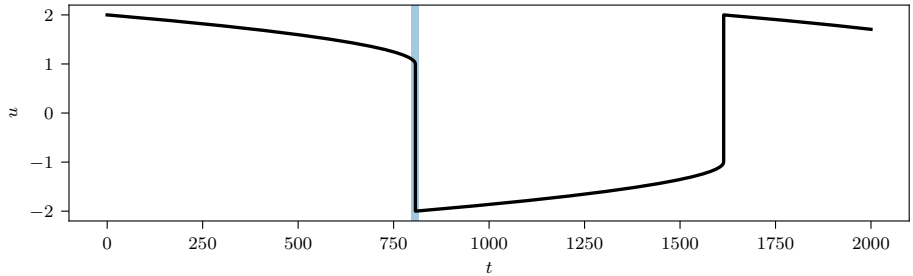
[Saupe et al., 2024a]

- Iterate until SDC iteration is converged
- Choose all nodes but one:
  $\tilde{\tau} = \{\tau_m = 1, \dots M-2, M\}$
- Interpolate u_m^k at $\tilde{\tau}$ to missing node τ_{M-1} to obtain $\tilde{u}_{M-1}^k$ of order $M-1$
- Order $M-1$ error estimate:
  $\epsilon = \|u_{M-1}^k - \tilde{u}_{M-1}^k\| \propto \Delta t^M$
- Plug ϵ and $p = M-1$ into step size equation from embedded Runge-Kutta



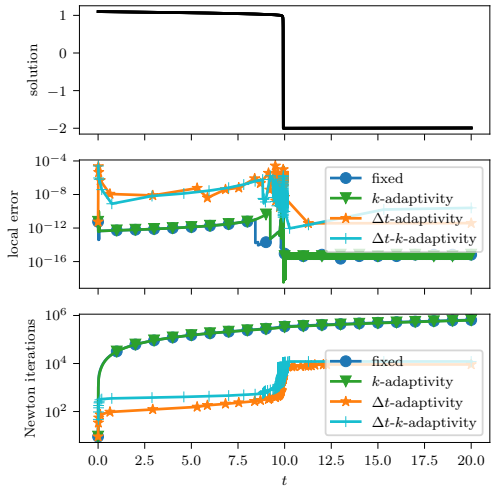
JÜLICH Forschungszentrum

Toy problem: Van der Pol oscillator



- Equation: $u_{tt} - \mu \left(1 - u^2\right) u_t + u = 0$
- Extreme separation of time-scales for $\mu >> 0$
- We focus on the shaded area here

Boosting computational efficiency with adaptivity



SDC strategies

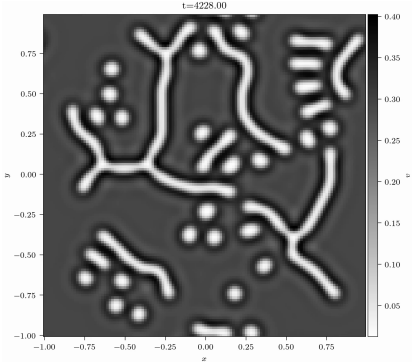
- fixed: fixed iteration number k and step size Δt
- k-adaptivity: adaptive k, fixed Δt
- Δt-adaptivity: adaptive Δt, fixed k
- Δt-k-adaptivity both adaptive

Results

- k-adaptivity: Can't tune k finely enough
- Need adaptive Δt to accommodate changes in timescale

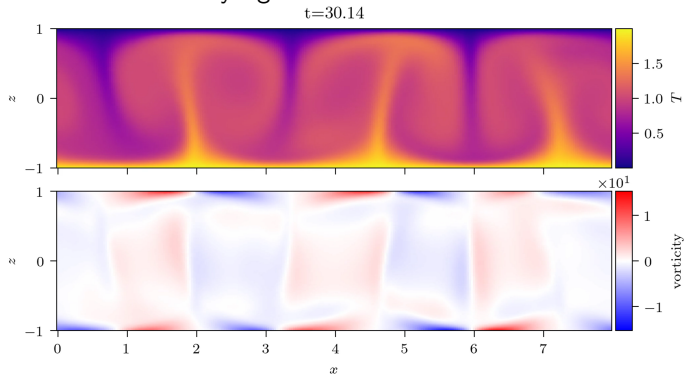
PDE benchmark problems

Gray-Scott



Here: $t \in [0,500]$

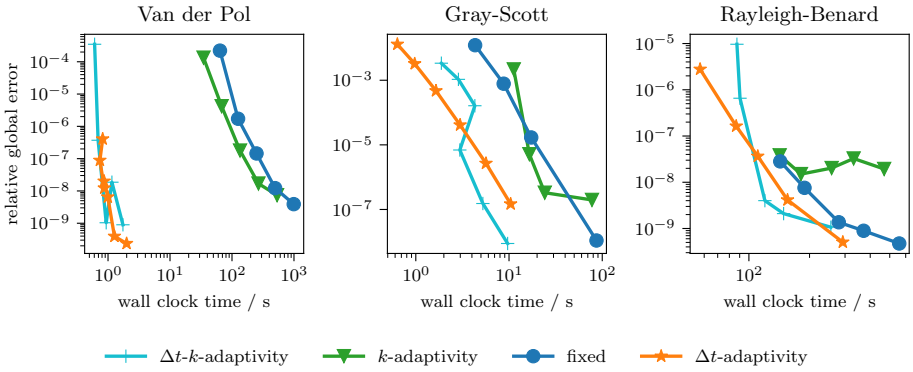
Rayleigh-Benard convection



Here: $t \in [10,16]$

JÜLICH Forschungszentrum

Boosting computational efficiency with adaptivity



- k-adaptivity gives only marginal benefit
- Δt-adaptivity works well for any accuracy and problem
- Δt-k-adaptivity works best for tight tolerance

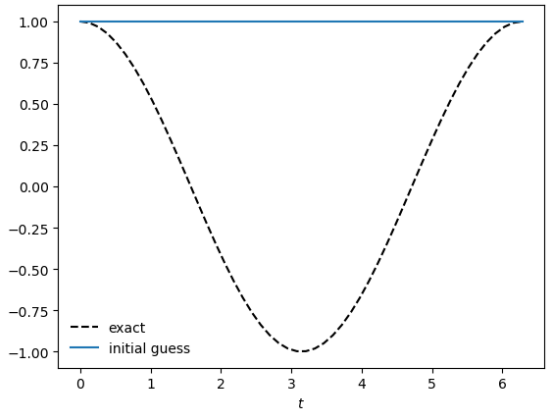
Further speedup by parallelization: Diagonal SDC

[van der Houwen and Sommeijer, 1991, Speck, 2018]

Choose Q_Δ diagonal

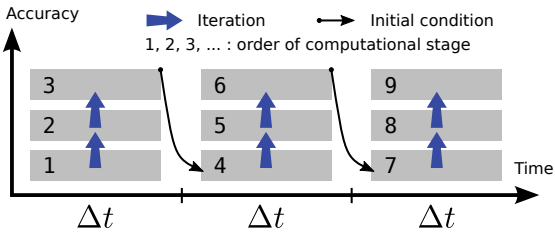
- Diagonal Q_Δ removed forward and backward coupling between u_m
- Update solution at all nodes simultaneously
- "MIN-SR-S" and "MIN-SR-NS" from [Čaklović et al., 2024] yield fast convergence

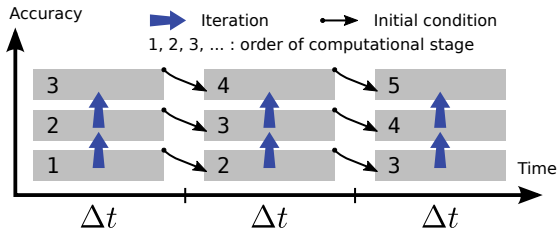
Example: $\partial_t u = iu,\ u(0) = 1,\ Re(u(t)) = \cos(t)$



Further speedup by parallelization: Pipelining

SDC Parallel-in-time extension: Block Gauß-Seidel SDC [Guibert and Tromeur-Dervout, 2007]



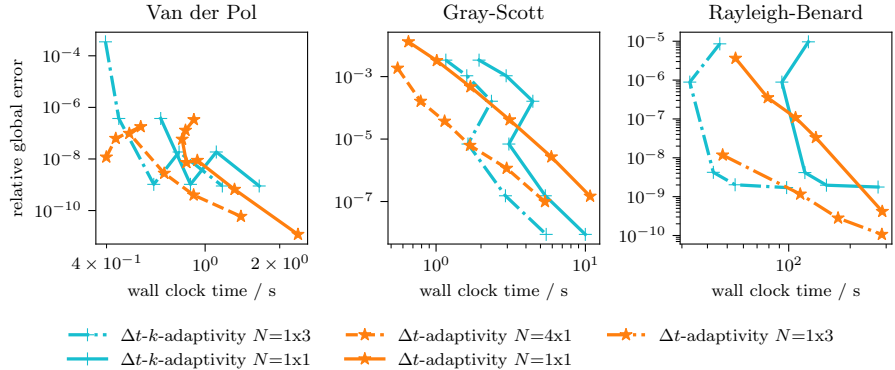


Start iteration on step as soon as one iteration has been performed on previous step

Figure provided by Thibaut Lunet

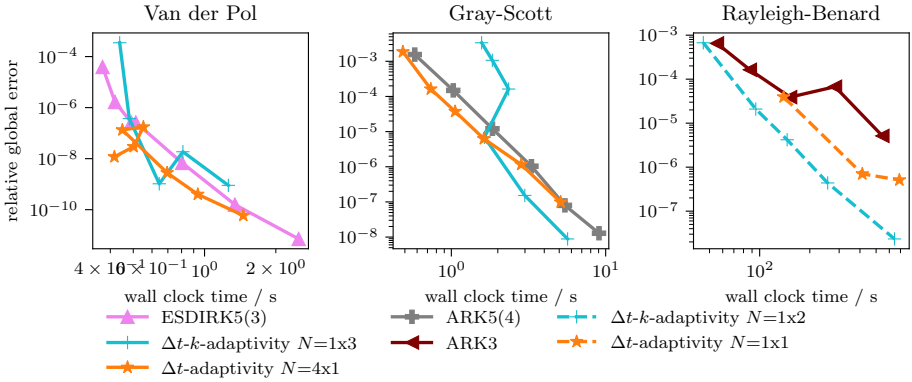
JÜLICH Forschungszentrum

Parallel-in-time speedup



- Number of tasks: N = [tasks in block-Gauß-Seidel]x[tasks in diagonal SDC]
- Diagonal SDC gives decent speedup most of the time
- Speedup by pipelining only for some problems
- Combination of both rarely gives speedup

Time-parallel SDC vs. reference Runge-Kutta method

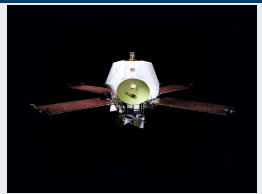


- At least mode of adaptive PinT SDC is always competitive with RKM for stiff problems
- In Rayleigh-Benard, no high order comparison RKM available, SDC still better at low order

Resilience

Tyranny of numbers

- Insane number of components
- Large chance of failure somewhere
- E.g.: Mariner 8 crashed because of a single faulty diode [JOHNSON, 2021]

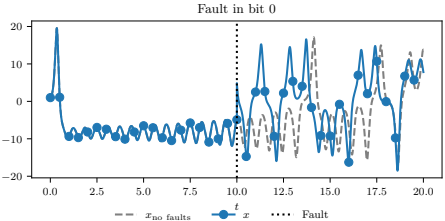


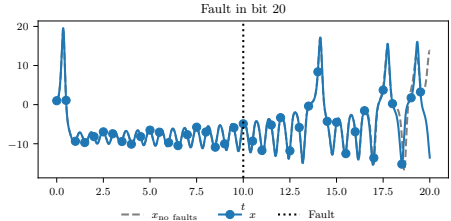
Faults in modern HPC

- Sources: Radiation [Lapinsky and Easty, 2006] and hardware damage [Schroeder et al., 2011]
- Multiple times each day in HPC machines [Glosli et al., 2007, Schroeder et al., 2011]

Boosting resilience by adaptivity

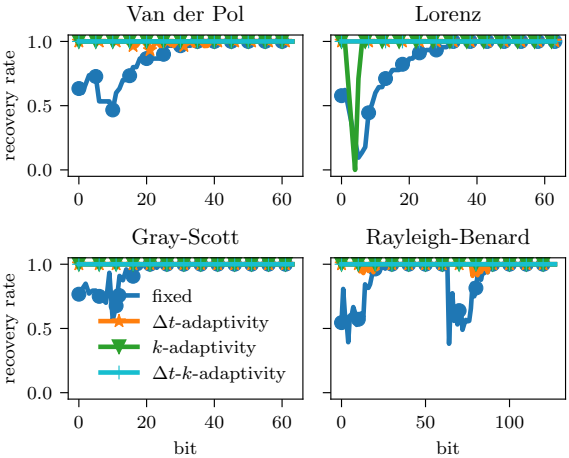
Lorenz attractor





- Flipping individual bits can silently corrupt simulation result
- Adaptive Runge-Kutta methods are naturally resilient [Benson et al., 2015]
- The same is true for adaptive SDC [Saupe et al., 2024b]

Boosting resilience by adaptivity



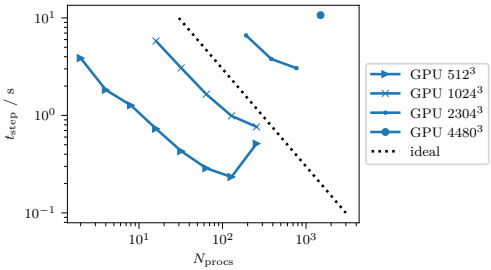
- Bits 0: sign, 1-12: exponent, 13-64: mantissa
- Note: We exclude faults to initial conditions or ones that cause crashes here
- k-adaptivity: Iteration may not converge from faulty initial guess
- Adaptivity greatly improves resilience

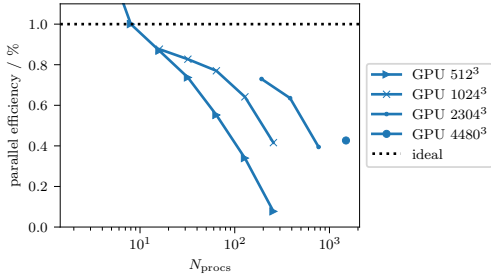
High performance computing (HPC)

Figure: The JUWELS booster module (left) and cluster module (right) at Jülich Supercomputing Centre. Image taken from [Alvarez, 2021, Figure 1] under CC BY 4.0 licence. Copyright: Forschungszentrum Jülich GmbH.

- Modern supercomputers are networks of compute nodes
- GPUs supply most of the compute power in current machines

Parallel scaling of 3D Gray-Scott implementation on GPUs

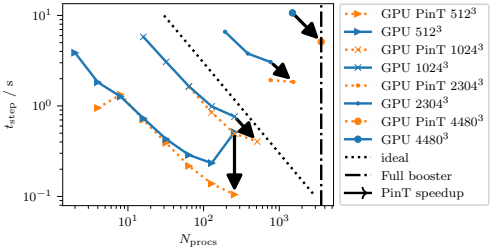


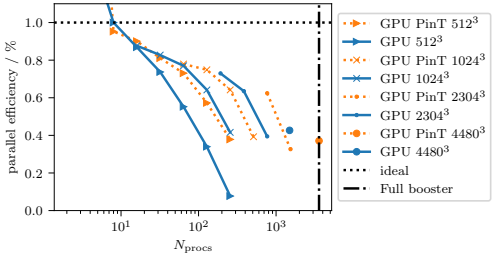


Use diagonal SDC to extend scaling

- Diagonal SDC is not too demanding of the network
- Improves strong scaling
- Enables scaling up to 3584 GPUs ($\approx 95\,\%$ of JUWELS booster)

Parallel scaling of 3D Gray-Scott implementation on GPUs

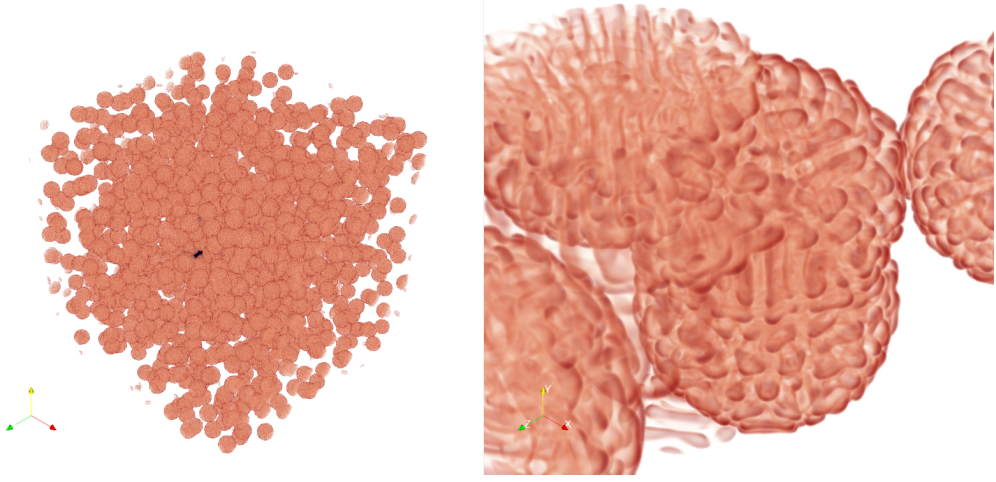




Use diagonal SDC to extend scaling

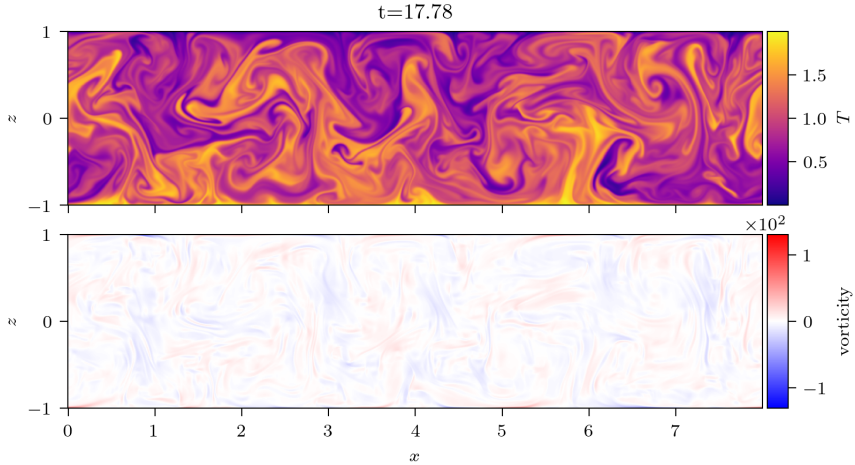
- Diagonal SDC is not too demanding of the network
- Improves strong scaling
- Enables scaling up to 3584 GPUs ($\approx 95\,\%$ of JUWELS booster)

Large space-time parallel Gray-Scott simulation



$N = 2304^3$ on $4 \times 192 = 768$ GPUs on JUWELS booster

Large space-time parallel Rayleigh-Benard simulation



$N = 4096 \times 1024$ on $4 \times 1024 = 4096$ CPUs on JURECA DC

JÜLICH Forschungszentrum

Conclusion

Summary

- Developed two algorithms for Δt-adaptivity in SDC
  - Demonstrated improved computational efficiency
  - Demonstrated speedup via time-parallelism
  - Showed that time-parallel adaptive SDC can outperform reference Runge-Kutta methods for PDEs
  - Demonstrated increased resilience against soft faults
- Developed space-time parallel implementation of Gray-Scott that scaled up to all of Juwels booster

Collaborators

- Sebastian Götschel
- Thibaut Lunet
- Daniel Ruprecht
- Robert Speck

Codes used / developed

pySDC

GPU port of mpi4py-fft

Sources I

Alvarez, D. (2021).

JUWELS Cluster and Booster: Exascale Pathfinder with Modular Supercomputing Architecture at Juelich Supercomputing Centre.
Journal of large-scale research facilities JLSRF, 7:A183.

Benson, A. R., Schmit, S., and Schreiber, R. (2015).

Silent error detection in numerical time-stepping schemes.
The International Journal of High Performance Computing Applications, 29(4):403–421.

Čaklović, G., Lunet, T., Götschel, S., and Ruprecht, D. (2024).

Improving efficiency of parallel across the method spectral deferred corrections.

Dutt, A., Greengard, L., and Rokhlin, V. (2000).

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

Glosli, J. N., Richards, D. F., Caspersen, K. J., Rudd, R. E., Gunnels, J. A., and Streitz, F. H. (2007).

Extending stability beyond CPU millennium: A micron-scale atomistic simulation of Kelvin-Helmholtz instability.
In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, SC '07, New York, NY, USA. Association for Computing Machinery.

Guibert, D. and Tromeur-Dervout, D. (2007).

Parallel deferred correction method for CFD problems.
In Kwon, J., Ecer, A., Satofuka, N., Periaux, J., and Fox, P., editors, *Parallel Computational Fluid Dynamics 2006*, pages 131–138. Elsevier Science B.V., Amsterdam.

Sources II

Hairer, E., Wanner, G., and Nørsett, S. P. (1993).
Solving Ordinary Differential Equations I, volume 8 of *Springer Series in Computational Mathematics*.
Springer Berlin Heidelberg, Berlin, Heidelberg.

JOHNSON, S. S. (2021).
SIRENS OF MARS: searching for life on another world.
PENGUIN BOOKS, S.l.
OCLC: 1198976118.

Lapinsky, S. E. and Easty, A. C. (2006).
Electromagnetic interference in critical care.
Journal of Critical Care, 21(3):267–270.

Lorenz, E. N. (1963).
Deterministic Nonperiodic Flow.
Journal of the Atmospheric Sciences, 20(2):130–141.

Saupe, T., Götschel, S., Lunet, T., Ruprecht, D., and Speck, R. (2024a).
Adaptive time step selection for spectral deferred correction.
Numerical Algorithms.

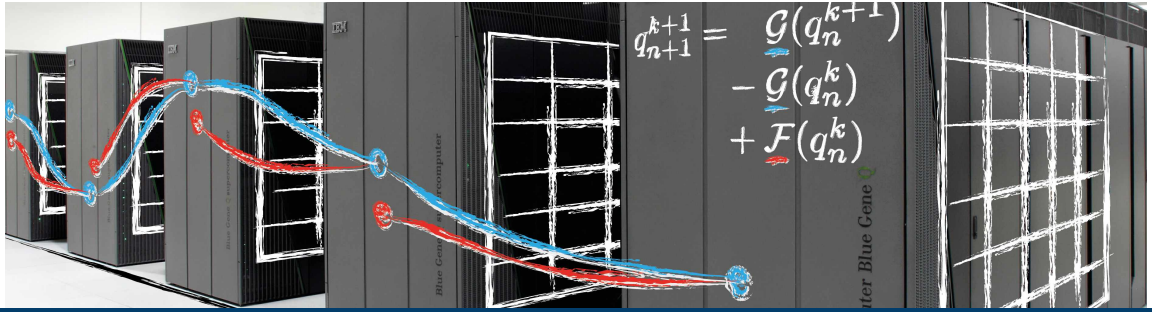
Saupe, T., Götschel, S., Lunet, T., Ruprecht, D., and Speck, R. (2024b).
Resilience against soft faults through adaptivity in spectral deferred correction.

Sources III

Schroeder, B., Pinheiro, E., and Weber, W.-D. (2011).
DRAM errors in the wild: A large-scale field study.
Commun. ACM, 54(2):100–107.

Speck, R. (2018).
Parallelizing spectral deferred corrections across the method.
Computing and Visualization in Science, 19(3):75–83.

van der Houwen, P. J. and Sommeijer, B. P. (1991).
Iterated runge–kutta methods on parallel computers.
SIAM Journal on Scientific and Statistical Computing, 12(5):1000–1028.



Efficient Massively Space-Time-Parallel Simulations with Adaptive Spectral Deferred Correction

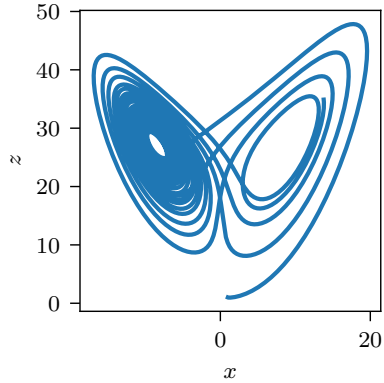
3.12.25 | Thomas Saupe | Jülich Supercomputing Centre

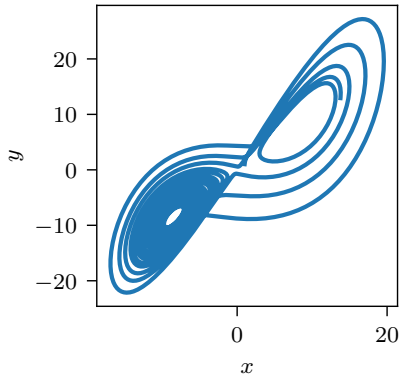
Benchmark problem II: Lorenz attractor [Lorenz, 1963]

- Equation:

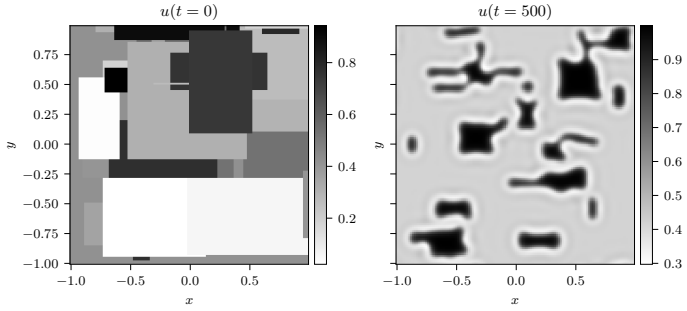
$$x_t = \sigma(y - x)$$
$$y_t = \rho x - y - xz$$
$$z_t = xy - \beta z$$

- Chaotic behavior (Butterfly effect)





Benchmark problem III: Gray-Scott



- Equation:

$$u_t = \nu_u \Delta u - uv^2 + F(1-u)$$
$$v_t = \nu_v \Delta v + uv^2 - (F+k)v$$

- Turing instability: Perturbations + diffusion = complex shapes

Benchmark problem III

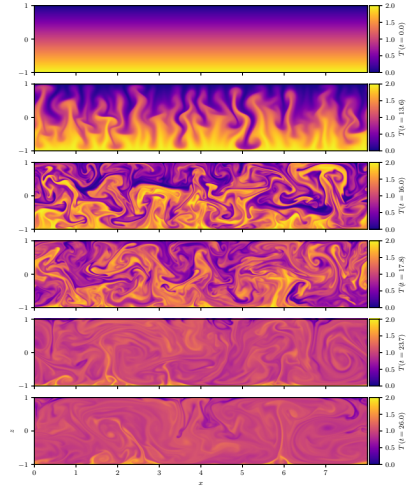
Rayleigh-Benard convection

Incompressible fluid dynamics:

$$
\begin{aligned}
u_t - \nu(u_{xx} + u_{zz}) + p_x &= -uu_x - vu_z \\
v_t - \nu(v_{xx} + v_{zz}) + p_z - T &= -uv_x - vv_z \\
T_t - \kappa(T_{xx} + T_{zz}) &= -uT_x - vT_z \\
u_x + v_z &= 0
\end{aligned}
$$

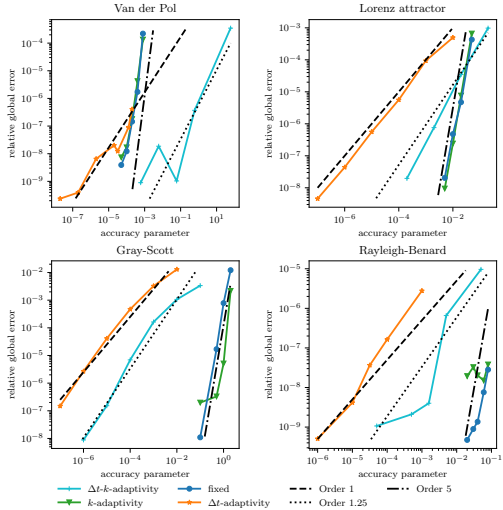
Space discretization

- Solve IMEX with Fourier + ultraspherical pseudo-spectral discretization
- Differential algebraic equation $\rightarrow$ need stiffly accurate integrator



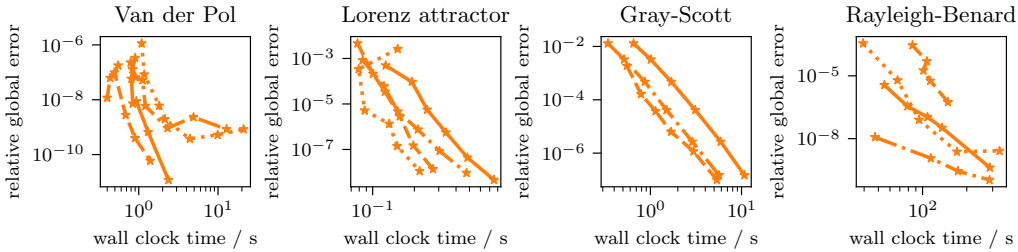
JÜLICH Forschungszentrum

“Order” of adaptive schemes



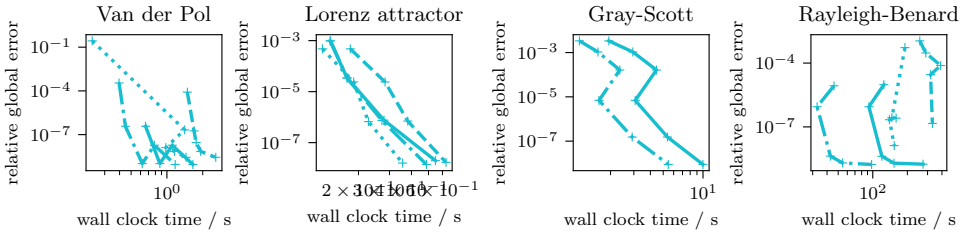
JÜLICH Forschungszentrum

Parallel-in-time Δt-adaptivity



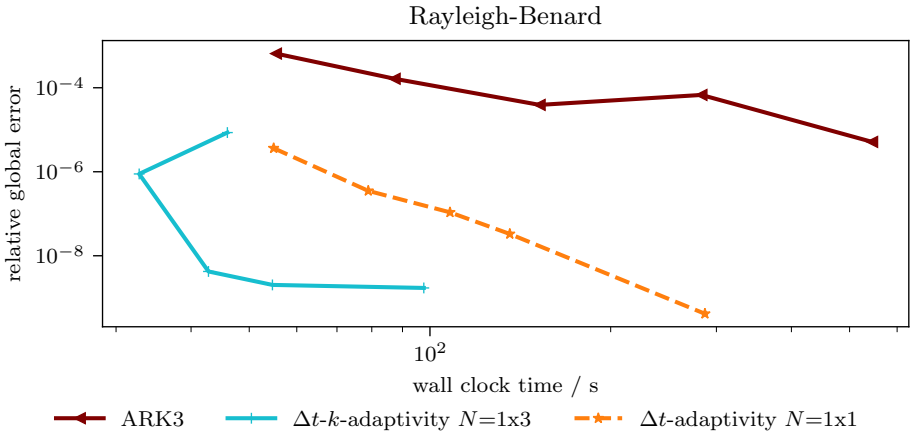
- Performance heavily depends on Q_Δ
- Don't know beforehand if speedup is possible
- At least one parallelization method always gives good speedup
- Combining diagonal SDC and block Gauß-Seidel SDC is seldom worthwhile

Parallel-in-time Δt-k-adaptivity



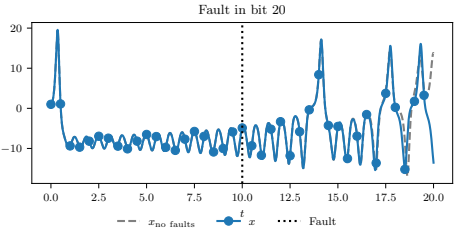
- Similar poperties w.r.t. parallelisation as Δt-adaptivity
- Here, we often use diagonal Q_Δ in serial as well
  $\rightarrow$ Get same result in diagonal SDC
- Particularly good speedup with diagonal SDC for PDEs

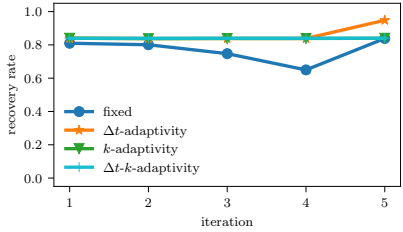
Order 5 SDC vs. order 3 RK for RBC

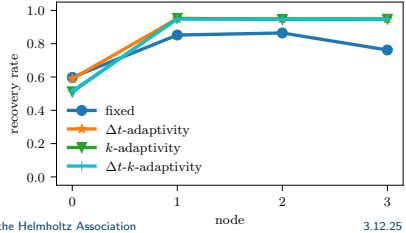


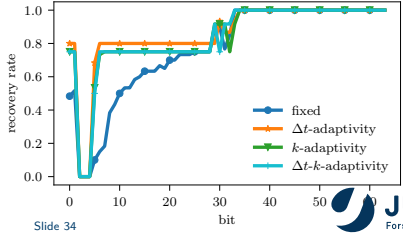
Boosting resilience by adaptivity

Lorenz attractor









JÜLICH Forschungszentrum

Porting mpi4py-fft to GPU: Alltoall in NCCL

N,r ← Number of GPUs, MPI rank ▷ Initialize variables
sendbufs, sendbufs ← {}, {}

NCCL group start ▷ Launch all communications in a single kernel
for i in 0, 1, ..., N: **do**
    send_to ← (r + i) % N
    recv_from ← (r - i + N) % N
    sendbufs[i] ← contiguous copy of data to send
    recvbufs[i] ← contiguous empty array
    NCCL receive from recv_from
    NCCL send to send_to
end for
NCCL group end ▷ Wait for all communication to finish before unpacking

copy recvbufs to destination array

Record all of this to CUDA graph!

Porting mpi4py-fft to GPU: Alltoall in NCCL

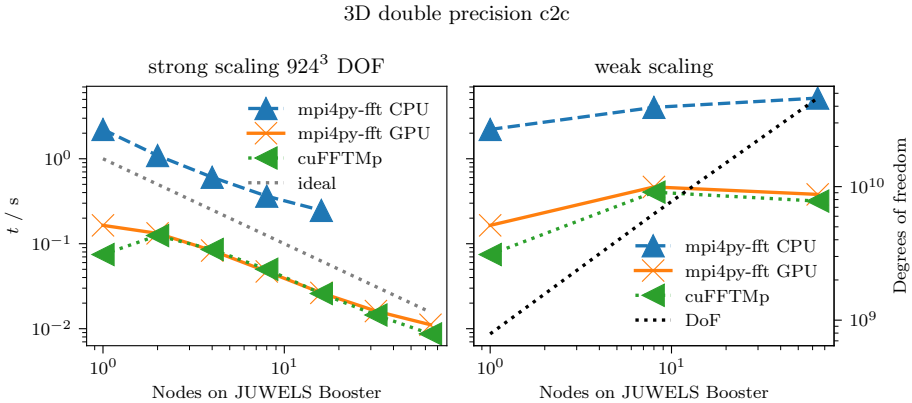
N,r ← Number of GPUs, MPI rank ▷ Initialize variables
sendbufs, sendbufs ← {}, {}

NCCL group start ▷ Launch all communications in a single kernel
for i in 0, 1, ..., N: **do**
    send_to ← (r + i) % N
    recv_from ← (r - i + N) % N
    sendbufs[i] ← contiguous copy of data to send
    recvbufs[i] ← contiguous empty array
    NCCL receive from recv_from
    NCCL send to send_to
end for
NCCL group end ▷ Wait for all communication to finish before unpacking

copy recvbufs to destination array

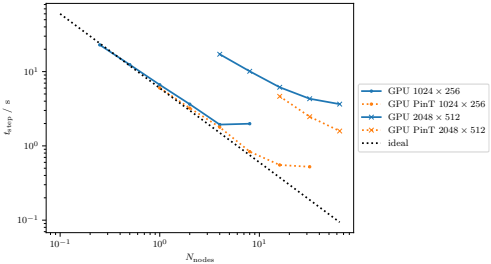
Record all of this to CUDA graph!

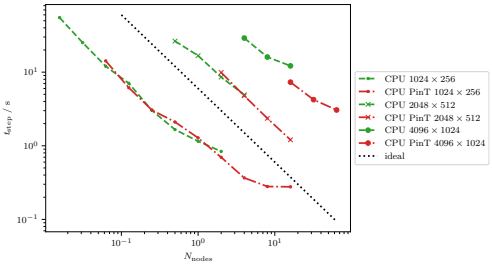
Parallel scaling of mpi4py-fft GPU port



Remarkably competitive with optimized NVIDIA competition cuFFTMp

Parallel scaling of RBC implementation





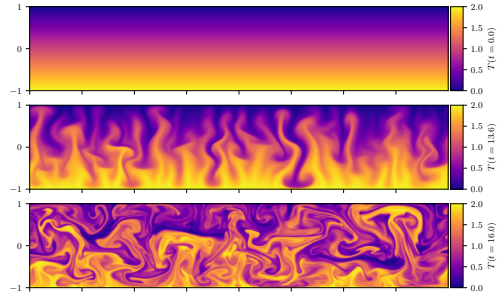
Open issues on GPUs

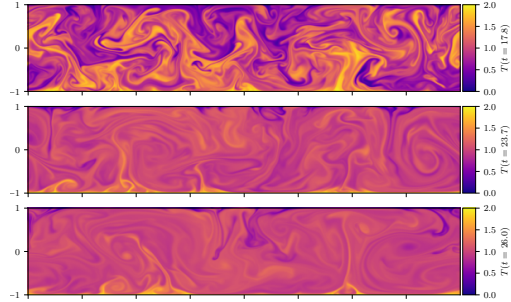
- linear solver is not available on GPU
- current implementation is faster on CPUs

Use diagonal SDC to extend scaling

- Circumvents space-decomposition limit
- Improves strong scaling
- Enables scaling up to 4096 CPUs

Large space-time parallel Rayleigh-Benard simulation





- Use Rayleigh number 2×10^7 on $N = 4096 \times 1024$ grid
- Use Δt-k-adaptivity with four Gauß-Radau nodes for step size selection
- Use $4 \times 1024 = 4096$ CPUs on JURECA DC (32 nodes)
- Use approx. 140k CPU hours to reach $t = 26$ because IMEX stability limit requires $\Delta t \approx 10^{-3}$