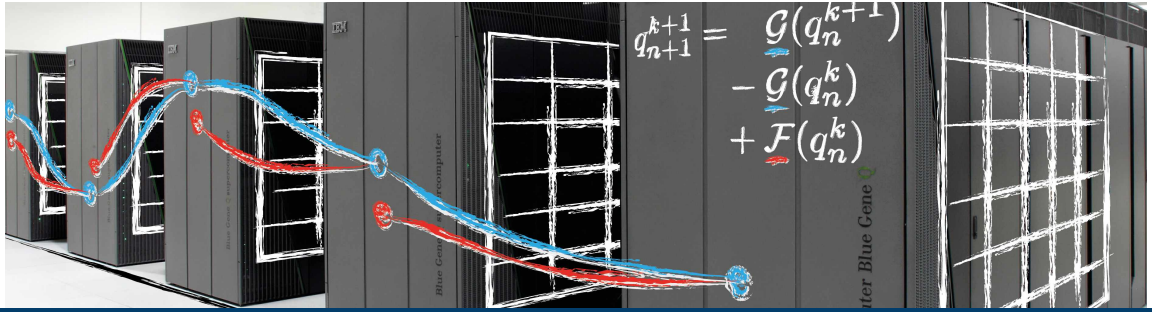


Adaptive Step Size in SDC and What's New With pySDC?

February 6, 2024 | Thomas Baumann | Jülich Supercomputing Centre

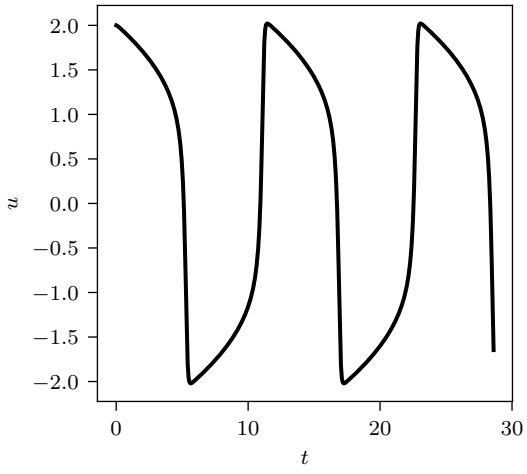
Agenda

- Part I: Science: Adaptive Step Size Selection in SDC
- Part II: How did pySDC evolve during TIME-X?

Part I: Adaptivity in SDC

Motivation: Problems with changing global timescale

Example: Van der Pol



- Models oscillating vacuum tubes:

$$u_{tt} - \mu\left(1 - u^2\right) u_t + u = 0$$

- Oscillations on two time scales
- Constant step size will over-resolve slow parts

Starting point: Embedded Runge-Kutta methods

- Use two sets of weights to get solutions of order p and $q < p$

Butcher tableau

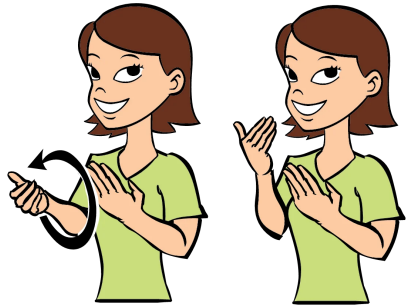
$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \\ & b_1^* & b_2^* & \dots & b_s^* \end{array}$$

- $\sum_i b_i k_i = u_{\text{exact}} + e + \mathcal{O}(\Delta t^{p+1})$
- $\sum_i b_i^* k_i = u_{\text{exact}} + e^* + \mathcal{O}(\Delta t^{q+1})$
- $\implies \sum_i (b_i - b_i^*) k_i = e^* + \mathcal{O}(\Delta t^{q+1})$

- Compute optimal step size to reach ϵ_{TOL}: $\Delta t_{\text{opt}} = 0.9 \left(\frac{\epsilon_{\text{TOL}}}{e^*}\right)^{1/q+1} \Delta t$
- Recompute current step if $e^* > \epsilon_{\text{TOL}}$, move on otherwise

What do we need to change for SDC?

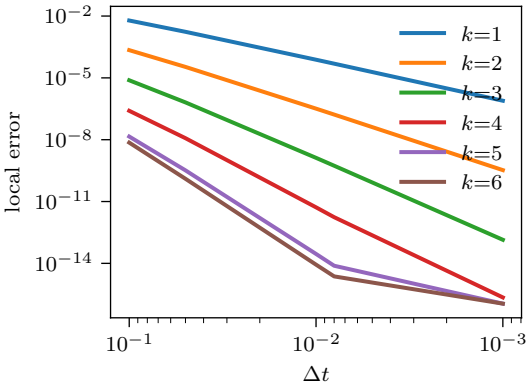
- Construct error estimate
- Done



Baby sign language for “easy”[1]

[1] https://babysignlanguage.com/dictionary/easy

Δt-Adaptivity

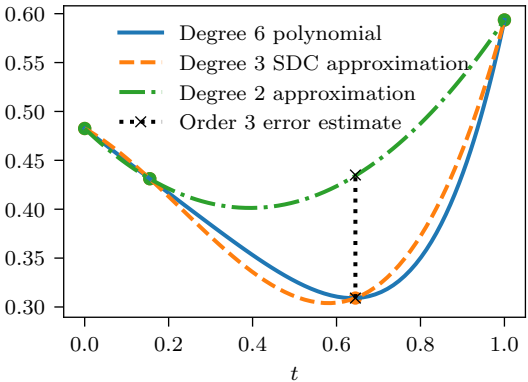


- Increment is error estimate
- Order: $\min(k, 2M - x)$

$\rightarrow$ Keep $k \leq 2M - x$ fixed, but vary Δt

Δt-k-Adaptivity

Use dense output property

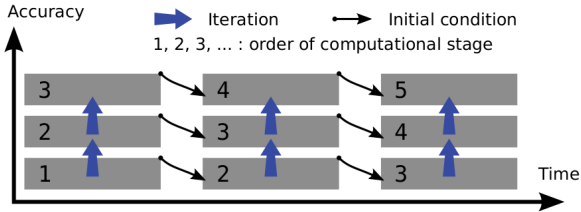


- Solve collocation problem exactly
- Pick one collocation node
- Interpolate from all other nodes to that node
- Interpolated value is lower accuracy
- Difference is estimate of the local error

$\rightarrow$ Vary both k and Δt

What about PinT SDC flavors?

Δt-adaptivity:
Block Gauß-Seidel SDC[2]



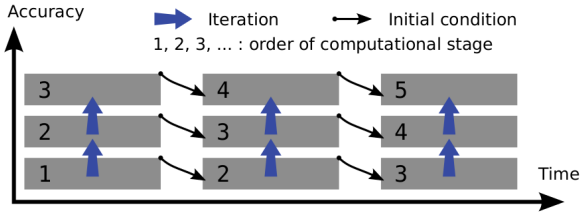
$$Q_\Delta = \begin{pmatrix} \tilde{q}_1 & 0 & 0 & \dots & 0 \\ 0 & \tilde{q}_2 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \dots & 0 \\ 0 & \dots & \dots & \dots & \tilde{q}_M \end{pmatrix}$$

Δt-k adaptivity:
Diagonal SDC
(See Thibaut's secrets)

[2]Image by Thibaut

What about PinT SDC flavors?

Δt-adaptivity:
Block Gauß-Seidel SDC[2]

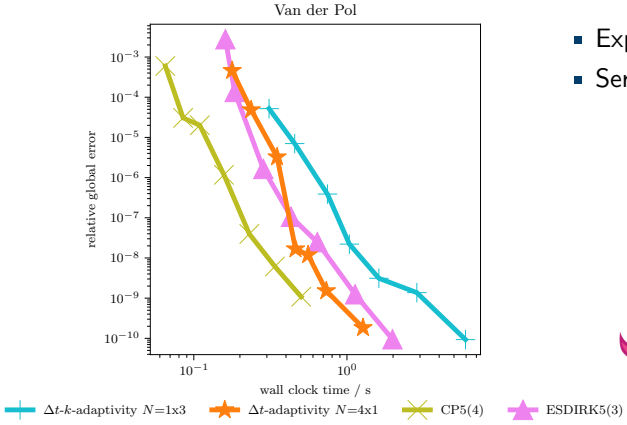


$$Q_\Delta = \begin{pmatrix} \tilde{q}_1 & 0 & 0 & \dots & 0 \\ 0 & \tilde{q}_2 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \dots & 0 \\ 0 & \dots & \dots & \dots & \tilde{q}_M \end{pmatrix}$$

Δt-k adaptivity:
Diagonal SDC
(See Thibaut's secrets)

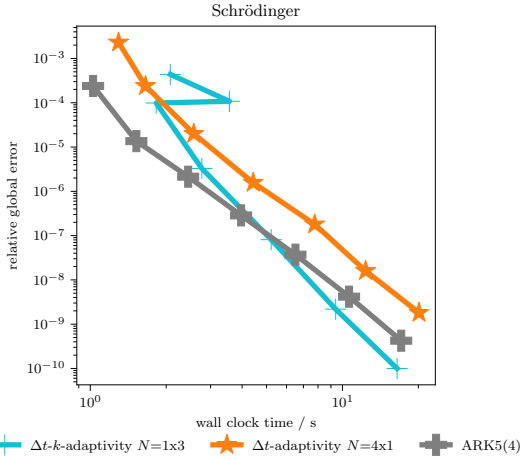
[2]Image by Thibaut

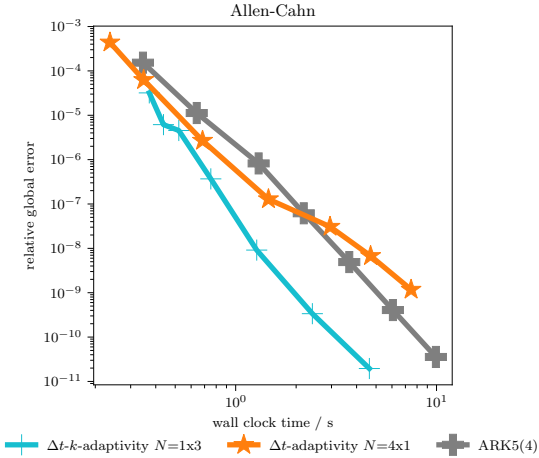
Does it work for Van der Pol?



- Explicit Cash-Carp is very good for this problem
- Serial DIRK method is as good as parallel SDC

Well, what about problems SDC is well suited to?





Adaptive SDC can outperform established Runge-Kutta methods for complicated PDEs!

We had to change the code a bit...

- Need to change simulation parameters during runtime based on current solution
- Want to measure wall times of diagonal SDC
- Want to compare to established methods
- ...

$\rightarrow$ Part II: pySDC Newsletter

Larger changes to the codebase

Easier prototyping

- New callback modules to do arbitrary things to the data at any time
- Added very generic finite difference framework to easily implement new problems

Prototyping performance

- MPI parallel implementation of diagonal SDC
- Added rudimentary GPU support
- Added many popular Runge-Kutta methods for comparisons

Main authors of these changes:

- Ikrom
- Lisa
- Robert
- Thibaut
- Thomas
- Timo Lenz

Projects with pySDC in the last three years

- SDC for second order problems (Ikrom)
- SDC for problems with discontinuities (Lisa)
- SDC for differential algebraic equations (Lisa & Kyrill Ho)
- Preconditioners for diagonal SDC (Gayatri & Thibaut)
- Porting pySDC to GPU (Timo Lenz & Thomas)
- Adaptivity and resilience in SDC (Thomas)
- Compression in SDC (Sansriti Ranjan)

Compression

Can we reduce the enormous memory footprints of collocation methods?

Still work in progress!

What is implemented so far?

- Unused stages are cached
- When cache fills up, stages are compressed
- So far only lossless compression

How well is it working?

- Compression causes very significant computational overhead
- Memory footprint is barely reduced

$\rightarrow$ Need to try again with lossy compression

Porting pySDC to GPU

How difficult is it to port this?

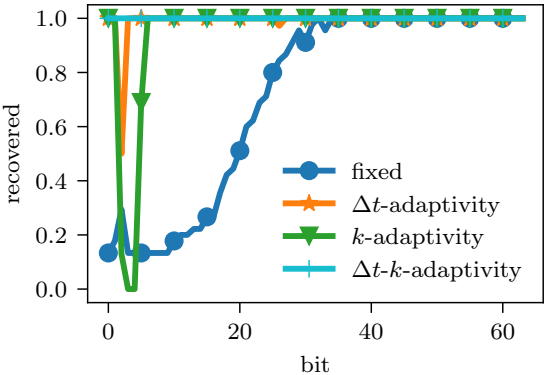
- Replace NUMPY with CUPY to put everything on the GPU
- When using MPI: Need to synchronize host and devices, cannot use Reduce

→ Works almost “out of the box” and is much faster for big problems!

Now working on: Really large space-time-parallel Allen-Cahn simulation with diagonal IMEX SDC and FFT on GPUs

Resilience

What happens when we flip bits while solving the Lorenz attractor?



- SDC with everything fixed: Poor resilience for significant bits
- Adaptive SDC: Good resilience while also increasing computational efficiency