pySDC Tutorial @ PinT12

Before we start...

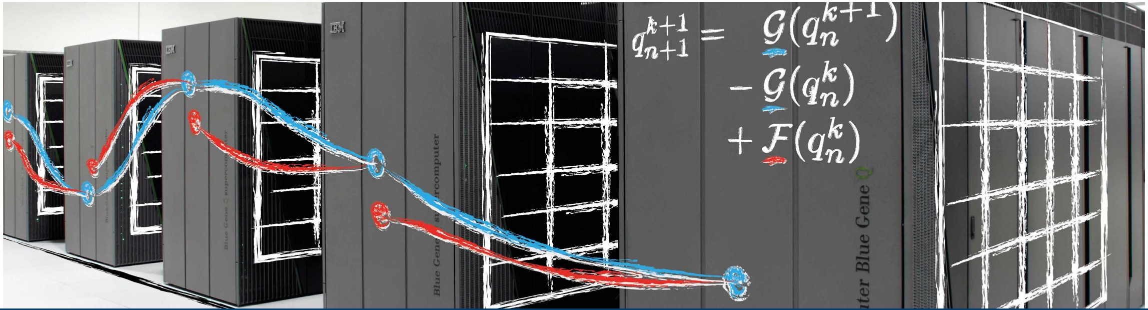
If you want to participate actively:

- Go to the README file and start the installation process:

  `http://bit.ly/pySDC23`

- Ideally, you end up with...
  - A suitable (and virtual!) Python environment
  - A working Jupyter Notebook installation
  - A copy of the `pySDC` code, e.g., by cloning from GitHub
  - A passing test suite
- Thomas will be here to help you

Don't worry if something doesn't work, the tutorial can also be done on your own, at your own pace.


$q_{n+1}^{k+1} = \mathcal{G}(q_n^{k+1}) - \mathcal{G}(q_n^k) + \mathcal{F}(q_n^k)$

pySDC Tutorial @ PinT12

July 19, 2023 | Robert Speck, Thomas Baumann | Jülich Supercomputing Centre

Acknowledgements

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955701. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Belgium, France, Germany, Switzerland.

This project has received funding from the German Federal Ministry of Education and Research through the ParaPhase project within the framework "IKT 2020 - Forschung für Innovationen" (project number 01IH15005A).

SDC and PFASST implementations

FAQ: "Is it hard to use SDC/PFASST?"

Yes

- ... if you already have a full-fledged application or
- ... if you need/want your own time integrator

No

- ... if your code allows access to the ODE's right-hand side etc. or
- ... if you have a lot of time (or Oompa Loompas)

To cover as many scenarios as possible, you can choose between (at least) 3 codes:

1. the prototyping framework pySDC
2. the standalone HPC code libpfasst
3. the DUNE module dune-PFASST

SDC and PFASST implementations

FAQ: "Is it hard to use SDC/PFASST?"

Yes

- ... if you already have a full-fledged application or
- ... if you need/want your own time integrator

No

- ... if your code allows access to the ODE's right-hand side etc. or
- ... if you have a lot of time (or Oompa Loompas)

To cover as many scenarios as possible, you can choose between (at least) 3 codes:

1. the prototyping framework pySDC — the "playground"
2. the standalone HPC code libpfasst — the "library"
3. the DUNE module dune-PFASST — the "specialist"

pySDC - the playground

Landing page: `https://parallel-in-time.org/pySDC`

Properties:

- purpose: prototyping, education, easy access, “test before you invest”
- not (very) optimized, but well-documented, Python

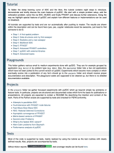
Features:

- many variants of SDC and PFASST
- many examples, from heat equation to particles in an electromagnetic field
- can use whatever data structure and solvers you want (e.g. FEniCS, PETSc)

Some pySDC features

Tutorials and examples

- Ships with a lot of examples
- Results clearly defined
- Copy from there ...
- ... no White Paper Syndrome



Tutorial

To flatten the steep learning curve of SDC and the likes, this tutorial contains eight steps to introduce, demonstrate and briefly discuss the main features of pySDC. We start with a simple problem setup, add the collocation problem, solve this via SDC, MLSDC and finally PFASST and discuss further extensions. Along the way we highlight special features of pySDC and explain how different features or implementations can be used (or disabled).

All tutorials are supported by tests and are run automatically after pushing to master. The results are shown within the description and can be found here (yes, yes, Jupyter notebooks would be awesome, just need to find someone to do it):

- Step-1: A first spatial problem
- Step-2: Data structures and my first sweeper
- Step-3: Statistics and a new sweeper
- Step-4: Multilevel SDC
- Step-5: PFASST
- Step-6: Advanced PFASST controllers
- Step-7: pySDC with external libraries
- Step-8: Advanced topics

Playgrounds

This folder gathers various small to medium experiments done with pySDC. They are for example grouped by application (e.g. Boris) or by problem type (e.g. ODEs). Also, the deprecated folder lists a few old applications which have not been ported to the current version of pySDC. Experiments which become more complex or which eventually evolve into a publication of any form should go to the projects folder and should receive proper documentation and description. The playground codes are supposed to be cleaned up, but there is no detailed documentation available.

Projects

In the projects folder we gather focussed experiments with pySDC which go beyond simple toy problems or feature tests. In particular, projects are structured and documented codes which form the basis for publications or presentations. All projects are expected to contain a README file describing the intention and content of the work. Parts of the Python scripts are supported by tests and checked for PEP8 conformity.

- Attempts to parallelize SDC
- Fault-tolerance with PFASST: node failures
- Fast-Wave-Slow-Wave SDC
- RDC: Rational Deferred Corrections
- Asymptotic convergence of PFASST
- Matrix-based versions of PFASST
- Second-order Problems
- What is the fastest SDC variant?
- Allen-Cahn problems from Bayreuth
- Performance analysis of pySDC

Tests

Most of the code is supported by tests, mainly realized by using the tutorial as the test routines with clearly defined results. Also, projects are accompanied by tests.

Github Action reports: CI pipeline for pySDC passing, test coverage results can be found here.

Parallel and serial

- Serial algorithms
- Pseudo-parallel algorithms
- Time-parallel algorithms
- Space-time parallel algorithms

PETSc+FEniCS integration

- Use their data structures, discretization, solvers, ...
- Use PETSc's parallelization
- Work in progress...

CI/CD/CT

- GitHub Pages...
- ...and GitHub Actions
- Core features testing
- Reproduce paper results

Why have more codes?

pySDC's pros

- many features from the SDC and PFASST universe
- code is close to formulas in publications
- well-documented, tutorials, many examples to copy from
- easy to install, easy to port, easy to use
- close-to-optimal time-to-simulation

pySDC's cons

- no memory optimization, no tuning for time-to-solution
- hard to convince people to use Python for production (on this level)
- hard to use within large, existing applications

Separation of concerns

Prototypes vs Specialists

Pro prototypes

- Focus on math, numerics, algorithms
- Fast(er) results here
- No predetermination of a “field”

Pro specialists

- Larger-scale applications possible
- Some problems only show up “in real life”
- HPC requires more thoughts/care

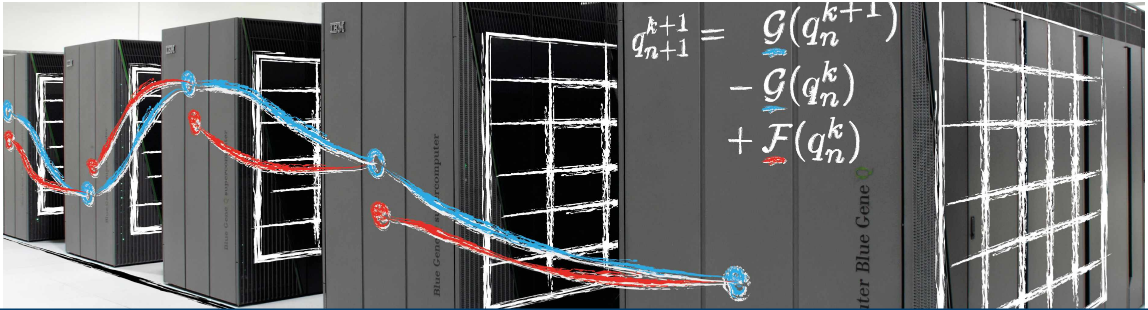
Alternative: Implement within existing time-integration framework (e.g. SUNDIALS, PETSc)

- After initial struggles, code may be used for prototyping
- After careful implementation, code can be used for applications
- Broad user base, established software engineering workflows

Key obstacle: severe initial porting effort (which will probably pay off big time, though)

And now..

Let’s get going!


$q_{n+1}^{k+1} = \mathcal{G}(q_n^{k+1}) - \mathcal{G}(q_n^k) + \mathcal{F}(q_n^k)$
Blue Gene

pySDC Tutorial @ PinT12

July 19, 2023 | Robert Speck, Thomas Baumann | Jülich Supercomputing Centre