Parallel-in-Time integration with pySDC
From prototyping to applications

April 16, 2019 | Robert Speck | Jülich Supercomputing Centre
Collaborators

- UNIVERSITY OF LEEDS
  - Daniel Ruprecht
- Università della Svizzera Italiana
  - Rolf Krause
- Institute of Computational Science
  - Oliver Sander
- BERGISCHE UNIVERSITÄT WUPPERTAL
  - Matthias Bolten
- You?
- BERKELEY LAB
  - Michael Minion

Member of the Helmholtz Association
Robert Speck
Slide 1
Limits of purely spatial parallelization

- Spatial parallelization reduces runtime **per time-step**
- Strong scaling saturates eventually because of communication
- Costs for **more time-steps** are not mitigated

**Figure:** Time-stepping to solve time-dependent partial differential equations.
Limits of purely spatial parallelization

Figure: Time-stepping to solve time-dependent partial differential equations.

- Spatial parallelization reduces runtime **per time-step**
- Strong scaling saturates eventually because of communication
- Costs for **more time-steps** are not mitigated
Limits of purely spatial parallelization

- Spatial parallelization reduces runtime per time-step
- Strong scaling saturates eventually because of communication
- Costs for more time-steps are not mitigated

Figure: Time-stepping to solve time-dependent partial differential equations.
Limits of purely spatial parallelization

- Spatial parallelization reduces runtime per time-step
- Strong scaling saturates eventually because of communication
- Costs for more time-steps are not mitigated

Figure: Time-stepping to solve time-dependent partial differential equations.
Limits of purely spatial parallelization

- Spatial parallelization reduces runtime per time-step
- Strong scaling saturates eventually because of communication
- Costs for more time-steps are not mitigated

→ Can we compute multiple time-steps simultaneously?

Figure: Time-stepping to solve time-dependent partial differential equations.
Parallel-in-Time ("PinT") approaches


- Interpolation-based approach (Nievergelt 1964)
- Predictor-corrector approach (Miranker, Liniger 1967)
- Parabolic or time multi-grid (Hackbusch 1984) and (Horton 1992)
- Multiple shooting in time (Kiehl 1994)
- Parallel Runge-Kutta methods (e.g. Butcher 1997)
- Parareal (Lions, Maday, Turinici 2001)
- PITA (Farhat, Chandesris 2003)
- Guided Simulations (Srinavasan, Chandra 2005)
- RIDC (Christlieb, Macdonald, Ong 2010)
- PFASST (Emmett, Minion 2012)
- MGRIT (Falgout et al 2014)
- ... and many more
Parallel-in-Time ("PinT") approaches

"50 years of parallel-in-time integration", M. Gander (CMCS, 2015)

- Interpolation-based approach (Nievergelt 1964)
- Predictor-corrector approach (Miranker, Liniger 1967)
- Parabolic or time multi-grid (Hackbusch 1984) and (Horton 1992)
- Multiple shooting in time (Kiehl 1994)
- Parallel Runge-Kutta methods (e.g. Butcher 1997)
- Parareal (Lions, Maday, Turinici 2001)
- PITA (Farhat, Chandesris 2003)
- Guided Simulations (Srinavasan, Chandra 2005)
- RIDC (Christlieb, Macdonald, Ong 2010)
- PFASST (Emmett, Minion 2012)
- MGRIT (Falgout et al 2014)
- ... and many more
A quick algebraic introduction to PFASST

Basic building block: spectral deferred corrections (SDC)

Consider the Picard form of an initial value problem on \([T_0, T_1]\)

\[ u(t) = u_0 + \int_{T_0}^{t} f(u(s)) \, ds , \]

discretized using spectral quadrature rules with nodes \(t_m\):

\[ u_m = u_0 + \Delta t QF(u) \approx u_0 + \int_{T_0}^{t_m} f(u(s)) \, ds , \]

then SDC methods can be seen as (clever) Gauß-Seidel iteration to solve this collocation problem for all \(u_m\).

⇒ Use this for block smoothing in space-time multigrid = PFASST
A quick algebraic introduction to PFASST

Basic building block: spectral deferred corrections (SDC)

Consider the Picard form of an initial value problem on $[T_0, T_1]$

$$u(t) = u_0 + \int_{T_0}^{t} f(u(s)) \, ds,$$

discretized using spectral quadrature rules with nodes $t_m$:

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

then SDC methods can be seen as (clever) Gauß-Seidel iteration to solve this collocation problem for all $u_m$.

⇒ Use this for block smoothing in space-time multigrid = PFASST
A quick visual introduction to PFASST
A quick visual introduction to PFASST
A quick visual introduction to PFASST

Member of the Helmholtz Association

Robert Speck

Slide 5
A quick visual introduction to PFASST

![Diagram showing computation time and sweep operations]

- coarse sweep
- fine sweep
- coarse comm.
- fine comm.

$t_0, P_0, t_1, P_1, t_2, P_2, t_3, P_3, t_4$

Member of the Helmholtz Association

Robert Speck

Slide 5
A quick visual introduction to PFASST

Member of the Helmholtz Association  Robert Speck  Slide 5
A quick visual introduction to PFASST

coarse sweep
fine sweep
course comm.
fine comm.

computation time

$t_0$ $P_0$ $t_1$ $P_1$ $t_2$ $P_2$ $t_3$ $P_3$ $t_4$

Member of the Helmholtz Association
Robert Speck
A quick visual introduction to PFASST

- Coarse sweep
- Fine sweep
- Coarse comm.
- Fine comm.

Computation time

$t_0$, $P_0$, $t_1$, $P_1$, $t_2$, $P_2$, $t_3$, $P_3$, $t_4$
A quick visual introduction to PFASST

Member of the Helmholtz Association

Robert Speck

Slide 5
A quick visual introduction to PFASST
A quick visual introduction to PFASST
A quick visual introduction to PFASST

computation time

coarse sweep
fine sweep
coarse comm.
fine comm.

$t_0\; P_0 \quad t_1 \quad P_1 \quad t_2 \quad P_2 \quad t_3 \quad P_3 \quad t_4$

predictor

Member of the Helmholtz Association
Robert Speck
A quick visual introduction to PFASST

- coarse sweep
- fine sweep
- coarse comm.
- fine comm.

Member of the Helmholtz Association

Robert Speck
A quick visual introduction to PFASST
A quick visual introduction to PFASST
A quick visual introduction to PFASST
A quick visual introduction to PFASST
A quick visual introduction to PFASST
Two examples

RS, D. Ruprecht et al. (SC 2012 and SC 2013)

- vortex particles with Barnes-Hut tree code in space
- reduced force calculation on coarse level

→ 7x additional speedup

- 3D heat equation with multigrid solver in space
- coarsening via reduction of grid points and discretization order

→ 4x improved speedup
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 3 codes:

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

Why Python?
- easy to learn, easy to use, easy to port
- expensive parts = spatial solvers = optimized C/Fortran anyway
- trading speed vs. time-to-simulation
### Other cool things

#### Fault tolerance playground
- PinT + ABFT
- Protect against bitflips
- Recover after data loss
- Testbed for ideas

#### Parallel SDC
- Diag. preconditioners
- Shared-memory parallelization in time
- Multi-parallel PinT?

#### PETSc integration
- PETSc’s data structures
- PETSc’s parallelization
- Integrators for Parareal?
- Work in progress...

#### Continuous integration
- GitHub Pages...
- ...and Travis-CI
- 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

pySDC’s cons

- no memory optimization, no tuning for speed
- hard to convince people to use Python for production
- hard to use within large, existing applications
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

**pySDC’s cons**
- no memory optimization, no tuning for speed
- hard to convince people to use Python for production
- hard to use within large, existing applications

To integrate SDC/PFASST into existing applications/frameworks, we need dedicated implementations.. the “specialists”. 
dune-PFASST - the specialist

Landing page: https://github.com/Parallel-in-Time/dune-PFASST

Properties:

- purpose: PFASST for DUNE-based finite element simulations
- a bit of documentation, implemented within DUNE, C++

Features:

- currently tuned for reaction-diffusion examples (toward phase-field problems)
- primarily focused on DUNE data structures
- a blueprint for application-oriented PinT implementations
Three takeaways

Parallel-in-Time integration with PFASST (and others) can help you to overcome scaling limits

A good place to start with SDC and PFASST, to run first examples and to test your ideas: pySDC

Libraries vs. specialists: community needs both to make progress in numerics, codes and applications
Three takeaways

Parallel-in-Time integration with PFASST (and others) can help you to overcome scaling limits

A good place to start with SDC and PFASST, to run first examples and to test your ideas: pySDC

Libraries vs. specialists: community needs both to make progress in numerics, codes and applications