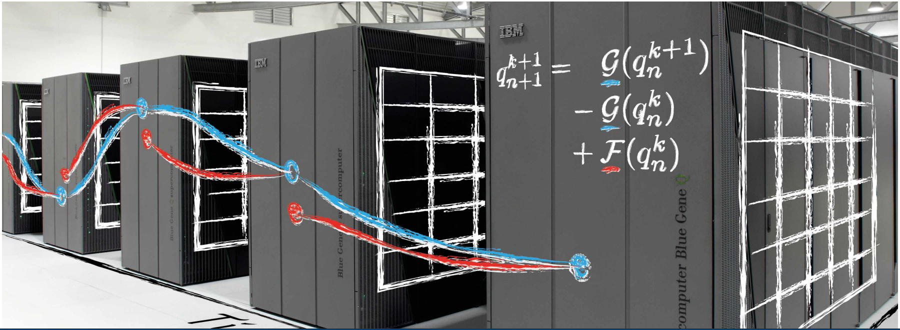


Parallel-in-Time Integration with PFASST
From prototyping to applications

June 5, 2019 | Robert Speck | Jülich Supercomputing Centre

Collaborators

UNIVERSITY OF LEEDS
Daniel Ruprecht

Università della Svizzera italiana — Faculty of Informatics — Institute of Computational Science ICS
Rolf Krause

TECHNISCHE UNIVERSITÄT DRESDEN
Oliver Sander

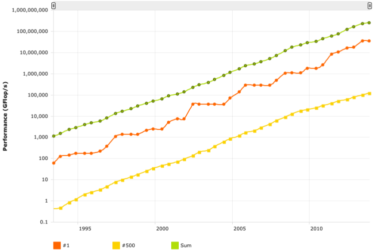
BERGISCHE UNIVERSITÄT WUPPERTAL
Matthias Bolten

You?

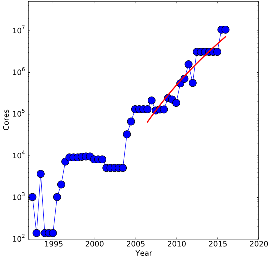
BERKELEY LAB
Michael Minion

Moore's law in HPC today

"The free lunch is over" (H.Sutter, 2005)



(a) Performance of the world's 500 most powerful supercomputers.



(b) Number of cores in the number one system in the Top 500 list.

- HPC systems already require multi-million way concurrency
- Need new numerical methods to provide this degree of parallelism

Limits of purely spatial parallelization

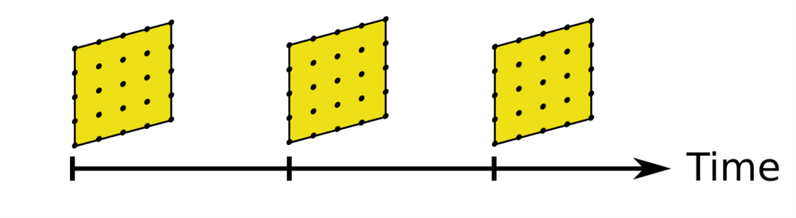


Figure: Time-stepping to solve time-dependent partial differential equations.

- 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

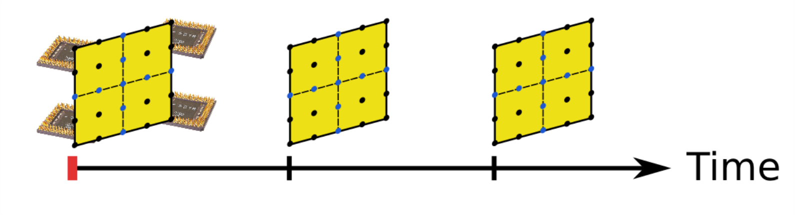


Figure: Time-stepping to solve time-dependent partial differential equations.

- 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

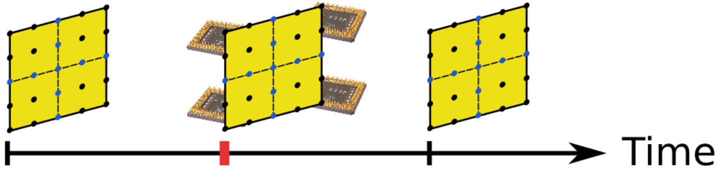


Figure: Time-stepping to solve time-dependent partial differential equations.

- 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

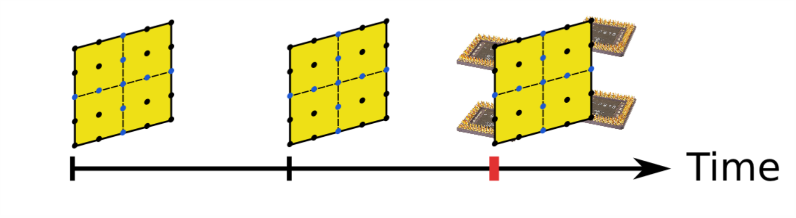


Figure: Time-stepping to solve time-dependent partial differential equations.

- 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

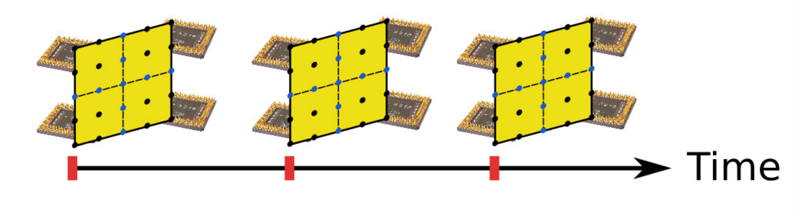


Figure: Time-stepping to solve time-dependent partial differential equations.

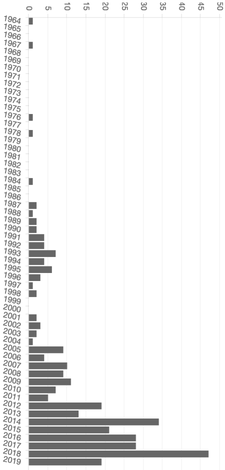
- Spatial parallelization reduces runtime **per time-step**
- Strong scaling saturates eventually because of communication
- Costs for **more time-steps** are not mitigated

→ Can we compute multiple time-steps simultaneously?

Parallel-in-Time ("PinT") approaches

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

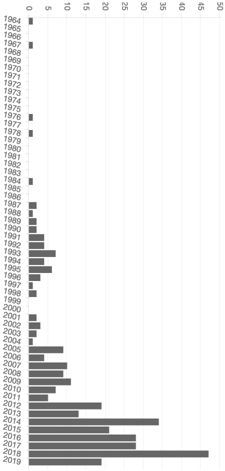
- Interpolation-based approach (Nievergelt 1964)
- Predictor-corrector approach (Miranker, Liniger 1967)
- Parabolic or time multi-grid (Hackbusch 1984) and (Horton 1992)
- Multiple shooting in time (Kiehl 1994)
- Parallel Runge-Kutta methods (e.g. Butcher 1997)
- Parareal (Lions, Maday, Turinici 2001)
- 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



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



<math>

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.

$\Rightarrow$ 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:

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

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

$\Rightarrow$ Use this for block smoothing in space-time multigrid = **PFASST**

A quick algebraic introduction to PFASST

Multigrid for the composite collocation problem

We now glue L time-steps together, using N to transfer information from step l to step $l+1$. We get the composite collocation problem:

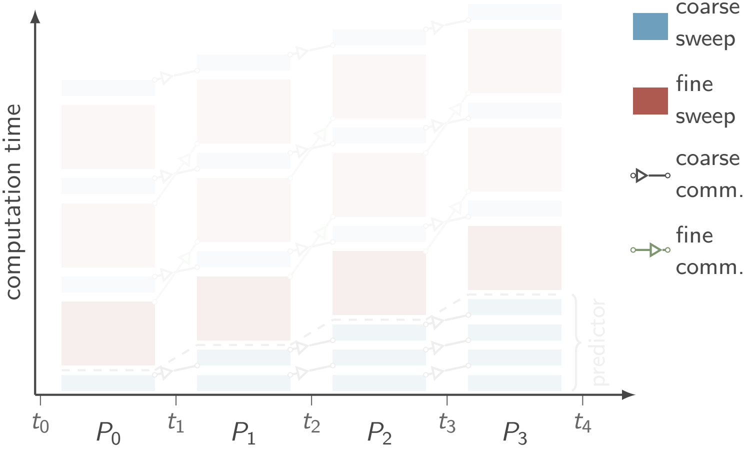
$$\begin{pmatrix} I-\Delta tQF & & & \\ -N & I-\Delta tQF & & \\ & \ddots & \ddots & \\ & & -N & I-\Delta tQF \end{pmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vdots \\ \vec{u}_L \end{pmatrix} = \begin{pmatrix} \vec{u}_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

PFASST:

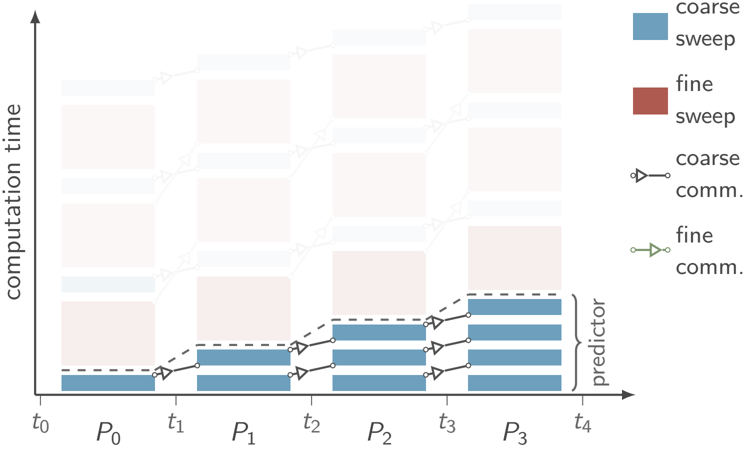
- use (linear/FAS) multigrid to solve this system iteratively
- smoother: parallel block Jacobi with SDC in the blocks
- coarse-level solver: serial block Gauß-Seidel with SDC in the blocks
- exploit cheapest coarse level to quickly propagate information forward in time

</math>

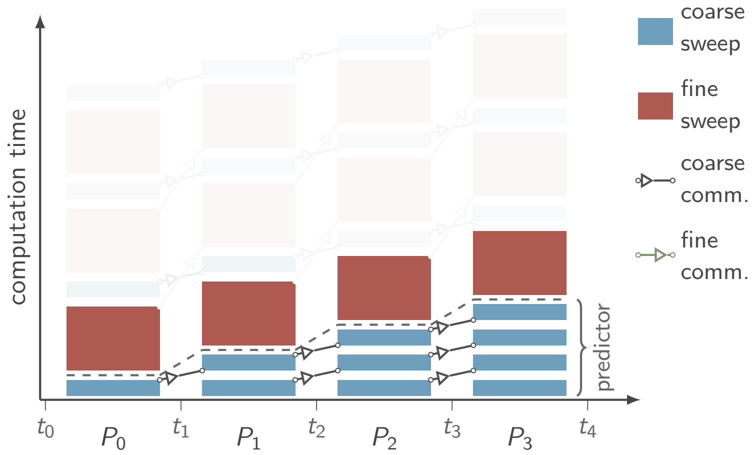
A quick visual introduction to PFASST



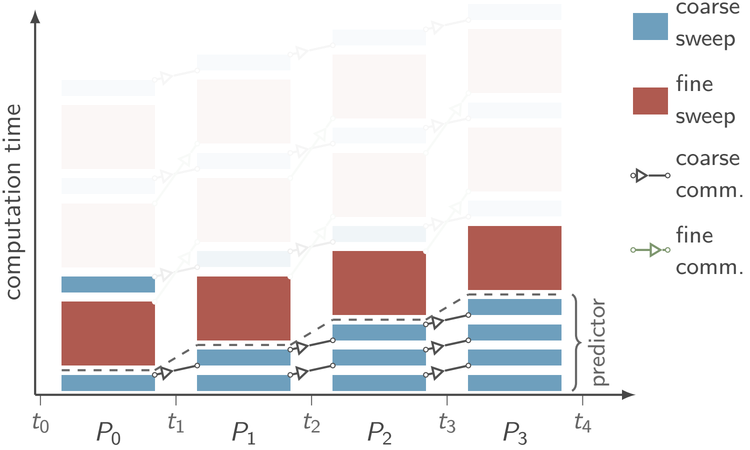
A quick visual introduction to PFASST



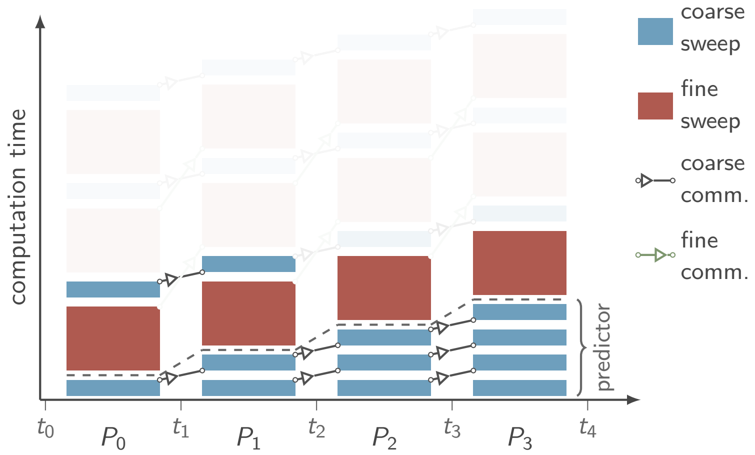
A quick visual introduction to PFASST



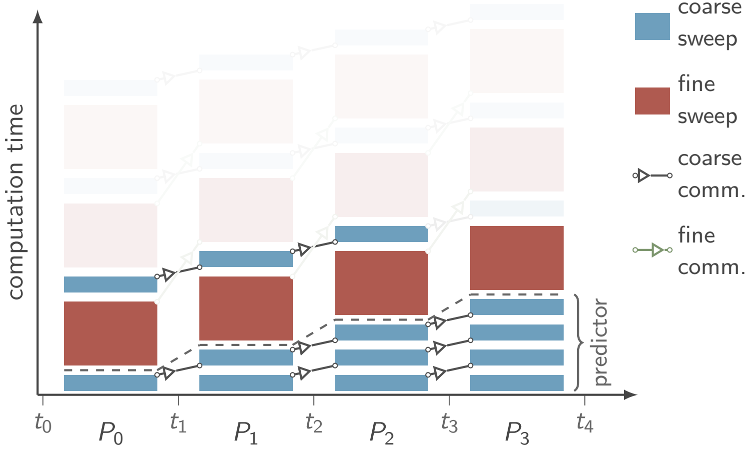
A quick visual introduction to PFASST



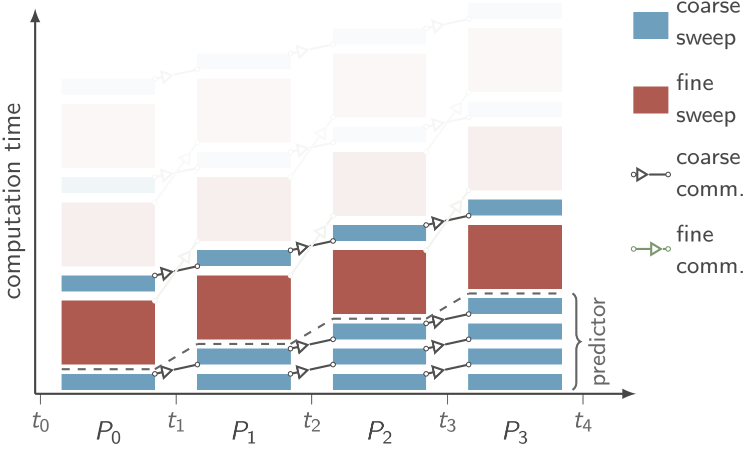
A quick visual introduction to PFASST



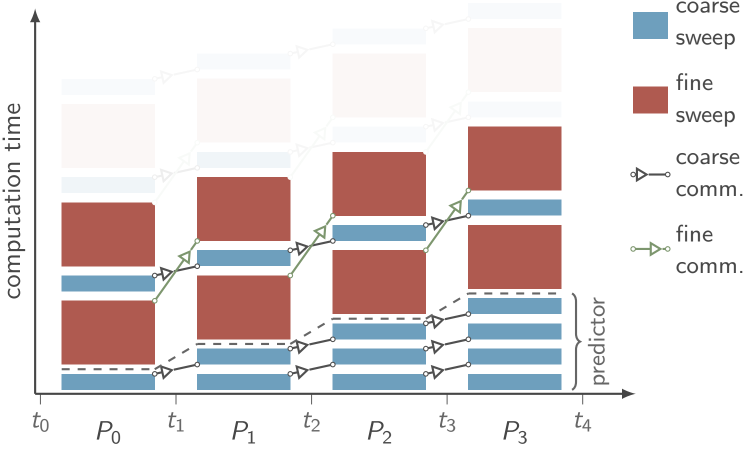
A quick visual introduction to PFASST



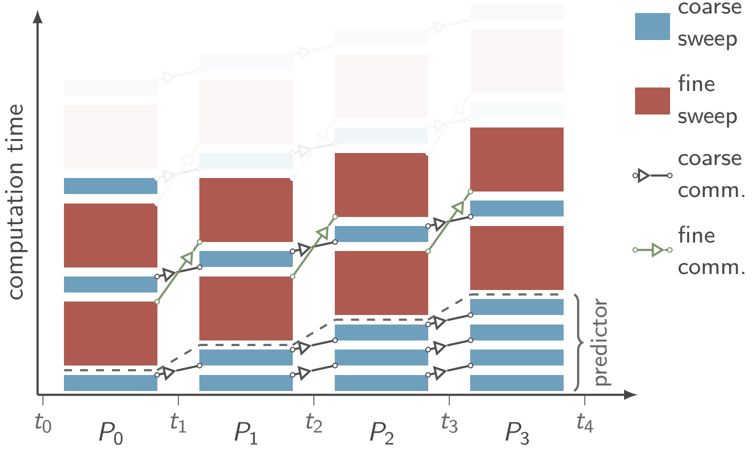
A quick visual introduction to PFASST



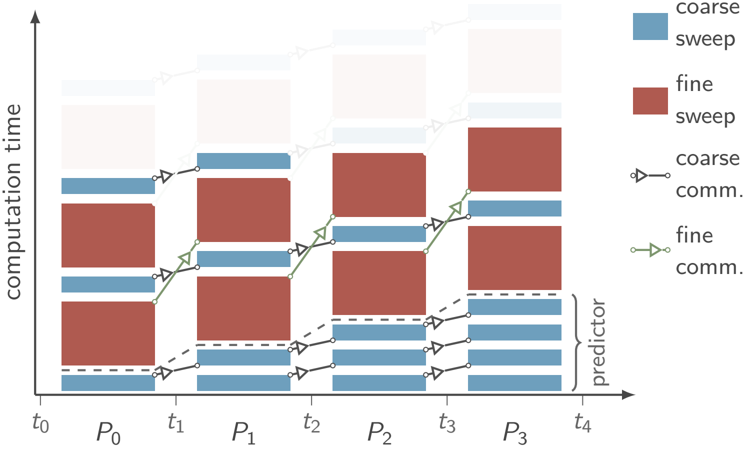
A quick visual introduction to PFASST



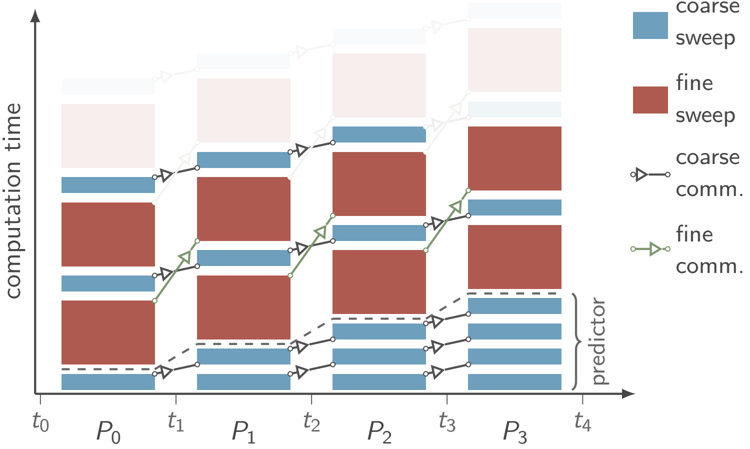
A quick visual introduction to PFASST



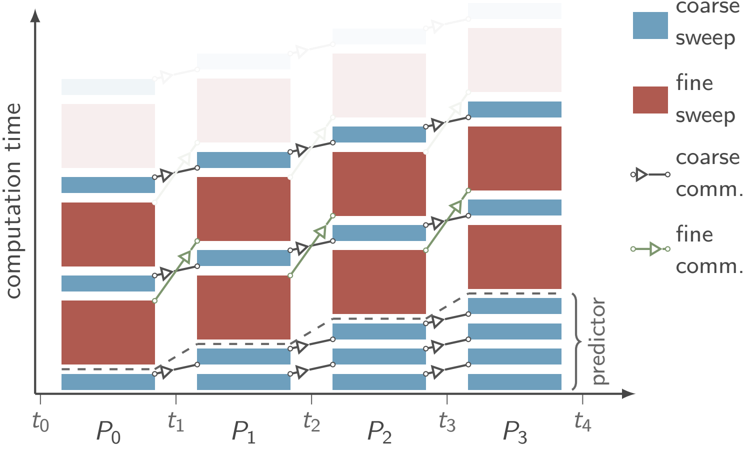
A quick visual introduction to PFASST



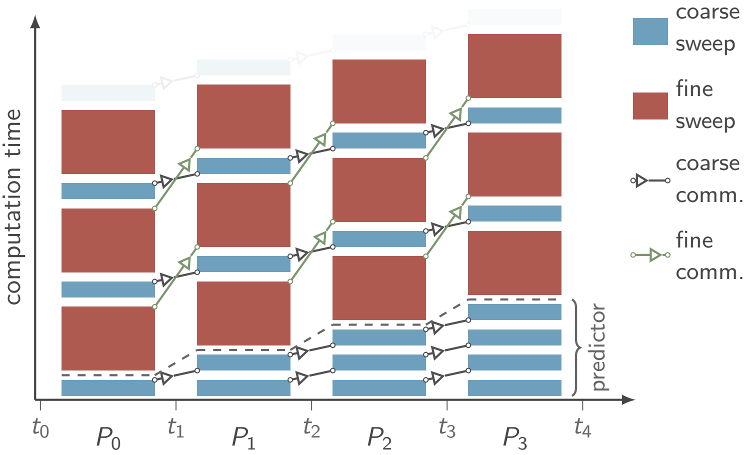
A quick visual introduction to PFASST



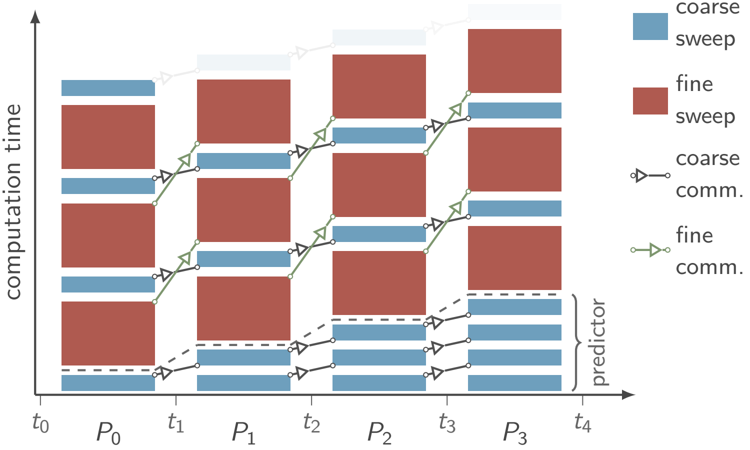
A quick visual introduction to PFASST



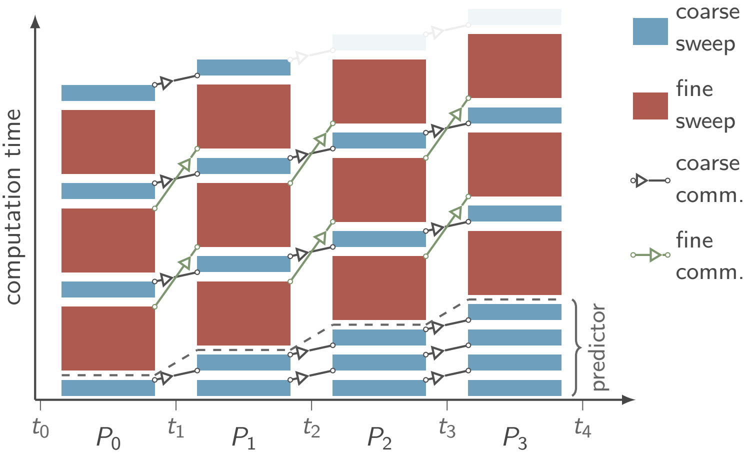
A quick visual introduction to PFASST



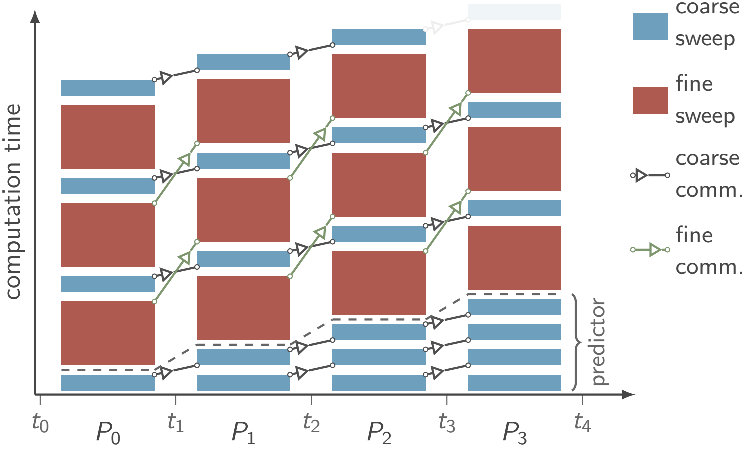
A quick visual introduction to PFASST



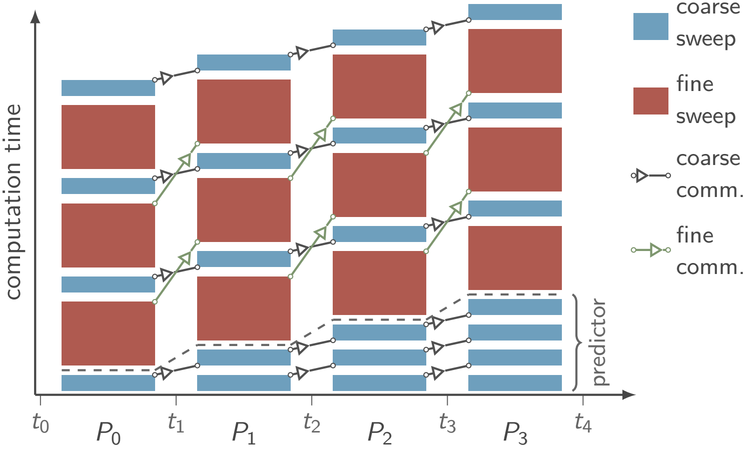
A quick visual introduction to PFASST



A quick visual introduction to PFASST



A quick visual introduction to PFASST



PFASST implementations

FAQ: "Is it hard to use 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 already work with spectral deferred corrections

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

PFASST implementations

FAQ: "Is it hard to use 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 already work with spectral deferred corrections

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: `http://www.parallel-in-time.org/pySDC`

Properties:

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

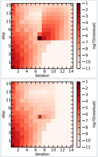
Features:

- 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)

Other cool things

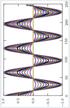
Fault tolerance playground

- PinT + ABFT
- Protect against bitflips
- Recover after data loss
- Testbed for ideas



Hamiltonian problems

- Newton's eqs of motion
- basis: velocity-Verlet
- From toy problems...
- ...to MD, someday?

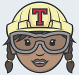


PETSc integration

- 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

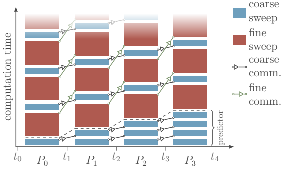
- many features from the SDC and PFASST universe
- code is close to formulas in publications
- well-documented, tutorials, many examples to copy from
- easy to install, easy to port, easy to use

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 PFASST into existing applications/frameworks, we need dedicated implementations.. the "specialists".

Three takeaways



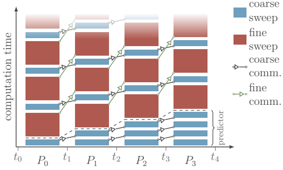
Parallel-in-Time integration with PFASST (and others) can help you to overcome scaling limits

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

images/lego-pile.jpg

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**

images/lego-pile.jpg

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

The PinT Community

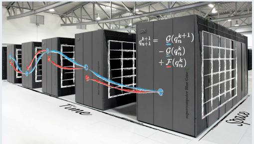
To learn more about PinT check out the website

`www.parallel-in-time.org`

and/or join one of the PinT Workshops, e.g.

9th Workshop on Parallel-in-Time Integration

- June 8-12, 2020
- Michigan, USA
- organized by Ben Ong and others



Also, there is a mailing list, join by writing to

`parallelintime+subscribe@googlegroups.com`