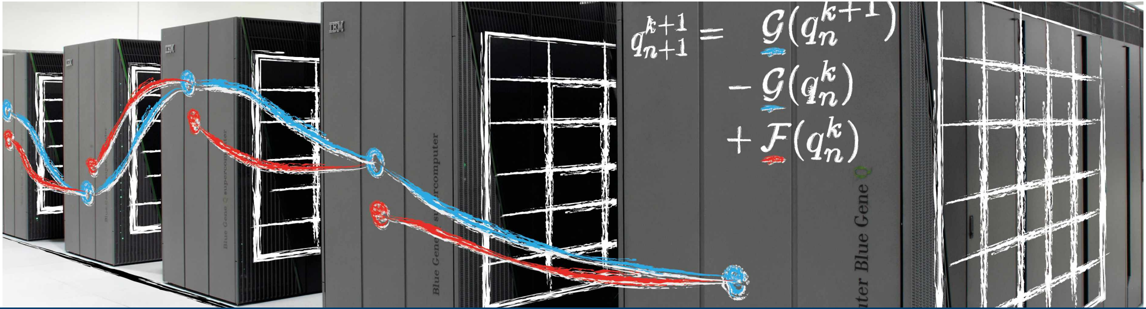


Parallel-in-Time Collocation Methods

June 10, 2024 | Robert Speck & Ruth Partzsch | Jülich Supercomputing Centre

Acknowledgements

SPONSORED BY THE
Federal Ministry of Education and Research

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 StroemungsRaum project within the SCALEXA framework (project number 16ME0708).

Collaborators

TUHH
Technische Universität Hamburg

Daniel Ruprecht

Università della Svizzera italiana | Faculty of Informatics | Institute of Computational Science ICS

Rolf Krause

Karlsruher Institut für Technologie

Martin Frank

BERGISCHE UNIVERSITÄT WUPPERTAL

Matthias Bolten

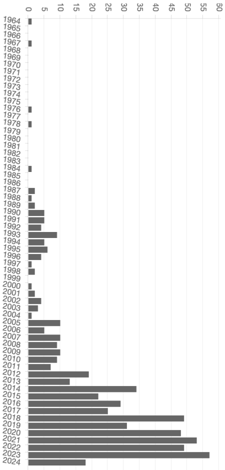
You?

Michael Minion

Parallel-in-Time ("PinT") approaches

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

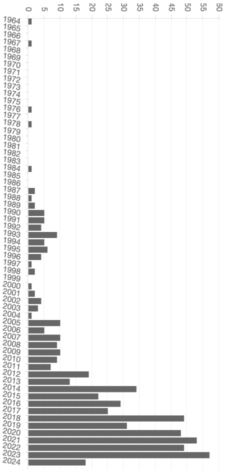
- Interpolation-based approach (Nievergelt 1964)
- Predictor-corrector approach (Miranker, Liniger 1967)
- Parabolic or time multi-grid (Hackbusch 1984)
- Multiple shooting in time (Kiehl 1994)
- Parallel Runge-Kutta methods (e.g. Butcher 1997)
- Parareal (Lions, Maday, Turinici 2001)
- PITA (Farhat, Chandesris 2003)
- Guided Simulations (Srinavasan, Chandra 2005)
- RIDC (Christlieb, Macdonald, Ong 2010)
- PFASST (Emmett, Minion 2012)
- MGRIT (Falgout et al 2014)
- ParaDIAG (Wu et al 2021)
- ...



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)
- 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)
- ParaDIAG (Wu et al 2021)
- ...



For this talk: the collocation problem

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:

$$u_m = u_0 + \Delta t \sum_{l=1}^{M} q_{m,l} f(u_l) \approx u_0 + \int_{T_0}^{t_m} f(u(s))ds,$$

$\Rightarrow$ corresponds to a fully implicit Runge-Kutta method on $[T_0, T_1]$.

How to solve this system (and more) in parallel?

For this talk: the collocation problem

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:

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

$\Rightarrow$ corresponds to a fully implicit Runge-Kutta method on $[T_0, T_1]$.

How to solve this system (and more) in parallel?

Four approaches

Following Kevin Burrage's terminology:

1. "Parallelization across the method": computation of the solution at all M stages at once
   1. using diagonalization of Q
   2. using preconditioned spectral deferred corrections

2. "Parallelization across the steps": computation of the solution at multiple steps at once
   1. using multilevel/multigrid techniques
   2. using diagonalization techniques

Parallelization across the method I

Diagonalization

For suitable choices of the M collocation nodes, Q can be diagonalized, i.e. for linear problems

$$(I - \Delta t Q F)(\vec{u}) = (I - \Delta t Q \otimes A)\vec{u} = (V_Q \otimes I)(I - \Delta t D_Q \otimes A)(V_Q \otimes I)^{-1}\vec{u}$$

with diagonal matrix D_Q.

Remarks:

- This is a direct solver for linear problems
- Extension to nonlinear problems via inexact Newton
- Classical approach to deal with fully-implicit RK methods
- Beware: D_Q has complex entries!

Parallelization across the method II

Spectral deferred corrections (serial, for now)

- standard Picard iteration is Richardson for $(I - \Delta t QF)(\vec{u}) = \vec{u}_0$, i.e.

$$\vec{u}^{k+1} = \vec{u}^k + \left(\vec{u}_0 - (I - \Delta t QF)(\vec{u}^k)\right)$$

- preconditioning: use simpler integration rule Q_Δ with

$$(I - \Delta t Q_\Delta F)(\vec{u}^{k+1}) = (I - \Delta t Q_\Delta F)(\vec{u}^k) + \left(\vec{u}_0 - (I - \Delta t QF)(\vec{u}^k)\right)$$

  This corresponds to **spectral deferred corrections (SDC)**!

- if the integration rule Q_Δ is implicit/explicit, the whole iteration will be implicit/explicit
- can also do IMEX, multi-implicit and (limited) multirate time-stepping, high-order Boris-SDC, adaptive time-stepping, fault-tolerant integration, ...

Parallelization across the method II

Spectral deferred corrections (serial, for now)

- standard Picard iteration is Richardson for $(I - \Delta t Q F)(\vec{u}) = \vec{u}_0$, i.e.

$$\vec{u}^{k+1} = \vec{u}^k + \left(\vec{u}_0 - (I - \Delta t Q F)(\vec{u}^k)\right)$$

- preconditioning: use simpler integration rule Q_Δ with

$$(I - \Delta t Q_\Delta F)(\vec{u}^{k+1}) = \vec{u}_0 + \Delta t \left(Q - Q_\Delta\right) F(\vec{u}^k)$$

  This corresponds to **spectral deferred corrections (SDC)**!

- if the integration rule Q_Δ is implicit/explicit, the whole iteration will be implicit/explicit
- can also do IMEX, multi-implicit and (limited) multirate time-stepping, high-order Boris-SDC, adaptive time-stepping, fault-tolerant integration, ...

Parallelization across the method II

Parallel SDC, with Ruth Schöbel, Daniel Ruprecht, Thibaut Lunet, Gayatri Caklovic and others

Idea: use diagonal Q_Δ to compute updates simultaneously for all collocation nodes

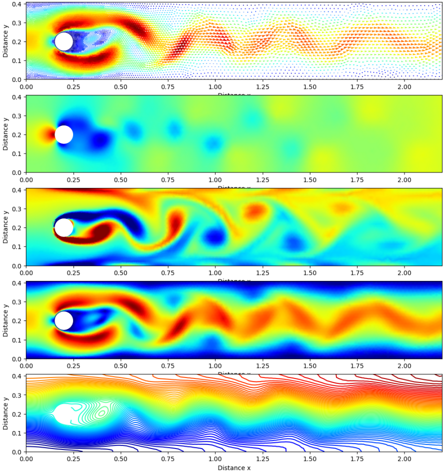
How to find a suitable Q_Δ?

1. Standard tricks like the diagonal of Q (don't work well)
2. Minimize $\rho(I - Q_\Delta^{-1} Q)$ to tune the iteration for the stiff limit (works well for stiff problems)
3. Use machine/reinforcement learning to find the "optimal" entries of Q_Δ for a given problem class

$\rightarrow$ New paper by Thibaut et al. has very promising results!

Parallel SDC for Navier-Stokes equations

IMEX SDC using a projection-based approach



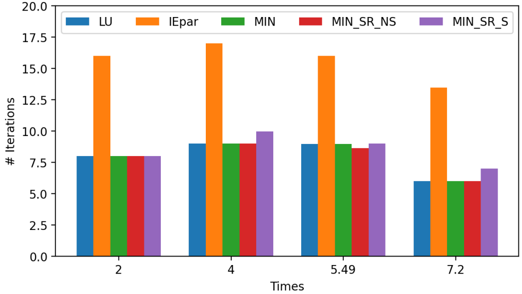


Figure: Left: Flow around the cylinder, DFG95 benchmark. Top: Number of iterations for different SDC preconditioners at selected time-steps. Smaller is better, blue is serial reference.

Parallelization across the method

Summary

Pros

- Pretty good parallel efficiency
- Simple to implement, simple to use, simple to analyze
- Can be easily combined with other parallelization strategies

Cons

- Parallelization depends on order of accuracy
- Small-scale parallelization only
- Nonlinear problems doable, but not straightforward

Parallelization across the steps I

Multigrid for the composite collocation problem

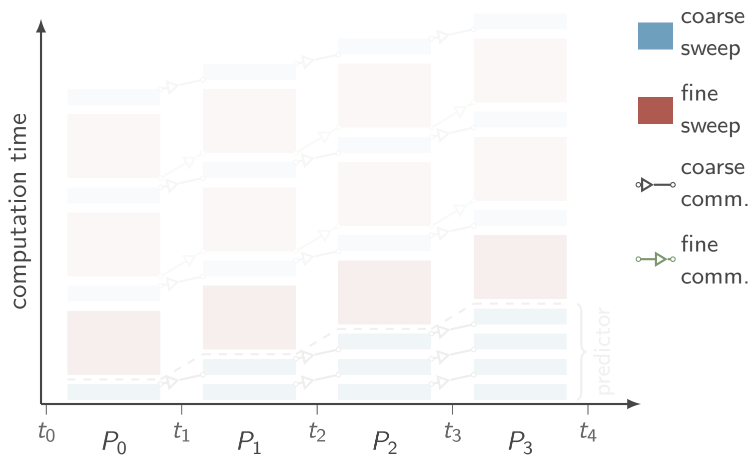
We now glue L time-steps together, using N to transfer information from step l to step $l+1$. We get the composite collocation problem:

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

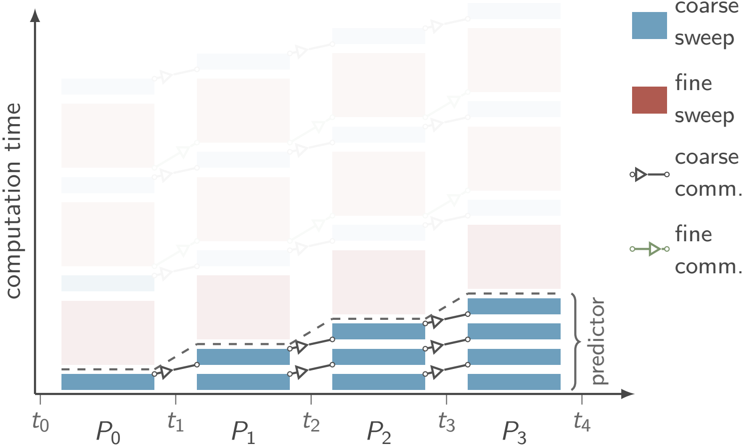
Parallel Full Approximation Scheme in Space and Time (PFASST, Minion and Emmett, 2012):

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

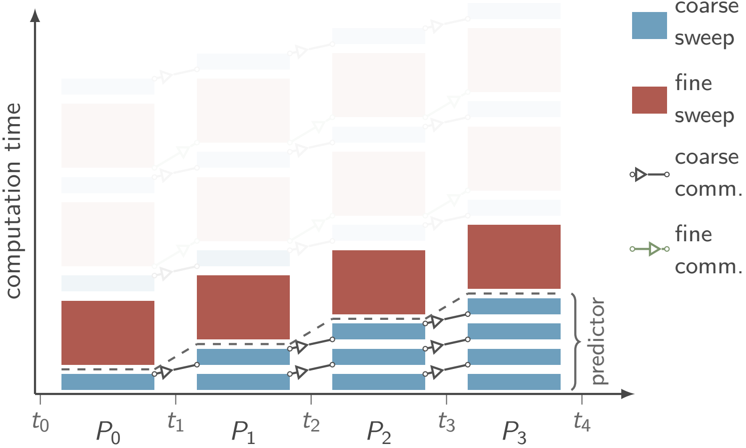
A quick visual introduction to PFASST



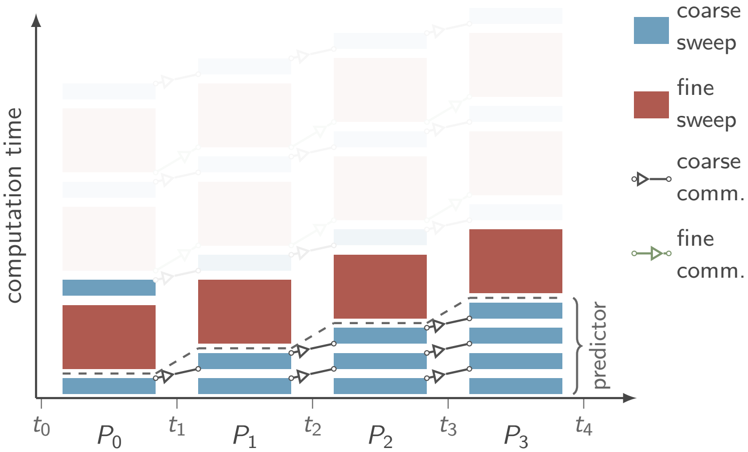
A quick visual introduction to PFASST



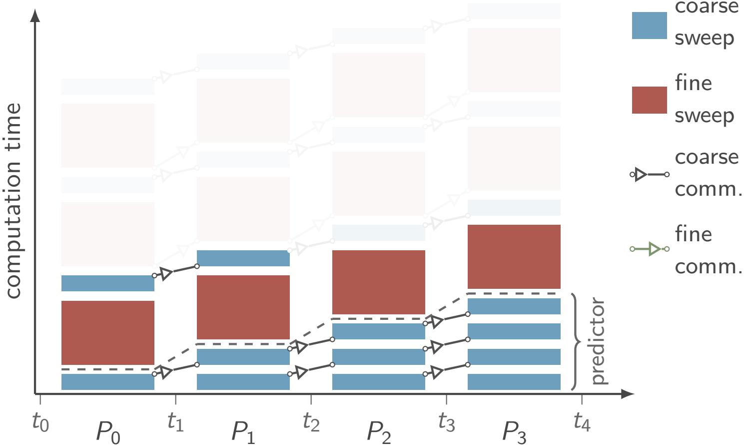
A quick visual introduction to PFASST



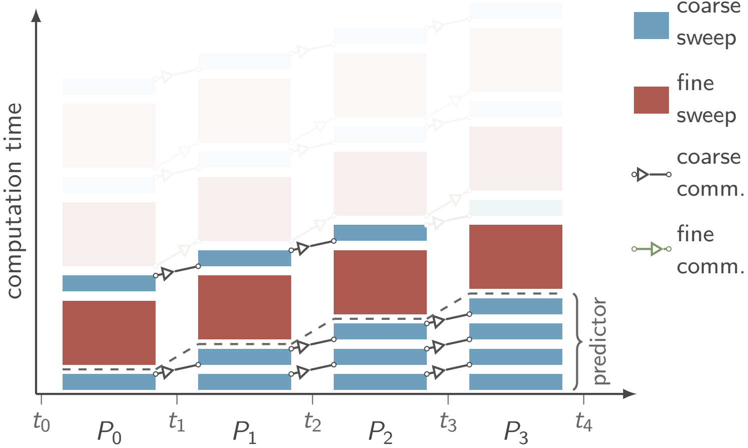
A quick visual introduction to PFASST



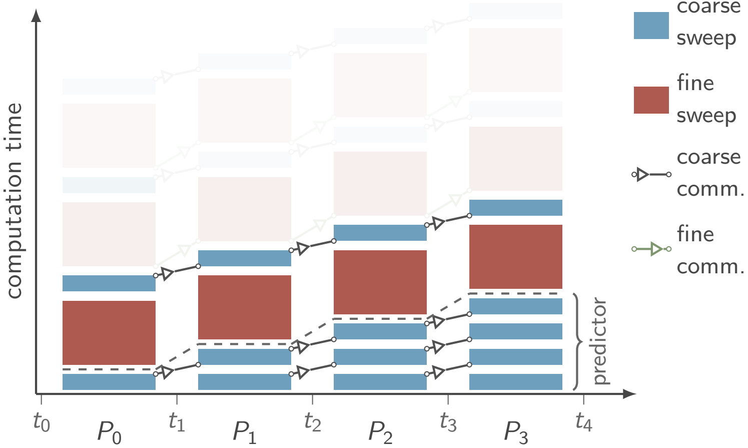
A quick visual introduction to PFASST



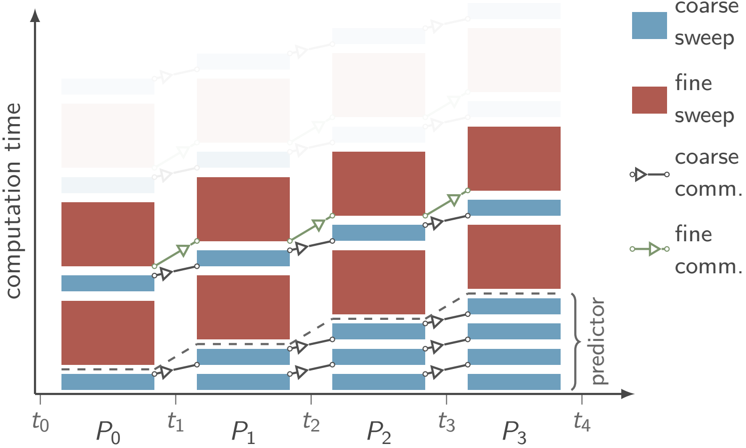
A quick visual introduction to PFASST



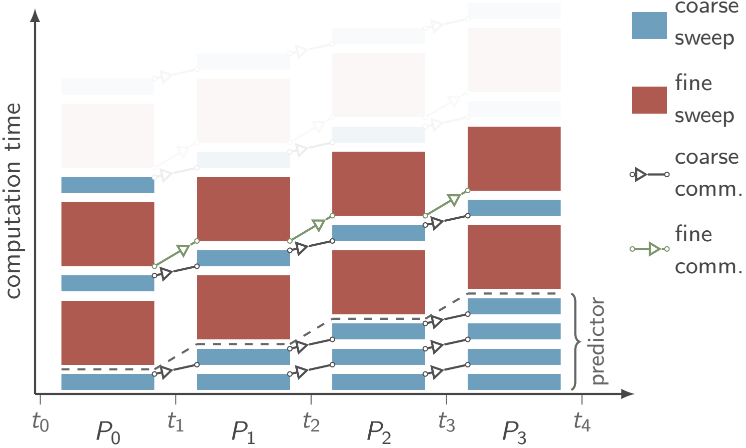
A quick visual introduction to PFASST



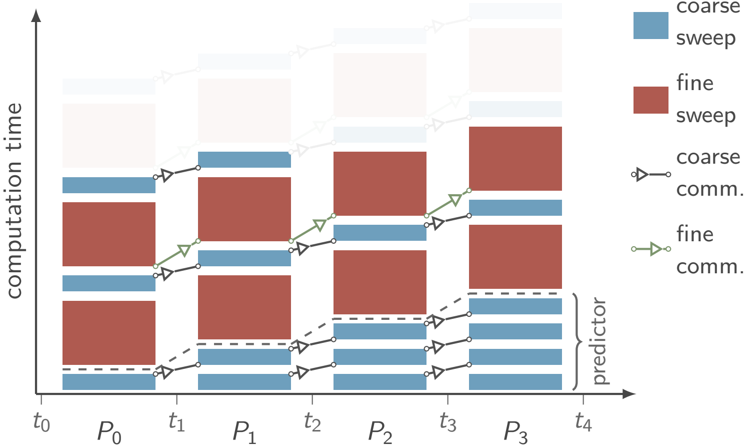
A quick visual introduction to PFASST



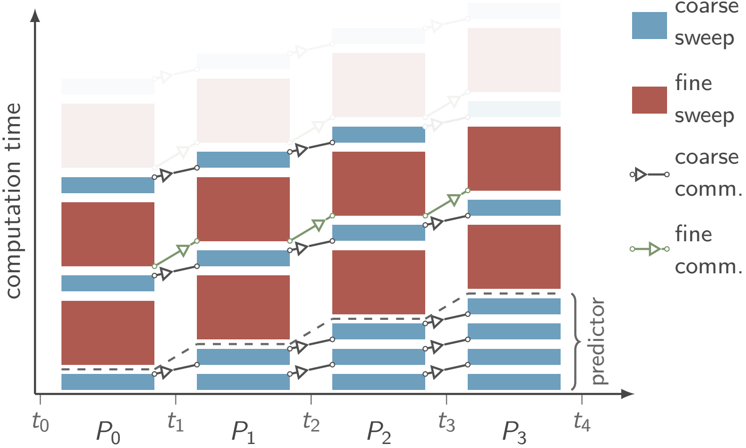
A quick visual introduction to PFASST



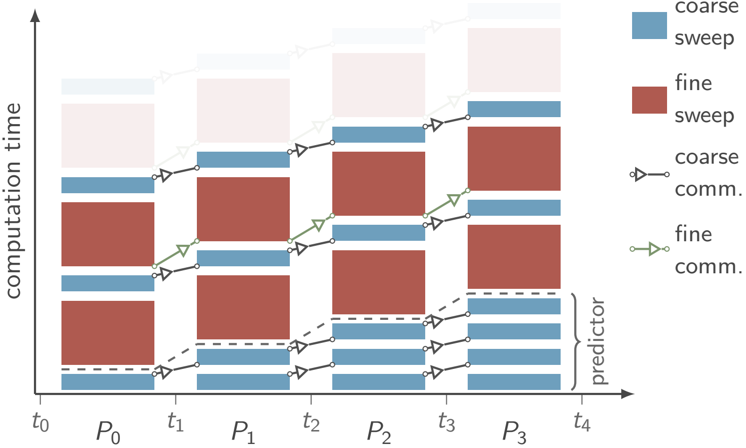
A quick visual introduction to PFASST



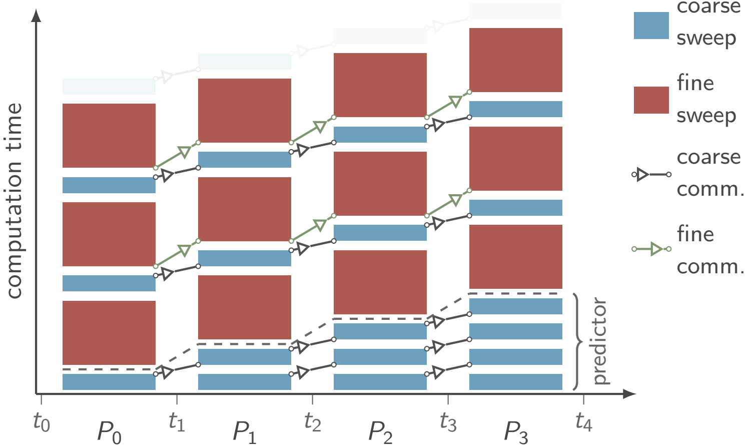
A quick visual introduction to PFASST



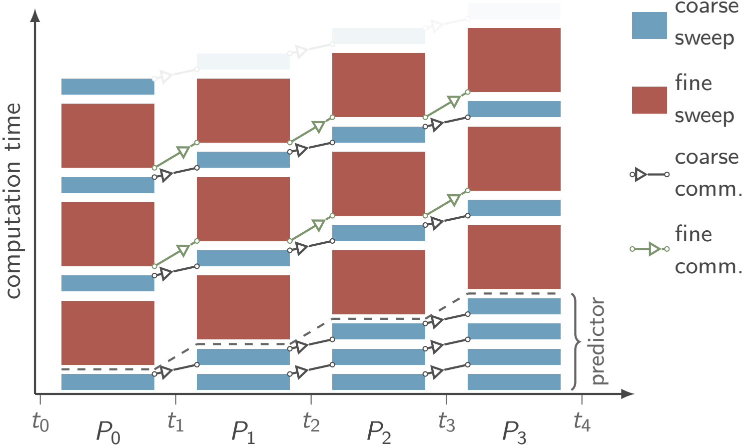
A quick visual introduction to PFASST



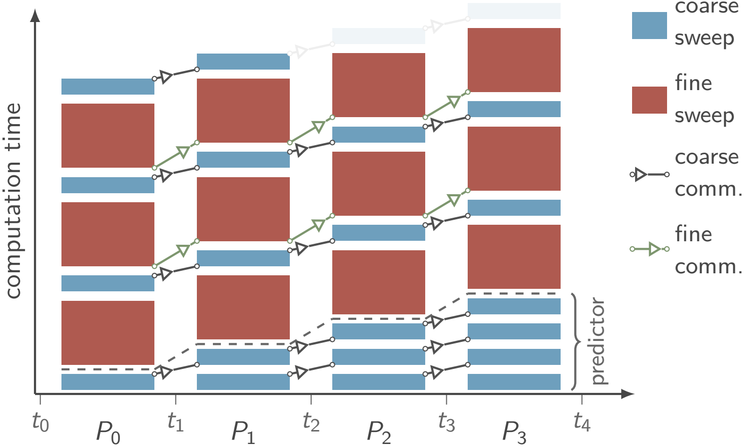
A quick visual introduction to PFASST



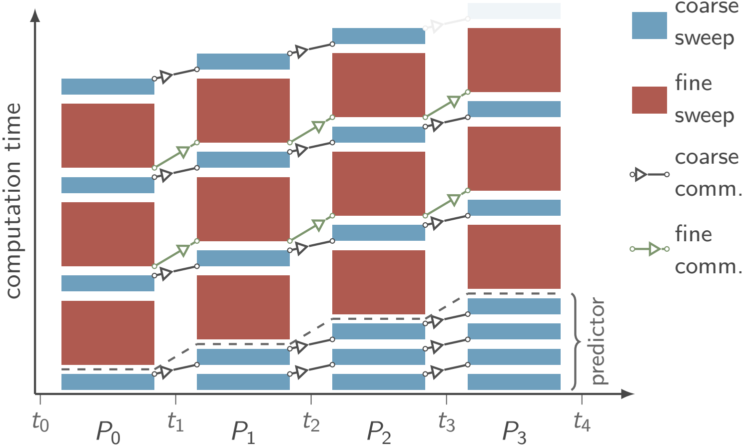
A quick visual introduction to PFASST



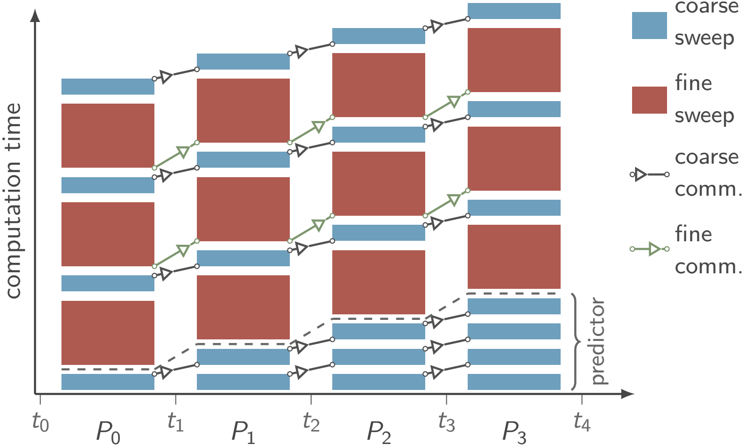
A quick visual introduction to PFASST



A quick visual introduction to PFASST



A quick visual introduction to PFASST



Coarsening in space and time

Space-time multilevel techniques, with Daniel Ruprecht and Michael Minion

Key to optimal efficiency: ratio between coarse and fine sweep

- coarsening strategies:
  1. reduction of temporal SDC nodes
  2. reduction of degrees-of-freedom in space
  3. reduced order in spatial discretization
  4. reduced implicit solve (if implicit integrator used)
  5. reduced physical representation
- precise balancing between aggressive coarsening and additional iterations crucial
- application-tailored coarsening in space and time required

Coarsening in space and time

Space-time multilevel techniques, with Daniel Ruprecht and Michael Minion

Key to optimal efficiency: ratio between coarse and fine sweep

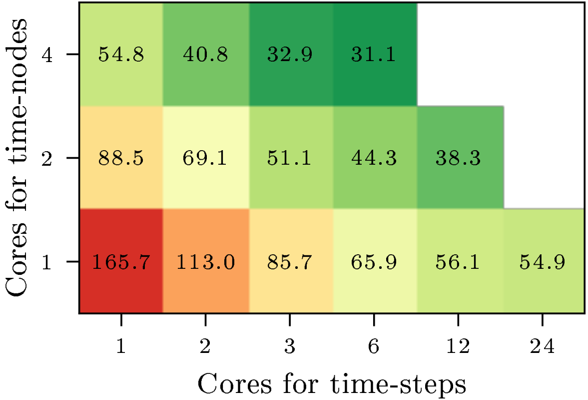
- coarsening strategies:
  1. reduction of temporal SDC nodes
  2. reduction of degrees-of-freedom in space
  3. reduced order in spatial discretization
  4. reduced implicit solve (if implicit integrator used)
  5. reduced physical representation
- precise balancing between aggressive coarsening and additional iterations crucial
- application-tailored coarsening in space and time required

One step further: PFASST-ER

PFASST + parallel SDC, with Ruth Schöbel

Idea: Use parallel SDC sweeps within parallel time-steps

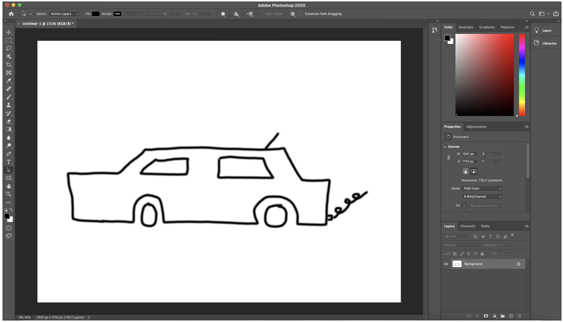
Example: 2D Allen-Cahn, fully-implicit, 256x256 DOFs in space, up to 24 available cores.



<commercial>

pySDC - Prototyping Spectral Deferred Corrections

Test before you invest at `https://parallel-in-time.org/pySDC`

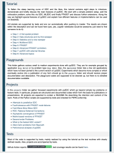


→

Some pySDC features

Tutorials and examples

- Ships with a lot of examples
- Many SDC flavors up to PFASST
- Problems beyond heat equation


Tutorial
Playgrounds
Projects
Tests

Parallel and serial

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

Python

- Interface compiled code for expensive spatial solves
- Implementation close to formulas

CI/CD/CT

- Well documented
- Well tested
- Works ~~on my machine~~ anywhere
- Reproduce paper results

Code separated into modules

Problem

- implicit Euler like solves
- evaluate right hand side
- initial conditions, maybe exact solution
- use your own datatype

Sweeper: Timestepping

- assembles and calls solves in problem class
- administers right hand side evaluations
- takes care of Q_Δ, splitting etc.
- DIRK methods available as sweepers

Callbacks: Modify anything at any time

- solution
- step size
- sweeper
- ...

Hooks: Extract anything at any time

- Newton / SDC iterations and f evaluations
- wall time
- error
- ...

</commercial>

Parallelization across the step with PFASST

Summary

Pros

- Can provide significant speedup over space-parallel codes
- Has been demonstrated to run on very large scales
- Code base is solid and (a bit) diverse
- Designed and works directly for nonlinear problems

Cons

- Theory is.. scarce
- Usage and implementation is usually a big obstacle (“non-non-intrusive”)
- Suitable coarsening strategies are not always easy to define
- Does not work well for hyperbolic problems

Back to the composite collocation problem

Let's go back to the problem PFASST is solving:

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

Compact notation:

$$(I - (I \otimes \Delta t Q)F - E \otimes N)(\vec{u}) = \vec{u_0}$$

Make it linear (for now):

$$(I - I \otimes \Delta t Q \otimes A - E \otimes N)\vec{u} = \vec{u_0}$$

Even more compactly written: $C\vec{u} = \vec{u_0}$

Parallelization through diagonalization

A ParaDIAG variant for the composite collocation problem, with Gayatri Caklovic

Preconditioned iteration:

$$\vec{u}^{k+1} = \vec{u}^k + P^{-1}(\vec{u}_0 - C\vec{u}^k)$$

If P^{-1} can be computed in a parallel way, then we have a PinT integrator.

Idea:

$$C = \begin{pmatrix} I - \Delta t Q \otimes A & & & \\ -N & I - \Delta t Q \otimes A & & \\ & \ddots & \ddots & \\ & & -N & I - \Delta t Q \otimes A \end{pmatrix}$$

Or, more compactly:

$$C_\alpha = I - I \otimes \Delta t Q \otimes A - E_\alpha \otimes N$$

Since E_α can be diagonalized, C_α can also be diagonalized and C_α^{-1} can be computed in parallel!

Parallelization through diagonalization

A ParaDIAG variant for the composite collocation problem, with Gayatri Caklovic

Preconditioned iteration:

$$\vec{u}^{k+1} = \vec{u}^k + P^{-1}(\vec{u}_0 - C\vec{u}^k)$$

If P^{-1} can be computed in a parallel way, then we have a PinT integrator.

Idea:

$$C_\alpha = \begin{pmatrix} I - \Delta t Q \otimes A & & & -\alpha N \\ -N & I - \Delta t Q \otimes A & & \\ & \ddots & \ddots & \\ & & -N & I - \Delta t Q \otimes A \end{pmatrix}$$

Or, more compactly:

$$C_\alpha = I - I \otimes \Delta t Q \otimes A - E_\alpha \otimes N$$

Since E_α can be diagonalized, C_α can also be diagonalized and C_α^{-1} can be computed in parallel!

Parallelization through diagonalization

A ParaDIAG variant for the composite collocation problem, with Gayatri Caklovic

Preconditioned iteration:

$$\vec{u}^{k+1} = \vec{u}^{k} + P^{-1}(\vec{u}_0 - C\vec{u}^{k})$$

If P^{-1} can be computed in a parallel way, then we have a PinT integrator.
Idea:

$$C_{\alpha} = \begin{pmatrix} I - \Delta t Q \otimes A & & & -\alpha N \\ -N & I - \Delta t Q \otimes A & & \\ & \ddots & \ddots & \\ & & -N & I - \Delta t Q \otimes A \end{pmatrix}$$

Or, more compactly:

$$C_{\alpha} = I - I \otimes \Delta t Q \otimes A - E_{\alpha} \otimes N$$

Since E_{α} can be diagonalized, C_{α} can also be diagonalized and C_{α}^{-1} can be computed in parallel!

Some subtleties

Looks great in theory, but it is not so easy..

1. What's α and how to choose it?
   - α small = rapid converence, but large diagonalization error
   - α large = small diagonalization error, but slow convergence

   Convergence analysis suggests adaptive strategy!

2. On each time-step we now have to solve

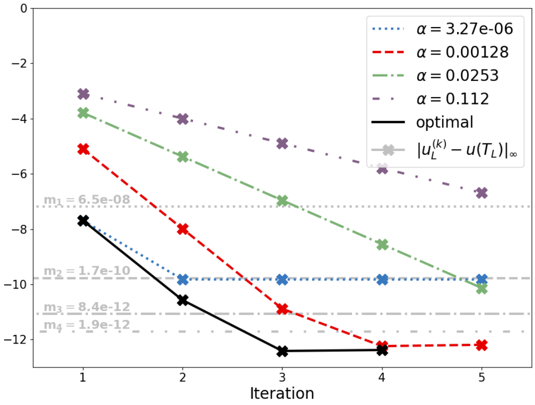
$$\big((d_l H + I) \otimes I - \Delta t Q \otimes A\big)\widetilde{u}_l = v_l, \quad d_l = -\alpha^{\frac{1}{L} e^{-2\pi i \frac{l-1}{L}}}, \quad 1 \leq l \leq L.$$

   Déjà vu: Diagonalize Q and solve parallel across the nodes!

3. How costly is the diagonalization?

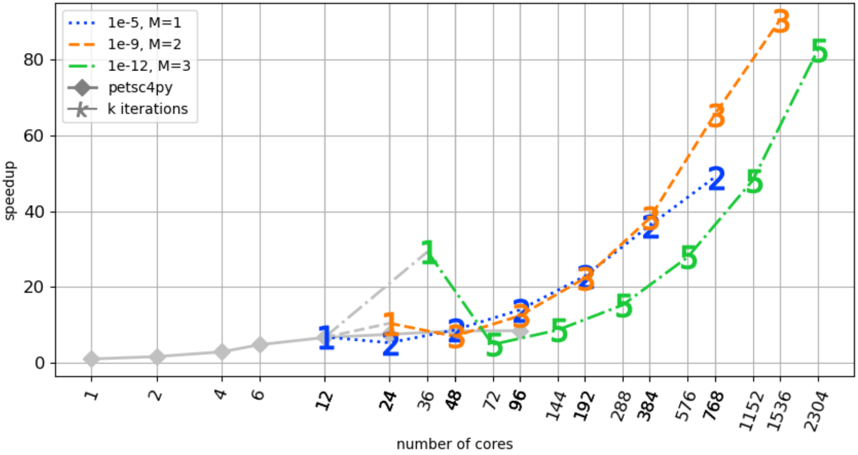
Adaptive α strategy

Advection equation in 2D, fixed α vs. adaptive α coming from the convergence analysis



Speedup

Advection equation in 2D, speedup for different accuracies and orders, space and 2x time



Parallelization across the steps with ParaDIAG

Is this IT?

Pros

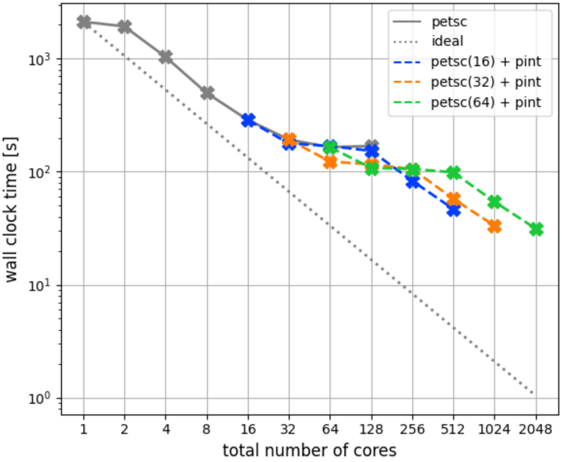
- Convergence speed is (often) very fast
- Can speed up low order/low accuracy and high order/high accuracy simulations
- Communication scheme is rather cheap (radix-2 butterfly in time)
- Once implemented, usage is simple (esp. no coarsening)
- Works well for hyperbolic problems

Cons

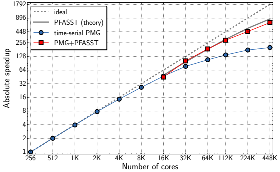
- Implementation and theory have a lot of hidden pitfalls
- Choice of α is somewhat fragile
- And: nonlinear problems are much more difficult and less efficient

Speedup for a nonlinear problem

Boltzmann equation in 3D, IMEX splitting, speedup in space and 2x time

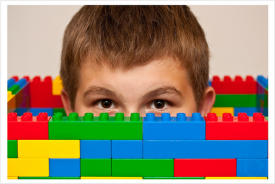


Three takeaways



Parallel-in-Time integration (PinT) can help to extend prevailing scaling limits

Collocation methods provide a fruitful and extensive playground for all sorts of parallelization strategies



Prototyping ideas, with real code, on real parallel machines, is crucial to find out about potential and limitations

The PinT Community

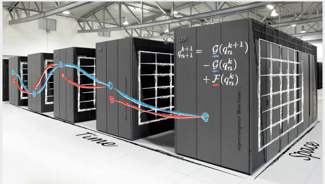
To learn more about PinT check out the website

www.parallel-in-time.org

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

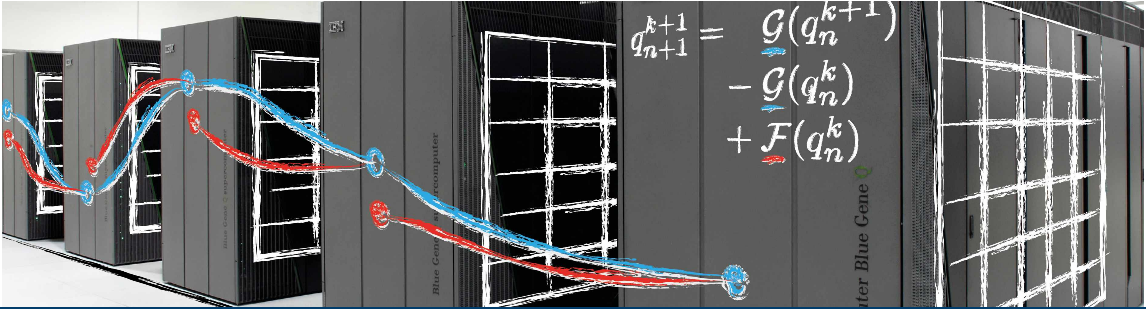
14th Workshop on Parallel-in-Time Integration

- July 7-12, 2025
- Edinburgh, UK
- organized by Jemma Shipton and others



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

parallelintime+subscribe@googlegroups.com



Parallel-in-Time Collocation Methods

June 10, 2024 | Robert Speck & Ruth Partzsch | Jülich Supercomputing Centre