

PyPinT

Towards a framework for rapid prototyping of iterative parallel-in-time algorithms

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck | 3rd Workshop on Parallel-in-Time Integration

Overview

- 1 Recap of existing Parallel-in-Time Approaches
- 2 The *PyPinT* Framework Explained
- 3 Goals of *PyPinT*
- 4 Proof of Concept



PyPinT

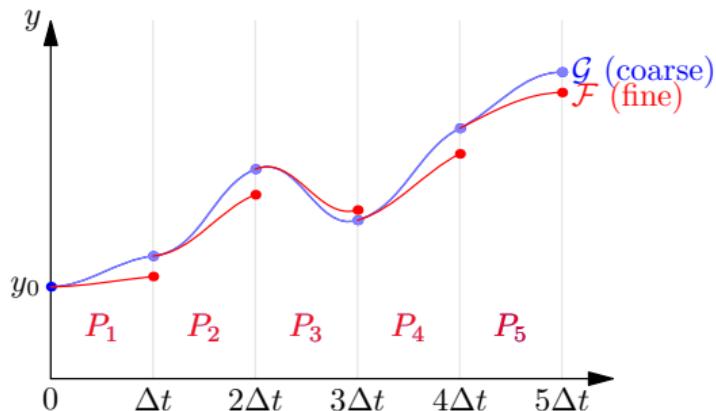
Part I: Existing Parallel-in-Time Approaches

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

Parareal

Y. Maday. 2008. *The Parareal in Time Algorithm.*

- coarse \mathcal{G} and fine \mathcal{F} propagators make parareal flexible and **modular**
- an initial value is improved **iteratively**
- order is controllable through the fine propagator \mathcal{F}



[Courtesy of M. Emmett, LBNL]

$$y_{m+1}^{k+1} = \mathcal{F}(y_m^k) + \mathcal{G}(y_m^{k+1}) - \mathcal{G}(y_m^k)$$

What is needed? **Two Integrators**

Deferred Corrections

V. Pereyra. 1966. *On Improving an Approximate Solution of a Functional Equation by Deferred Corrections.* Numerische Mathematik. 8(4) 376–391.

$$y'(t) = f(t, y(t)), \quad y(0) = y_0$$

Using the residual

$$r(t) = f(t, \tilde{y}(t)) - \tilde{y}'(t)$$

to compute the error

$$e'(t) = f(t, \tilde{y} + e) - f(t, \tilde{y}) + r'(t)$$

$$e(0) = 0$$

for the next update

$$\tilde{y}_{j+1} = \tilde{y}_j + e_j$$

Deferred Corrections

and

Integral Deferred Corrections

A. Dutt, L. Greengard, V. Rokhlin. 2000. *Spectral Deferred Correction Methods for Ordinary Differential Equations*. BIT Numerical Mathematics 40(2) 241–266.

$$y'(t) = f(t, y(t)), \quad y(0) = y_0$$

$$y(t) = y_0 + \int_0^t f(\tau, y(\tau)) d\tau, \quad y(0) = y_0$$

Using the residual

$$r(t) = f(t, \tilde{y}(t)) - \tilde{y}'(t)$$

to compute the error

$$\begin{aligned} e'(t) &= f(t, \tilde{y} + e) - f(t, \tilde{y}) + r'(t) \\ e(0) &= 0 \end{aligned}$$

for the next update

$$\tilde{y}_{j+1} = \tilde{y}_j + e_j$$

Using the residual

$$r(t) = y_0 - \tilde{y}(t) + \int_0^t f(\tau, y(\tau)) d\tau$$

to compute the error

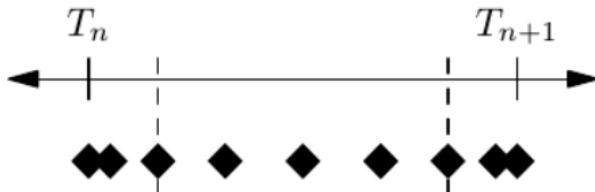
$$e(t) = \int_0^t f(\tau, \tilde{y} + e) - f(\tau, \tilde{y}) d\tau + r(t)$$

for the next update

$$\tilde{y}_{j+1} = \tilde{y}_j + e_j$$

Spectral Deferred Corrections

A. Dutt, L. Greengard, V. Rokhlin. 2000. *Spectral Deferred Correction Methods for Ordinary Differential Equations*. BIT Numerical Mathematics 40(2) 241–266.



One sweep is performed by

$$u_{n+1}^{k+1} = u_n^{k+1} + \Delta t [F(u_{n+1}^{k+1}) - F(u_{n+1}^k)] + \mathcal{I}_n^{n+1}(F(\mathbf{u}^k))$$

new main constituent is the quadrature operator

$$\mathcal{I}_n^{n+1}\mathbf{y} = \sum_{i=1}^m \omega_i \tilde{y}_i \approx \int_{t_n}^{t_{n+1}} y(\tau) d\tau$$

What is needed? Quadrature, Evaluation of F (, Space Solver) → new Integrator

Multilevel SDC

R. Speck, D. Ruprecht, M. Emmett, M. L. Minion, M. Bolten, R. Krause. 2013. *A Multi-Level Spectral Deferred Correction Method*. arXiv Math Numerical Analysis.

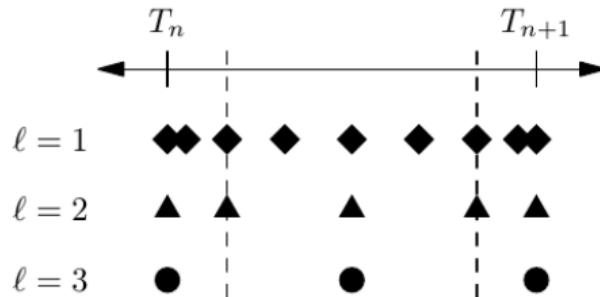
Slightly different sweep

$$u_{n+1}^{k+1} = u_n^{k+1} + \Delta t \left[F(u_{n+1}^{k+1}) - F(u_{n+1}^k) \right] + \mathcal{I}_n^{n+1} \left(F(\mathbf{u}^k) \right) + \tau_n^k$$

and multiple level, correction τ_n^k based on information from different levels

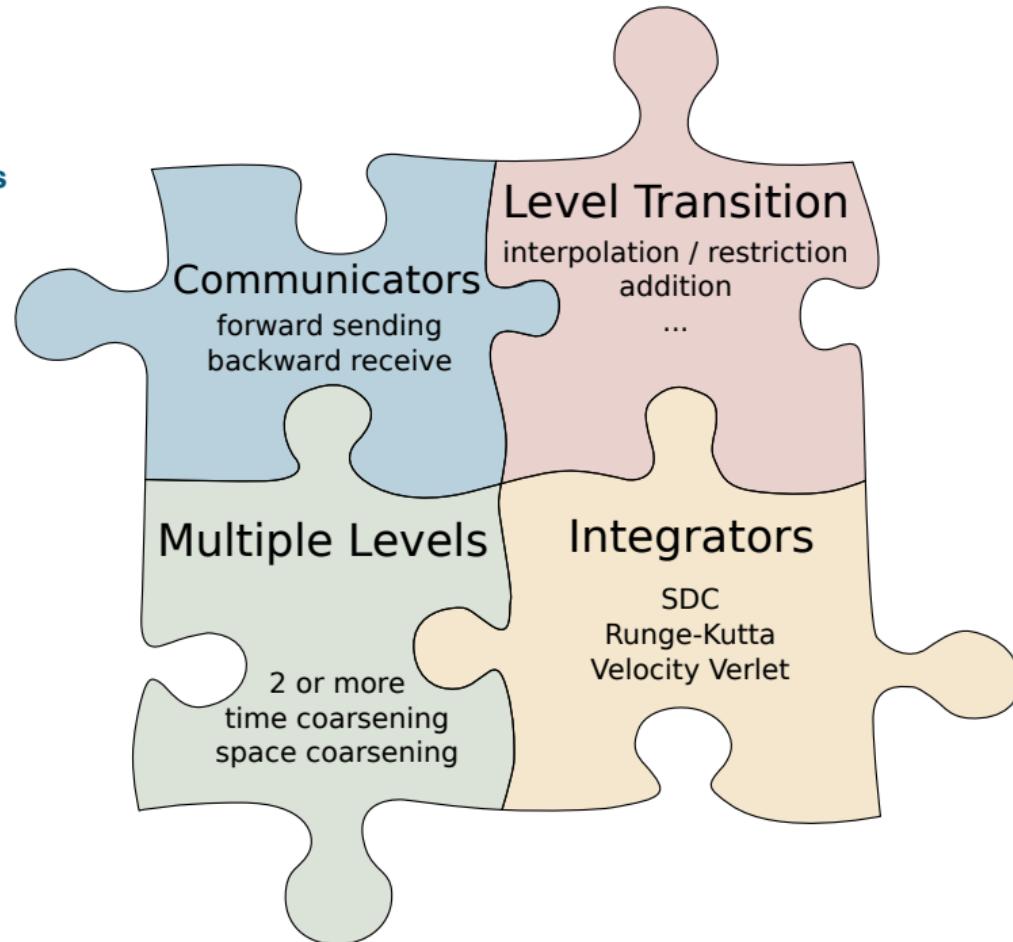
What is needed?

- same as SDC
- **Interpolation** (space and time)
- **Restriction** (space and time)



PFASST

Building Blocks





PyPinT

Part II: The PyPinT Framework Explained

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

Basic Concept

- *Python ≥3.2* as language of choice
 - for ease of use and extensibility (cf. *NumPy*, *SciPy*)
- modular building blocks
 - for fast exchange of algorithms' building blocks
- well-conceived and intuitive abstract interfaces
 - for reusable code ensuring DRY principle
- integrated analyzation tools
 - for introspection and plotting (cf. *matplotlib*)
- usage of a sophisticated unit-testing framework
 - nobody writes bug-free code

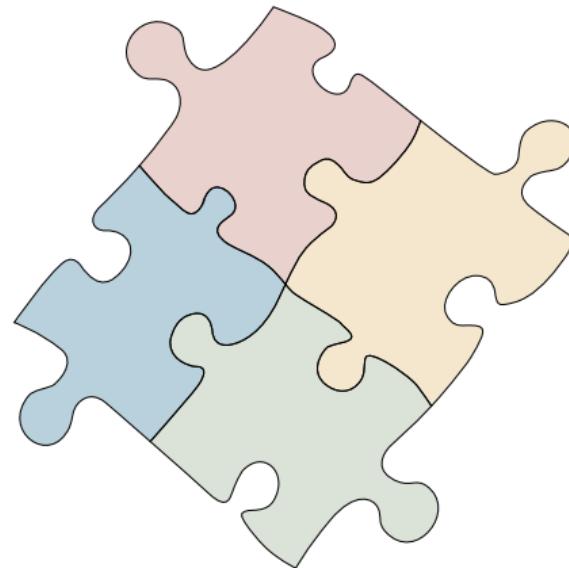


Modules

Abstract Modelling of PinT Algorithms

pypint

- ~.problems
- ~.solvers
- ~.integrators
- ~.communicators
- ~.multi_level_providers
- ~.solutions
- ~.plugins

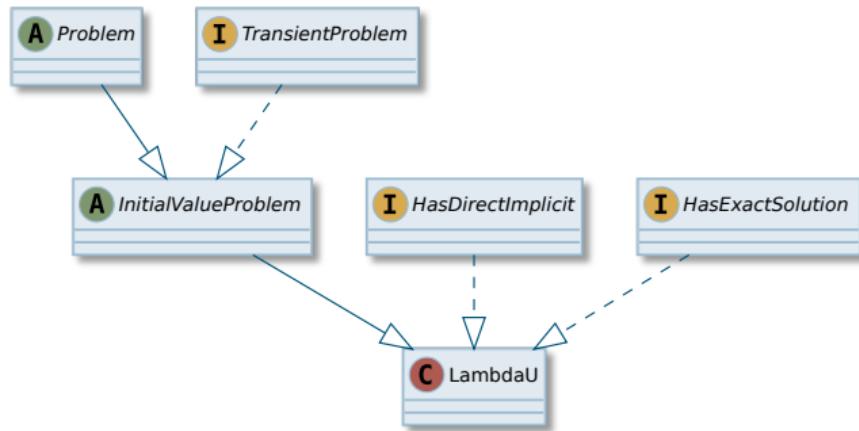


Modules

Abstract Modelling of PinT Algorithms

```
pypint
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- interfaces for problem setups
- generic problem with specializations via mixins



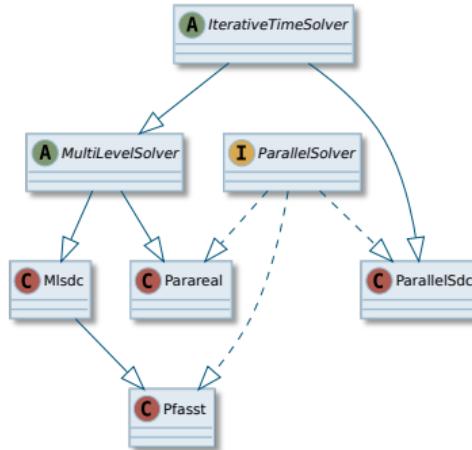
Modules

Abstract Modelling of PinT Algorithms

pypint

```
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- interfaces for iterative time solvers
- providing generic building blocks of solvers



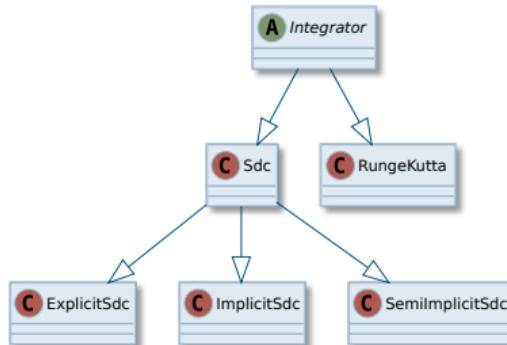
Modules

Abstract Modelling of PinT Algorithms

pypint

```
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- abstract interface for integrators of ODEs
e.g. *Runge-Kutta*, *Expl./Impl. Euler*, *SDC* etc.
- unified generic interface for executing integrator
via method `.apply(*args, **kwargs)`



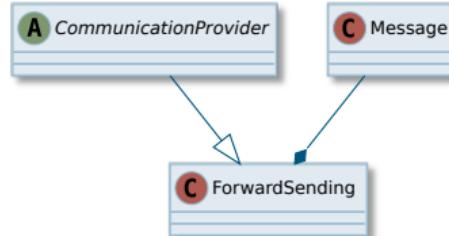
Modules

Abstract Modelling of PinT Algorithms

pypint

```
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- generic abstract interfaces for communication patterns
e.g. implemented as *forward sending*, etc.
- extendable basic message buffer
holding data, solver flags and other meta information

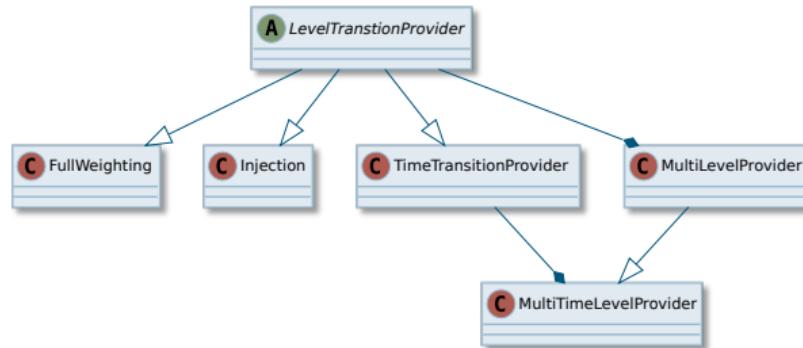


Modules

Abstract Modelling of PinT Algorithms

```
pypint
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- abstract interface for generic level transitions
e.g. *full weighting, injection, Lagrange-polynomial integration* etc.
- containers for multiple levels
providing access to integrators of each level
- unified generic interface for transitions between levels
via methods `.restrict(data, fine_id, coarse_id)` and
`.prolongate(data, coarse_id, fine_id)`



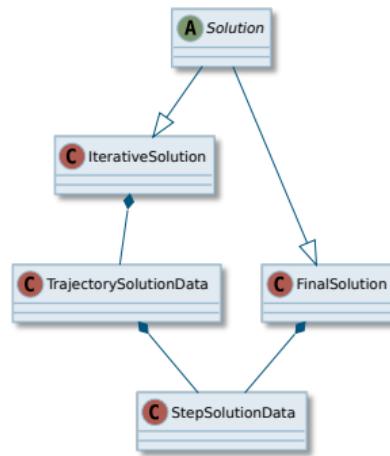
Modules

Abstract Modelling of PinT Algorithms

pypint

```
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- containers for solution data + meta information
 - e.g. time-node/step-wise, trajectory (consecutive time nodes)
- container interface for complete solutions
 - e.g. only last time node of last iteration or all nodes of all iterations



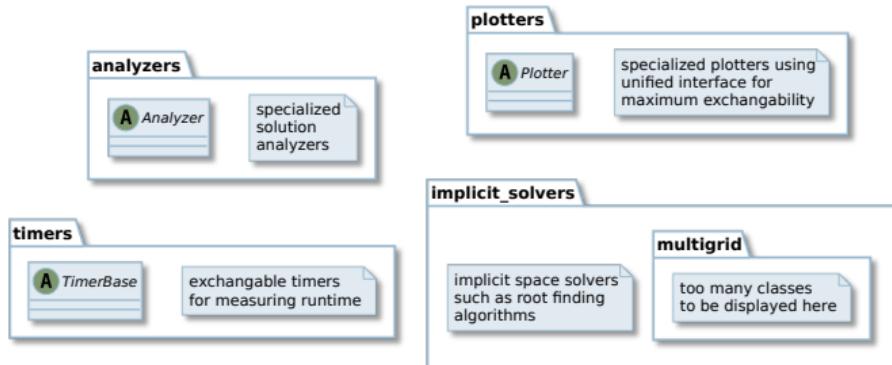
Modules

Abstract Modelling of PinT Algorithms

pypint

- ~.problems
- ~.solvers
- ~.integrators
- ~.communicators
- ~.multi_level_providers
- ~.solutions
- ~.plugins

- containers for solution data + meta information
 - e.g. time-node/step-wise, trajectory (consecutive time nodes)
- container interface for complete solutions
 - e.g. only last time node of last iteration or all nodes of all iterations





PyPinT

Part III: Goals for PyPinT

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

Goals for PyPinT

- providing generic framework for many PinT algorithms

Goals for PyPinT

- providing generic framework for many PinT algorithms
⇒ easy comparability of different algorithms' building blocks

Goals for PyPinT

- providing generic framework for many PinT algorithms
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces

Goals for PyPinT

- providing generic framework for many PinT algorithms
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces
⇒ fast exchangability and prototyping of different building blocks

Goals for PyPinT

- providing generic framework for many PinT algorithms
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces
⇒ fast exchangability and prototyping of different building blocks
- well-tested and well-documented implementations of PinT algorithms

Goals for PyPinT

- providing generic framework for many PinT algorithms
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces
⇒ fast exchangability and prototyping of different building blocks
- well-tested and well-documented implementations of PinT algorithms
⇒ suited for educational use

Goals for PyPinT

- providing generic framework for many PinT algorithms
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces
⇒ fast exchangability and prototyping of different building blocks
- well-tested and well-documented implementations of PinT algorithms
⇒ suited for educational use
- open-source hosted on  GitHub

Goals for PyPinT

- providing generic framework for many PinT algorithms
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces
⇒ fast exchangability and prototyping of different building blocks
- well-tested and well-documented implementations of PinT algorithms
⇒ suited for educational use
- open-source hosted on  GitHub
⇒ reaching young & rising developers and scientists promoting PinT algorithms



PyPinT

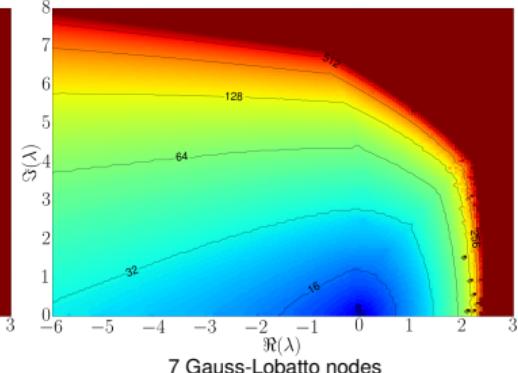
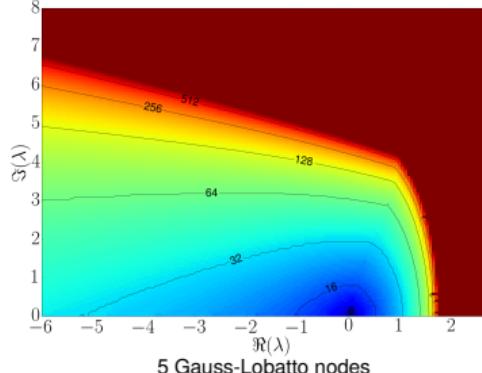
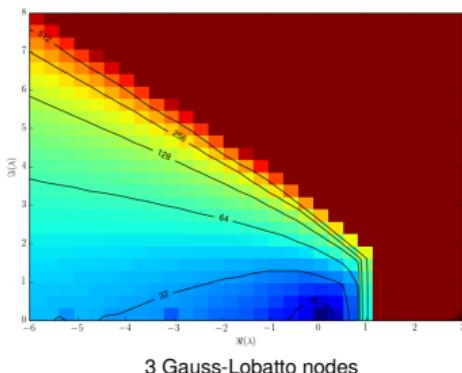
Part IV: Proof of Concept

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

Implementation of SDC and MLSDC

Convergence Regions (number iterations for residual $\leq 10^{-14}$) of $u'(x, t) = \lambda u(x, t)$, $t \in [0, 1]$, $\lambda \in \mathbb{C}$

SDC

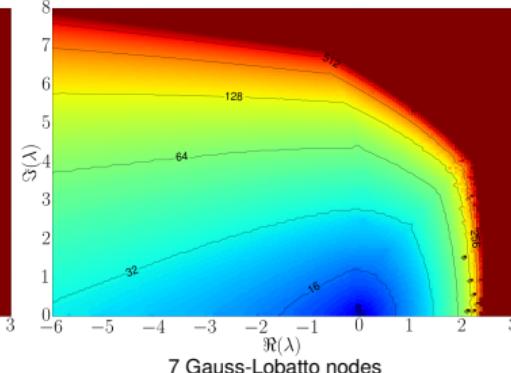
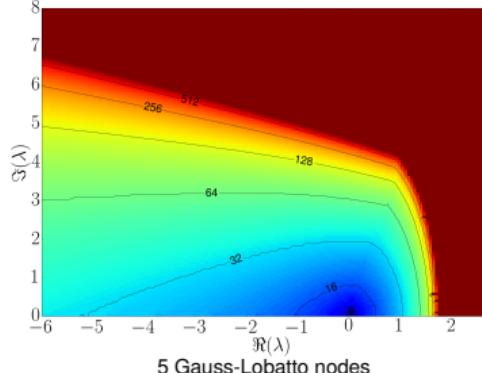
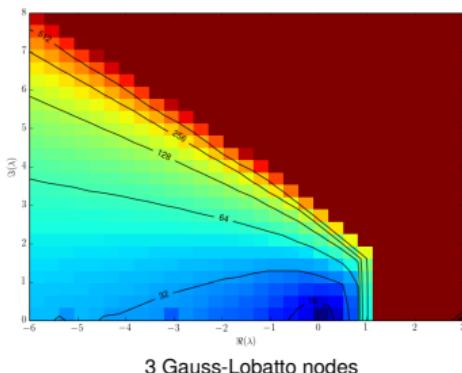


MLSDC

Implementation of SDC and MLSDC

Convergence Regions (number iterations for residual $\leq 10^{-14}$) of $u'(x, t) = \lambda u(x, t)$, $t \in [0, 1]$, $\lambda \in \mathbb{C}$

SDC



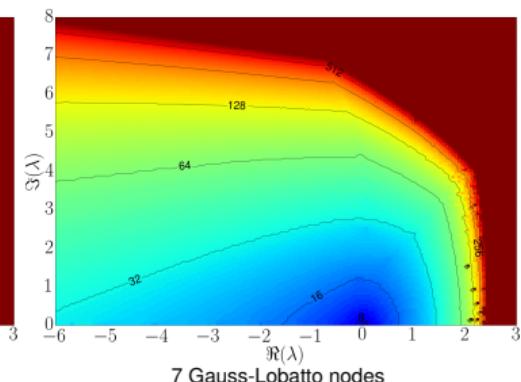
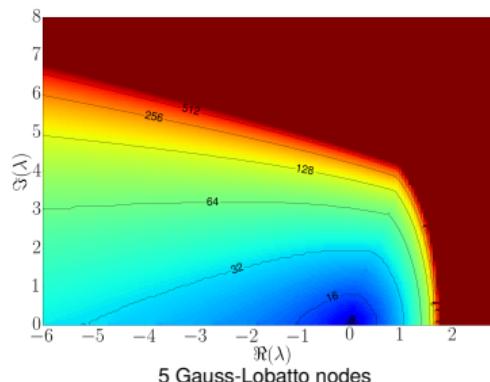
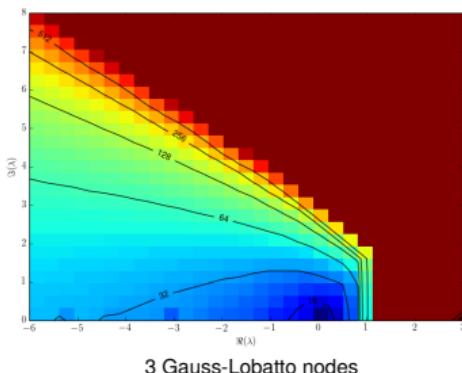
MLSDC



Implementation of SDC and MLSDC

Convergence Regions (number iterations for residual $\leq 10^{-14}$) of $u'(x, t) = \lambda u(x, t)$, $t \in [0, 1]$, $\lambda \in \mathbb{C}$

SDC



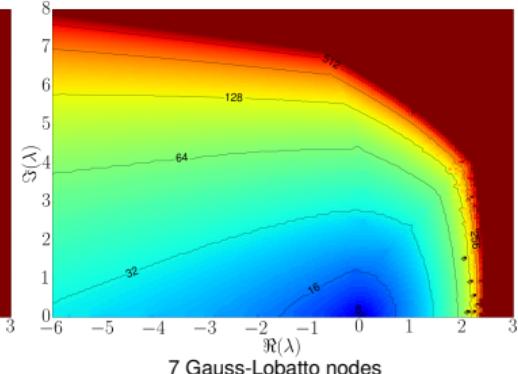
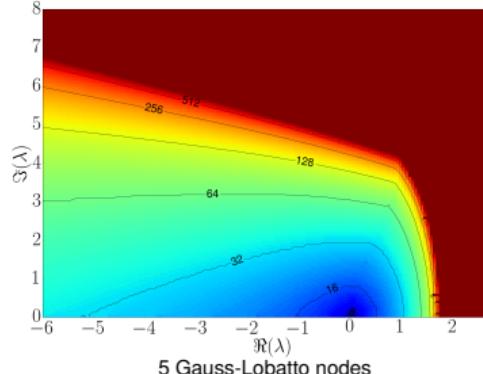
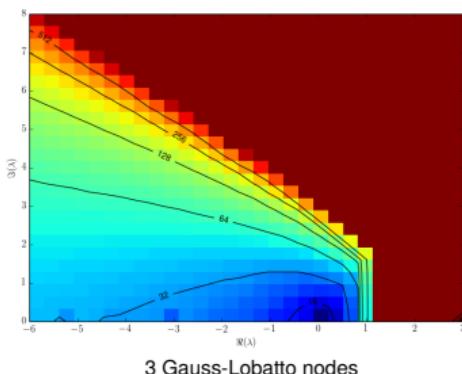
MLSDC



Implementation of SDC and MLSDC

Convergence Regions (number iterations for residual $\leq 10^{-14}$) of $u'(x, t) = \lambda u(x, t)$, $t \in [0, 1]$, $\lambda \in \mathbb{C}$

SDC



MLSDC



Implementation of SDC and MLSDC

Python Script for a SDC run with 3 Gauss-Lobatto nodes

```
1 prob = LambdaU(complex(-1.0, 1.0))
2 comm = ForwardSendingMessaging()
3 solver = Sdc(communicator=comm)
4 solver.init(problem=prob)
5 quadr = QuadratureBase(nodes=GaussLobatto(num_nodes=3),
6                           weights=Polynomial(coeffs=[1]))
7 integr = SemiImplicitSdc(quadrature=quadr)
8 solution = solver.run(integrator=integr)
9 # plotting via matplotlib.pyplot
```

Advantages of *PyPinT*

“towards” ...

- playground for rapid prototyping of PinT algorithms
 - no need to implement e.g. matrix inversion or quadrature on your own
⇒ saves your precious time
- use of Python's full feature set
 - easy parallelism
via Python's own (e.g. `multiprocessing`) or 3rd-party modules (e.g. `mpi4py`)
 - built-in portability
runs on Unix and MacOS (should run on Windows too)
 - interactive computing
integrates well with IPython

IP[y]: IPython
Interactive Computing



Thank you for your attention!

Questions?

(now or later)

PyPinT is on  GitHub: <https://github.com/Parallel-in-Time/PyPinT>

Torbjörn Klatt
Juelich Supercomputing Centre
Building 16.3, Room 022
Tel: +49 2461 61 96452
eMail: t.klatt@fz-juelich.de

Dieter Moser
Juelich Supercomputing Centre
Building 16.3, Room 022
Tel: +49 2461 61 96453
eMail: d.moser@fz-juelich.de

Robert Speck *
Juelich Supercomputing Centre
Building 16.3, Room 131
Tel: +49 2461 61 1644
eMail: r.speck@fz-juelich.de

* corresponding author

The Aviles-Giga Problem

An example from *real* life

$$u_t = \frac{1}{\varepsilon} \underbrace{\nabla \cdot \left(\left(|\nabla u|^2 - 1 \right) \nabla u \right)}_{\mathcal{N}} - \varepsilon \underbrace{\nabla^4 u}_{\mathcal{L}}$$

The Aviles-Giga Problem

An example from *real* life

$$u_t = \frac{1}{\varepsilon} \underbrace{\nabla \cdot \left(\left(|\nabla u|^2 - 1 \right) \nabla u \right)}_{\mathcal{N}} - \varepsilon \underbrace{\nabla^4 u}_{\mathcal{L}}$$

- very stiff linear part \mathcal{L}
- highly nonlinear part \mathcal{N}
- computed on a periodic domain $\mathbb{T} = (0, 2\pi)^2$

The Aviles-Giga Problem

An example from *real* life

$$u_t = \frac{1}{\varepsilon} \underbrace{\nabla \cdot \left(\left(|\nabla u|^2 - 1 \right) \nabla u \right)}_{\mathcal{N}} - \varepsilon \underbrace{\nabla^4 u}_{\mathcal{L}}$$

- very stiff linear part \mathcal{L}
- highly nonlinear part \mathcal{N}
- computed on a periodic domain $\mathbb{T} = (0, 2\pi)^2$

But what is it good for?

The Aviles-Giga Problem

An example from *real* life

$$u_t = \frac{1}{\varepsilon} \underbrace{\nabla \cdot \left(\left(|\nabla u|^2 - 1 \right) \nabla u \right)}_{\mathcal{N}} - \varepsilon \underbrace{\nabla^4 u}_{\mathcal{L}}$$

- very stiff linear part \mathcal{L}
- highly nonlinear part \mathcal{N}
- computed on a periodic domain $\mathbb{T} = (0, 2\pi)^2$

But what is it good for?

It is a simple model for Pattern Forming Mechanisms in magnetic bulk