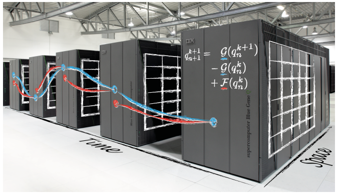
PyPinT

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


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

PyPinT

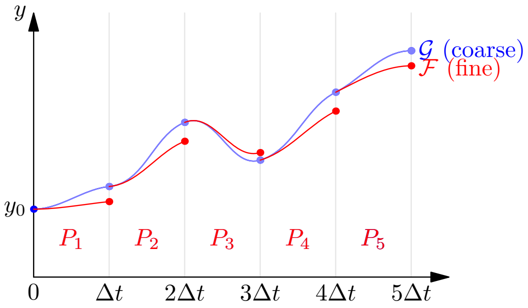
Part I: Existing Parallel-in-Time Approaches

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.

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

Using the residual

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

to compute the error

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

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.

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

Using the residual

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

to compute the error

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

for the next update

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

$$y(t) = y_0 + \int_0^t f\left(\tau, y\left(\tau\right)\right) \mathrm{d}\tau, \quad y\left(0\right) = y_0$$

Using the residual

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

to compute the error

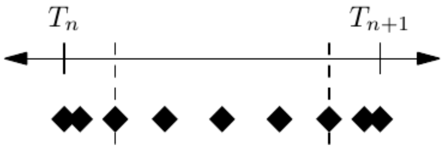
$$e(t) = \int_0^t f(\tau, \tilde{y} + e) - f\left(\tau, \tilde{y}\right) \mathrm{d}\tau + r\left(t\right)$$

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

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

new main constituent is the quadrature operator

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

What is needed? Quadrature, Evaluation of F (, Space Solver) $\longrightarrow$ 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.

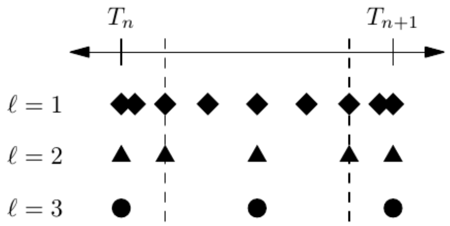
Slightly different sweep

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

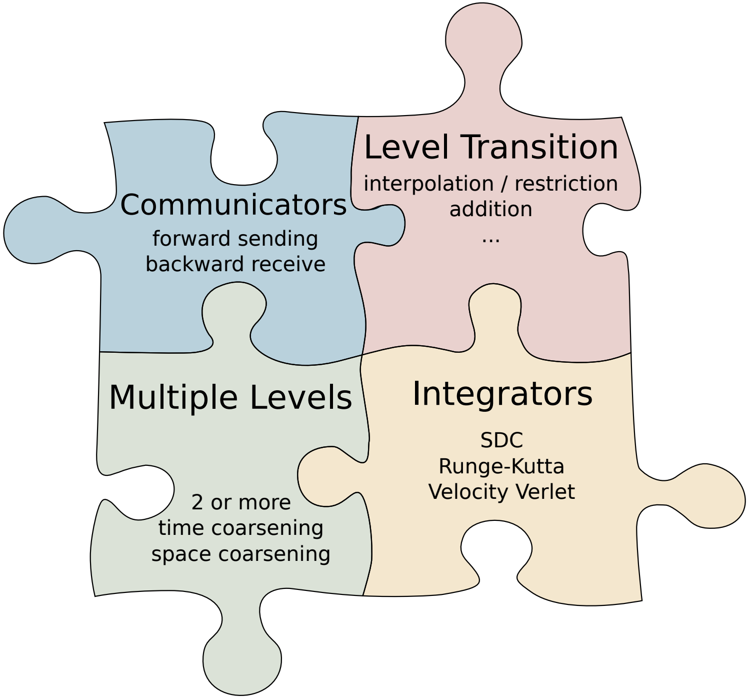
and multiple level, correction τ_n^k based on information from different levels

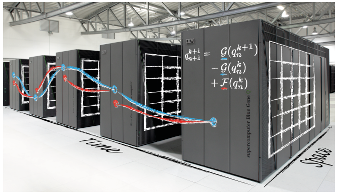
What is needed?

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



PFASST
Building Blocks




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

PyPinT

Part II: The PyPinT Framework Explained

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

Basic Concept

- *Python* $\geq$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

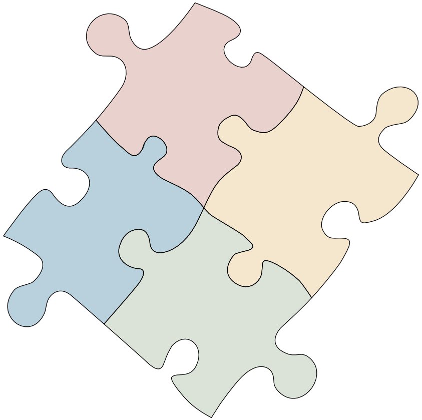
  for introspection and plotting (cf. *matplotlib*)
- usage of a sophisticated unit-testing framework

  nobody writes bug-free code

Modules

Abstract Modelling of PinT Algorithms

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

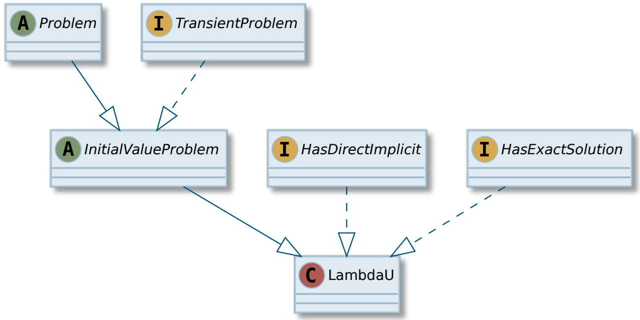


Modules

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

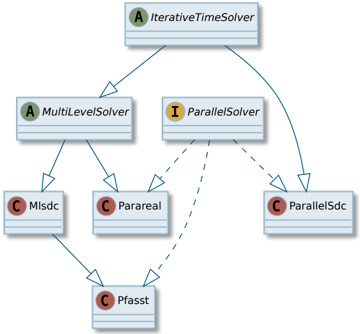


Modules

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

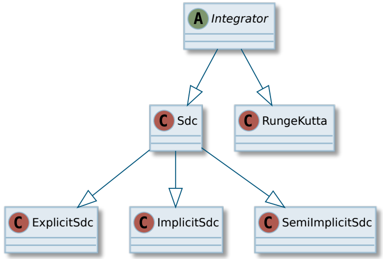


Modules

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)`

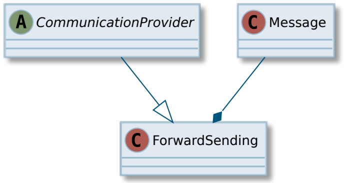


Modules

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

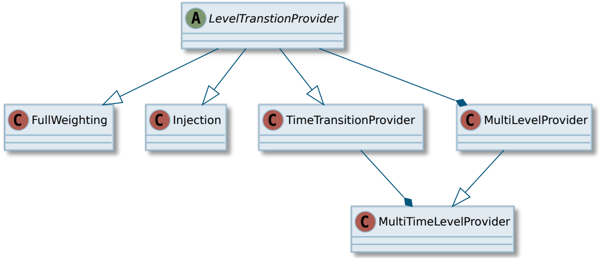


Modules

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)`

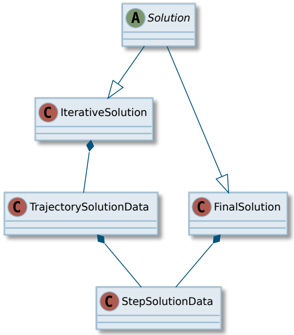


Modules

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

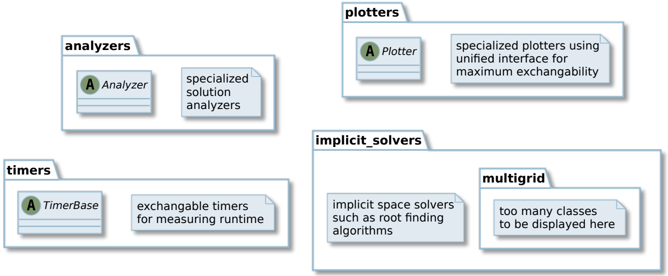


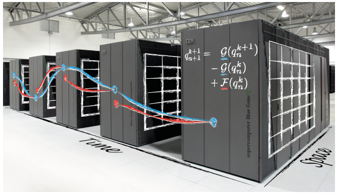
Modules

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


analyzers
A Analyzer
specialized solution analyzers
plotters
A Plotter
specialized plotters using unified interface for maximum exchangability
timers
A TimerBase
exchangable timers for measuring runtime
implicit_solvers
implicit space solvers such as root finding algorithms
multigrid
too many classes to be displayed here


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

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
  $\Rightarrow$ easy comparability of different algorithms' building blocks

Goals for PyPinT

- providing generic framework for many PinT algorithms
  $\Rightarrow$ easy comparability of different algorithms' building blocks

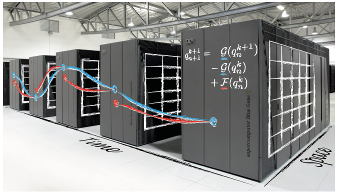
- intuitive user-expandability through well-defined interfaces
  $\Rightarrow$ fast exchangability and prototyping of different building blocks

Goals for PyPinT

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

Goals for PyPinT

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


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

PyPinT

Part IV: Proof of Concept

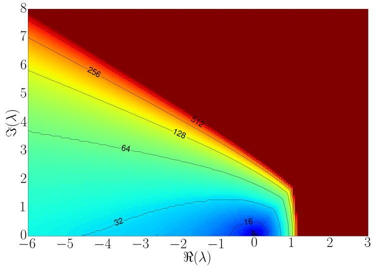
May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

Implementation of SDC and MLSDC

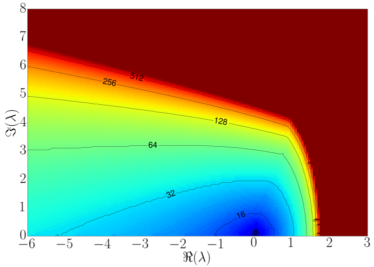
Convergence Regions (number iterations for residual $\leq 10^{-14}$) of $u'(x,t) = \lambda u(x,t)$, $t \in [0,1]$, $\lambda \in \mathbb{C}$

SDC

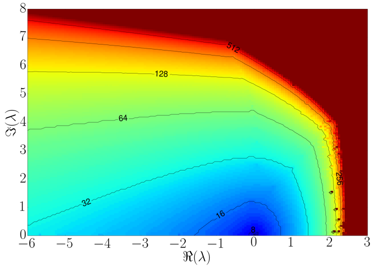
3 Gauss-Lobatto nodes



5 Gauss-Lobatto nodes



7 Gauss-Lobatto nodes

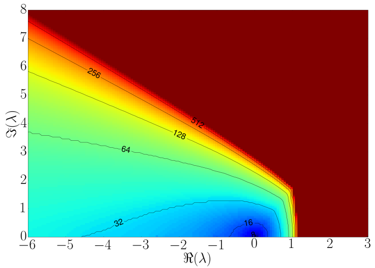


Implementation of SDC and MLSDC

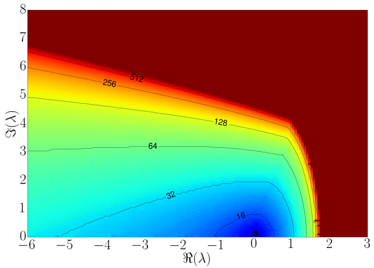
Convergence Regions (number iterations for residual $\leq 10^{-14}$) of $u'(x,t) = \lambda u(x,t)$, $t \in [0,1]$, $\lambda \in \mathbb{C}$

SDC

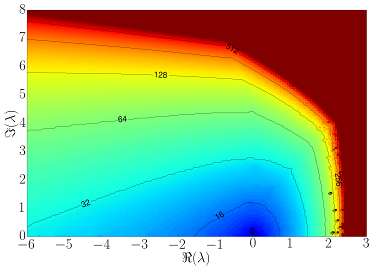
3 Gauss-Lobatto nodes



5 Gauss-Lobatto nodes



7 Gauss-Lobatto nodes

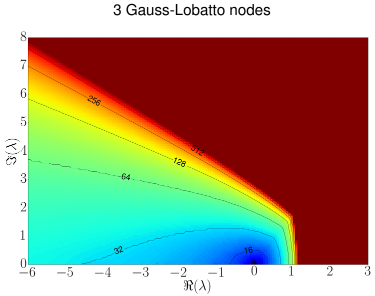


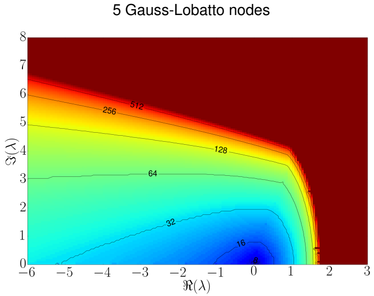
MLSDC

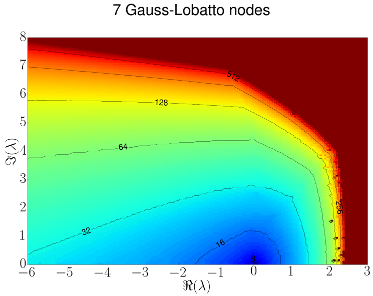
Implementation of SDC and MLSDC

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





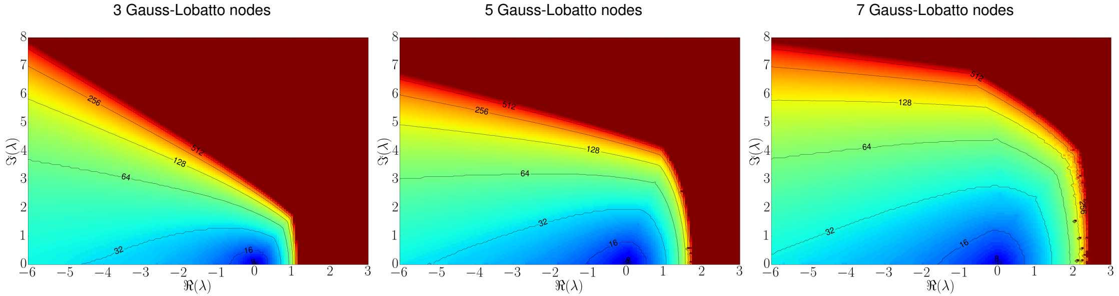


MLSDC

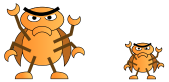
Implementation of SDC and MLSDC

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



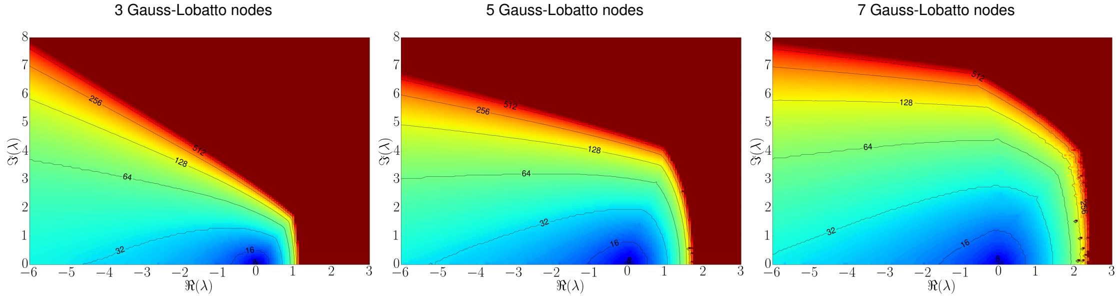
MLSDC



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

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

The Aviles-Giga Problem

An example from *real* life . . .

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

- 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 . . .

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

- 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 . . .

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

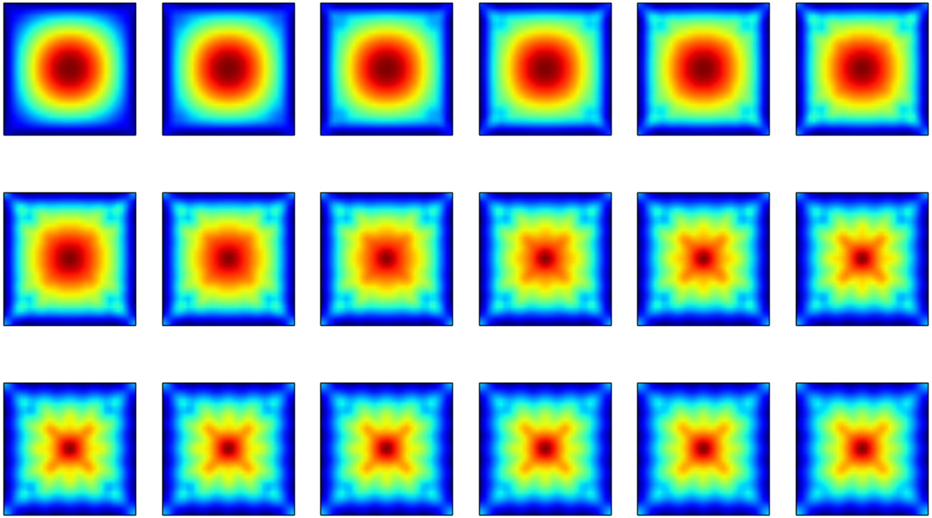
- highly nonlinear part $\mathcal{N}$
- computed on a periodic domain $\mathbb{T} = (0, 2\pi)^2$

But what is it good for?

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

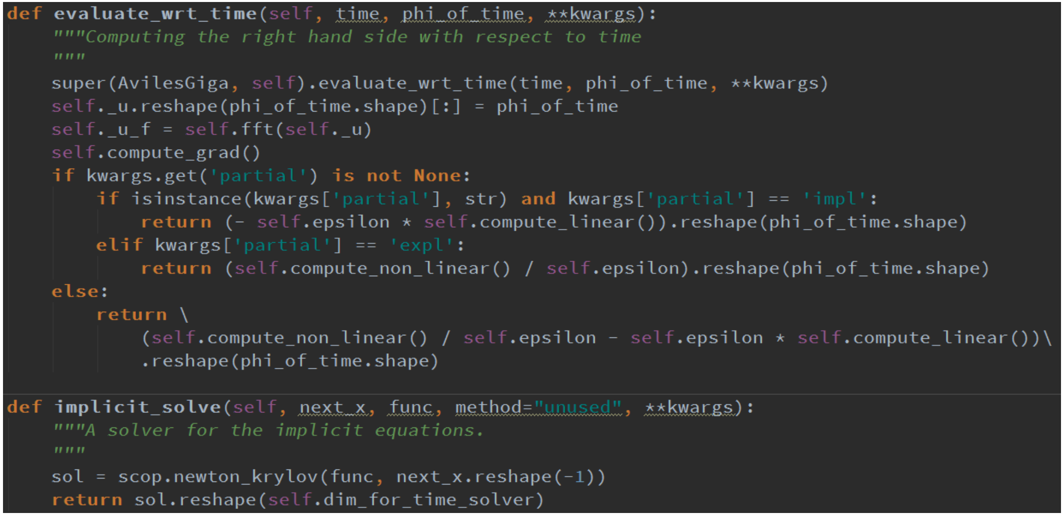
The Aviles-Giga Problem

. . . for nice plots . . .



The Aviles-Giga Problem

. . . with less code



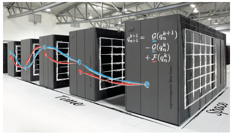
```python
def evaluate_wrt_time(self, time, phi_of_time, **kwargs):
    """Computing the right hand side with respect to time
    """
    super(AvilesGiga, self).evaluate_wrt_time(time, phi_of_time, **kwargs)
    self._u.reshape(phi_of_time.shape)[:] = phi_of_time
    self._u_f = self.fft(self._u)
    self.compute_grad()
    if kwargs.get('partial') is not None:
        if isinstance(kwargs['partial'], str) and kwargs['partial'] == 'impl':
            return (- self.epsilon * self.compute_linear()).reshape(phi_of_time.shape)
        elif kwargs['partial'] == 'expl':
            return (self.compute_non_linear() / self.epsilon).reshape(phi_of_time.shape)
    else:
        return \
            (self.compute_non_linear() / self.epsilon - self.epsilon * self.compute_linear())\
            .reshape(phi_of_time.shape)

def implicit_solve(self, next_x, func, method="unused", **kwargs):
    """A solver for the implicit equations.
    """
    sol = scop.newton_krylov(func, next_x.reshape(-1))
    return sol.reshape(self.dim_for_time_solver)
```

Advantages of *PyPinT*

"towards" . . .

- playground for rapid prototyping of PinT algorithms
  - no need to implement e.g. matrix inversion or quadrature on your own
    $\Rightarrow$ 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


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

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