The Parallel Full Approximation Scheme in Space and Time (PFASST) algorithm

M. Emmett and M. L. Minion

Department of Mathematics
University of North Carolina at Chapel Hill

February 6 2012

Overview

Two main parts:

1. PFASST: Parallel in time method by Michael Minion at UNC-CH:
   - parallel spectral deferred corrections (SDC)
   - full approximation scheme
   - hybrid iterations
2. Applications
   - Advection/diffusion and viscous Burger's eqn
   - Wave eqn
   - Kuramoto-Silvashinsky
   - Thin-film core-annular
   - 2d vorticity formulation of Navier-Stokes
   - 3d pseudo-spectral projection method for Navier-Stokes
   - Atmospheric Boussinesq eqns

When might time parallelisation for PDEs be attractive?

- When spatial parallelisation for a fixed problem size is saturated and more processors are available.
- When a very large number of time steps are required for a given simulation.
- When many realisations of the same time dependent problem must be computed sequentially (e.g. optimisation, UQ)
- To "easily" parallelise shared memory (OpenMP) codes across many multi-core processors.
- To "easily" parallelise serial PDE codes on multi-core machines.

Goal: construct parallel methods in the temporal direction that achieve "adequate" parallel efficiency.

Strategies for time parallelisation of ODEs

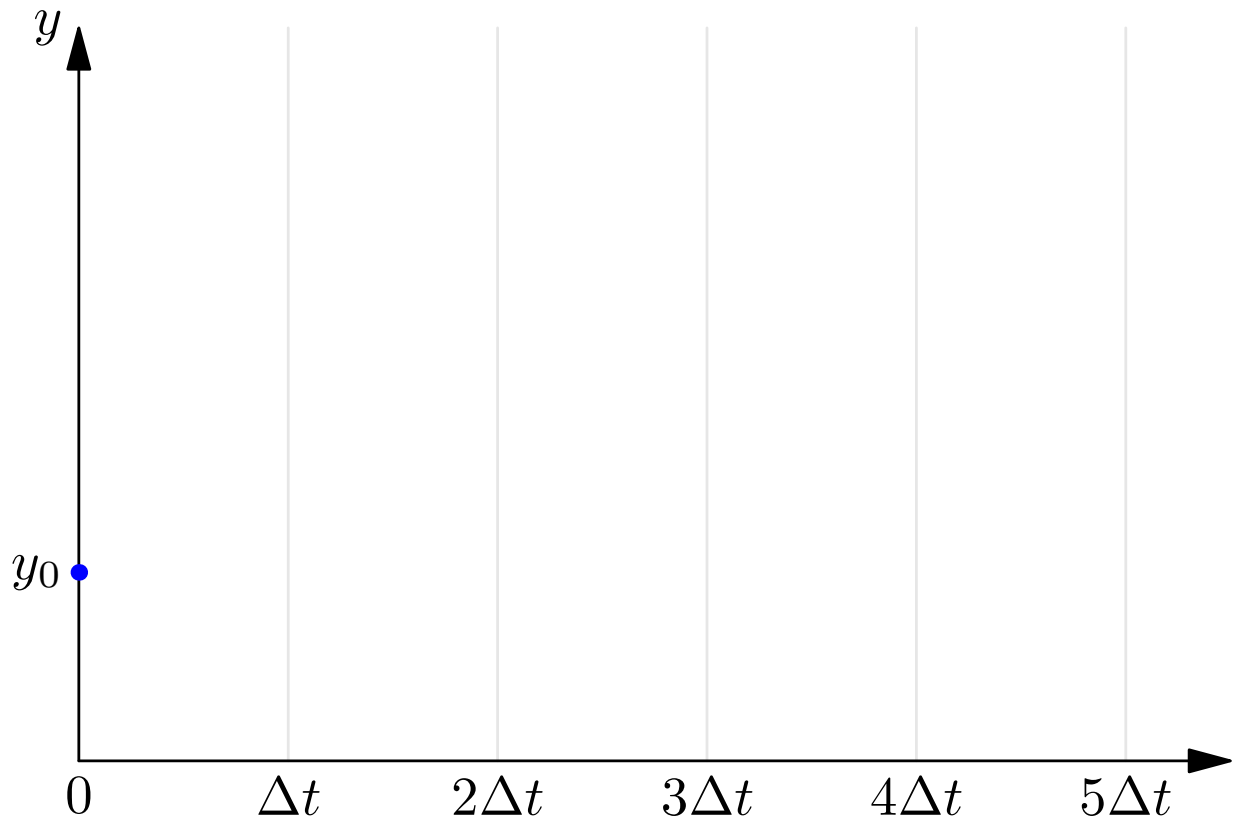
Early methods to parallelise the solution of ODEs date to the 1960s and can be roughly classified by

- Across the Method (e.g. parallelisation of Runge-Kutta methods)
- Across the Problem (e.g. wave-form relaxation)
- Across the Time Domain (e.g. Parareal, PITA, PFASST)

Of the three, only methods that use parallelisation across time have been shown to be efficient for many processors.

Parallelisation across time

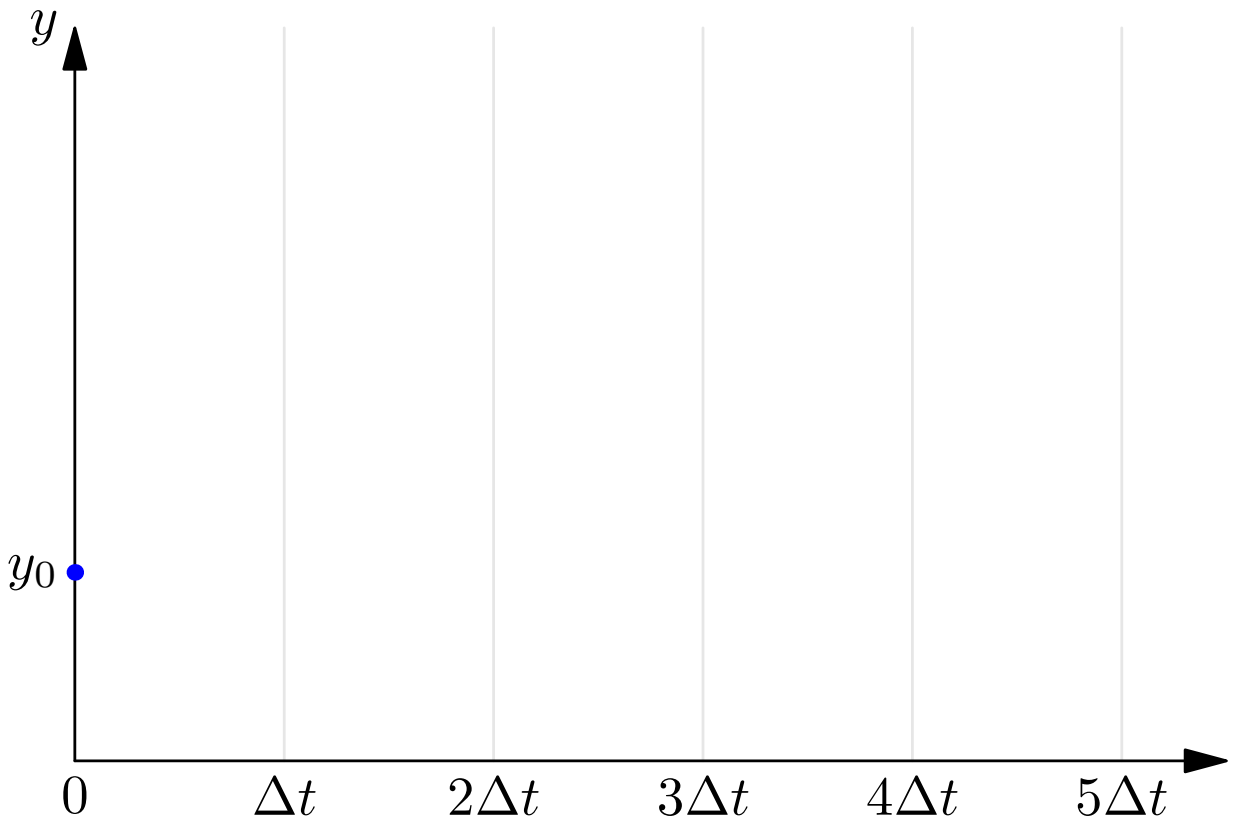
General idea:



We want to solve $y' = F(t, y)$ with $y(0) = y_0$

Parallelisation across time

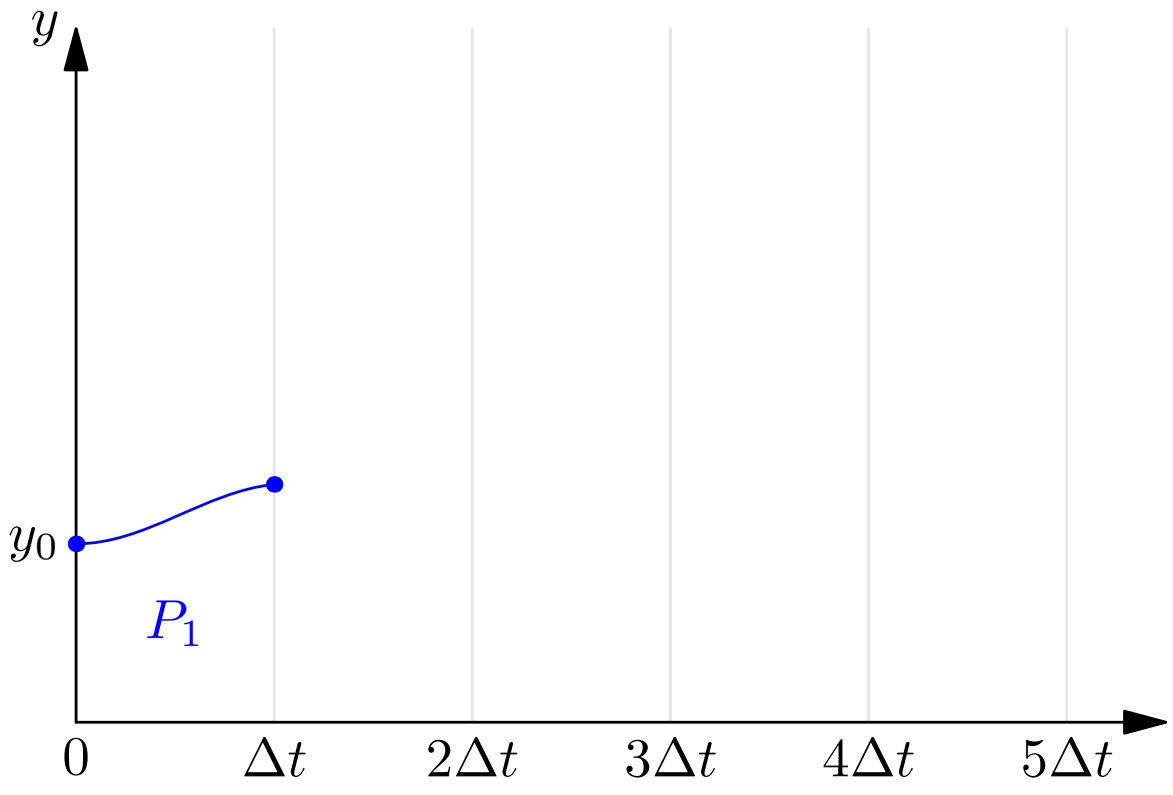
General idea:



We assign each processor a time step

Parallelisation across time

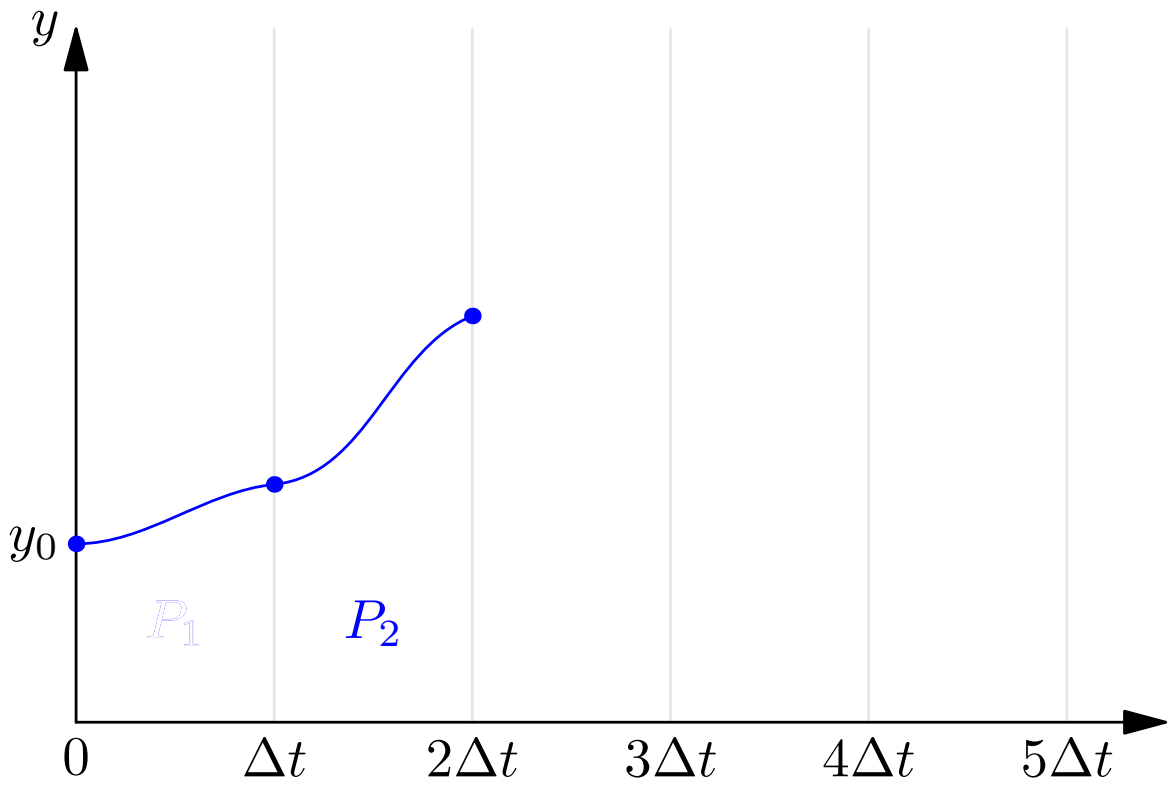
General idea:



A provisional solution is computed in serial

Parallelisation across time

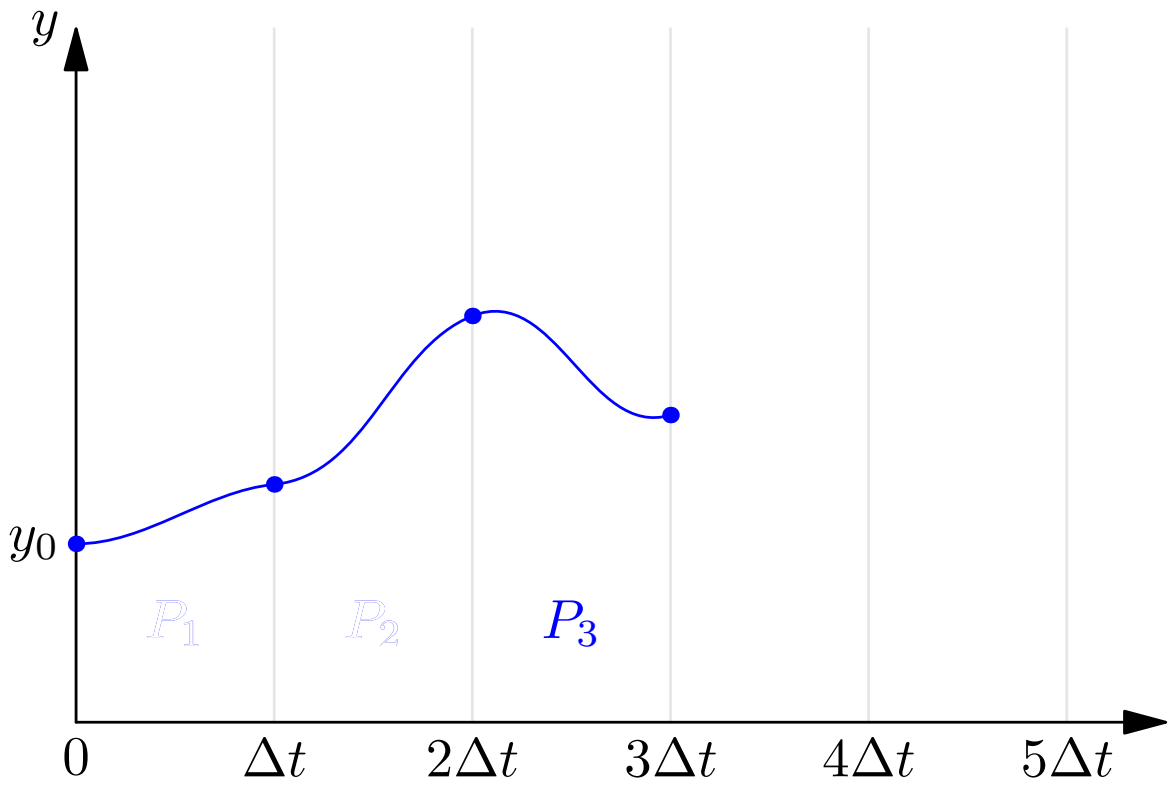
General idea:



Each processor propagates the solution

Parallelisation across time

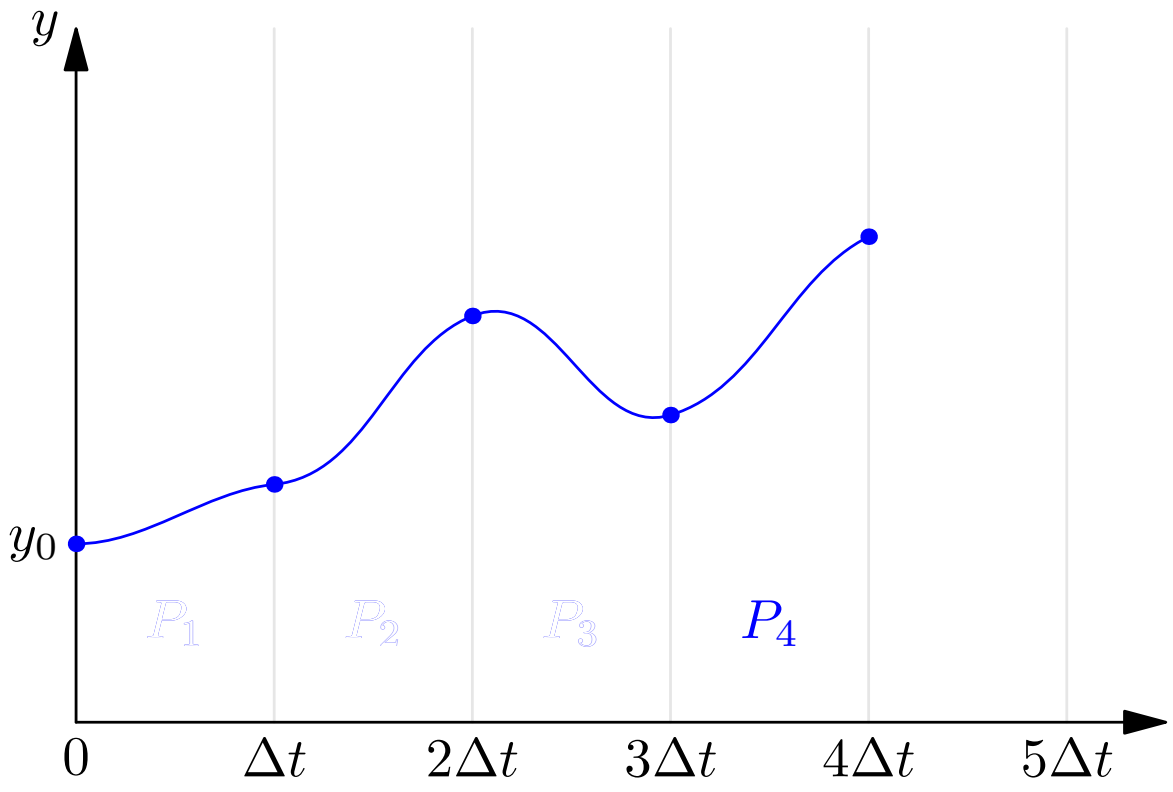
General idea:



The initial condition for the next step is passed forward in time

Parallelisation across time

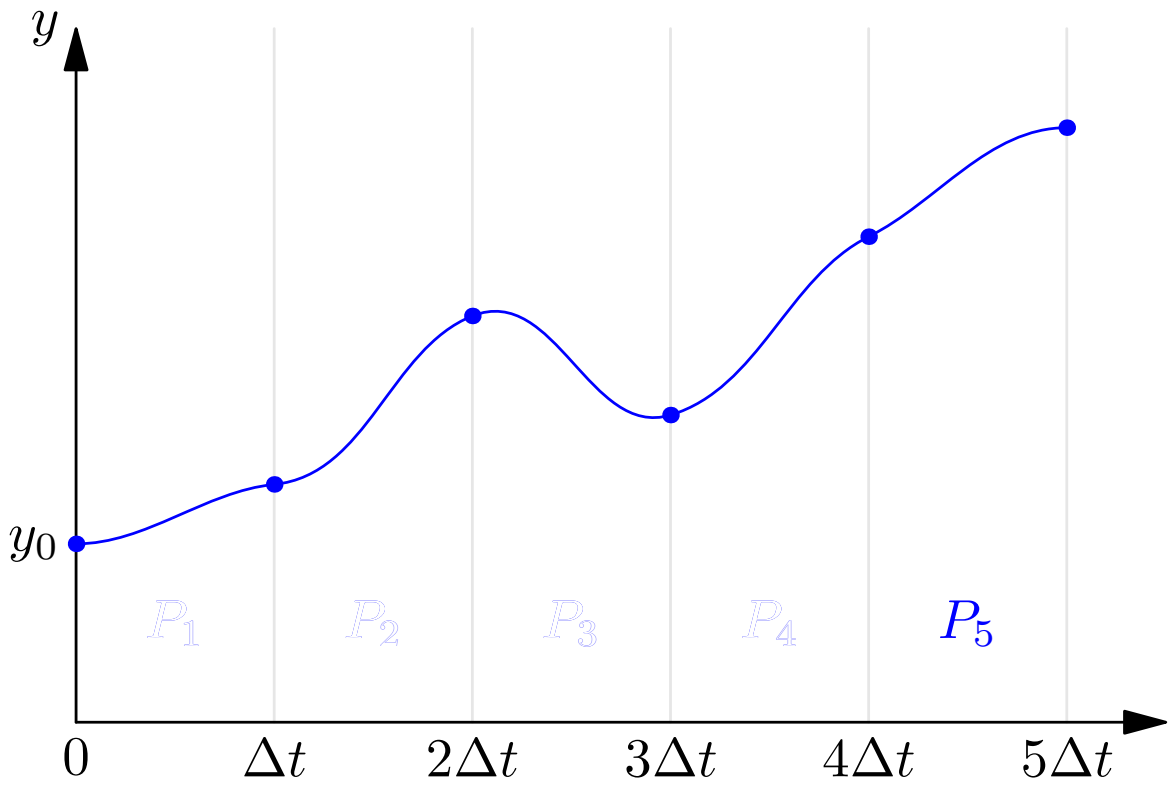
General idea:



The initial condition for the next step is passed forward in time

Parallelisation across time

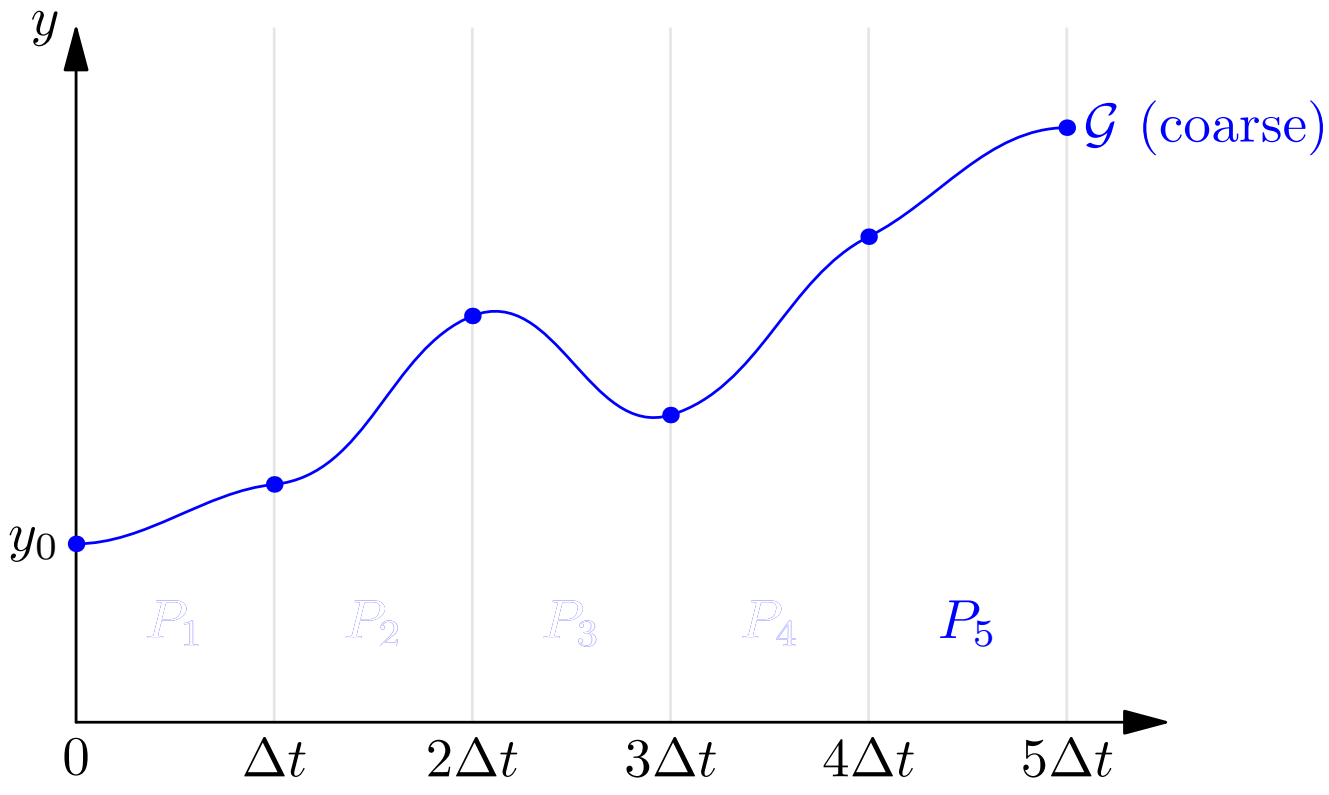
General idea:



The initial condition for the next step is passed forward in time

Parallelisation across time

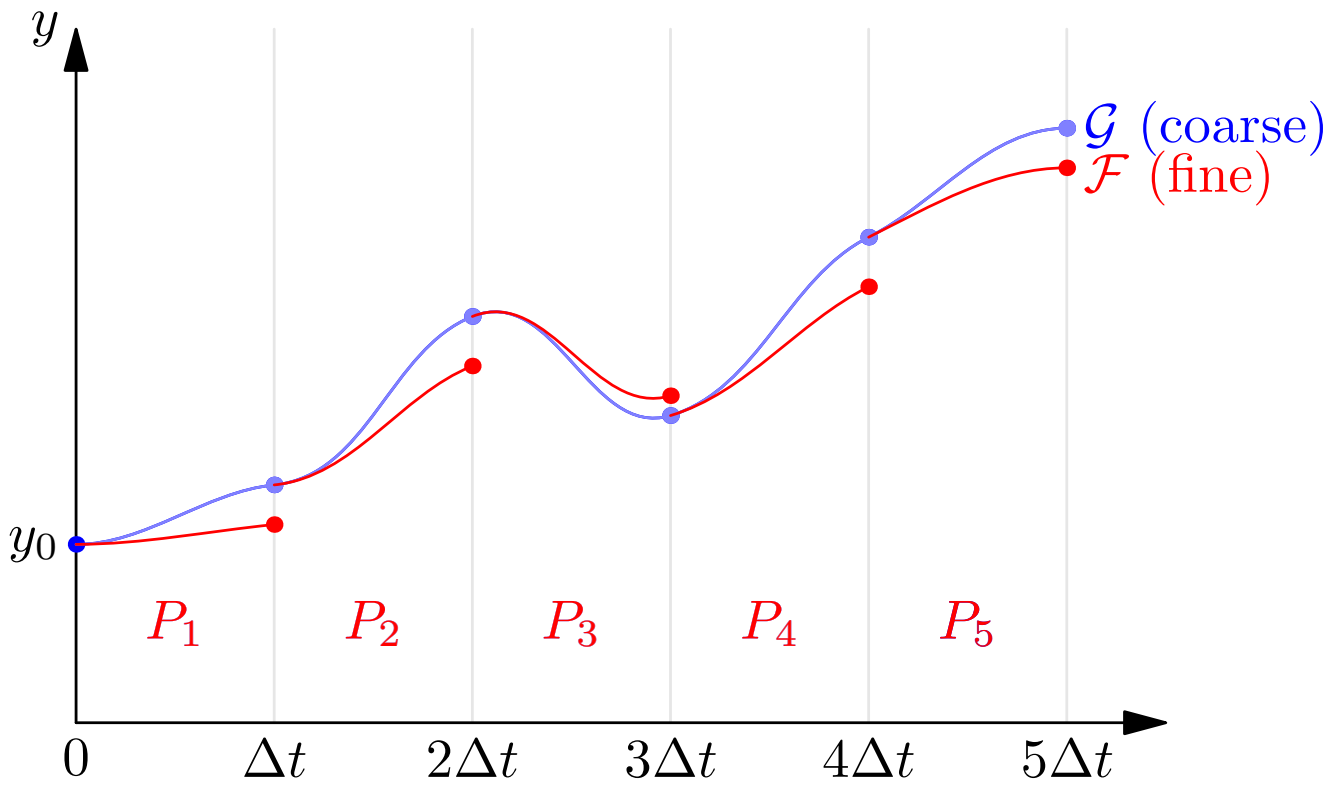
General idea:



Finally each processor has an (approximate) initial condition

Parallelisation across time

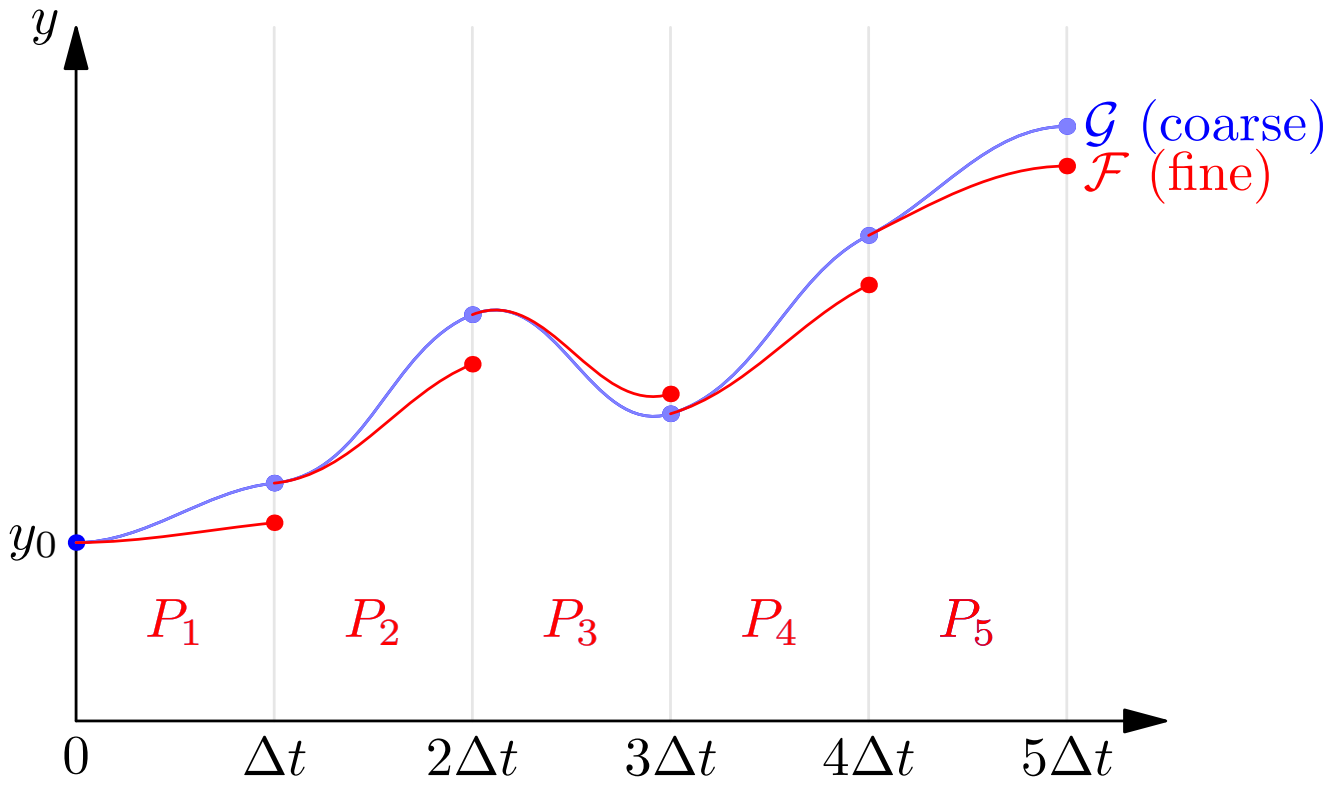
General idea:



Accurate solutions can be computed on each processor simultaneously
(but with the wrong initial conditions)

Parallelisation across time

General idea:



To get a more accurate solution than coarse, we need to somehow improve the initial value on each time slice.

Parallel in time - general idea

All of the parallel in time methods share the general idea:

- use a *coarse* propagator $\mathcal{G}$ to obtain approximate initial values on the coarse time-grid
- use a *fine* propagator $\mathcal{F}$ to obtain a more accurate solution
- apply an iterative procedure (involving $\mathcal{G}$ and $\mathcal{F}$) to smooth out the discontinuities

What makes PFASST unique?

- uses the SDC time-stepping scheme (which is iterative)
- the $\mathcal{G}/\mathcal{F}$ and SDC iterations are combined into one hybrid iteration
- multiple levels of spatial and temporal refinements are used in a multi-grid-esque time/space hierarchy to decrease the relative cost of $\mathcal{G}$ vs $\mathcal{F}$.

Spectral Deferred Corrections (SDC)

Consider the ODE:

$$y' = f(t, y) \quad \text{or equivalently} \quad y = y_0 + \int_{t_0}^{t} f(\tau, y)\, d\tau.$$

Perform the integration using Gaussian quadrature:

$$y(t_{n+1}) \approx y_n + \Delta t \sum_{m=1}^{M} q_m f\left(t_m, y(t_m)\right)$$

More compactly:

$$\mathbf{Y} = \mathbf{Y}_0 + \Delta t \mathbf{S}\mathbf{F}(\mathbf{Y})$$

Spectral Deferred Corrections (SDC)

SDC is an **iterative** time-stepping method for solving

$$y' = f(t, y) \implies \mathbf{Y} = \mathbf{Y}_0 + \Delta t \mathbf{S}\mathbf{F}(\mathbf{Y})$$

Can build very high-order SDC integrators using simple time-steppers: the Forward-Euler SDC method is

$$y_{m+1}^{k+1} = y_m^{k+1} + \Delta t_m \left[f(t_m, y_m^{k+1}) - f(t_m, y_m^k) \right] + S_m^{m+1}$$

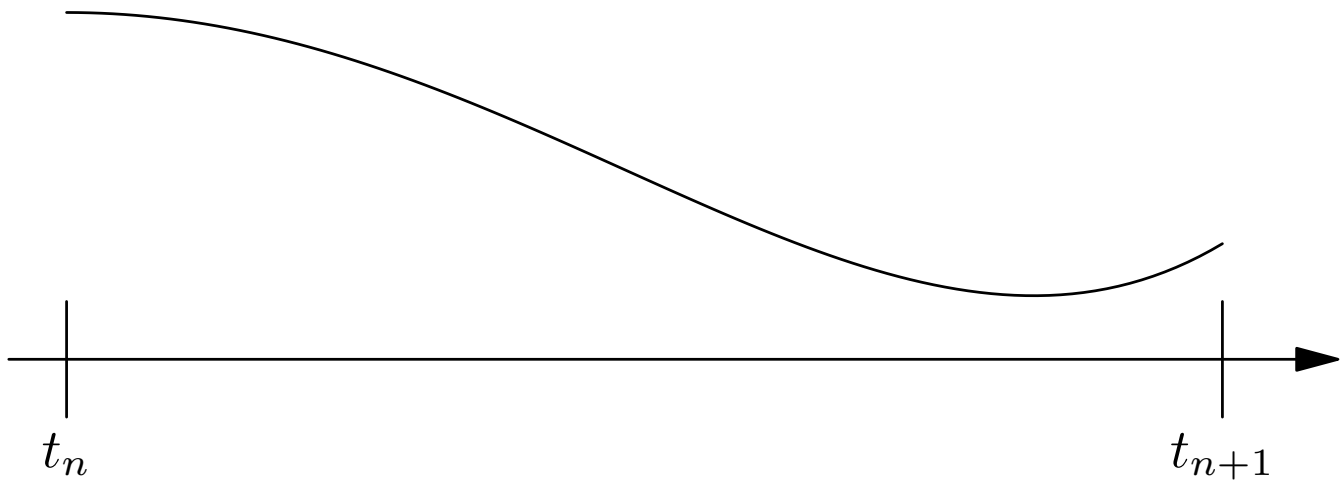
where

$$S_m^{m+1} \approx \int_{t_m}^{t_{m+1}} f(\tau, y^k(\tau))\, d\tau.$$

Spectral Deferred Corrections (SDC)

SDC is an **iterative** time-stepping method for solving

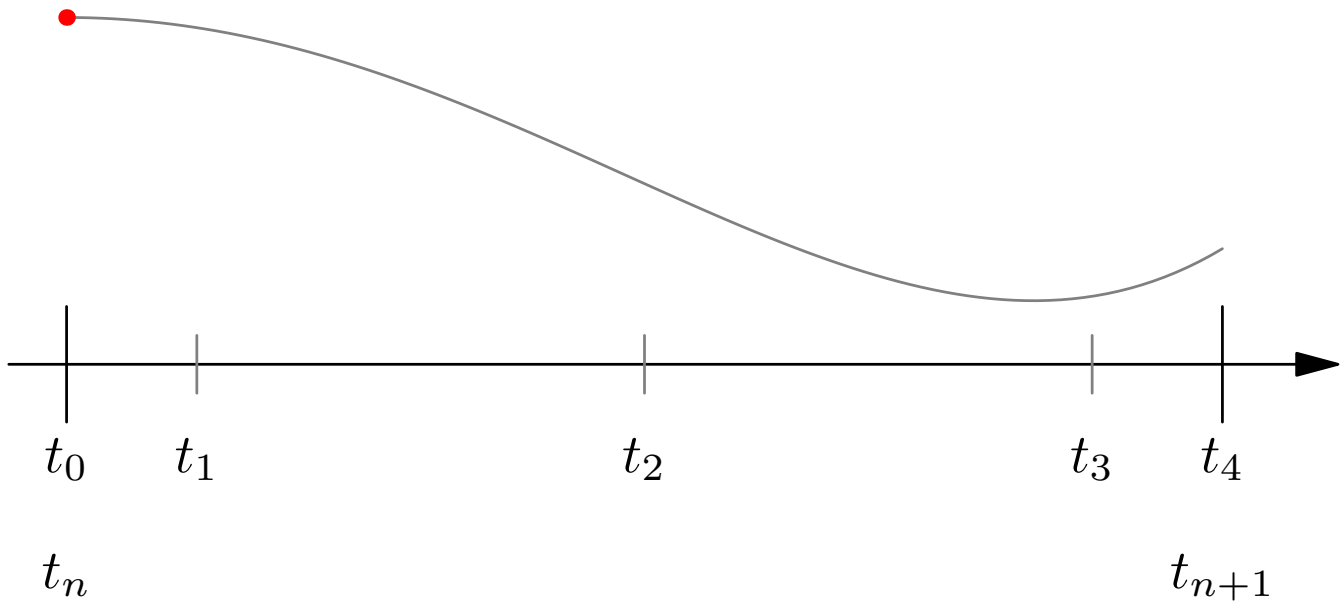
$$y' = f(t, y).$$



Spectral Deferred Corrections (SDC)

SDC is an **iterative** time-stepping method for solving

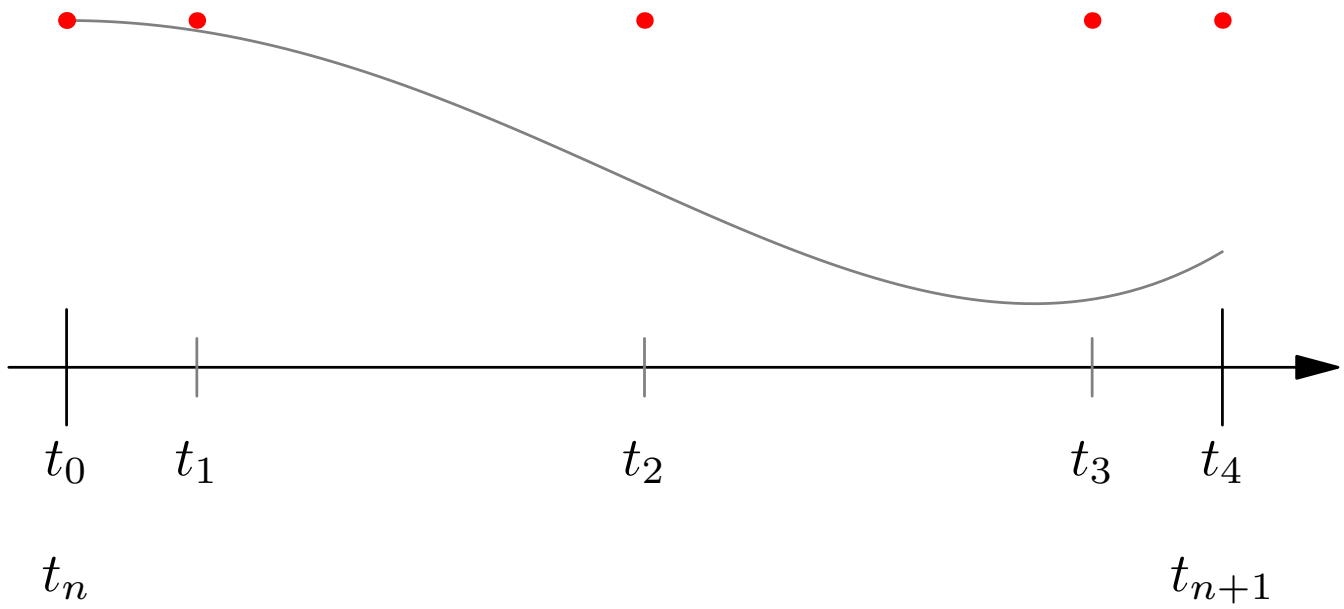
$$y' = f(t, y).$$



Spectral Deferred Corrections (SDC)

SDC is an **iterative** time-stepping method for solving

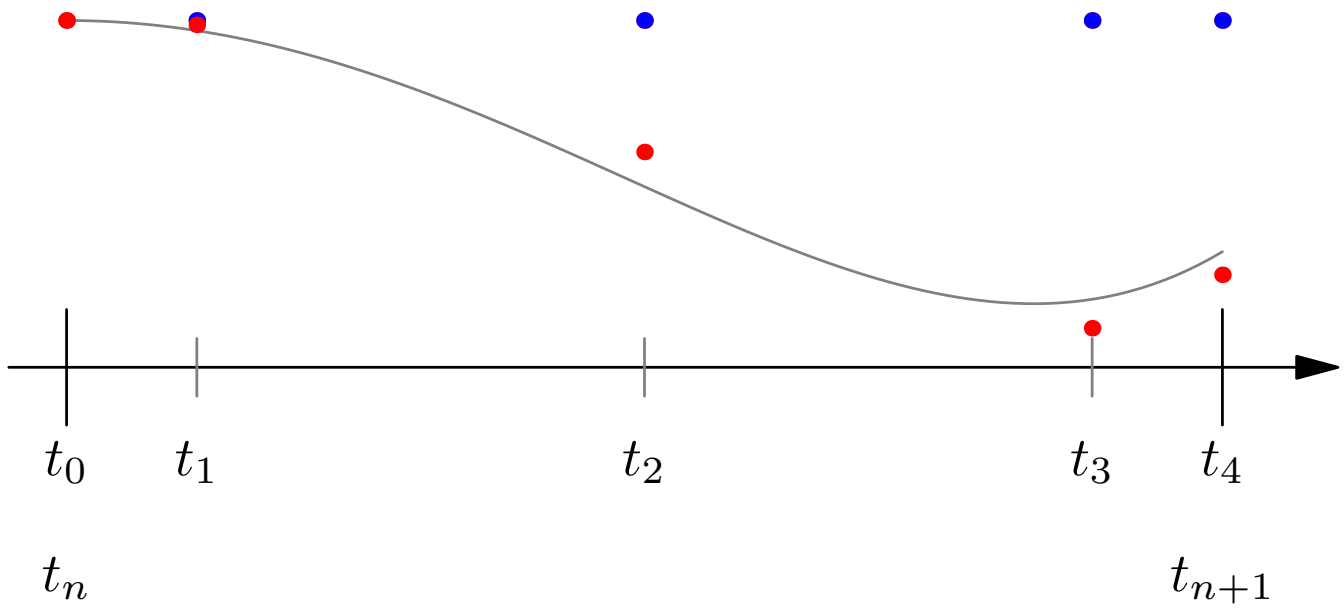
$$y' = f(t, y).$$



Spectral Deferred Corrections (SDC)

SDC is an **iterative** time-stepping method for solving

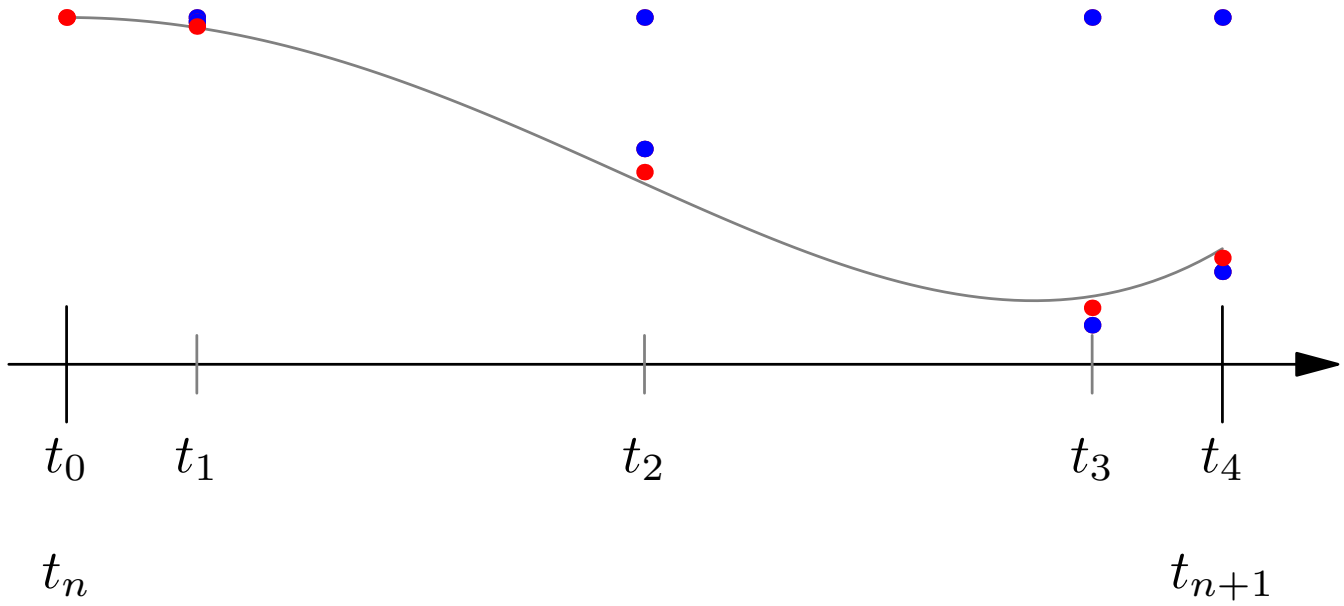
$$y' = f(t, y).$$



Spectral Deferred Corrections (SDC)

SDC is an **iterative** time-stepping method for solving

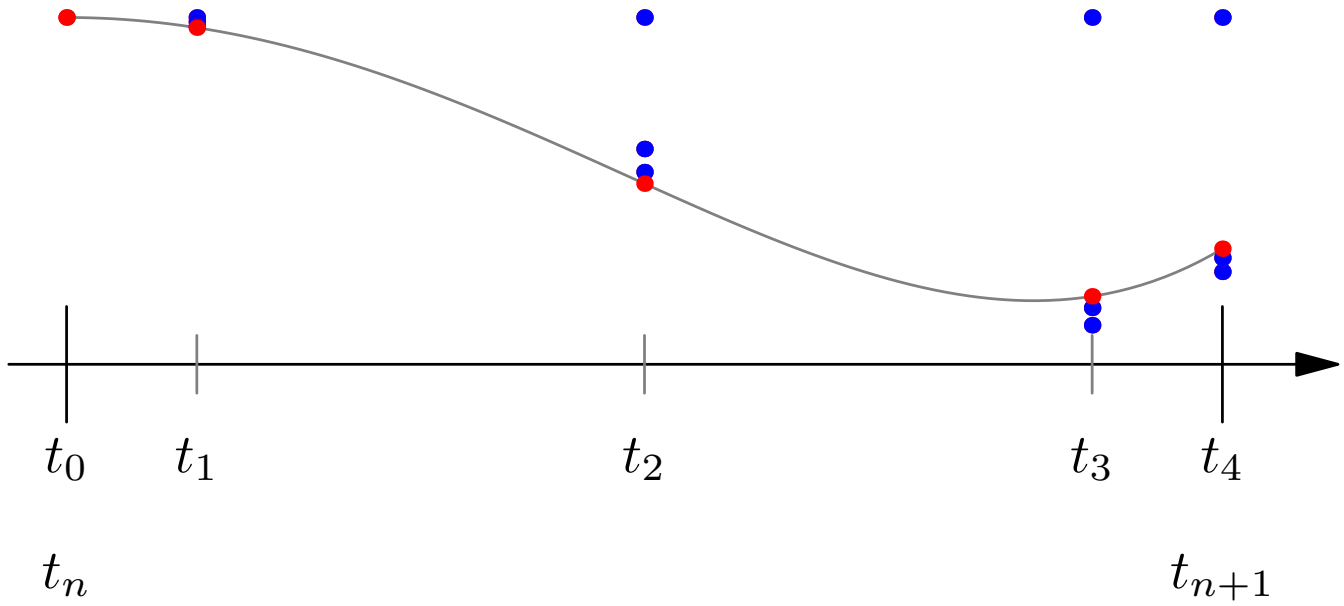
$$y' = f(t, y).$$



Spectral Deferred Corrections (SDC)

SDC is an **iterative** time-stepping method for solving

$$y' = f(t, y).$$



Full Approximation Scheme (FAS)

FAS corrections allow coarse solutions to obtain the accuarcy of fine solutions.

Consider the non-linear equation:

$$A(x) = b \quad \text{with residual eqn} \quad A(\tilde{x} + e) = A(\tilde{x}) + r.$$

In a FAS multi-grid approach, given a fine solution $\tilde{x}^F$, the coarse residual eqn is re-written according to

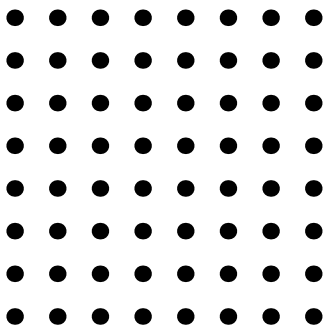
$$\begin{aligned} A^G(\tilde{x}^G + e^G) &= A^G(\tilde{x}^G) + r^G \\ &= A^G(\tilde{x}^G) + T_F^G(b^F - A^F(\tilde{x}^F)) \\ &= b^G + A^G(\tilde{x}^G) - T_F^G A^T(\tilde{x}^F) \\ &= b^G + \tau. \end{aligned}$$

That is, the coarse grid eqn is: $A(x^G) = b^G + \tau$ instead of $A(x^G) = b^G$.

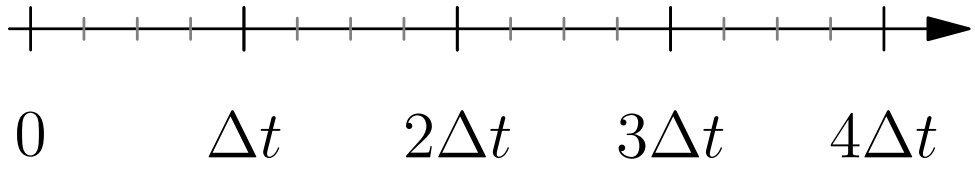
PFASST grid hierarchy

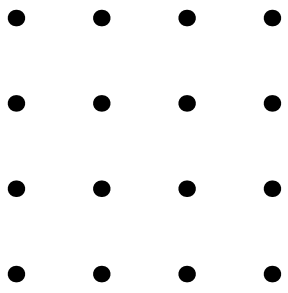
For a 2D problem with two PFASST levels, we (may) have refinement in

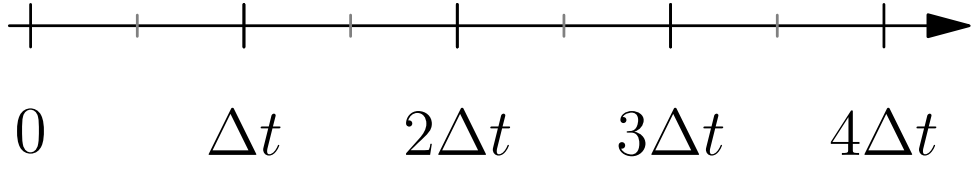
space



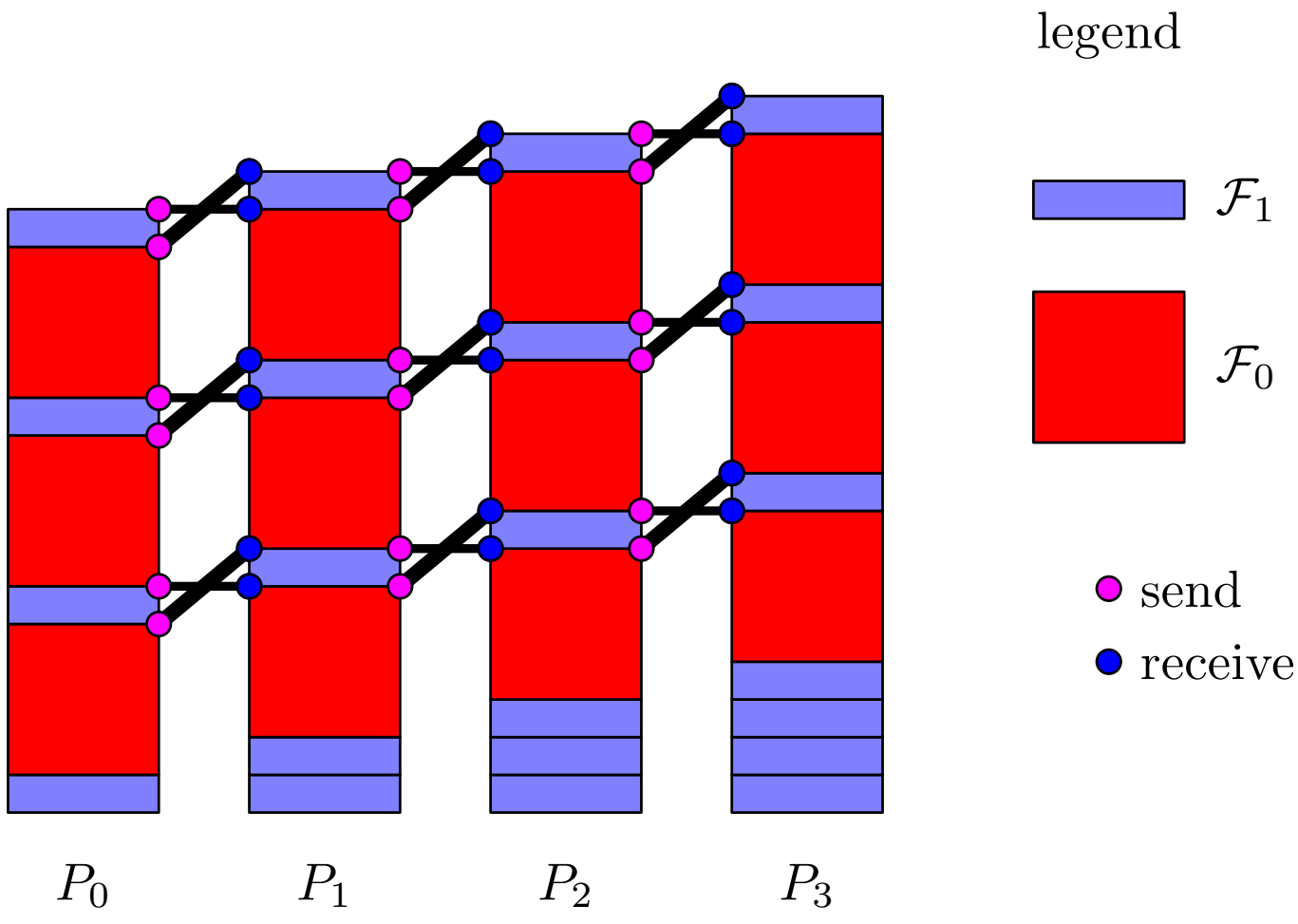
time



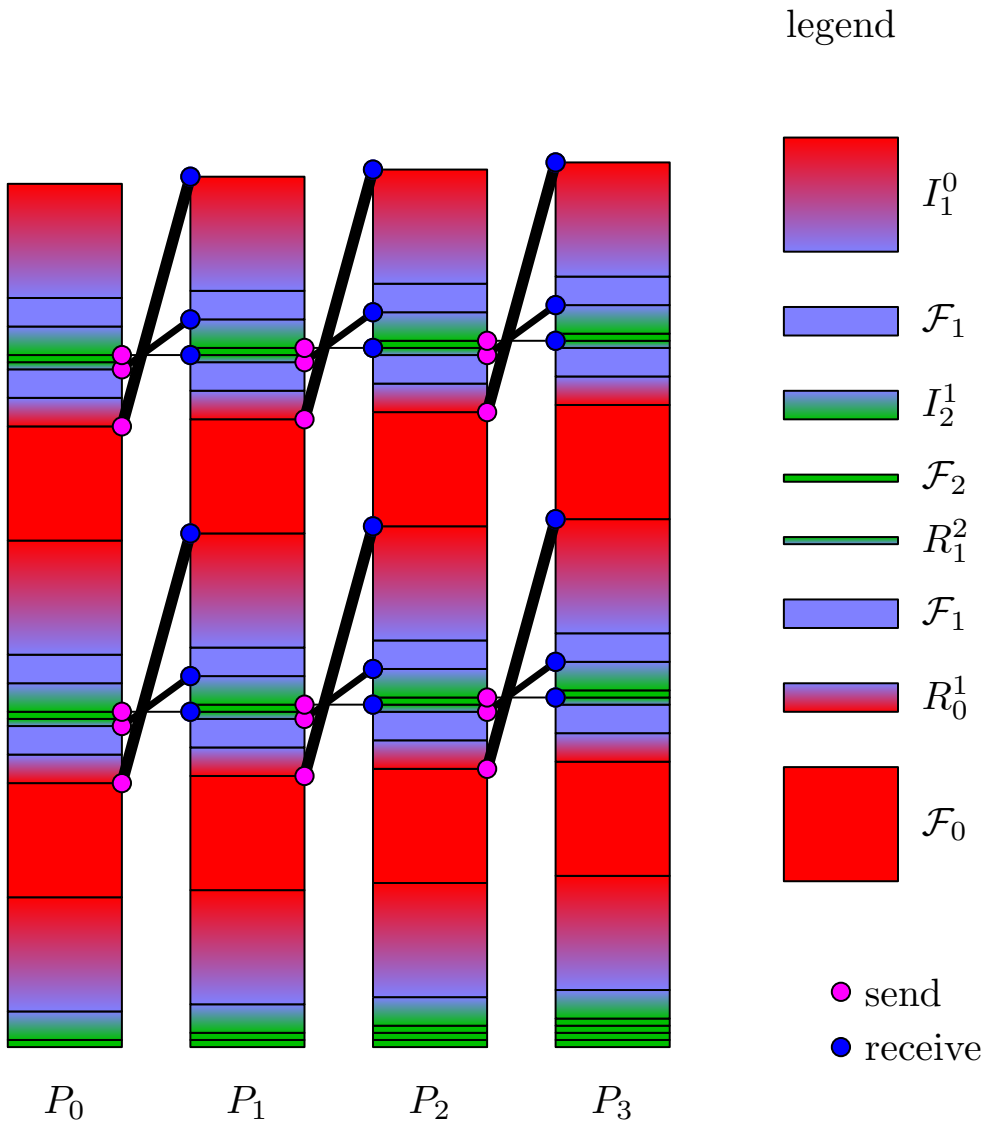




Two-level PFASST diagram



Three-level PFASST diagram (full)



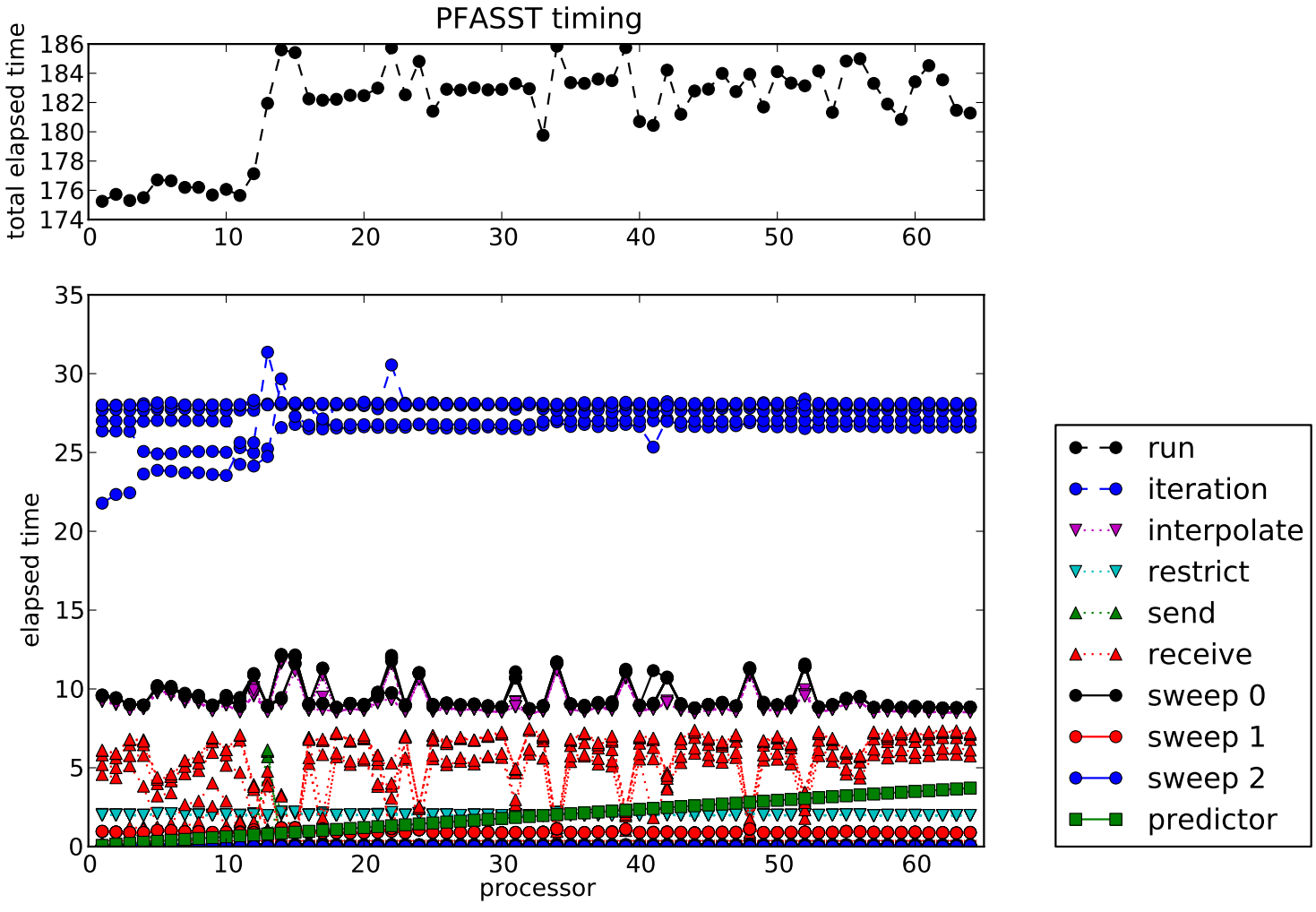
PFASST in action

Two level, 8 processor run...

PFASST in action

Three level, 16 processor run...

PFASST timing



PFASST

Two light-weight PFASST implementations:

`http://www.unc.edu/~memmett/pfasst/`

- PyPFASST: Python + Numpy + mpi4py. Available on `github.com`.
- F90PFASST: Fortran + MPI.

Works with method-of-lines discretisations: you provide

$$\frac{d}{dt} Q_i = F_{\text{exp}}(\mathbf{Q}, t) + F_{\text{imp}}(\mathbf{Q}, t).$$

Can combine with spatial parallelisation techniques as well:

- MPI
- PETSc
- OpenMP

PFASST applications so far

The PFASST algorithm has been successfully applied to various problems including:

- Advection/diffusion and viscous Burger's eqn
- Wave eqn
- Kuramoto-Silvashinsky
- Thin-film core-annular
- 2d vorticity formulation of Navier-Stokes
- 3d pseudo-spectral projection method for Navier-Stokes
- Atmospheric Boussinesq eqns

We have run tests using anywhere from 4 to 512 processors.

Some speedups

Some sample timings:

- 1d KS: 512 pt grid, 12x speedup on 64 cores (20% efficiency)
- 2d vorticity: 128^2 grid, 6.81x speedup on 16 cores (43% efficiency)
- 2d KS: 512^2 pt grid, 70x speedup on 128 cores (54% efficiency)
- 3d scalar AD: 128^3 grid, 5.25x speedup on 8 cores (66% efficiency)

We have shown that the theoretical efficiency of PFASST behaves like

$$E = \frac{K_s}{K_p(1+\beta_0)}.$$

Note that for parareal the theoretical efficiency is more like

$$E = \frac{1}{K_p(1+\beta_0)}.$$

Kuromoto-Silvashinsky

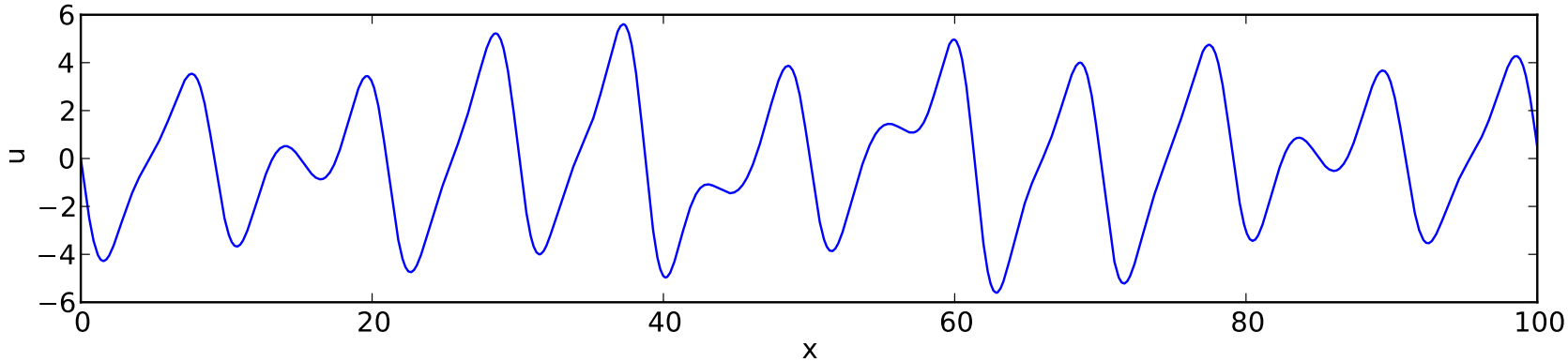
The 1d KS equation is

$$u_t + uu_x + u_{xx} + u_{xxxx} = 0$$

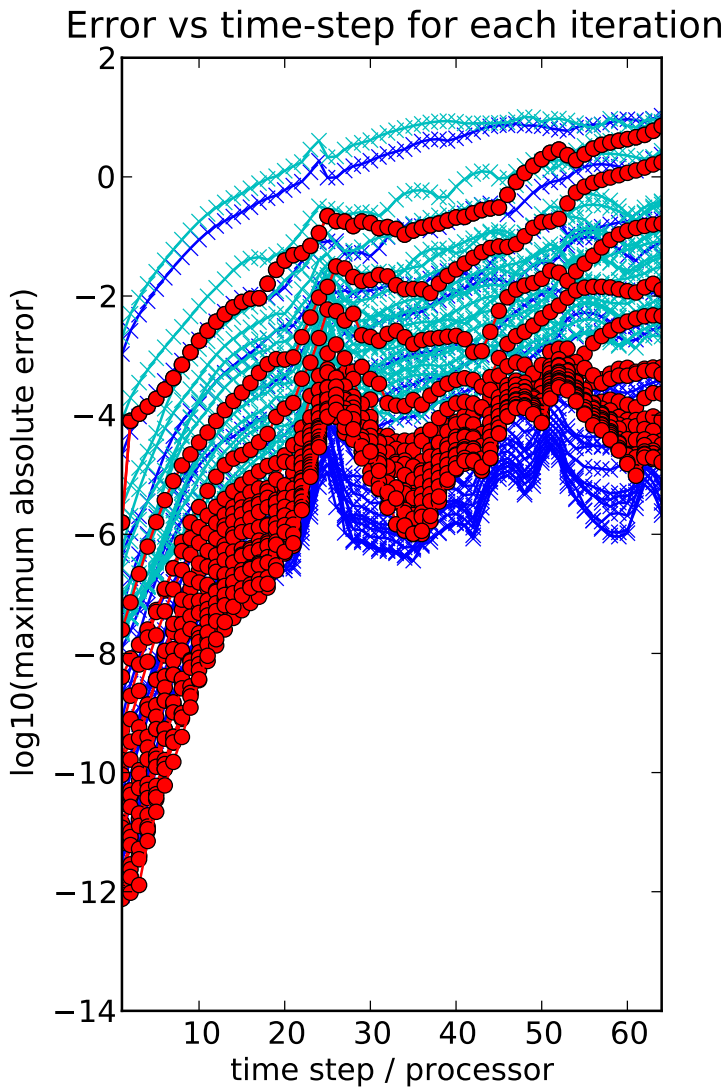
Periodic domain of length $L = 100$ with initial conditions

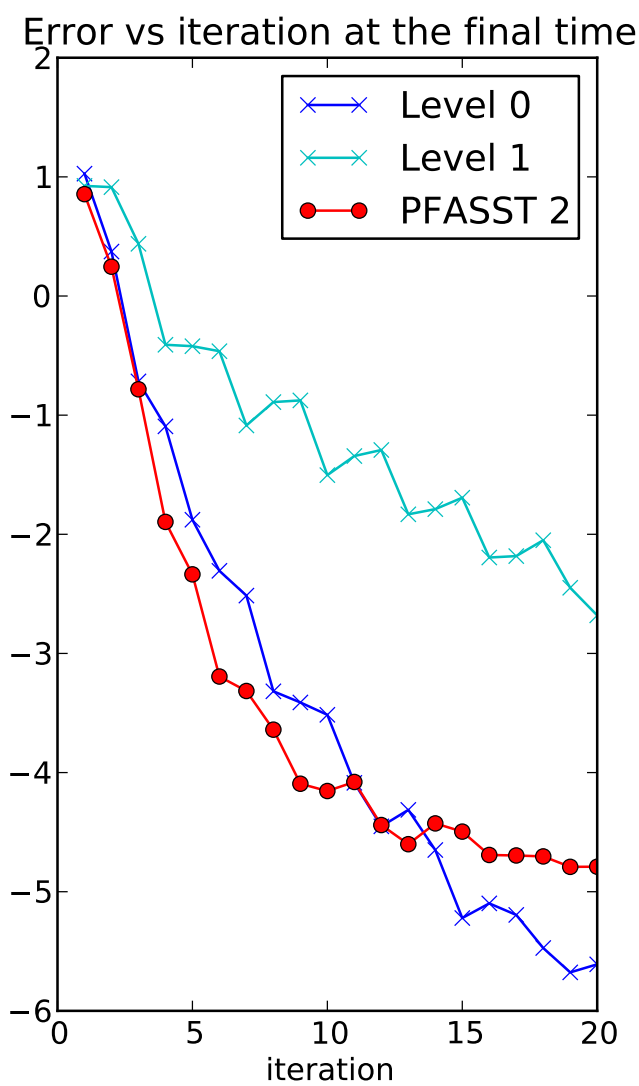
$$u_0(x) = 0.1\sin(6\pi x/L) + 0.2\sin(8\pi x/L) + 0.3\sin(14\pi x/L).$$

Solution at time $t = 64$



Kuromoto-Silvashinsky





Vorticity

Vorticity formulation of the 2D incompressible Navier-Stokes equation is

$$\partial_t \omega + \mathbf{u} \cdot \nabla \omega = \nu \nabla^2 \omega$$

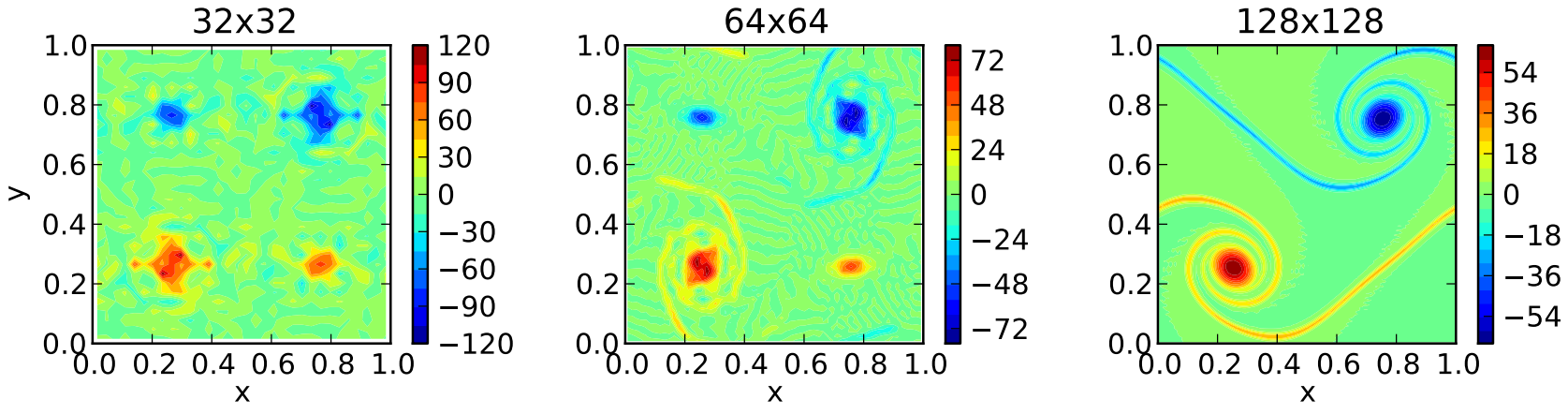
Doubly periodic domain $[0, 1] \times [0, 1]$ with initial conditions

$$u_0(x, y) = -1.0 + \tanh\big(\rho(0.75 - y)\big) + \tanh\big(\rho(y - 0.25)\big)$$
$$v_0(x, y) = -\delta \sin\big(2\pi(x + 0.25)\big)$$

where $\rho = 100$ and $\delta = 0.05$.

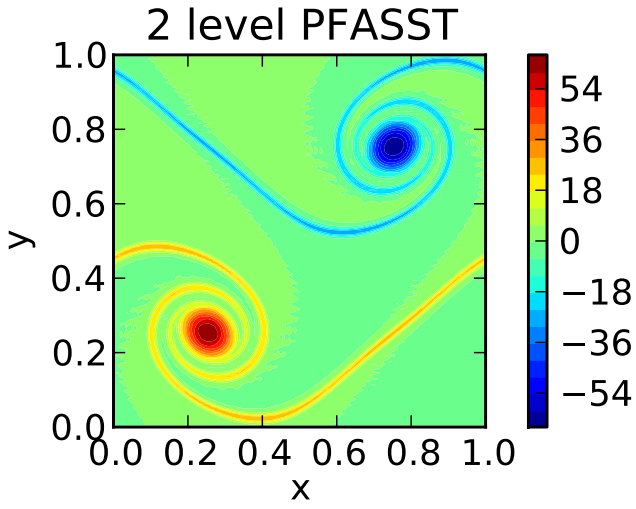
Vorticity

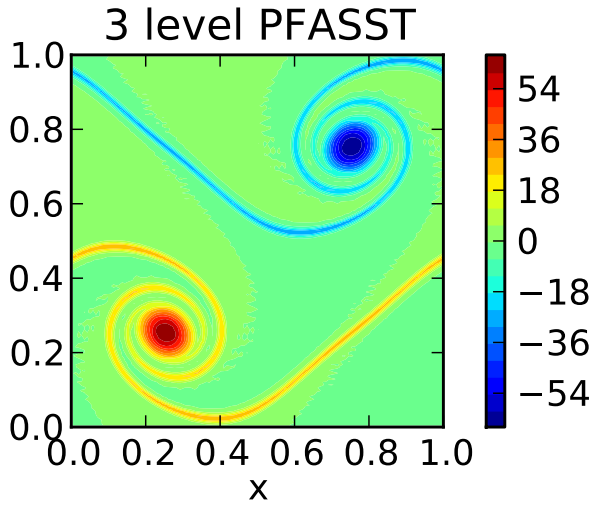
For small grids the solution is severely under-resolved.



Vorticity

Despite the under-resolution of the coarser levels, the multi-level PFASST solutions are essentially the same as the high-resolution serial run.





Navier-Stokes

Constant density incompressible NS:

$$\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u} = \nu \nabla^2 \mathbf{u} - \nabla p$$
$$\nabla \cdot \mathbf{u} = 0$$

Auxiliary variable projection method:

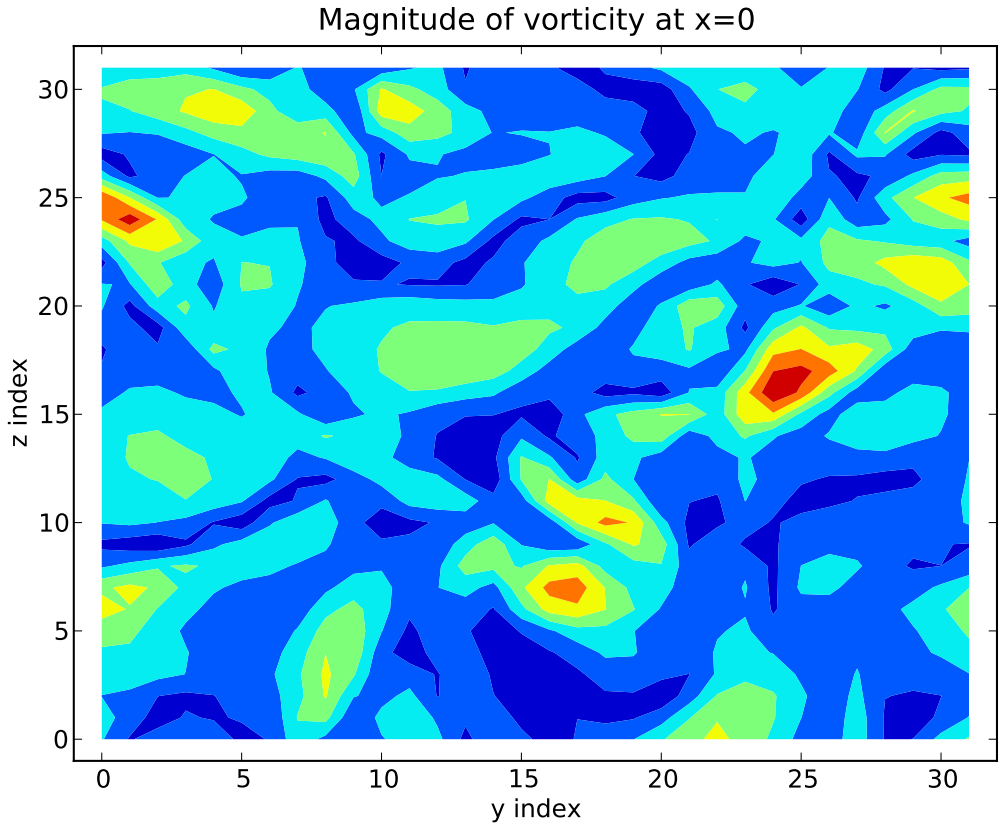
$$\mathbf{u}^*_t + \mathbf{u} \cdot \nabla \mathbf{u} = \nu \nabla^2 \mathbf{u}^* - \nabla q$$

where $P(\mathbf{u}^*) = \mathbf{u}$.

Time dependent forcing on a 3d cube:

$$\mathbf{f}(t, \mathbf{x}) = \sum_{|\mathbf{k}| \leq 4} \mathbf{a}_\mathbf{k} \cos(\omega t - \phi_t) \cos(k_x x - \phi_x) \cos(k_y y - \phi_y) \cos(k_z z - \phi_z)$$

Navier-Stokes



Thin film core-annular

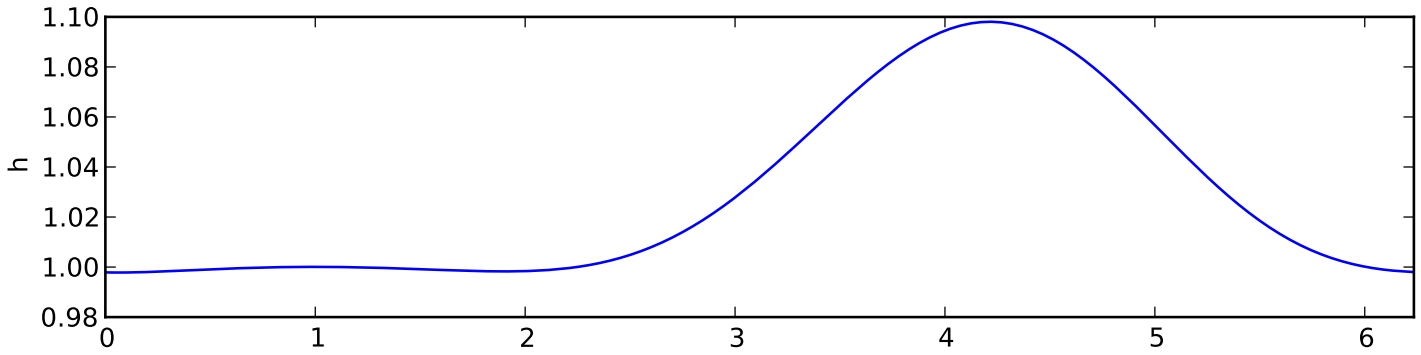
The thin-film core-annular equation is

$$h_t + hh_z + S\left(h^3(h_z + h_{zzz})\right)_z = 0$$

where S is some constant. Solver:

- pseudo-spectral, fully implicit
- uses PETSc matrix-free SNES to perform non-linear solves

Solution at time $t = 4$



VARDEN-SDC

Work in progress...

VARDEN-SDC:

- zero-Mach number fluid flow solver
- auxiliary variable projection method
- variable density Poisson multi-grid solver
- fourth order finite-volume discretisation
- semi-implicit SDC time-stepping
- work in progress: non-trivial boundary conditions; adaptive mesh refinement; and high-order multi-grid solver.

VARDEN-SDC is built upon BoxLib, which also serves as a basis for many other software projects at the CCSE. BoxLib facilitates:

- parallel spatial domain decomposition
- adaptive mesh refinement

PyClaw?

I'm here until Saturday at 2am... could it be this easy?

```
import levitate

...

class PFASSTPyCLAW(pfasst.imex.IMEXFEval):

  ...

  def f1_evaluate(self, q, t, f1, **kwargs):
    """Evaluate explicit piece."""

    self.solver.state.q = q
    f1[...] = self.solver.dqdt(self.solver.state)
```

Future work

Some ideas:

- Hybridize implicit solve iterations with PFASST iterations
- Multi-physics to further reduce ratio of $\mathcal{G}$ and $\mathcal{F}$
- Krylov deferred corrections (KDC)
- Uncertainty quantification

We'd like to try different spatial discretisation techniques with PFASST, and are always looking for collaborators...

We have a paper describing the two-level PFASST scheme that has been accepted for publication in CAMCoS. I would be happy to forward this to anyone that is interested.

Thanks!

Thanks to Dr. M. L. Minion for his guidance and patience.

This work was supported by the Director, DOE Office of Science, Office of Advanced Scientific Computing Research, Office of Mathematics, Information, and Computational Sciences, Applied Mathematical Sciences Program, under contract DE-SC0004011.