PINT 2019

PFASST-ER: combining the parallel full approximation scheme in space and time with parallelization across the method

Ruth Schöbel[1] · Robert Speck[2]

© The Author(s) 2020

Abstract

To extend prevailing scaling limits when solving time-dependent partial differential equations, the parallel full approximation scheme in space and time (PFASST) has been shown to be a promising parallel-in-time integrator. Similar to space–time multigrid, PFASST is able to compute multiple time-steps simultaneously and is therefore in particular suitable for large-scale applications on high performance computing systems. In this work we couple PFASST with a parallel spectral deferred correction (SDC) method, forming an unprecedented doubly time-parallel integrator. While PFASST provides global, large-scale "parallelization across the step", the inner parallel SDC method allows integrating each individual time-step "parallel across the method" using a diagonalized local Quasi-Newton solver. This new method, which we call "PFASST with Enhanced concuRrency" (PFASST-ER), therefore exposes even more temporal concurrency. For two challenging nonlinear reaction-diffusion problems, we show that PFASST-ER works more efficiently than the classical variants of PFASST and can use more processors than time-steps.

Keywords Parallel-in-time integration · Parallel full approximation scheme in space and time · Spectral deferred corrections · Parallelization across the method · Parallelization across the step · Quasi-Newton

1 Introduction

The efficient use of modern high performance computing systems for solving space–time-dependent differential equations has become one of the key challenges in computational science. Exploiting the exponentially growing number of processors using traditional techniques for spatial parallelism becomes problematic when, for example, for a fixed problem size communication costs start to dominate. Parallel-in-time integration methods have recently been shown to provide a promising way to extend these scaling limits, see e.g. [7,18,22] to name but a few examples.

✉ Ruth Schöbel
   ruth.schoebel@tu-dresden.de

   Robert Speck
   r.speck@fz-juelich.de

[1] Institut für Numerische Mathematik, TU Dresden, Dresden, Germany

[2] Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Jülich, Germany

As one example, the "Parallel Full Approximation Scheme in Space and Time" (PFASST) by Emmett and Minion [6] allows one to integrate multiple time-steps simultaneously by using inner iterations of spectral deferred corrections (SDC) on a space–time hierarchy. It works on the so called composite collocation problem, where each time-step includes a further discretization through quadrature nodes. This "parallelization across the steps" approach [3] targets large-scale parallelization on top of saturated spatial parallelization of partial differential equations (PDEs), where parallelization in the temporal domain acts as a multiplier for standard parallelization techniques in space. In contrast, "parallelization across the method" approaches [3] try to parallelize the integration within an individual time-step. While this typically results in smaller-scale parallelization in the time-domain, parallel efficiency and applicability of these methods are often more favorable. Most notably, the "revisionist integral deferred correction method" (RIDC) by Christlieb et al. [4] makes use of integral deferred corrections (which are indeed closely related to SDC) in order to compute multiple iterations in a pipelined way. In [19], different approaches for parallelizing SDC across the method have been discussed, allowing the simultaneous computation of updates on

multiple quadrature nodes. A much more structured and complete overview of parallel-in-time integration approaches can be found in [8]. The Parallel-in-Time community website (https://parallel-in-time.org) offers a comprehensive list of references.

The key goal of parallel-in-time integrators is to expose additional parallelism in the temporal domain in the cases where classical strategies like parallelism in space are either already saturated or not even possible. In [5] the classical Parareal method [15] is used to overcome the scaling limit of a space-parallel simulation of a kinematic dynamo on up to 1600 cores. The multigrid extension of Parareal, the "multigrid reduction in time" method (MGRIT), has been shown to provide significant speedup beyond spatial parallelization [7] for a multitude of problems. Using PFASST, a space-parallel N-body solver has been extended in [22] to run on up to 262,244 cores, while in [18] it has been coupled to a space-parallel multigrid solver on up to 458,752 cores.

So far, parallel-in-time methods have been implemented and tested either without any additional parallelization techniques or in combination with spatial parallelism. The goal for this work is to couple two different parallel-in-time strategies in order to extend the overall temporal parallelism exposed by the resulting integrator. To this end, we take the diagonalization idea for SDC presented in [19] (parallel across the method) and use it within PFASST (parallel across the steps). In this way we create an algorithm that computes approximations for different time-steps simultaneously but also works in parallel on each time-step itself. Doing so we combine the advantages of both parallelization techniques and create the "Parallel Full Approximation Scheme in Space and Time with Enhanced concuRrency" (PFASST-ER), an unprecedented doubly time-parallel integrator for PDEs. In the next section we will first introduce SDC and PFASST from an algebraic point of view, following [1,2]. We particularly focus on nonlinear problems and briefly explain the application of a Newton solver within PFASST. Then, this Newton solver is modified in Sect. 3 so that by using a diagonalization approach the resulting Quasi-Newton method can be computed in parallel across the quadrature nodes of each time-step. In Sect. 4, we compare different variants of this idea to the classical PFASST implementation using two nonlinear reaction-diffusion test cases. We show parallel runtimes for different setups and evaluate the impact of the various Newton and diagonalization strategies. Section 5 concludes this work with a short summary and an outlook.

2 Parallelization across the steps with PFASST

We focus on an initial value problem

$$u_t = f(u), \quad u(0) = u_0 \tag{1}$$

with $u(t), u_0, f(u) \in \mathbb{R}$. In order to keep the notation simple, we do not consider systems of initial value problems for now, where $u(t) \in \mathbb{R}^N$. Necessary modifications will be mentioned where needed. In a first step, we now discretize this problem in time and review the idea of single-step, time-serial spectral deferred corrections (SDC).

2.1 Spectral deferred corrections

For one time-step on the interval $[t_l, t_{l+1}]$ the Picard formulation of Eq. (1) is given by

$$u(t) = u_{l,0} + \int_{t_0}^{t} f(u(s))ds, \ t \in [t_l, t_{l+1}]. \tag{2}$$

To approximate the integral we use a spectral quadrature rule. We define M quadrature nodes $\tau_{l,1}, \ldots, \tau_{l,M}$, which are given by $t_l \leq \tau_{l,1} < \cdots < \tau_{l,M} = t_{l+1}$. We will in the following explicitly exploit the condition that the last node is equal to the right integral boundary. Quadrature rules like Gauß-Radau or Gauß-Lobatto quadrature satisfy this property. We can then approximate the integrals from t_l to the nodes $\tau_{l,m}$, such that

$$u_{l,m} = u_{l,0} + \Delta t \sum_{j=1}^{M} q_{m,j} f(u_{l,j}),$$

where $u_{l,m} \approx u(\tau_{l,m})$, $\Delta t = t_{l+1} - t_l$ and $q_{m,j}$ represent the quadrature weights for the interval $[t_l, \tau_{l,m}]$ such that

$$\sum_{j=1}^{M} q_{m,j} f(u_{l,j}) \approx \int_{t_l}^{\tau_{l,m}} f(u(s))ds.$$

We combine these M equations into one system

$$(\mathbf{I} - \Delta t \mathbf{Q} f)(\boldsymbol{u}_l) = \boldsymbol{u}_{l,0}, \tag{3}$$

which we call the "collocation problem". Here, $\boldsymbol{u}_l = (u_{l,1}, \ldots, u_{l,M})^T \approx (u(\tau_{l,1}), \ldots, u(\tau_{l,M}))^T \in \mathbb{R}^M$, $\boldsymbol{u}_{l,0} = (u_{l,0}, \ldots, u_{l,0})^T \in \mathbb{R}^M$, $\mathbf{Q} = (q_{ij})_{i,j} \in \mathbb{R}^{M \times M}$ is the matrix gathering the quadrature weights and the vector function $\boldsymbol{f} : \mathbb{R}^M \to \mathbb{R}^M$ is given by

$$\boldsymbol{f}(\boldsymbol{u}_l) = (f(u_{l,1}), \ldots, f(u_{l,M}))^T.$$

To simplify the notation we define

$$\mathbf{C}_f^{\mathrm{coll}}(\boldsymbol{u}_l) := (\mathbf{I} - \Delta t \mathbf{Q} f)(\boldsymbol{u}_l).$$

We note that for $u(t) \in \mathbb{R}^N$, we need to replace $\mathbf{Q}$ by $\mathbf{Q} \otimes \mathbf{I}_N$, where $\otimes$ denotes the Kronecker product.

System (3) is dense and a direct solution is not advisable, in particular if f is a nonlinear operator. The spectral deferred correction method solves the collocation problem in an iterative way. While it has been derived originally from classical deferred or defect correction strategies, we here follow [10,17,27] to present SDC as preconditioned Picard iteration. A standard Picard iteration is given by

$$\mathbf{u}_l^{k+1} = \mathbf{u}_l^k + (\mathbf{u}_{l,0} - \mathbf{C}_f^{\text{coll}}(\mathbf{u}_l^k))$$

for $k = 0, \ldots, K$, and some initial guess $\mathbf{u}_l^0$.

In order to increase range and speed of convergence, we now precondition this iteration. The standard approach to preconditioning is to define an operator $\mathbf{P}_f^{\text{sdc}}$, which is easy to invert but also close to the operator of the system. We define this "SDC preconditioner" as

$$\mathbf{P}_f^{\text{sdc}}(\mathbf{u}_l) := (\mathbf{I} - \Delta t \mathbf{Q}_\Delta f)(\mathbf{u}_l)$$

so that the preconditioned Picard iteration reads

$$\mathbf{P}_f^{\text{sdc}}(\mathbf{u}_l^{k+1}) = (\mathbf{P}_f^{\text{sdc}} - \mathbf{C}_f^{\text{coll}})(\mathbf{u}_l^k) + \mathbf{u}_{l,0}. \tag{4}$$

The key for defining $\mathbf{P}_f^{\text{sdc}}$ is the choice of the matrix $\mathbf{Q}_\Delta$. The idea is to choose a "simpler" quadrature rule to generate a triangular matrix $\mathbf{Q}_\Delta$ such that solving System (4) can be done by forward substitution. Common choices include the implicit Euler method or the so-called "LU-trick", where the LU decomposition of $\mathbf{Q}^T$ with

$$\mathbf{Q}_\Delta^{\text{LU}} = \mathbf{U}^{\mathbf{T}} \quad \text{for} \quad \mathbf{Q}^T = \mathbf{LU} \tag{5}$$

is used [27].

System (4) establishes the method of spectral deferred corrections, which can be used to approximate the solution of the collocation problem on a single time-step. In the next step, we will couple multiple collocation problems and use SDC to explain the idea of the parallel full approximation scheme in space and time.

2.2 Parallel full approximation scheme in space and time

The idea of PFASST is to solve a "composite collocation problem" for multiple time-steps at once using multigrid techniques and SDC for each step in parallel. This composite collocation problem for L time-steps can be written as

$$\begin{pmatrix} \mathbf{C}_f^{\text{coll}} & & & \\ -\mathbf{H} & \mathbf{C}_f^{\text{coll}} & & \\ & \ddots & \ddots & \\ & & -\mathbf{H} & \mathbf{C}_f^{\text{coll}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_L \end{pmatrix} = \begin{pmatrix} \mathbf{u}_{0,0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix},$$

where the matrix $\mathbf{H} \in \mathbb{R}^{M \times M}$ on the lower subdiagonal transfers the information from one time-step to the next one. It takes the value of the last node $\tau_{l,M}$ of an interval $[t_l, t_{l+1}]$, which is by requirement equal to the left boundary t_{l+1} of the following interval $[t_{l+1}, t_{l+2}]$, and provides it as a new starting value for this interval. Therefore, the matrix $\mathbf{H}$ contains the value 1 on every position in the last column and zeros elsewhere. To write the composite collocation problem in a more compact form we define the vector $\mathbf{u} = (\mathbf{u}_1, \ldots, \mathbf{u}_L)^T \in \mathbb{R}^{LM}$, which contains the solution at all quadrature nodes at all time-steps, and the vector $\mathbf{b} = (\mathbf{u}_{0,0}, \mathbf{0}, \ldots, \mathbf{0})^T \in \mathbb{R}^{LM}$, which contains the initial condition for all nodes at the first interval and zeros elsewhere. We define $\mathbf{F} : \mathbb{R}^{LM} \to \mathbb{R}^{LM}$ as an extension of f so that $\mathbf{F}(\mathbf{u}) = (f(\mathbf{u}_1), \ldots, f(\mathbf{u}_L))^T$. Then, the composite collocation problem can be written as

$$\mathbf{C}_F(\mathbf{u}) = \mathbf{b}. \tag{6}$$

with

$$\mathbf{C}_F(\mathbf{u}) = (\mathbf{I} - \Delta t(\mathbf{I}_L \otimes \mathbf{Q})\mathbf{F} - \mathbf{E} \otimes \mathbf{H})(\mathbf{u}),$$

where the matrix $\mathbf{E} \in \mathbb{R}^{L \times L}$ just has ones on the first subdiagonal and zeros elsewhere. If $u \in \mathbb{R}^N$, we need to replace $\mathbf{H}$ by $\mathbf{H} \otimes \mathbf{I}_N$.

SDC can be used to solve the composite collocation problem by forward substitution in a sequential way, which means to solve one time-step after each other using the previous solution as initial value of the current time-step. The parallel-in-time integrator PFASST, on the other hand solves the composite collocation problem by calculating on all time-steps simultaneously and is therefore an attractive alternative. The first step from SDC towards PFASST is the introduction of multiple levels, which are representations of the problem with different accuracies in space and time. In order to simplify the notation we focus on a two-level scheme consisting of a fine and a coarse level. Coarsening can be achieved for example by reducing the resolution in space, by decreasing the number of quadrature nodes on each interval or by solving implicit systems less accurately. Especially a coarsening through the reduction of quadrature points does not seem to be worthwhile for our idea to parallize the belonging calculations, since there would no longer be a full employment regarding the calculations on the coarse grid, but instead individual processors would have to communicate larger amounts of data. For this work, we only consider coarsening

in space, i.e., by using a restriction operator R on a vector $u \in \mathbb{R}^N$ we obtain a new vector $\tilde{u} \in \mathbb{R}^{\tilde{N}}$. Vice versa, the interpolation operator T is used to interpolate values from $\tilde{u}$ to u. Operators, vectors and numbers on the coarse level will be denoted by a tilde to avoid further index cluttering. Thus, the composite collocation operator on the coarse-level is given by $\tilde{\mathbf{C}}_F$. While $\mathbf{C}_F$ is defined on $\mathbb{R}^{LMN}$, $\tilde{\mathbf{C}}_F$ acts on $\mathbb{R}^{LM\tilde{N}}$ with $\tilde{N} \leq N$, but as before we will neglect the space dimension in the following notation. The extension of the spatial transfer operators to the full space–time domain is given by $\mathbf{R} = \mathbf{I}_{LM} \otimes R$ and $\mathbf{T} = \mathbf{I}_{LM} \otimes T$.

The main goal of the introduction of a coarse level is to move the serial part of the computation to this hopefully cheaper level, while being able to run the expensive part in parallel. For that, we define two preconditioners: a serial one with a lower subdiagonal for the coarse level and a parallel, block-diagonal one for the fine level. The serial preconditioner for the coarse level is defined by

$$\tilde{\mathbf{P}}_F = \begin{pmatrix} \tilde{\mathbf{P}}_f^{\text{sdc}} & & & \\ -\tilde{\mathbf{H}} & \tilde{\mathbf{P}}_f^{\text{sdc}} & & \\ & \ddots & \ddots & \\ & & -\tilde{\mathbf{H}} & \tilde{\mathbf{P}}_f^{\text{sdc}} \end{pmatrix},$$

or, in a more compact way, by

$$\tilde{\mathbf{P}}_F(\tilde{u}) = \left(\tilde{\mathbf{I}} - \Delta t (\mathbf{I}_L \otimes \tilde{\mathbf{Q}}_\Delta) \tilde{\mathbf{F}} - \mathbf{E} \otimes \tilde{\mathbf{H}} \right)(\tilde{u}).$$

Inverting this corresponds to a single inner iteration of SDC (a "sweep") on step 1, then sending forward the result to step 2, an SDC sweep there and so on. The parallel preconditioner on the fine level then simply reads

$$\mathbf{P}_F(u) = (\mathbf{I} - \Delta t (\mathbf{I}_L \otimes \mathbf{Q}_\Delta) \mathbf{F})(u).$$

Applying $\mathbf{P}_F$ on the fine level leads to L decoupled SDC sweeps, which can be run in parallel.

For PFASST, these two preconditioners and the levels they work on are coupled using a full approximation scheme (FAS) known from nonlinear multigrid theory [25]. Following [1] one iteration of PFASST can then be formulated in four steps:

1. the computation of the FAS correction τ^k, including the restriction of the fine value to the coarse level

$$\tau^k = \tilde{\mathbf{C}}_F(\mathbf{R}u^k) - \mathbf{R}\mathbf{C}_F(u^k),$$

2. the coarse sweep on the modified composite collocation problem on the coarse level

$$\tilde{\mathbf{P}}_F(\tilde{u}^{k+1}) = (\tilde{\mathbf{P}}_F - \tilde{\mathbf{C}}_F)(\tilde{u}^k) + \tilde{b} + \tau^k, \qquad (7)$$
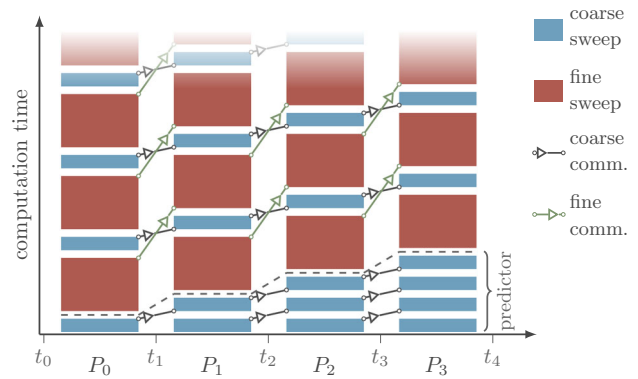

Fig. 1 Schematic view of PFASST on four processors. The figure was created with `pfasst-tikz` [14]

3. the coarse grid correction applied to the fine level value

$$u^{k+\frac{1}{2}} = u^k + \mathbf{T}(\tilde{u}^{k+1} - \mathbf{R}u^k), \qquad (8)$$

4. the fine sweep on the composite collocation problem on the fine level

$$\mathbf{P}_F(u^{k+1}) = (\mathbf{P}_F - \mathbf{C}_F)(u^{k+\frac{1}{2}}) + b. \qquad (9)$$

In Fig. 1, we see a schematic representation of the described steps. The time-step parallel procedure, which we describe here is also the same for all PFASST versions, that we will introduce later. It is common to use as many processors as time-steps: In the given illustration four processors work on four time-steps. Therefore the temporal domain is divided into four intervals, which are assigned to four processors $P_0, \ldots, P_3$. Every processor performs SDC sweeps on its assigned interval on alternating levels. The big red blocks represent fine sweeps, given by Eq. (9), and the small blue blocks coarse sweeps, given by Eq. (7).

The coarse sweep over all intervals is a serial process: after a processor finished its coarse sweeps, it sends forward its results to the next processor, which takes this result as an initial value for its own coarse sweeps. We see the communication in the picture represented by small arrows, which connect the coarse sweeps of each interval. In (7), the need for communication with a neighboring process is obvious, because $\tilde{\mathbf{P}}_F$ is not a (block-) diagonal matrix, but has entries on its lower block-diagonal. $\mathbf{P}_F$ on the other hand is block-diagonal, which means that the processors can calculate on the fine level in parallel. We see in (9) that there is only a connection to previous time-steps through the right-hand side, where we gather values from the previous time-step and iteration but not from the current iteration. The picture shows this connection by a fine communication, which forwards data from each fine sweep to the following fine sweep of the right neighbor. The fine and coarse calculations on every

processor are connected through the FAS corrections, which in our formula are part of the coarse sweep.

2.3 PFASST-Newton

For each coarse and each fine sweep within each PFASST iteration, System (7) and System (9), respectively, need to be solved. If f is a nonlinear function these systems are nonlinear as well. The obvious and traditional way to proceed in this case is to linearize the problem locally (i.e. for each time-step, at each quadrature node) using Newton's method. This way, PFASST is the outer solver with an inner Newton iteration. For triangular $\mathbf{Q}_\Delta$, the mth equation on the lth time-step on the coarse level reads

$$(1 - \Delta t\, \tilde{q}^{\Delta}_{l,m} \tilde{f})(\tilde{u}^{k+1}_{l,m}) = \tilde{u}^{k+1}_{l,0} + \Delta t \sum_{n=1}^{m-1} \tilde{q}^{\Delta}_{l,n} \tilde{f}(\tilde{u}^{k+1}_{l,n})$$
$$+ \tilde{c}(\tilde{u}^k)_{l,m},$$

where $\tilde{u}^{k+1}_{0,0} = \tilde{u}_{0,0}$ and $\tilde{c}(\tilde{u}^k)_{l,m}$ is the mth entry the lth block of $\tilde{c}(\tilde{u}^k) := (\tilde{\mathbf{P}}_F - \tilde{\mathbf{C}}_F)(\tilde{u}^k) + \tau^k$. This term gathers all values of the previous iteration. The first summand of the right-hand side of the coarse level equation corresponds to $\tilde{b}$ and $\tilde{\mathbf{H}}$, while the following sum comes from the lower triangular structure of $\tilde{\mathbf{Q}}_\Delta$.

For time-step l these equations can be solved one by one using Newton iterations and forward substitution. This is inherently serial, because the solution on the mth quadrature node depends on the solution at all previous nodes through the sum. Thus, while running parallel across the steps, each solution of the local collocation problem is found in serial. In the next section, we will present a novel way of applying Newton's method, which allows one to parallelize this part across the collocation nodes, joining parallelization across the step with parallelization across the method.

3 PFASST-ER

From the perspective of a single time-step $[t_l, t_{l+1}]$ or processor P_l, Eq. (7) on the coarse level for this step reads

$$\tilde{\mathbf{P}}^{\mathrm{sdc}}_f(\tilde{u}^{k+1}_l) - \tilde{u}^{k+1}_{l,0} = (\tilde{\mathbf{P}}^{\mathrm{sdc}}_f - \tilde{\mathbf{C}}^{\mathrm{coll}}_f)(\tilde{u}^k_l) + \tau^k_l,$$

where τ^k_l is the lth component of τ^k, belonging to the interval $[t_l, t_{l+1}]$. Note that the serial dependency is given by the term $\tilde{u}^{k+1}_{l,0}$, so that it does not depend on the solution $\tilde{u}^{k+1}_l$ of this equation and can thus be considered as part of a given right-hand side. On the fine level, this is even simpler, because there we have to solve

$$\mathbf{P}^{\mathrm{sdc}}_f(u^{k+1}_l) = (\mathbf{P}^{\mathrm{sdc}}_f - \mathbf{C}^{\mathrm{coll}}_f)(u^{k+\frac{1}{2}}_l) + u^{k+\frac{1}{2}}_{l,0},$$

where the $u^{k+\frac{1}{2}}_{l,0}$-term is independent of the current iteration (which, of course, leads to the parallelism on the fine level).

As we have seen above, the typical strategy would be to solve these systems line by line, node by node, using forward substitution and previous PFASST iterates as initial guesses. An alternative approach has been presented in [19], where each SDC iteration can be parallelized across the nodes. While this is trivial for linear problems, nonlinear ones require the linearization of the full equations, not node-wise as before. For the fine sweep, let

$$\mathbf{G}^{\mathrm{sdc}}_f(v) := \mathbf{P}^{\mathrm{sdc}}_f(v) - (\mathbf{P}^{\mathrm{sdc}}_f - \mathbf{C}^{\mathrm{coll}}_f)(u^{k+\frac{1}{2}}_l) - u^{k+\frac{1}{2}}_{l,0}$$

then a Newton step for $\mathbf{G}^{\mathrm{sdc}}_f(v) = 0$ is given by

$$\nabla \mathbf{G}^{\mathrm{sdc}}_f(v^j)e^j = -\mathbf{G}^{\mathrm{sdc}}_f(v^j),$$
$$v^{j+1} = v^j + e^j,$$

for Jacobian matrix $\nabla \mathbf{G}^{\mathrm{sdc}}_f(v^j)$ of $\mathbf{G}^{\mathrm{sdc}}_f$ evaluated at v^j. We have

$$\nabla \mathbf{G}^{\mathrm{sdc}}_f(v^j) = \nabla \mathbf{P}^{\mathrm{sdc}}_f(v^j)$$
$$= \mathbf{I} - \Delta t \mathbf{Q}_\Delta \nabla f(v^j)$$

for Jacobian matrix $\nabla f(v^j)$ of f evaluated at v^j which in turn is given by

$$\nabla f(v^j) = \mathrm{diag}(f'(v^j_1), \ldots, f'(v^j_M))^T.$$

There is still no parallelism to exploit, but when we replace the full Jacobian matrix $\nabla f(v^j)$ by the approximation $f'(v_{l,0})\mathbf{I}_M$, which is the derivative of f at the initial value for the current time-step, we can use

$$\nabla \mathbf{G}^{\mathrm{sdc}}_f(v^j) \approx \nabla \mathbf{G}^{\Delta\text{-QN}}_f(v_{l,0}) := \mathbf{I} - f'(v_{l,0})\Delta t \mathbf{Q}_\Delta$$

to establish a Quasi-Newton iteration as

$$\nabla \mathbf{G}^{\Delta\text{-QN}}_f(v_{l,0})e^j = -\mathbf{G}^{\mathrm{sdc}}_f(v^j),$$
$$v^{j+1} = v^j + e^j.$$

This decouples the evaluation of the Jacobian matrix from the current quadrature nodes and now $\mathbf{Q}_\Delta$ can be diagonalized, so that the inversion of $\nabla \mathbf{G}^{\Delta\text{-QN}}_f(v_{l,0})$ can be parallelized across the nodes. Note that there are other options for approximating the full Jacobian matrix. Most notably, in [9] the mean over all Jacobian matrices is used (there across the time-steps). We did not see any impact on the convergence when following this strategy, most likely because the number of quadrature nodes is typically rather low. The advantage of using the

initial value is that it reduces the number of evaluations of the Jacobian matrix, which also includes communication time.

Provided that $\mathbf{Q}_\Delta$ is diagonalizable, we can decompose it by $\mathbf{Q}_\Delta = \mathbf{V}_\Delta \mathbf{\Lambda}_\Delta \mathbf{V}_\Delta^{-1}$, where $\mathbf{\Lambda}_\Delta = \text{diag}((\mathbf{Q}_\Delta)_{ii})$ contains the eigenvalues $(\mathbf{Q}_\Delta)_{ii} \in \mathbb{R}$ of $\mathbf{Q}_\Delta$ and $\mathbf{V}$ contains its eigenvectors.

Using the given diagonalization the algorithm reads:

1. replace $\boldsymbol{r}^j = -\mathbf{G}_f^{\text{sdc}}(\boldsymbol{v}^j)$ by $\bar{\boldsymbol{r}}^j = -\mathbf{V}_\Delta^{-1}\mathbf{G}_f^{\text{sdc}}(\boldsymbol{v}^j)$ (serial),
2. solve $\left(\mathbf{I} - f'(v_{l,0})\Delta t\,\mathbf{\Lambda}_\Delta\right)\bar{\boldsymbol{e}}^j = \bar{\boldsymbol{r}}^j$ (parallel in M),
3. replace $\bar{\boldsymbol{e}}^j$ by $\boldsymbol{e}^j = \mathbf{V}_\Delta \bar{\boldsymbol{e}}^j$ (serial),
4. set $\boldsymbol{v}^{j+1} = \boldsymbol{v}^j + \boldsymbol{e}^j$ (parallel in M).

This can be iterated until a certain threshold is reached and then set $\boldsymbol{u}_l^{k+1} = \boldsymbol{v}^J$ to obtain the solution of the equation for the fine sweep. On the coarse level, the procedure is very similar, with a slightly different definition of $\tilde{\mathbf{G}}_{\tilde{f}}^{\text{sdc}}(\tilde{\boldsymbol{v}})$. In practice, choosing only a single Newton iteration (i.e. $J = 1$) is sufficient, because this is only the inner solver for an outer PFASST iteration. In all cases we have studied so far, using more inner iterations does not lead to a faster overall method.

This linearization and diagonalization strategy immediately suggests a second approach: instead of using $\mathbf{Q}_\Delta$ for the preconditioner, we can use the original quadrature matrix $\mathbf{Q}$ directly. The intention of using $\mathbf{Q}_\Delta$ in the first place was to obtain a preconditioner which allowed inversion using forward substitutions. Now, with diagonalization in place, this is no longer necessary. Instead, we can use

$$\mathbf{P}_f^{\text{coll}} := \mathbf{C}_f^{\text{coll}}$$

and thus

$$\mathbf{G}_f^{\text{coll}}(\boldsymbol{v}) := \mathbf{C}_f^{\text{coll}}(\boldsymbol{v}) - \boldsymbol{u}_{l,0}^{k+\frac{1}{2}}.$$

Note that this is just the lth block of the original composite collocation problem. Following the same ideas as before, we end up with

$$\nabla\mathbf{G}_f^{\text{coll}}(\boldsymbol{v}^j) \approx \nabla\mathbf{G}_f^{\text{QN}}(v_{l,0}) := \mathbf{I} - f'(v_{l,0})\Delta t\mathbf{Q},$$

which can be diagonalized using $\mathbf{Q} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$, where $\mathbf{\Lambda}$ is a diagonal matrix with eigenvalues $\lambda_i(\mathbf{Q}) \in \mathbb{C}$. The same idea can be applied to the coarse level sweep, of course. As a result, the original nonlinear SDC sweeps within PFASST are now replaced by Quasi-Newton iterations which can be done parallel across the nodes. We note that using simplified or Quasi-Newton methods for solving implicit Runge-Kutta schemes is a standard approach, as e.g. [26] shows. We further refer to [19] for more details on the idea of parallel SDC sweeps with $\mathbf{Q}$ and $\mathbf{Q}_\Delta$.

The question now is, how much the approximation of the Jacobians affects the convergence and runtime of the method and how all this compares to standard PFASST iterations. It is well known that for suitable right-hand sides and initial guesses the standard, unmodified Newton method converges quadratically while the Quasi-Newton method as well as SDC show linear convergence, see e.g. [11,13,24]. We will examine the impact of these approaches in the following section along the lines of two numerical examples. A more rigorous mathematical analysis is currently ongoing work, as it can be embedded into a larger convergence theory for PFASST with inner Newton-type solvers.

4 Numerical results

We apply PFASST and PFASST-ER to two different, rather challenging reaction-diffusion problems, starting with a detailed analysis of the parallelization strategies for the Allen–Cahn equation and highlighting differences to these findings for the Gray-Scott equations.

4.1 Allen–Cahn equation

We study the two-dimensional Allen–Cahn equation, which is given by

$$u_t = \Delta u + \frac{1}{\varepsilon^2}u(1 - u) \tag{10}$$

on the spatial domain $[-0.5, 0.5]^2$ and with initial condition

$$u_0 = \tanh\left(\frac{R_0 - (x^2 + y^2)}{\sqrt{2}\varepsilon}\right),$$

and periodic boundary conditions. We use simple second-order finite differences for discretization in space and take 256 elements in each dimension on the fine level and 128 on the coarse one. We furthermore use $M = 4$ Gauß-Radau nodes, set $\varepsilon = 0.04$, $\Delta t = 0.001 < \varepsilon^2$ and stop the simulation after 24 time-steps at $T = 0.024$. The initial condition describes a circle with a radius $R_0 = 0.25$, see e.g. [28].

Note that since our focus is on the temporal parallelization, the temporal resolution was chosen to be quite high in contrast to the spatial resolution. Errors in space and time are not balanced here (and in the following example), with the spatial error being much higher. This has been done deliberately to avoid higher computational costs. By increasing the accuracy in space, we would increase the amount of large parallelizable computations in relation to communication. This would in turn improve the overall parallel efficiency, which would in the end lead to even better scaling results. However, when using parallelization in space, all processors have very few
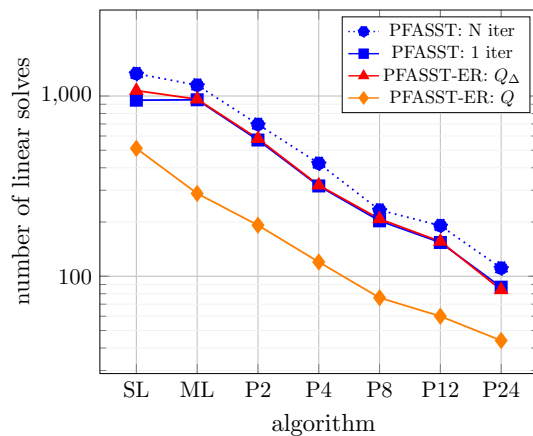
Fig. 2 Number of linear solves for the Allen–Cahn example, all methods run serial on the nodes

Fig. 3 Time to solution for Allen–Cahn with parallelization only across time-steps

degrees-of-freedom anyway, so our results may even reflect a "real" situation better.

The results we present in the following were computed with pySDC [20,21] on the supercomputer JURECA [12]. We run a serial single-level simulation using SDC ("SL" in the plots), a serial multi-level simulation using multi-level SDC ("ML", which is PFASST on one processor, see [23]) and parallel simulations with 2, 4, 8, 12 and 24 processors ("P2" to "P24"), all until a given residual tolerance of 10^{-10} is reached.

If less processors than time-steps are used, the time domain is split into blocks of parallel PFASST runs. These are handled sequentially, using the solution of the previous block as the initial data for the next one. For example, 6 processors work on the first 6 time-steps until convergence and the solution is used as new initial condition for the next block of 6 time-steps. This is repeated until all 24 time-steps have been completed.

In Fig. 2 we show the maximum number of linear solves which were performed by the slowest processor (i.e. last processor in time) for different versions of the solvers, aggregated over all its time-steps and quadrature nodes, over all outer and inner iterations.

Here, two versions of the original PFASST algorithm are run: The first one performs exactly one inner Newton iteration in every PFASST iteration; this version is labeled as "PFASST: 1 iter". In contrast, "PFASST: N iter" performs as many inner Newton iterations required so that the residual of the nonlinear inner problem is less than 10^{-11}. Both PFASST versions use the quadrature matrix $\mathbf{Q}_\Delta^{LU}$ from Eq. (5) inside the preconditioner. For PFASST-ER we also show two variants: The PFASST-ER algorithm, which uses the original $\mathbf{Q}$ inside the preconditioner is labeled as "PFASST-ER: Q" and the one which uses $\mathbf{Q}_\Delta^{LU}$ is labeled as "PFASST-ER: Q_Δ". Solving the innermost linear systems is done using GMRES with a tolerance of 10^{-12} in all cases.
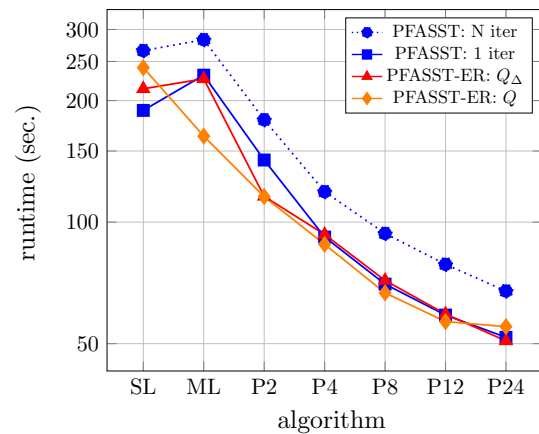
We can see that performing more than one inner Newton iteration ("PFASST: N iter" vs. "PFASST: 1 iter") does not improve the convergence of the overall algorithm. Although it is possible that by increasing the number of inner Newton iterations the number of outer PFASST iterations decreases, the total effort, which can be measured by the total number of linear solves, increases due to a higher number of inner Newton iterations.

Using the Quasi-Newton approach with the same preconditioner instead of the classical Newton solver ("PFASST-ER: Q_Δ" vs. "PFASST: 1 iter") shows little effect on the total iteration numbers, but using the original quadrature matrix $\mathbf{Q}$ instead of $\mathbf{Q}_\Delta^{LU}$ inside the preconditioner ("PFASST-ER: Q" vs. "PFASST-ER: Q_Δ") greatly reduces the number of iterations.

However, without parallelization one iteration of PFASST-ER with $\mathbf{Q}$ is in general more expensive than one iteration of the other algorithms, because it requires the solution of a full system via diagonalization instead of stepping through a triangular system via forward substitution.

In Fig. 3, we thus examine whether the lower number of more expensive iterations actually pays off. The plot shows results for the same setup as Fig. 2, but now we focus on the runtime instead of the iteration numbers. We only consider parallelization across the time-steps to compare the impact of the algorithmic change first. We see that despite the fact that the iterations are more expensive, PFASST-ER with $\mathbf{Q}$ already in this example shows a lower runtime than the original PFASST method. This is also true when using $\mathbf{Q}_\Delta$ instead of $\mathbf{Q}$.

At this point, we have not yet considered the additional direction of concurrency exposed by PFASST-ER. For that, we next compare different distributions of up to 24 cores on the 4 quadrature nodes and the 24 time-steps. All divisions of 24 were tested with all possible distributions.
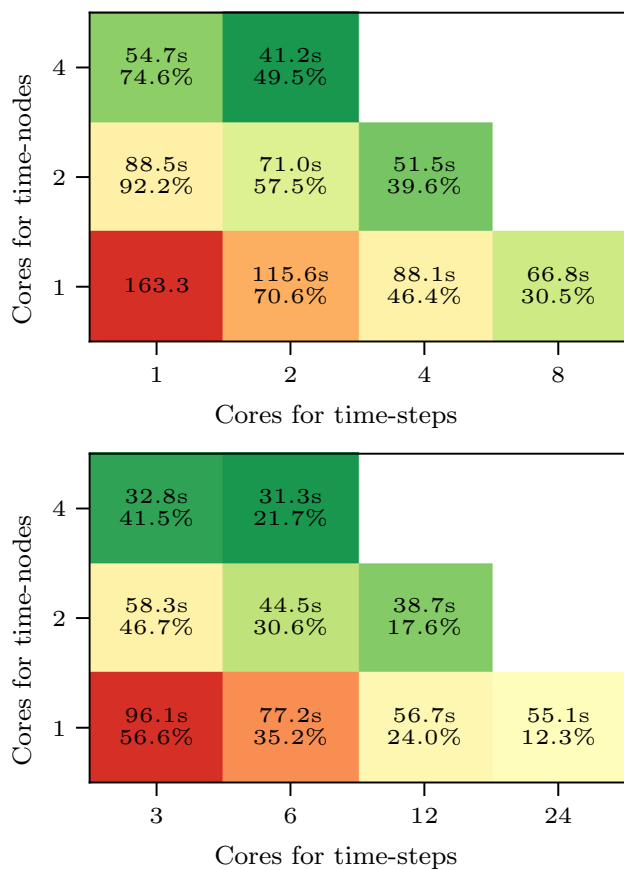
Fig. 4 Runtimes in seconds (first number) and efficiencies (second number) with different distribution of cores using PFASST-ER with $\mathbf{Q}$ for the Allen–Cahn equation

The two plots in Fig. 4 show different combinations of cores used for step-parallelization (x-axis) and for node-parallelization (y-axis) with PFASST-ER and $\mathbf{Q}$. Multiplying the numbers on both axes gives the total number of cores used for this simulation. This is also the reason why there are two plots, because not all combinations are actually possible or meaningful. Within each colored block the total runtime (in seconds, first number) and parallel efficiencies (second number) for this setup are given. We can see that using all available cores for parallelization across the step is by far not the most efficient choice. In turn, more than 4 cores cannot be used for parallelization across the nodes, although 4 gives the best speedup. Indeed, the best combination for this problem is to maximize node-parallelization first and then add step-parallelization (31.3 seconds with 4 cores on the nodes and 6 on the steps, lower picture). This is about 1.8 times faster than using 24 cores for the steps alone and more than 5 times faster than the serial PFASST-ER run.

Although using $\mathbf{Q}$ instead of $\mathbf{Q}_\Delta$ in PFASST-ER is faster for this example, it is quite revealing to repeat the simulations using $\mathbf{Q}_\Delta$. These results are shown in Fig. 5 and it is obvious that using as many cores as possible for the parallelization

across the nodes now is not the optimal strategy. Here, using 2 cores on the nodes and 12 on the steps is the most efficient combination, albeit still significantly slower than using PFASST-ER with $\mathbf{Q}$, even with the same combination. The reason for this potentially surprising result is that solving the innermost linear systems heavily depends on the structure of these systems, in particular when using an iterative solver like GMRES. Moreover, initial guesses are a crucial factor, too. For PFASST-ER, we use the current solution at node zero of the respective time-step as the initial guess. This is particularly suitable for the closest first nodes, but potentially less so for later ones. While both effects did not lead to significant variations in the time spent for solving the linear systems when using $\mathbf{Q}$, it does produce a severe load imbalance when using $\mathbf{Q}_\Delta$. More specifically, using 4 cores for the nodes and only 1 for the time-steps, i.e. exploiting only parallelization across the nodes, the first core takes about 118.2 seconds for all linear system solves together at the first node, while the last core takes about 194.6 seconds on the last node. Therefore, using 2 cores on the nodes, which enables a better load distribution is the ideal choice. One possibility would be that core 1 deals with nodes 1 and 4 and core 2 with 2 and 3, but because node 3 and 4 are very close to each other and the corresponding calculations are almost equally expensive also an alternating distribution is an ideal choice. This is precisely what has been done for Fig. 5, leading to the best speedup with 2 cores on the nodes. For other examples, an optimal distribution might be more difficult to find.

In Fig. 6 we now summarize the best results: PFASST with one inner Newton iteration in comparison to PFASST-ER using $\mathbf{Q}_\Delta$ and 2 cores on the nodes and PFASST-ER using $\mathbf{Q}$ with 4 cores on the node. The plot shows the simulation time for each variant based on the number of processors used in total. We see that PFASST-ER is always much more time efficient in doing the calculations than PFASST, with another significant gain when using $\mathbf{Q}$ instead of $\mathbf{Q}_\Delta$. Now, since PFASST-ER adds another direction of parallelization compared to PFASST, we can not only increase parallel efficiency as shown, but also extend the number of usable cores to obtain a better time-to-solution. This has been done in Fig. 7: taking 48 or 96 cores in total further reduces the computing time for 24 time-steps. With PFASST-ER, the number of resources that can be used for parallel-in-time integration is no more limited by the number of time-steps, but can be increased by the factor given by the number of quadrature nodes.

4.2 Gray-Scott equations

The second example we present here is the Gray-Scott system [16], which is given by
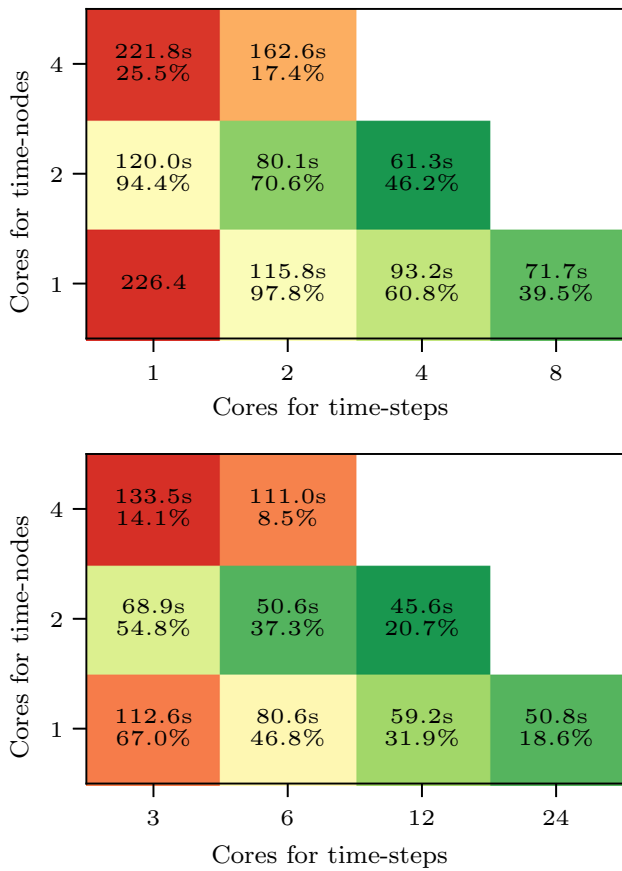
$$u_t = D_u \Delta u - 2uv + F(1 - u),$$

Fig. 5 Runtimes in seconds (first number) and efficiencies (second number) with different distribution of cores using PFASST-ER with $\mathbf{Q}_\Delta$ for the Allen–Cahn equation
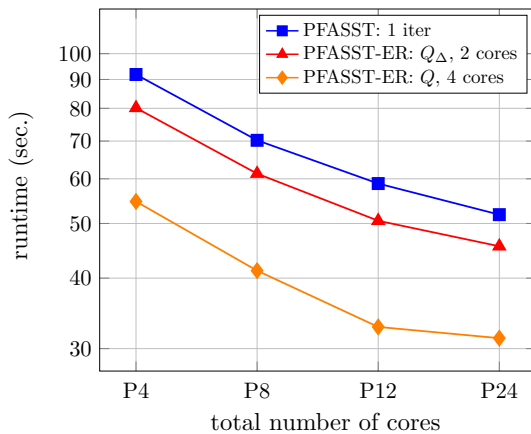

Fig. 6 Runtimes for the three best variants, Allen–Cahn example

$$v_t = D_v \Delta v + 2uv - (F + K)v,$$

on the spatial domain $[0, 1] \times [0, 1]$, with periodic boundary conditions. As initial condition we choose a circle with radius 0.05 centred in the spatial domain, where $u = 0.5$ and $v = 0.25$ on the inside, and $u = 1.0$ and $v = 0$ outside of
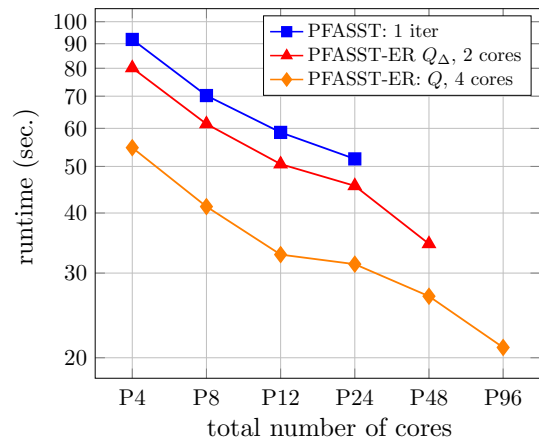
Fig. 7 Runtimes for different number of processors, Allen–Cahn example
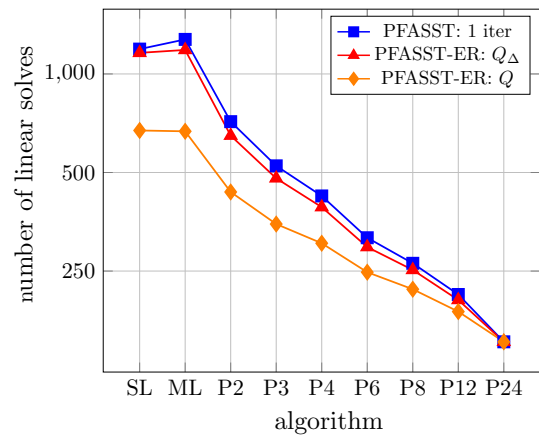
Fig. 8 Number of linear solves for the Gray-Scott example, all methods run serial on the nodes

this circle. We use $D_u = 10^{-4}$, $D_v = 10^{-5}$ and set a feed rate of $F = 0.0367$ and a kill rate of $K = 0.0649$. This leads after some time to a process similar to cellular division and is known as "mitosis". We discretize the spatial domain with 128 points in each dimension on the fine level and with 64 on the coarse one, using standard finite differences. We discretize every time-step of size $\Delta t = 1$ with 4 quadrature nodes and run the simulation again for 24 time-steps.

The results are similar to the ones for the Allen–Cahn equation in the previous section. We will omit the case of PFASST with more than one inner Newton iteration, though.

We start again by looking at the total number of linear solves the different algorithms need to perform. Figure 8 shows the number of linear solves for the methods, which run until a residual tolerance of 10^{-12} is reached. The results look quite similar to the ones for the previous example, with one critical difference: The difference between the $\mathbf{Q}$-variant of PFASST-ER and the other algorithms becomes smaller more rapidly the more parallel time-steps are used. There
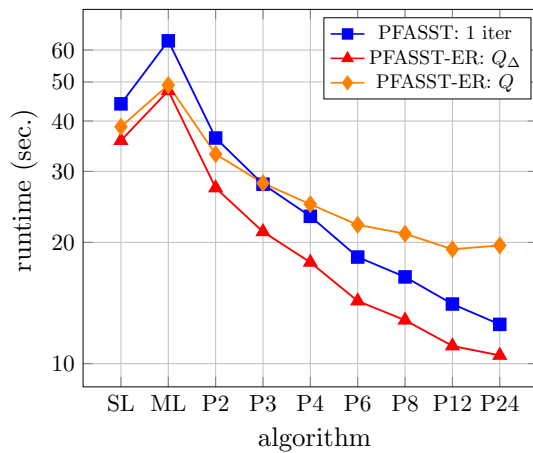
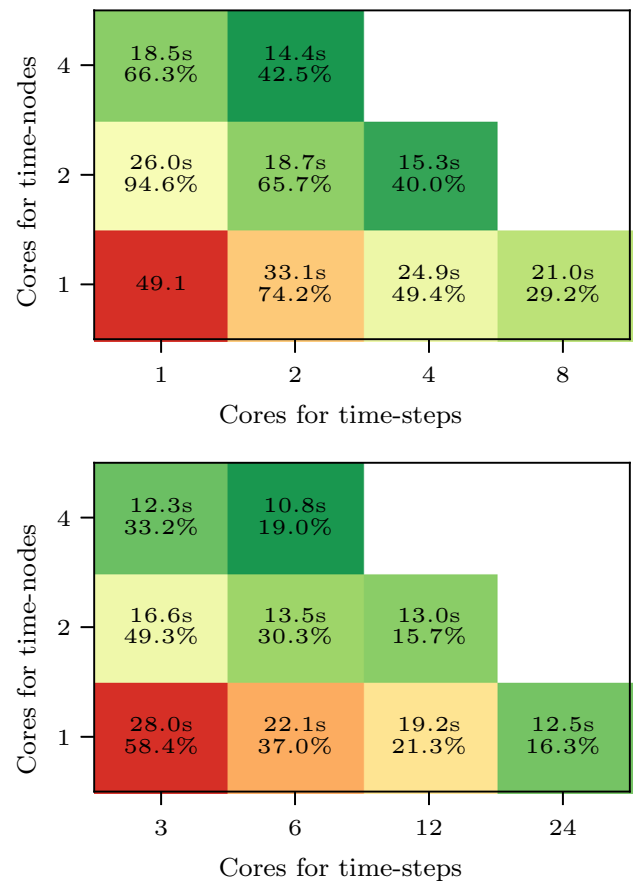**Fig. 9** Time to solution for Gray-Scott with parallelization only across time-steps

Fig. 10 Runtimes in seconds (first number) and efficiencies (second number) with different distribution of cores using PFASST-ER with **Q** for the Gray-Scott equations

is no obvious explanation (at least, obvious to us) for this behavior, though. The more time steps are approximated simultaneously, the less suitable u_0 works as initial value for more distant time-steps. Although the full Newton and the Quasi-Newton methods differ by an order of convergence in theory, in our scenario this seems relevant only for good initial values. One can expect that the runtime will increase when using PFASST-ER with **Q**, while it stayed about the same in the case of the Allen–Cahn example.

This is precisely what we can see in Fig. 9. The more parallel time-steps are run, the less efficient PFASST-ER with **Q** in this variant becomes. Already at 3 parallel steps, it is as costly as the original PFASST version, at least when parallelization across the nodes is not considered.

Now, adding node-parallelization, the findings are again similar to the ones in the previous section: Figure 10 shows that PFASST-ER with **Q** is still more efficient than using PFASST. In particular, using more cores on the nodes is better and the best combination is again 4 cores on the nodes and 6 on the steps. Again, this changes when considering PFASST-ER with $\mathbf{Q}_\Delta$ as in Fig. 11, where the ideal setup uses only 2 cores on the nodes, but 12 on the steps. This is again due to load imbalances of the innermost linear solves. However, note the key difference to the previous results: The fastest run of the $\mathbf{Q}_\Delta$-variant is now faster than the one of the **Q**-variant.

In Fig. 12 we now give an overview of the best results: If we use parallelism across the nodes in a suitable way, both PFASST-ER versions are more efficient based on the simulation time than the classical PFASST algorithm. Both can be used to extend the scaling capabilities beyond the number of time-steps, and both scale rather well in this regime. Note, however, that the $\mathbf{Q}_\Delta$-variant can here only leverage 2×24 cores. It is then faster than the **Q**-variant with twice as many cores.

5 Conclusion and outlook

Today's supercomputers are designed with an ever increasing number of processors. Therefore we need our software and the underlying numerical algorithms to handle this increasing degree of parallelism. Time-parallel integrators are one promising research direction, with quite a number of different approaches. Some approaches parallelize each individual time-step and others act on multiple time-steps simultaneously. In this paper we have introduced a solver that works in parallel across the method as well as across the steps. More precisely, we combine node-parallel spectral deferred corrections with the parallel full approximation scheme in space and time. While PFASST allows one to compute multiple time-steps simultaneously and target large-scale parallelism in time, the new version called PFASST-ER presented here extends this idea with an efficient small-scale parallelization for every single time-step itself. The scaling studies show that a combination of both concepts seems to be the most efficient way to solve time-dependent PDEs. Here we tested two different preconditioners: ones using the traditional, triangular
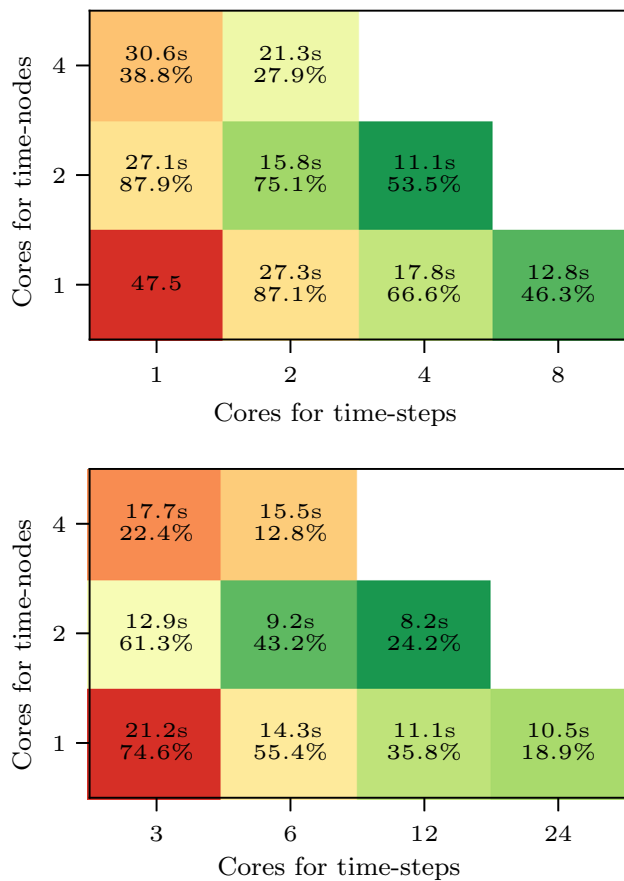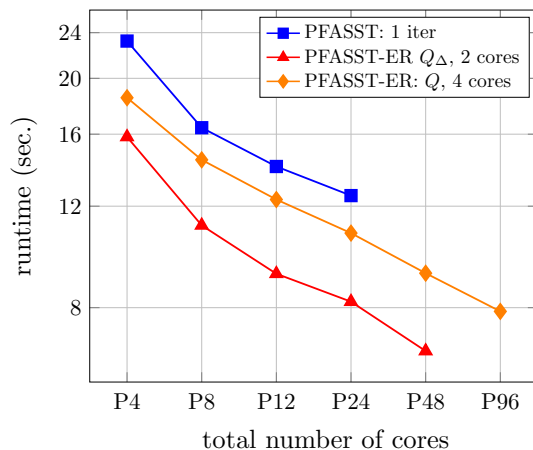
Fig. 11 Runtimes in seconds (first number) and efficiencies (second number) with different distribution of cores using PFASST-ER with $\mathbf{Q}_\Delta$ for the Gray-Scott equations

Fig. 12 Runtimes for the three best variants, Gray-Scott example

quadrature matrix $\mathbf{Q}_\Delta$, generated by a LU-decomposition and one using the original matrix $\mathbf{Q}$. Both can be diagonalized and used as parallel-across-the-node preconditioners. For the $\mathbf{Q}_\Delta$-preconditioner, we saw load imbalances when using an inner iterative linear solver, but by grouping nodes we still can speed up the simulation beyond the number

of parallel time-steps. For the $\mathbf{Q}$-preconditioner, the overall number of iterations was lower and time-to-solution was faster. Adding node-parallelization, parallel efficiency can be increased and speedup extended when compared to PFASST. Both PFASST-ER versions lead in the end to better scaling results than the classical PFASST algorithm. PFASST-ER Q especially offers an almost equal distribution of work for iterative linear solvers with respect to the individual quadrature nodes of a time-step. This advantage makes this algorithm particularly flexible and can be used for any number of quadrature points.

PFASST-ER is particularly favorable if an increase in parallelism across the steps would lead to a severe increase in the number of iterations. This could be due to e.g. the type of the equation or the coarsening strategy. During our experiments we saw that it is not clear *a priori* which combination of node- and step-parallelization is the most efficient one. This could lead to many, potentially irrelevant runs to find the sweet spot. Here, a performance model and a suitable convergence theory are needed to at least narrow down the relevant options. This has to be accompanied by more numerical tests, relating e.g. model parameters with load imbalances, to identify the limits of this approach.

References

1. Bolten, M., Moser, D., Speck, R.: A multigrid perspective on the parallel full approximation scheme in space and time. Numer. Linear Algebra Appl. **24**(6), e2110 (2017). https://doi.org/10.1002/nla.2110
2. Bolten, M., Moser, D., Speck, R.: Asymptotic convergence of the parallel full approximation scheme in space and time for linear problems. Numer. Linear Algebra Appl. **25**(6), e2208 (2018). https://doi.org/10.1002/nla.2208
3. Burrage, K.: Parallel methods for ODEs. Adv. Comput. Math. **7**, 1–3 (1997)

4. Christlieb, A.J., Macdonald, C.B., Ong, B.W.: Parallel high-order integrators. SIAM J. Sci. Comput. **32**(2), 818–835 (2010)
5. Clarke, A.T., Davies, C.J., Ruprecht, D., Tobias, S.M.: Parallel-in-time integration of kinematic dynamos (2019). arXiv:1902.00387 [physics.comp-ph]
6. Emmett, M., Minion, M.L.: Toward an efficient parallel in time method for partial differential equations. Commun. Appl. Math. Comput. Sci. **7**, 105–132 (2012)
7. Falgout, R.D., Friedhoff, S., Kolev, T.V., MacLachlan, S.P., Schroder, J.B., Vandewalle, S.: Multigrid methods with space–time concurrency. Comput. Vis. Sci. **18**(4–5), 123–143 (2017)
8. Gander, M.J.: 50 years of time parallel time integration. In: Multiple Shooting and Time Domain Decomposition. Springer (2015). https://doi.org/10.1007/978-3-319-23321-5_3
9. Gander, M.J., Halpern, L., Ryan, J., Tran, T.T.B.: A direct solver for time parallelization. In: Dickopf, T., Gander, M.J., Halpern, L., Krause, R., Pavarino, L.F. (Eds) Domain Decomposition Methods in Science and Engineering XXII, pp. 491–499. Springer (2016). https://doi.org/10.1007/978-3-319-18827-0_50
10. Huang, J., Jia, J., Minion, M.: Accelerating the convergence of spectral deferred correction methods. J. Comput. Phys. **214**(2), 633–656 (2006)
11. Jackson, K .R., Kværnø, A., Nørsett, S .P.: The use of butcher series in the analysis of newton-like iterations in Runge–Kutta formulas. Appl. Numer. Math. **15**(3), 341–356 (1994)
12. Jülich Supercomputing Centre. JURECA: General-purpose supercomputer at Jülich Supercomputing Centre. J. Large-Scale Res. Facil. **2**(A62) (2016). https://doi.org/10.17815/jlsrf-2-121
13. Kelley, C.T.: Iterative Methods for Linear and Nonlinear Equations. Number 16 in Frontiers in Applied Mathematics. SIAM (1995)
14. Koehler, F.: PFASST TikZ. https://github.com/Parallel-in-Time/pfasst-tikz (2015)
15. Lions, J.-L., Maday, Y., Turinici, G.: A "parareal" in time discretization of PDE's. Comptes Rendus de l'Académie des Sciences—Series I: Mathematics. https://doi.org/10.1016/S0764-4442(00)01793-6
16. Pearson, J.E.: Complex patterns in a simple system. Science **261**(5118), 189–192 (1993)
17. Ruprecht, D., Speck, R.: Spectral deferred corrections with fast-wave slow-wave splitting. SIAM J. Sci. Comput. **38**(4), A2535–A2557 (2016)
18. Ruprecht, D., Speck, R., Emmett, M., Bolten, M., Krause, R.: Poster: extreme-scale space-time parallelism. In: Proceedings of the 2013 Conference on High Performance Computing Networking, Storage and Analysis Companion, SC '13 Companion (2013). http://sc13.supercomputing.org/sites/default/files/PostersArchive/tech_posters/post148s2-file3.pdf
19. Speck, R.: Parallelizing spectral deferred corrections across the method. Comput. Vis. Sci. **19**(3–4), 75–83 (2018). https://doi.org/10.1007/s00791-018-0298-x
20. Speck, R.: Algorithm 997: pySDC-prototyping spectral deferred corrections. ACM Trans. Math. Softw. (2019). https://doi.org/10.1145/3310410
21. Speck, R.: Website for pySDC (2019). https://parallel-in-time.org/pySDC/. Accessed 27 November 2019
22. Speck, R., Ruprecht, D., Krause, R., Emmett, M., Minion, M., Winkel, M., Gibbon, P.: A massively space–time parallel N-body solver. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pp. 92:1–92:11. IEEE Computer Society Press, Los Alamitos, CA, USA (2012). ISBN 978-1-4673-0804-5. http://dl.acm.org/citation.cfm?id=2388996.2389121. event-place: Salt Lake City, Utah
23. Speck, R., Ruprecht, D., Emmett, M., Minion, M.L., Bolten, M., Krause, R.: A multi-level spectral deferred correction method. BIT Numer. Math. **55**, 843–867 (2015). https://doi.org/10.1007/s10543-014-0517-x
24. Tang, T., Xie, H., Yin, X.: High-order convergence of spectral deferred correction methods on general quadrature nodes. J. Sci. Comput. **56**(1), 1–13 (2013)
25. Trottenberg, U., Oosterlee, C., Schuller, A.: Multigrid. Academic Press, London (2000)
26. Wanner, G., Hairer, E.: Solving Ordinary Differential Equations II. Springer, Berlin (1996)
27. Weiser, M.: Faster SDC convergence on non-equidistant grids by DIRK sweeps. BIT Numer. Math. **55**(4), 1219–1241 (2014)
28. Zhang, J., Du, Q.: Numerical studies of discrete approximations to the Allen–Cahn equation in the sharp interface limit. SIAM J. Sci. Comput. **31**(4), 3042–3063 (2009). https://doi.org/10.1137/080738398.