Algorithm 956: PAMPAC, A Parallel Adaptive Method for Pseudo-Arclength Continuation

D. A. ARULIAH and LENNAERT VAN VEEN, University of Ontario Institute of Technology
ALEX DUBITSKI, Amadeus R&D, Toronto

Pseudo-arclength continuation is a well-established method for generating a numerical curve approximating the solution of an underdetermined system of nonlinear equations. It is an inherently sequential predictor-corrector method in which new approximate solutions are extrapolated from previously converged results and then iteratively refined. Convergence of the iterative corrections is guaranteed only for sufficiently small prediction steps. In high-dimensional systems, corrector steps are extremely costly to compute and the prediction step length must be adapted carefully to avoid failed steps or unnecessarily slow progress. We describe a parallel method for adapting the step length employing several predictor-corrector sequences of different step lengths computed concurrently. In addition, the algorithm permits intermediate results of correction sequences that have not converged to seed new predictions. This strategy results in an aggressive optimization of the step length at the cost of redundancy in the concurrent computation. We present two examples of convoluted solution curves of high-dimensional systems showing that speed-up by a factor of two can be attained on a multicore CPU while a factor of three is attainable on a small cluster.

Categories and Subject Descriptors: G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—Continuation (homotopy) methods; G.4 [**Mathematical Software**]: Parallel and Vector Implementations

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Pseudo-arclength continuation, parallel computing, adaptivity

ACM Reference Format:
D. A. Aruliah, Lennaert van Veen, and Alex Dubitski. 2016. Algorithm 956: PAMPAC, a parallel adaptive method for pseudo-arclength continuation. ACM Trans. Math. Softw. 42, 1, Article 8 (February 2016), 18 pages.
DOI: http://dx.doi.org/10.1145/2714570

1. INTRODUCTION

Continuation or *homotopy* problems arise naturally in numerous application domains. They are used to study the parameter dependence of solutions of nonlinear problems by continuously morphing between different systems of equations. For instance, in the numerical study of the Navier-Stokes equations for fluid motion, the Reynolds number Re often appears as a parameter. Certain computations—for example, those of time-periodic solutions or travelling waves—are significantly less challenging at low Reynolds numbers than at high Reynolds numbers, in which dynamical processes occur on a larger range of spatial scales. Continuation can be used to extend the results obtained at some low Reynolds number into the physically more interesting regime

This work is supported by the Natural Sciences and Engineering Research Council.
Authors' addresses: D. A. Aruliah, Continuum Analytics, #200-375 Water St, Vancouver, BC, Canada, V6B 0M9; email: dhavide@continuum.io; L. van Veen, Faculty of Science, UOIT, 2000 Simcoe St N, Oshawa, ON, Canada, L1H 7K4; email: Lennaert.vanVeen@uoit.ca; A. Dubitski, 284 Westway Crescent, Concord, ON, Canada, L4K 5S9; email: huazdsfcvy@gmail.com.

ACM 0098-3500/2016/02-ART8 $15.00
DOI: http://dx.doi.org/10.1145/2714570

by constructing a homotopy between turbulent flows with distinct characteristics. Homotopies are also used to compute and contrast similarities and differences of flows in disparate geometries (see Kawahara et al. [2012] and references therein).

In mathematical terms, homotopy—or continuation problems—are nonlinear systems of equations for which the number of equations is one fewer than the number of variables. That is, given a mapping $\mathbf{F} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, the equation

$$\mathbf{F}(\mathbf{x}, \lambda) = \mathbf{0} \tag{1}$$

with vector $\mathbf{x} \in \mathbb{R}^n$ and scalar $\lambda \in \mathbb{R}$ defines a continuation problem. In geometric terms, Equation (1) implicitly defines a one-dimensional curve in $\mathbb{R}^{n+1}$ under suitable smoothness and consistency properties of the mapping $\mathbf{F}$.

The essential idea of continuation, or homotopy, is to follow this curve of solutions in Equation (1). We assume that the parameter λ lies in some specified interval $[\lambda_{\min}, \lambda_{\max}] \subseteq \mathbb{R}$. Often, the problem can be formulated so that solving Equation (1) for $\mathbf{x}$ is easy for some $\lambda^* \in [\lambda_{\min}, \lambda_{\max}]$ and that the goal is to obtain a solution $\mathbf{x}$ when $\lambda = \lambda_{\min}$ or $\lambda = \lambda_{\max}$. Thus, continuation is the process of gradually morphing the solution of a straightforward problem into the solution of a formidable problem in small parameter increments.

Numerical continuation refers to families of numerical algorithms for generating points on the solution curve. Natural continuation and pseudo-arclength continuation are examples of predictor-corrector methods [Allgower and Georg 2003; Govaerts 2000; Kuznetsov 1998]. In the prediction stage, a putative new point on the curve is computed; in the correction stage, the putative candidate is iteratively refined until a solution for Equation (1) is found. Such algorithms are inherently sequential: previously computed points are used to predict the next solution point on the curve. Parallelization can be introduced within corrector iterations but the predictor steps need to be computed in sequence.

More importantly, predictor-corrector methods rely on adaptive step-size selection to make optimal progress in moving along the curve of solutions [Allgower and Georg 2003]. Strategies for adapting step-size selection are largely heuristic: when corrector steps fail to converge, the predictor step is rejected, the step size is reduced, and a new prediction step is generated. This turns out to be the principal bottleneck in many continuation problems: computation time devoted to nonconvergent corrector steps is wasted. As such, identifying strategies to improve the performance of predictor-corrector schemes by reducing the cost of failed predictor steps is a significant challenge in modern high-performance computing for numerical continuation problems.

We describe in the present work a parallel software library and the underlying algorithms that extend adaptive predictor-corrector methods to amortize the cost of rejected predictor steps. Specifically, we compute several predictor steps of different step sizes in parallel on distinct processors. At the same time, intermediate corrector iterates can seed new predictor steps. This strategy is most effective when corrector steps are costly. In particular, the time for a single corrector iteration should be much larger than the communication time between processors and should not depend sensitively on the step size. Moreover, the curve defined by Equation (1) should have a curvature that varies dramatically so that the optimal continuation step size also changes significantly along the solution curve.

2. BACKGROUND

We briefly review existing numerical continuation algorithms—notably, natural parameter and pseudo-arclength continuation—before describing our parallel adaptive algorithm. The prototypical problem is of the form in Equation (1), where $\lambda \in [\lambda_{\min}, \lambda_{\max}] \subseteq \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{F} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$. We sometimes write the underlying Equation (1) in the form

$$\mathbf{F}(\mathbf{z}) = \mathbf{0}, \text{ where } \mathbf{z} = (\mathbf{x}, \lambda) \in \mathbb{R}^{n+1}. \tag{2}$$

That is, we treat the concatenation of the n-vector $\mathbf{x}$ and the scalar λ as a single $(n+1)$-vector $\mathbf{z}$ while using the same symbol $\mathbf{F}$ to denote the mapping. Technically, this notation is ambiguous, but the meaning is clear from the context. Also, we denote appropriately sized matrices representing Jacobian derivatives by

$$\mathbf{F}_\lambda \equiv \frac{\partial \mathbf{F}}{\partial \lambda} \in \mathbb{R}^{n\times 1}, \quad \mathbf{F}_\mathbf{x} \equiv \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \in \mathbb{R}^{n\times n}, \quad \text{and} \quad \mathbf{F}_\mathbf{z} \equiv \frac{\partial \mathbf{F}}{\partial \mathbf{z}} \in \mathbb{R}^{n\times (n+1)}, \quad \text{respectively.} \tag{3}$$

For concreteness, the goal is to find a vector $\mathbf{x} = \mathbf{x}(\lambda_{\max})$ satisfying $F(\mathbf{x}(\lambda_{\max}), \lambda_{\max}) = \mathbf{0}$ when λ initially starts from $\lambda^* = \lambda_{\min}$ (i.e., the curve is traversed with λ increasing, at least initially). As a first obvious method for constructing the numerical curve of solutions, one can increment the parameter λ gradually from $\lambda_{\min}$ to $\lambda_{\max}$. That is, given a point $(\mathbf{x}, \lambda) \in \mathbb{R}^{n+1}$ known to lie on the curve, a new point on the curve $(\boldsymbol{\xi}, \lambda + h)$ is found

(1) by incrementing λ by a small amount $h > 0$; and
(2) by solving the n equations $\mathbf{F}(\boldsymbol{\xi}, \lambda + h) = \mathbf{0}$ for the unknown $\boldsymbol{\xi} \in \mathbb{R}^n$.

This approach is referred to as *natural parameter continuation* (or, more simply, *natural continuation*) [Allgower and Georg 2003; Govaerts 2000; Kuznetsov 1998]. Algorithm 1 is a high-level description of natural continuation. Fixing the known parameter value $\lambda \mapsto \lambda + h$ in the underdetermined nonlinear system $\mathbf{F}(\boldsymbol{\xi}, \lambda) = \mathbf{0}$ yields a closed $n \times n$ system in the n unknown components of $\boldsymbol{\xi}$. Intuitively, if h is sufficiently small, the vector $\boldsymbol{\xi}$ should be easy to obtain using a quadratically convergent Newton iteration (e.g., see Kelley [2003]).

ALGORITHM 1: Natural Parameter Continuation

```
   Input: $[\lambda_{\min}, \lambda_{\max}] \subseteq \mathbb{R}$; vector $\mathbf{x}^{(0)} \in \mathbb{R}^n$, where $\mathbf{F}(\mathbf{x}^{(0)}, \lambda_{\min}) = \mathbf{0}$; step size $h > 0$
   Output: vector $\mathbf{x}^{(k)} \in \mathbb{R}^n$ and scalar $\lambda^{(k)} \geq \lambda_{\max}$, such that $\mathbf{F}(\mathbf{x}^{(k)}, \lambda^{(k)}) = \mathbf{0}$
1  $k \leftarrow 0$
2  $\lambda^{(0)} \leftarrow \lambda_{\min}$                                % initialization
3  while $\lambda^{(k)} < \lambda_{\max}$ do          % loop to generate successive points on curve
4      $\lambda \leftarrow \lambda^{(k)} + h$                                % predictor step
5      Iteratively solve $n \times n$ nonlinear system of equations
                         $\mathbf{F}(\boldsymbol{\xi}, \lambda) = \mathbf{0}$
       to obtain $\boldsymbol{\xi} \in \mathbb{R}^n$ starting from initial iterate $\boldsymbol{\xi}^{(0)} = \mathbf{x}^{(k)}$     % corrector steps
6      if iteration in line 5 converges to $\boldsymbol{\xi}$ then           % accept next point on curve
7          $\mathbf{x}^{(k+1)} \leftarrow \boldsymbol{\xi}$
8          $\lambda^{(k+1)} \leftarrow \lambda$
9          $k \leftarrow k + 1$
10     else
11         Reduce step size $h$                        % reject predictor step & repeat
12 return $\mathbf{x}^{(k)}, \lambda^{(k)}$
```

Natural continuation is conceptually simple and easy to implement; however, it breaks down when the solution curve admits a *fold point* (i.e., a point $(\mathbf{x}, \lambda)$ where the Jacobian matrix $\mathbf{F}_\mathbf{x}(\mathbf{x}, \lambda)$ is singular; see Dickson et al. [2007] for an alternative characterization of fold points). At a fold point, systematically incrementing λ, as in Algorithm 1, yields an inconsistent system of nonlinear equations that cannot be solved regardless of how small h is. Fold points do occur in practical continuation problems; thus, other continuation strategies need to be devised [Yang and Keller 1986; Doedel et al. 2009].

Pseudo-arclength continuation (as outlined in Algorithm 2; see Allgower and Georg [2003], Dickson et al. [2007], and Keller [1977]) is a standard approach to circumvent fold point singularities. Under the assumption that both $\mathbf{x}$ and λ are smooth functions of arclength, pseudo-arclength continuation uses the unit direction $\mathbf{T} \in \mathbb{R}^{n+1}$ tangent to the curve for prediction. The term "pseudo-arclength" applies because the step size h—that is, the Euclidean distance in $\mathbb{R}^{n+1}$ between successive points on the numerical curve—approximates the arclength as measured along the curve. The tangent direction $\mathbf{T}$ can be determined by computing a null vector of the $n \times (n+1)$ Jacobian matrix $\mathbf{F_z}$ or by computing finite differences between successive points on the curve [Allgower and Georg 2003]. The underdetermined nonlinear system of Equation (1) is then closed by requiring that the solution $(\mathbf{x}, \lambda)$ sought must lie in the hyperplane orthogonal to the tangent direction $\mathbf{T}$ at distance h from the last known point; see Line 7 of Algorithm 2.

ALGORITHM 2: Pseudo-arclength Continuation

```
   Input: $[\lambda_{\min}, \lambda_{\max}] \subseteq \mathbb{R}$; vector $\mathbf{x}^{(0)} \in \mathbb{R}^n$ where $\mathbf{F}(\mathbf{x}^{(0)}, \lambda_{\min}) = \mathbf{0}$; step size $h > 0$
   Output: vector $\mathbf{x}^{(k)} \in \mathbb{R}^n$ and scalar $\lambda^{(k)} \geq \lambda_{\max}$ such that $\mathbf{F}(\mathbf{x}^{(k)}, \lambda^{(k)}) = \mathbf{0}$
 1 $k \leftarrow 0$
 2 $\lambda^{(0)} \leftarrow \lambda_{\min}$                                        % initialization
 3 $\mathbf{z}^{(0)} \leftarrow (\mathbf{x}^{(0)}, \lambda^{(0)}) \in \mathbb{R}^{n+1}$
 4 Determine approximate tangent vector $\mathbf{T}^{(0)} \in \mathbb{R}^{n+1}$ at $\mathbf{z}^{(0)}$
 5 while $\lambda_{\min} < \lambda^{(k)} < \lambda_{\max}$ do          % loop to generate successive points on curve
 6     $\mathbf{w} \leftarrow \mathbf{z}^{(k)} + h\mathbf{T}^{(k)}$                                 % predictor step
 7     Iteratively solve $(n+1) \times (n+1)$ nonlinear system of equations
           $\mathbf{F}(\boldsymbol{\zeta}) = \mathbf{0},$
           $\mathbf{T}^{(k)T}(\boldsymbol{\zeta} - \mathbf{z}^{(k)}) = h$
       to obtain $\boldsymbol{\zeta} \in \mathbb{R}^{n+1}$ starting from initial iterate $\boldsymbol{\zeta}^{(0)} = \mathbf{w}$          % corrector steps
 8     if iteration in Line 7 converges to $\boldsymbol{\zeta}$ then            % accept next point on curve
 9         $\mathbf{x}^{(k+1)} \leftarrow \boldsymbol{\zeta}_{1:n}$                          % extract subvector from $\boldsymbol{\zeta}$
10         $\lambda^{(k+1)} \leftarrow \zeta_{n+1}$                        % extract last element from $\boldsymbol{\zeta}$
11         $\mathbf{z}^{(k+1)} \leftarrow (\mathbf{x}^{(k+1)}, \lambda^{(k+1)})$
12         Determine approximate tangent vector $\mathbf{T}^{(k+1)} \in \mathbb{R}^{n+1}$ at $\mathbf{z}^{(k+1)}$
13         Heuristically adjust the step size $h$
14         $k \leftarrow k+1$
15     else
16         Reduce step size $h$                    % reject predictor step $\mathbf{w}$ and repeat
17 return $\mathbf{z}^{(k)} = (\mathbf{x}^{(k)}, \lambda^{(k)})$
```

Both natural continuation and pseudo-arclength continuation fit into a broader framework of *predictor-corrector methods* [Allgower and Georg 2003]. Predictor-corrector methods involve three important components:

—a *predictor step* of a prescribed step size;
—a sequence of *corrector steps* (alternatively, *corrector iterations*); and
—an adaptive step-size selection strategy.

The predictor step is used to seed the iterative solution of a nonlinear system of equations (successive iterates being called corrector steps). There are a variety of ways in which the predictor and corrector steps can be chosen (see Allgower and Georg [2003] and Doedel et al. [2009]). For instance, the predictor step in natural continuation comprises incrementing the scalar λ by h; in pseudo-arclength continuation, prediction involves traversing a distance h along $\mathbf{T}$, the direction tangent to the curve. Whatever the particular predictor-corrector strategy, when the step size h is too large, the inner corrector iterations can stagnate or diverge (either because the initial predictor is too far from a solution or perhaps because the system is inconsistent). In either event, the predictor step is rejected and the step size is reduced according to a simple heuristic such as $h \leftarrow th$, where t is a user-specified parameter satisfying $0 < t < 1$. Consult Allgower and Georg [2003, Ch. 6] for more detailed strategies for adapting step sizes.

The structure of predictor-corrector methods has two significant consequences:

(1) *The most expensive part of the algorithm is computation of the corrector steps* (especially when each iteration requires the solution of a linear system of equations, e.g., as in Newton's method).
(2) *Rejected predictor steps are costly* because numerous corrector steps are computed prior to rejection.

For moderate-sized nonlinear systems, the linear systems to be solved at each corrector iteration are amenable to dense, direct linear algebra solvers (e.g., as found in the LAPACK library [Anderson et al. 1999]). As such, failed predictor steps may not be so punitive. However, for large-scale systems, time (and possibly memory) requirements for each corrector iteration grows algebraically; thus, using direct solvers becomes infeasible. In that case, *Krylov subspace iterations* (e.g., see Saad and Schultz [1986]) can be applied within Newton iterations to determine the corrector steps within a continuation algorithm. This process is referred to as *Newton-Krylov continuation* [Sánchez et al. 2004; Knoll and Keyes 2004]). The convergence of Krylov subspace methods depends critically on the properties of the Jacobian matrix and may require preconditioning. Even using a well-tuned Krylov subspace method, individual corrector steps can take hours—in some cases up to a day [Gibson 2014]—in systems with upward to 10,000 unknowns. The penalty incurred for failed predictor steps is prohibitive. Given that rejected predictor steps can result from using a large step size, an obvious strategy is to use very small step sizes. Using small step sizes impedes progress along the curve from $\lambda_{\min}$ to $\lambda_{\max}$, however, both in natural and in pseudo-arclength continuation. Thus, efficient step-size adaptivity requires trade-offs between these conflicting concerns.

To achieve this balance, we developed a parallel software library—Parallel Adaptive Method for Pseudo-Arclength Continuation (PAMPAC)—that permits adaptive step-size selection within the predictor-corrector framework of pseudo-arclength continuation. The parallel algorithm implemented can be used for different kinds of correction steps and leads to speed-up for problems in which the computation of a correction iteration takes much longer than communication between processors. It is particularly effective for continuation problems in which the solution curve exhibits large variations in curvature (i.e., where the optimal step size changes wildly along the curve).

The parallelism established by PAMPAC is independent of the way the corrector steps are computed. In the examples included in Section 4, the corrector steps are Newton-Raphson steps, computed using a direct linear solver in the first example and an iterative solver in the second. In both examples, the individual corrector steps are computed on a single processor. In many applications, the corrector steps themselves can be parallelized. In the software package AUTO [Doedel et al. 2009], for instance, the structure of the linear problems arising from Newton-Raphson iterations is exploited to solve for the corrector steps in parallel. In applications in fluid dynamics, similar to our second example, the underlying simulation code may run in parallel using domain decomposition or parallel Fourier transforms. As long as the just described conditions hold, the PAMPAC algorithm can be combined perfectly well with parallelism of the corrector steps.

3. SCHEME FOR PARALLELIZATION

There are two essential ways to achieve parallelism in the selection of step sizes for predictor steps. The first is most obvious: use a concurrent sequence of predictor steps of different step sizes $t_1 h, t_2 h, \dots, t_W h$ for some prescribed positive scalars $\{t_\alpha\}_{\alpha=1}^{W}$. That is, given W processes, an initial point $\mathbf{z} \in \mathbb{R}^{n+1}$ known to satisfy $\mathbf{F}(\mathbf{z}) = \mathbf{0}$ up to some fixed tolerance, and a unit tangent direction $\mathbf{T} \in \mathbb{R}^{n+1}$, each process α computes a predictor step $\boldsymbol{\zeta}_\alpha^{(0)} = \mathbf{z} + t_\alpha h \mathbf{T}$. A maximum step size $h_{\max}$ may have to be set to avoid spurious convergence to a remote branch of the continuation curve. In that instance, fewer than W predictor steps would be computed. Once a prediction $\boldsymbol{\zeta}_\alpha^{(0)}$ is determined by process α, the process computes a sequence of corrector steps $\{\boldsymbol{\zeta}_\alpha^{(0)}, \boldsymbol{\zeta}_\alpha^{(1)}, \boldsymbol{\zeta}_\alpha^{(2)}, \dots\}$. Subsequent corrector iterations do not require interprocess communications, thus can be carried out concurrently by distinct processes. Each process α maintains its own iteration counter ν_α as well as the iterates $\boldsymbol{\zeta}_\alpha^{(\nu_\alpha)}$ and the associated nonlinear residuals $\mathbf{r}_\alpha^{(\nu_\alpha)} \equiv \mathbf{F}(\boldsymbol{\zeta}_\alpha^{(\nu_\alpha)})$.

A second strategy for parallel step-size adaptivity is to use intermediate computations to seed new predictions. That is, suppose process α is initialized using the predictor $\boldsymbol{\zeta}_\alpha^{(0)} = \mathbf{z}_\alpha + h_\alpha \mathbf{T}_\alpha$ and proceeds to compute a sequence of correction steps $\{\boldsymbol{\zeta}_\alpha^{(0)}, \boldsymbol{\zeta}_\alpha^{(1)}, \boldsymbol{\zeta}_\alpha^{(2)}, \dots\}$. Rather than waiting for the corrector iterations to converge, the intermediate iterates can be used to compute a normalized secant direction $\widehat{\mathbf{T}}_\alpha$ in the direction of $\boldsymbol{\zeta}_\alpha^{(\nu_\alpha)} - \mathbf{z}_\alpha$ for some iterate ν_α. Assuming that the corrector iterates are sufficiently close to the curve of solutions, the new secant direction $\widehat{\mathbf{T}}_\alpha$ approximates a tangent to the curve and can be handed off to another process β to generate a new predictor point $\mathbf{z}_\beta + h_\beta \mathbf{T}_\beta$, where $\mathbf{z}_\beta = \boldsymbol{\zeta}_\alpha^{(\nu_\alpha)}$, $h_\beta = t_\ell h_\alpha$ for some $\ell \in \{1, \dots, W\}$, and $\mathbf{T}_\beta = \widehat{\mathbf{T}}_\alpha$.

To manage concurrent processes when both these strategies are employed, we represent each process by a node in a rooted tree with a master process at the root (e.g., see Knuth [1997], for a discussion of trees and related algorithms). The user specifies the tree's width $W > 0$ and depth $D > 0$; the width corresponds to the number of scalars $t_1, t_2, \dots, t_W$ that multiply the step length h and the depth corresponds to the number of extrapolated predictor steps computed from intermediate corrector iterates. New nodes, that is, new corrector sequences, are seeded only if physical processors are available; the queueing of multiple processes on a single processor hinders the continuation.

Each node in the tree corresponds to a potential computational process. A node α is associated with a current iteration counter ν_α and a current corrector iterate $\boldsymbol{\zeta}_\alpha^{(\nu_\alpha)} \in \mathbb{R}^{n+1}$. The node α is generated with

$$\nu_\alpha \leftarrow 0 \text{ and } \boldsymbol{\zeta}_\alpha^{(0)} \leftarrow \mathbf{z}_\alpha^{\text{init}} + h_\alpha^{\text{init}} \mathbf{T}_\alpha^{\text{init}}, \tag{4}$$

where the data $\mathbf{z}_\alpha^{\text{init}}$, $\mathbf{T}_\alpha^{\text{init}}$, and h_α^{init} are determined from the parent node (i.e., the previous point on or near the continuation curve). When created, the node α also records ν_α^{init}—the number of corrector iterations that its parent had computed prior to spawning node α. Node α also maintains a real parameter h_α representing the base step size that α uses to construct new predictor steps. Initially, $h_\alpha \leftarrow h_\alpha^{\text{init}}$, that is, the base step size node that α uses to make predictor steps is the same as the step size used to initialize node α. However, in the event that all child nodes spawned from α lead to divergent sequences, h_α is reduced by some constant scaling factor $t \in (0, 1)$ before spawning more predictions. The scaling factor t is fixed to ensure that all new prediction steps are at a distance shorter than the shortest of the steps that just failed:

$$t = 0.9\,\frac{t_{\min}}{t_{\max}}, \text{ where } t_{\min} = \min_{1\le k\le W} t_k \text{ and } t_{\max} = \max_{1\le k\le W} t_k.$$

This precaution ensures that new predictors generated do not repeat earlier computations (after the failed child nodes are all deleted). If the base step size h_{α_r} of the root node is reduced below some user-specified threshold H_MIN, the continuation algorithm halts.

Each node α is assigned a state $s_\alpha \in \{\texttt{CONVERGED}, \texttt{CONVERGING}, \texttt{PROGRESSING}, \texttt{FAILED}\}$ as the algorithm proceeds. The state s_α of node α is determined using $\|\mathbf{r}_\alpha^{(\nu_\alpha)}\|_2$, that is, the 2-norm of the nonlinear residual $\mathbf{r}_\alpha^{(\nu_\alpha)} = \mathbf{F}(\boldsymbol{\zeta}_\alpha^{(\nu_\alpha)}) \in \mathbb{R}^n$ at the current corrector iterate $\boldsymbol{\zeta}_\alpha^{(\nu_\alpha)}$. Given positive parameters TOL_RESIDUAL, GAMMA, and MU, the rules are as follows:

(1) $s_\alpha \leftarrow$ CONVERGED if $\|\mathbf{r}_\alpha^{(\nu_\alpha)}\|_2 \le$ TOL_RESIDUAL (i.e., the current iterate $\boldsymbol{\zeta}_\alpha^{(\nu_\alpha)}$ is deemed to have converged to a point on, or sufficiently near, the solution curve);
(2) $s_\alpha \leftarrow$ CONVERGING if $\|\mathbf{r}_\alpha^{(\nu_\alpha)}\|_2^{\texttt{GAMMA}} \le$ TOL_RESIDUAL (i.e., the *next iterate* $\boldsymbol{\zeta}_\alpha^{(\nu_\alpha+1)}$ is expected to have converged);
(3) $s_\alpha \leftarrow$ FAILED if $\nu_\alpha >$ MAX_ITER or $\|\mathbf{r}_\alpha^{(\nu_\alpha)}\|_2 >$ MU$\|\mathbf{r}_\alpha^{(\nu_\alpha-1)}\|_2$ (i.e., the maximum number of corrector iterations is exceeded or the reduction of the residual is insufficient in consecutive corrector iterations); and
(4) $s_\alpha \leftarrow$ PROGRESSING otherwise.

The efficiency of the parallelization depends on the preceding criteria for assigning states to nodes.

New nodes are assigned state PROGRESSING by default; the states are reassessed after the current corrector iterates and the corresponding nonlinear residuals are computed on every node. The specific criteria for assigning a CONVERGED, CONVERGING, or FAILED state to a node depend on the nature of the corrector steps and the continuation problem at hand. The user chooses these criteria by providing problem-dependent parameters TOL_RESIDUAL, MAX_ITER, GAMMA, and MU. The parameters TOL_RESIDUAL and MAX_ITER are standard, as expected in any iterative solver. The other parameters GAMMA and MU are based on the asymptotic behavior of the user's choice of corrector sequences.

For instance, when using Newton's method for corrector steps, we expect the decrease in the residual to be quadratic (or at least superlinear) near the solution curve; in that case, it makes sense to choose GAMMA $\in (1, 2)$ in the Criterion (2) to assign state CONVERGING to "nearly converged nodes," for example, ones within the basin of attraction of Newton's method. For predictor steps too far from the solution curve, we often see linear convergence only; it makes sense, then, to choose MU $\in (0, 1)$ in the Criterion (3) to assign state FAILED to a node when it is not converging sufficiently quickly. Nodes with state FAILED get deleted from the tree since computing more corrector steps likely leads to divergence or spurious convergence on another branch of the continuation curve.

Algorithm 3 is a high-level summary of the essential steps underlying our parallel adaptive algorithm. The main conceptual pieces are the concurrent computation of corrector steps from predictor steps of various step lengths, extrapolation from intermediate corrector steps, and management of the parallel computations using a rooted tree with states assigned to each node.

The outermost loop of Algorithm 3 corresponds to accepting and logging valid points along the continuation curve. The nested loop spanning Lines 3 to 9 describe the generation of new predictor points and their distribution over available processors. The processors compute individual corrector steps concurrently in Line 3, requiring synchronization before advancing to Line 10. This implies that the parallelization scheme

ALGORITHM 3: Essential Parallel Adaptive Algorithm

```
Input: $[\lambda_{\min}, \lambda_{\max}] \subseteq \mathbb{R}$; vector $\mathbf{x}^* \in \mathbb{R}^n$ where $\mathbf{F}(\mathbf{x}^*, \lambda_{\min}) = \mathbf{0}$; step size $h \neq 0$; tangent
       direction $\mathbf{T}^* \in \mathbb{R}^{n+1}$, where $\|\mathbf{T}^*\|_2 = 1$; tolerance TOL_RESIDUAL $> 0$; GAMMA $> 0$;
       MU $> 0$; depth $D > 0$; width $W > 0$; positive scaling parameters $\{t_1, t_2, \ldots, t_W\}$
Output: vector $\mathbf{x}^{(k)} \in \mathbb{R}^n$ and scalar $\lambda^{(k)} \geq \lambda_{\max}$ such that $\mathbf{F}(\mathbf{x}^{(k)}, \lambda^{(k)}) = \mathbf{0}$

1  Seed root node $\alpha_r$ with data $(\mathbf{z}_{\alpha_r}, \widehat{\mathbf{T}}_{\alpha_r}, h_{\alpha_r}) \leftarrow ((\mathbf{x}^*, \lambda_{\min}), \mathbf{T}^*, h)$      % initialization
2  repeat                                             % loop to generate successive points on curve
3      foreach leaf node $\alpha$ do                                            % Spawn new nodes
4          if depth of $\alpha < D$ then
5              if $\alpha \neq \alpha_r$ then
6                  $\widehat{\mathbf{T}}^{\text{init}}_{\alpha} \leftarrow \left\|\boldsymbol{\zeta}^{(\nu_\alpha)}_{\alpha} - \mathbf{z}^{\text{init}}_{\alpha}\right\|_2^{-1} \left(\boldsymbol{\zeta}^{(\nu_\alpha)}_{\alpha} - \mathbf{z}^{\text{init}}_{\alpha}\right)$      % secant direction
7              for $\ell = 1 : W$ do
8                  if processors are available then
9                      Assign processor $\beta_\ell$ the data
                           $\mathbf{z}^{\text{init}}_{\beta_\ell} \leftarrow \boldsymbol{\zeta}^{\nu_\alpha}_{\alpha}, \quad \widehat{\mathbf{T}}^{\text{init}}_{\beta_\ell} \leftarrow \widehat{\mathbf{T}}_{\alpha}, \quad h^{\text{init}}_{\beta_\ell} \leftarrow t_\ell h_\alpha, \quad \nu^{\text{init}}_{\beta_\ell} \leftarrow \nu_\alpha,$
                           $\nu_{\beta_\ell} \leftarrow 0, \quad \boldsymbol{\zeta}^{(0)}_{\beta_\ell} \leftarrow \boldsymbol{\zeta}^{\nu_\alpha}_{\alpha} + t_\ell h_\alpha \widehat{\mathbf{T}}_{\alpha}, \quad s_{\beta_\ell} \leftarrow$ PROGRESSING
10     Compute single corrector steps concurrently on all PROGRESSING and CONVERGING nodes
       of the tree.
11     Traverse nodes of tree updating residuals
           $\mathbf{r}^{(\nu_\alpha)}_{\alpha} \leftarrow \mathbf{F}(\boldsymbol{\zeta}^{(\nu_\alpha)}_{\alpha})$
       and updating states $s_\alpha$ accordingly.
12     PruneTree($\alpha_r$), deleting FAILED nodes and eliminating redundant subtrees.
13     while root node $\alpha_r$ has a single child, $\beta$, and $\beta$ is CONVERGED do      % update root
14         Write the solution on $\alpha_r$ to disk
15         Update root: $\alpha_r \leftarrow \beta$
16 until root node $\alpha_r$ has $\lambda^{(\alpha_r)} \geq \lambda_{\max}$ or $\lambda^{(\alpha_r)} \leq \lambda_{\min}$
```

is efficient only when the time required for a corrector step is much greater than the communication costs and when the time for the concurrent corrector computations is roughly the same on all processors. At a high level, the algorithm is mostly straightforward; further explanation is required to understand the pruning algorithm in Line 12.

Concurrent computations are managed by pruning the tree in two stages. In the first stage, all subtrees rooted at FAILED nodes are removed. The corrector sequences associated with FAILED nodes are deemed to have made insufficient progress; as such, computing more corrector steps likely leads to divergence or spurious convergence to a point on another branch of the continuation curve. Other non-FAILED nodes are unaltered in this first stage.

Figure 1 provides a graphical illustration of the first stage of pruning. In the leftmost tree, the root node represents a valid CONVERGED solution point with three child nodes (one CONVERGING and two PROGRESSING nodes). The center tree represents the state after concurrent corrector steps are computed for all CONVERGING and PROGRESSING nodes (requiring 12 active processes in this case). At the end of the corrector step, four nodes are now CONVERGED, two are CONVERGING, and four are FAILED. The first stage of pruning is simply to delete all FAILED nodes and their associated subtrees, which leads to the configuration shown in the rightmost tree in Figure 1.

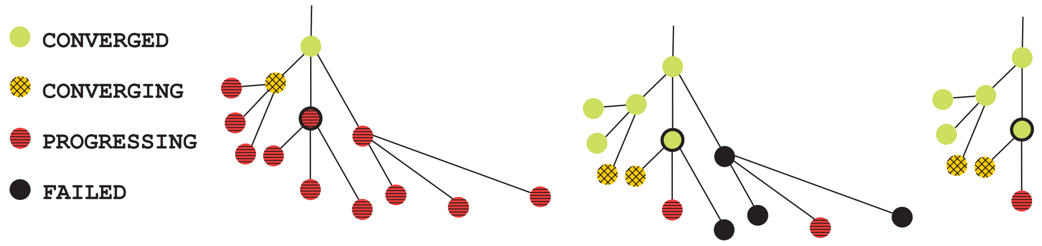

Fig. 1. Illustration of the first stage of the pruning algorithm on a tree of width 3 and depth 3. The links are drawn from left to right in the order of increasing step length. In the left tree, `CONVERGED` root node representing a valid solution point and two levels of child nodes. Each of the nodes at depth 2 has three `PROGRESSING` child nodes that have indeterminate convergence status as of yet. In the center tree, concurrent corrector steps have been computed on all `CONVERGING` and `PROGRESSING` nodes. After computing and assessing the residuals on all of the nodes, some have been assigned state `CONVERGED`, `CONVERGING`, `FAILED`, or `PROGRESSING` (i.e., it is undecided whether this corrector sequence will converge successfully). Finally, in the right tree, all the `FAILED` nodes and their attached subtrees have been deleted.

The second stage of pruning-Line 12 of Algorithm 3—involves comparing child nodes of a given parent node with viable corrector iterates and deciding which to keep and which to delete (along with associated subtrees). Doing so permits allocation of computing resources to nodes that are deemed to yield the greatest benefit. The greatest gain is determined by balancing the longest distance travelled along the continuation curve (i.e., pseudo-arclength) against the least amount of computational work (measured in corrector iterations). `PROGRESSING` child nodes are kept by default—it is unclear whether they will yield convergent corrector sequences or not. When deciding which `CONVERGED` or `CONVERGING` child nodes to keep, a more sophisticated criterion is applied that requires a few definitions.

Definition 3.1. A *path* P in the computation tree is a connected subtree in which each node has at most one child node.

A path in the computation tree corresponds to a putative segment of the continuation curve. An ordered sequence of connected nodes (e.g., $P = (\alpha_1, \alpha_2, \ldots, \alpha_N)$) represents a path in the computation tree.

Being able to identify paths in the computation tree, there are two important measures needed to control our algorithm.

Definition 3.2. The *initialization length* of the path $P = (\alpha_1, \alpha_2, \ldots, \alpha_N)$ is

$$L(P) = \sum_{k=1}^{N} h_{\alpha_k}^{\text{init}}. \tag{5}$$

Definition 3.3. The *iteration cost* of the path $P = (\alpha_1, \alpha_2, \ldots, \alpha_N)$ is

$$I(P) = \mu_{\alpha_1} \text{ where } \mu_{\alpha_k} = \begin{cases} \nu_{\alpha_k}, & \text{if } k = N, \\ \max\left(\nu_{\alpha_k}, \mu_{\alpha_{k+1}} + \nu_{\alpha_{k+1}}^{\text{init}}\right), & \text{otherwise.} \end{cases} \tag{6}$$

That is, $L(P)$ is the pseudo-arclength of the segment of the curve associated with the nodes of P in sequence—assuming that the points associated with the nodes all lie on the continuation curve. Similarly, $I(P)$ is the accumulated number of corrector steps computed to attain the current state of the entire path.

It is useful to identify two specific kinds of paths in the tree to choose between "converged" and "nearly converged" corrector sequences (represented on the tree by `CONVERGED` and `CONVERGING` nodes, respectively).

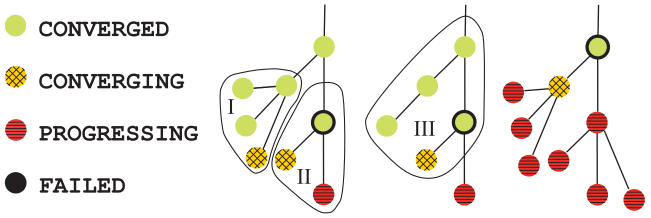

Fig. 2. Illustration of the pruning algorithm on a tree of width 3 and depth 3. For each subtree, the path that is expected to yield the fastest progress is retained with any PROGRESSING nodes. This path is first selected in the smaller subtrees 1 and 2, then in the larger subtree 3. After pruning, the CONVERGED node at the root of subtree 2 in the left tree becomes the new root node and three new leaves are spawned from both of its children (as shown in the middle tree).

Definition 3.4. A *valid path* is a path in which all nodes are CONVERGED.

Definition 3.5. A *viable path* is a path in which all nodes are either CONVERGED or CONVERGING.

Note that a valid path is, by definition, a viable path also.

Thus, before the second stage of pruning, valid and viable paths are identified recursively (Line 12 of Algorithm 3). Naturally, a valid path comprises nodes associated with points known to be on the continuation curve (within the desired tolerance). Viable paths consist of nodes associated with points that are either known to be on the continuation curve or points that are likely to be on the curve in the next concurrent corrector iteration. As such, when comparing a node's CONVERGED and CONVERGING child nodes, Algorithm 4 chooses the node associated with the longest viable or valid path (measured in pseudo-arclength).

In the event that distinct valid and viable paths exist, Algorithm 4 decides whether to keep the best valid or best viable path. In particular, to choose between the longest valid path $P_{\text{valid}}(\alpha)$ and the longest viable path $P_{\text{viable}}(\alpha)$ extending below node α, the rate of progress along the curve is estimated by computing

$$\frac{L(P_{\text{valid}}(\alpha))}{I(P_{\text{valid}}(\alpha))} \quad \text{and} \quad \frac{L(P_{\text{viable}}(\alpha))}{I(P_{\text{viable}}(\alpha))+1}. \tag{7}$$

The algorithm keeps whichever child node leads to the path with the greatest rate of progress as computed in Equation (7); the other CONVERGED and CONVERGING child nodes are deleted (with their associated subtrees).

Figure 2 illustrates Algorithm 4 for a tree of width 3 and depth 3 continuing where Figure 1 stopped. The step lengths in this example are $\{t_k\}_{k=1}^{W} = \{1/4, 1, 3/2\}$. The leftmost tree in Figure 2 represents the initial state; the root CONVERGED node represents a solution that has been computed after two corrector steps, while its child nodes have been spawned after one step. First, subtrees I and II are examined to choose which child to keep. Subtree II in the leftmost tree has only a single viable path; thus, it is left alone. Subtree I in the leftmost tree has two valid and three viable paths from its root CONVERGED node. The longest valid and longest viable paths are compared using Equation (7) and only the path with the greatest "speed" is kept. The longest valid path took two corrector steps to construct and gives a gain of $t_1h + t_2t_1h = h/2$ in arc length, while the longest viable path is expected to yield a gain of $t_1h+t_3t_1h = 5h/8$ in three steps. Since $(h/2)/2 > (5h/8)/3$, the longest valid path is retained. The result after choosing between paths is the middle tree in Figure 2 with subtree III hanging off the root CONVERGED node. The subtree III has one valid and two viable paths; retaining the all-green (valid) subtree, on the left, would lead to a gain of $(1 + t_1 + t_2t_1)h = 3h/2$ in three corrector steps. By contrast, retaining the CONVERGED and CONVERGING (viable but not valid) subtree on the right should give a gain of $(1 + t_2 + t_1 t_2)h = 9h/4$ in four corrector steps. Since $(3h/2)/3 = h/2 < 9h/16 = (9h/4)/4$, the CONVERGED and CONVERGING subtree is retained. Finally, the next CONVERGED node on the retained path becomes the new root node yielding the rightmost tree. The CONVERGING and PROGRESSING leaf nodes from the middle tree are used to compute new predictor points (the leaf nodes in the rightmost tree).

4. NUMERICAL EXAMPLES

We present two examples to test the performance of the parallel algorithm. The first example concerns travelling waves in a 1 + 1 dimensional, nonlinear, partial differential equation (PDE) and is distributed with the PAMPAC library in the `examples` directory. The second concerns time-periodic solutions to the Navier–Stokes equation on a three-dimensional, periodic domain. In both test cases, the number of unknowns is in the thousands, but the corrector step is computed differently. In the first test case, we compute and LU-decompose the dense Jacobian matrix, while in the second case, an inexact Krylov subspace method is used. An additional difference is that the first test case was implemented in C, but needs to be linked to external numerical libraries, whereas the second is a self-contained legacy FORTRAN code. In spite of these differences, a similar set of parameters gives rise to similar speed-up.

The numerical experiments were run on a cluster with QDR Infiniband interconnects and 2.2GHz AMD Opteron processors. In addition, the wall-clock time scaling was verified for up to eight processes on a desktop machine with two quad-core Intel Xeon X5482 processors.

4.1. Travelling Waves in a Modified Kuramoto-Sivashinsky Equation

The Kuramoto-Sivashinsky equation was derived independently in different contexts, most notably by Kuramoto and Tsuzuki [1976] for describing phase dynamics in

ALGORITHM 4: PruneTree(α_r)

```
   Input: root node $\alpha_r$ of computation tree
 1 foreach node $\alpha$ in a depth-first traversal of computation tree rooted at $\alpha_r$ do
 2     if $s_\alpha$ = FAILED then
 3         delete subtree rooted at $\alpha$
 4 foreach node $\alpha$ in a depth-first traversal of computation tree rooted at $\alpha_r$ do
 5     compute paths $P_{\text{viable}}(\alpha)$ & $P_{\text{valid}}(\alpha)$,the respective viable and valid paths of longest
       initialization path length rooted at node $\alpha$
 6 foreach node $\alpha$ in a depth-first traversal of computation tree rooted at $\alpha_r$ do
 7     $P_{\text{best}}(\alpha) \leftarrow \emptyset$
 8     if $P_{viable}(\alpha) \neq \emptyset$ then
 9         $P_{\text{best}}(\alpha) \leftarrow P_{\text{viable}}(\alpha)$
10         if $P_{valid}(\alpha) \neq \emptyset$ and $P_{valid}(\alpha) \neq P_{viable}(\alpha)$ then
11             $S_{\text{valid}}(\alpha) \leftarrow L(P_{\text{valid}}(\alpha))/I(P_{\text{valid}}(\alpha))$
12             $S_{\text{viable}}(\alpha) \leftarrow L(P_{\text{viable}}(\alpha))/\left[I(P_{\text{viable}}(\alpha)) + 1\right]$
13             if $S_{valid}(\alpha) \geq S_{viable}(\alpha)$ then
14                 $P_{\text{best}}(\alpha) \leftarrow P_{\text{valid}}(\alpha)$
15     foreach child node $\beta$ of node $\alpha$ do
16         if $\beta \notin P_{best}(\beta)$ and $s_\beta \in$ {CONVERGING, CONVERGED} then
17             delete subtree rooted at $\beta$
18 return
```

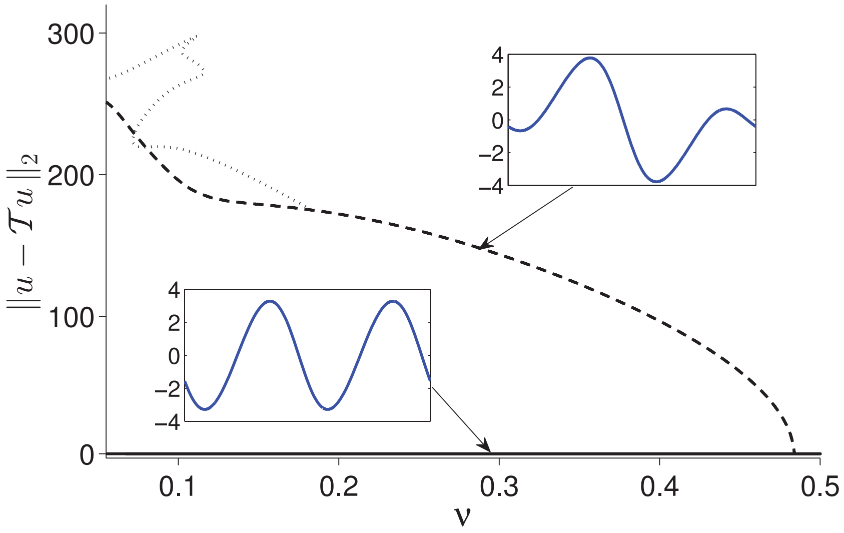

Fig. 3. Partial bifurcation diagram of the discretization of PDE model 8. Solid line: branch of equibria invariant under symmetries $\mathcal{S}$ and $\mathcal{T}$. Dashed line: branch of equilibria invariant only under $\mathcal{S}$. The dotted line is the branch of travelling wave solutions shown in detail in Figure 4. Shown is a measure for the deviation from invariance under the shift $\mathcal{T}$ versus the viscosity.

reaction–diffusion systems and by Sivashinsky [1977] for describing flame front dynamics. The similarity of its quadratic nonlinearity to that of the Navier–Stokes equation makes it a popular test case for numerical methods for PDEs. We add a second nonlinear term to obtain

$$u_t + uu_x + u_{xx} + \lambda u_{xxxx} - A\sin(u) = 0. \tag{8}$$

The extra nonlinear term breaks the equivariance under Galileo boosts so that we can compute families of travelling waves with a uniquely determined wave speed. The modified equation is still equivariant under translations and under the reflection symmetry given by $\mathcal{S}: \ (u, x) \to (-u, -x)$.

We consider this PDE on a periodic domain with $u \in C^4(S)$, fixing $A = 8.09$ and considering the viscosity, $\lambda > 0$, as control parameter. There is always a trivial solution at $u \equiv 0$, from which equilibria branch off with increasing wave number for decreasing viscosity. In Figure 3, a branch with wave number two is shown. This branch is symmetric under the reflection S, as well as under the shift $\mathcal{T}$ over half the domain. At $\lambda \approx 0.48$, a family of equilibria branches off in a bifurcation that breaks the translational symmetry. Subsequently, at $\lambda \approx 0.18$, a family of travelling waves is created, shown in detail in Figure 4.

This family has a number of fold points and the wave changes its shape rapidly along the branch. Therefore, its computation by the traditional pseudo-arclength continuation approach suffers from many failed steps. Moreover, the wave becomes increasingly localized for small values of the viscosity. In order to resolve it correctly, we need fine discretization, which will result in time-consuming corrector steps.

We compute the travelling waves as $u(x, t) = w(x - ct)$, which results in the following Boundary Value Problem (BVP):

$$-cw' + ww' + w'' + \lambda w'''' - A\sin(w) = 0; \quad w(0) = w(2\pi).$$

The linear terms in this equation are efficiently computed in Fourier space, while the nonlinear terms are best computed on a regular periodic grid. Thus, we approximate

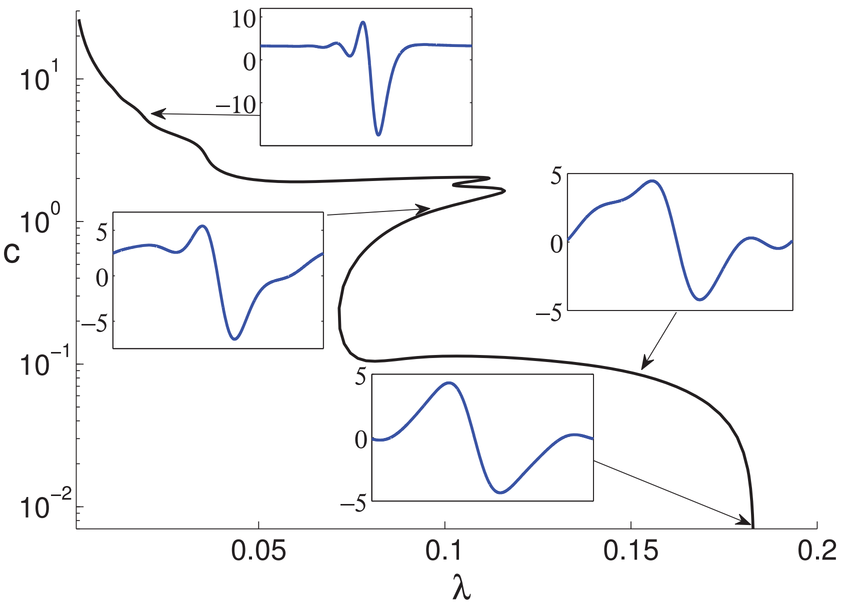

Fig. 4. The branch of travelling wave solutions to Equation (8) used for testing the parallel continuation algorithm. Shown is the wave speed, c, versus the control parameter, λ. The inlays show snapshots of the solution $u(x, t)$ at four points along the continuation curve. At $c = 0$, the wave bifurcates from an equilibrium, which is symmetric under the reflection symmetry. For $\lambda \lesssim 0.01$, the solution becomes strongly localized.

solutions as

$$w_j = \sum_{k=0}^{n-1} a_k e^{ikx_j}, \text{ where } x_j = \frac{2\pi}{n} j, \ j = 0, \dots, n-1$$

and switch between the representations $\{w_j\}$ and $\{a_k\}$ by the discrete Fourier transform, implemented using the GNU Scientific library [Gough 2009]. The computational cost of each transform is $O(n \ln n)$. The resolution is fixed to $n = 2048$, fine enough to resolve the solutions shown in Figure 4 and avoid aliasing issues.

The continuation problem now takes the form $F(a_0, \dots, a_{n-1}, c, \lambda) = 0$, where F and its derivatives are evaluated using the pseudo-spectral method outlined earlier. Since we have an extra unknown, the wave speed c, we must add an extra equation to ensure uniqueness of the corrector steps. We impose that the Newton update steps be orthogonal to w_x, the generator of the symmetry group of translations. This condition ensures that the successive iterates under corrector steps do not slide along the x-direction. Because of the second nonlinear term in Equation (8), the Jacobian matrix $\boldsymbol{F_x}$ is dense. The test code takes $O(n^2)$ flops to compute this matrix and a further $O(n^3)$ flops to LU-decompose it when solving for the Newton update step. Thus, the execution of a single Newton step in this test has a complexity of $O(n^3)$ and takes a few seconds on a single 2.2GHz CPU, rendering the MPI communication time negligible. In Table I, all parameters relevant for the numerical computation are given.

The wall-clock time for the computation of the entire curve shown in Figure 4 using various tree structures and step-size distributions is shown in Figure 5. The maximal speed-up is about a factor of three, obtained with a tree depth and width of three. In that computation, there is a lot of redundancy, as many processors will be working on very similar approximate solutions. A speed-up of more than a factor of two, however, can be obtained using as few as three processors with the same step-size multiplication

Table I. System and Algorithm Parameters for the Numerical Experiments with the Modified Kuramoto-Sivashinsky Equation

A	8.09	amplitude of second nonlinear term
$[\lambda_0,\ \lambda_1]$	[0.1828, 0.001]	range of the continuation parameter
n	2048	# grid points
$\{t_i\}_{i=1}^{3}$	{0.75, 1, 2}	step-size multipliers
$H_{\max}$	2000	maximal step size
$H_{\min}$	10^{-2}	minimal step size
H_{init}	100	initial step size
$\nu_{\max}$	4	maximal number or Newton iterations
μ	0.5	minimal linear residual decrease
γ	2.0	expected order of residual decrease
$r_{\max}$	$5 \cdot 10^{-7}$	tolerance for the nonlinear problem

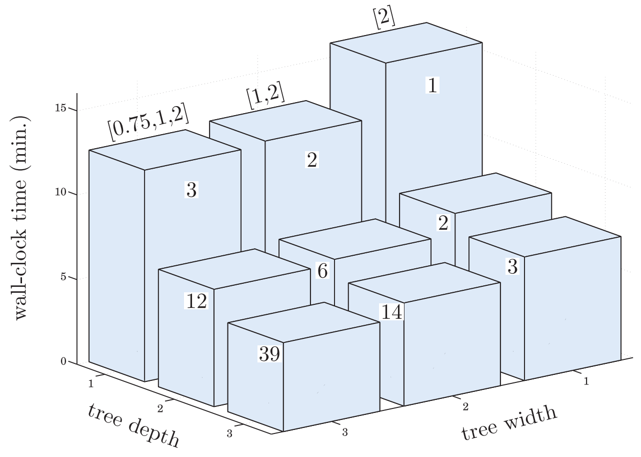

Fig. 5. Wall-clock time for the computation of the continuation curve shown in Figure 4 using a tree width and depth up to three. The numbers between brackets denote the multipliers for the step-size, $\{t_i\}_{i=1}^{W}$ and the integer denotes the number of CPUs concurrently processing corrector steps.

factor. The three depth seems to impact the efficiency much more strongly than the tree width, but this is likely problem-dependent. Some initial experimentation will be necessary for each individual problem to determine a near-optimal strategy for a given number of available processors.

In this example, the computational complexity of all corrector steps is equal. Consequently, the only real source of overhead in the computation is the parallel processing of many approximate solutions, only a few of which are actually used for the construction of the continuation curve. In the next example, the difference in wall-clock time taken by concurrent corrector steps is an additional source of overhead.

4.2. Periodic Solutions in a Turbulent Flow

The second test problem concerns the continuation of time-periodic solutions to the Navier–Stokes equation for fluid motion. We consider an incompressible, viscous fluid in a box with periodic boundary conditions in every direction. In the simulation code, the unknown variables are the Fourier coefficients of the vorticity field, truncated to a finite number. Energy is input by keeping fixed in time some coefficients with small wave numbers, corresponding to large spatial scales. At small spatial scales, energy is dissipated by viscous processes. The resulting turbulent flow is statistically stationary and exhibits a cascade of energy across spatial scales. Although the statistical description of this cascade is well developed (e.g., see Monin and Yaglom [2007, Ch. 7]), the dynamics of this process are largely unknown.

In van Veen et al. [2006], time-periodic solutions of this flow are considered as building blocks of turbulence. The idea is to study the dynamics and parameter dependence of such building blocks and distill from the results a hypothesis on the dynamical processes that contribute to the energy cascade. The essential difficulty in computation is that a very large number of degrees of freedom is required to simulate the flow accurately. Even for weakly turbulent flow, about $n \simeq 10^6$ degrees of freedom are needed. This number increases algebraically with Taylor's microscale Reynolds number, Re_λ, which, in turn, increases with decreasing viscosity. Symmetry arguments reduce the number of degrees of freedom to $n \simeq 10^4$, making the study of the resulting *high symmetric flow* feasible [Kida 1985].

After symmetry reduction, the resulting system is a set of n coupled, nonlinear ODEs with a single parameter: the kinematic viscosity, here denoted by λ, which determines the Reynolds number of the flow. Time stepping is done by the pseudo-spectral method, employing the fourth-order Runge-Kutta-Gill scheme. In the results presented here, the spatial resolution is fixed at 2^7 points in every direction that, after dealiasing and symmetry reduction, gives $n = 6370$ variables. The kinematic viscosity is varied in the range $0.0045 \geq \lambda \geq 0.0035$, which corresponds to a Taylor microscale Reynolds number Re_λ in the range $57 \leq \mathrm{Re}_\lambda \leq 68$. The square of this dimensionless number can be compared to the commonly used geometric Reynolds number Re.

Periodic solutions are computed as fixed points of an iterated Poincaré map, that is, as solutions of a nonlinear system:

$$\mathcal{P}^{(k)}(\mathbf{x}, \lambda) - \mathbf{x} = \mathbf{F}(\mathbf{x}, \lambda) = \mathbf{0}. \tag{9}$$

In Equation (9), $\mathbf{x}$ is the vector of Fourier coefficients of the vorticity field and k is the discrete period of the orbit. The Poincaré plane of intersection is a coordinate plane on which one of the low wavenumber Fourier coefficients equals its time-averaged value. A solution of discrete period $k = 5$ is filtered from turbulent data at the highest viscosity. At this viscosity, the flow is relatively quiescent. Subsequently, pseudo-arclength continuation is used to track the periodic solution to the more turbulent regime. In the correction step of pseudo-arclength continuation, the linear problem associated with the Newton-Raphson iteration is solved in an inner Generalised Minimal Residual (GMRES) iteration [Saad and Schultz 1986]. The combination of pseudo-arclength continuation with a Krylov subspace iteration is called *Newton-Krylov continuation* and was first implemented by Sánchez et al. [2004]. Each linear problem within this continuation takes about 20 inner GMRES iterations to solve. In turn, each inner GMRES iteration requires integrating a system of n ODEs modeling the flow along an approximately periodic orbit. On a single 2.2GHz CPU, one corrector step takes about 10min for 12 GMRES iterations at low Reynolds numbers, to 20min for 24 GMRES iterations at higher Reynolds numbers. Thus, the MPI communication time is again negligible. In contrast to the first example, however, a possible source of overhead is the difference in wall-clock time taken by concurrent corrector steps. Although all linear problems solved at the same time are close, in the sense that their difference is linear in the continuation step size, variations in the number of GMRES iterations may occur. In the computation to follow, a maximal relative difference of 1 out of 5 iterations was observed. The difference was at most 1 for 90% of the concurrent steps and the total idle CPU time was below 10% of the total CPU time. All system and algorithm parameters used to generate the results are listed in Table II.

Figure 6 shows the numerical curve produced by pseudo-arclength continuation. The points shown are computed using a naïve scheme for step-size control: the step size

Table II. System and Algorithm Parameters for the Numerical Experiments with High-Symmetric Flow

$[\lambda_0, \lambda_1]$	[0.0045, 0.0035]	range of the continuation parameter
N	128^3	spatial resolution
n	6370	# variables
$\{t_i\}_{i=1}^3$	{0.75, 1, 2}	step-size multipliers
h_i	0.01	initial step-size
q	0.5	minimal redisual decrease is $\|\boldsymbol{r}^{(\alpha,\nu_\alpha)}\|_2 < q\|\boldsymbol{r}^{(\alpha,\nu_\alpha-1)}\|_2$
t_f	0.5	step-size multiplier if all child nodes fail
TOL	10^{-6}	tolerance for the nonlinear problem
GMRESTOL	10^{-5}	relative residual tolerance for GMRES
Δt	5×10^{-3}	step-size for 4th order Runge-Kutta-Gill time stepping

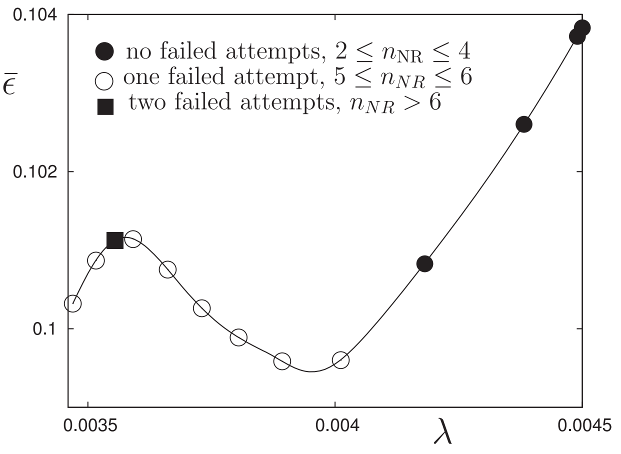

Fig. 6. Continuation of a periodic solution in high-symmetric flow by serial pseudo-arclength continutation. The continuation parameter is the kinematic viscosity, λ, and the time-mean energy dissipation rate $\bar{\epsilon}$ is shown on the vertical axis. Dots, circles, and squares denote points computed after zero, one and two failed correction steps, respectively.

is doubled after each successful step, halved after each failed step. Twelve points are computed along the curve at the cost of 55 Newton-Krylov iterations. Out of these 55 correction steps, 15 steps were rejected; thus, over a quarter of the time (as measured on a wall clock) was wasted. A more careful strategy based on a local estimate of the curvature yields 18 points computed on the numerical curve at a cost of 49 Newton iterations with four rejected steps. In Figure 7, the wall-clock time required by PAMPAC is shown for tree configurations up to width and depth three. As compared to the first test case, the tree width, that is, the number of different step lengths attempted in parallel, is of greater influence. However, a speed-up by a factor of two is again obtained using three CPUs and the maximal speed-up by a factor of three is obtained on 39 CPUs.

5. CONCLUSION

In the computation of parameterized solutions to discretized PDEs, most elements have been optimized for efficiency. For instance, when studying Navier–Stokes flow, we often use pseudo-spectral time stepping in combination with fast Fourier transforms. The linear systems that we must solve to find Newton update steps are handled by Krylov subspace methods, whose convergence can be sped up by a host of preconditioners. In contrast, the continuation algorithm, which forms the outer loop of the computation,

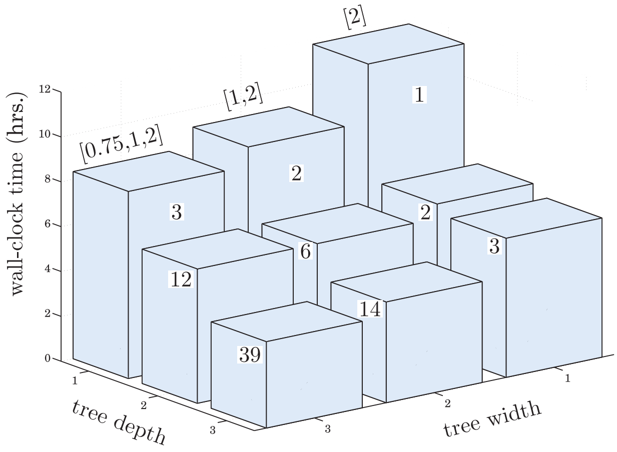

Fig. 7. Wall-clock time in hours versus tree width and depth. The numbers between brackets are the step-length multiplication factors $\{t_i\}_{i=1}^{W}$ and the integer on each data bar is the number of CPUs concurrently processing corrector steps.

is essentially the same as that used for small sets of ODEs. For such small systems, selecting an unnecessarily small step size, or an overly large step size that leads to diverging corrector steps, will cost seconds or minutes of computation time. For large systems, this may cost days or weeks.

In this article, we have presented an elegant, recursive algorithm that combines two strategies for aggressively optimizing step size and minimizing computation time. The first is to try several step sizes in parallel; the second is to predict new solutions from a sequence of corrector steps before this sequence has converged.

Two test cases, different in the type of solution computed, linear solving, and, implementation, demonstrate that the continuation can be sped up by a factor of two using only three processors, and by a factor of three using 39. Since multicore processors are a standard feature of new desktop computers and cluster computers are available in many places, we expect that the gain in wall-clock time will increasingly outweigh the heavy use of CPU time.

Implicit time stepping for PDEs could be another interesting application of PAMPAC. In that application, a nonlinear equation must be solved in every time step. As in homotopy-type problems, optimizing the step size is of critical importance. In addition to conditions on the residual of the nonlinear problem, some control of the local discretization error will have to be included in the parallel algorithm. This is the topic of ongoing research.

ACKNOWLEDGMENTS

The computations presented in Section 4 were made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca) and Compute/Calcul Canada. The authors would also like to thank the anonymous referees for their careful reading and thoughtful comments that improved the final article.

REFERENCES

Eugene L. Allgower and Kurt Georg. 2003. *Introduction to Numerical Continuation Methods*. *Classics in Applied Mathematics*, Vol. 45. SIAM, Philadelphia, PA.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. 1999. *LAPACK Users' Guide* (3rd ed.). SIAM, Philadelphia, PA.

K. I. Dickson, C. T. Kelley, I. C. F. Ipsen, and I. G. Kevrekidis. 2007. Condition estimates for pseudo-arclength continuation. *SIAM ournal on Numerical Analysis* 45, 1, 263–276. http://dx.doi.org/10.1137/060654384

E. J. Doedel, A. R. Champneys, F. Dercole, T. F. Fairgrieve, Yu. A. Kuznetsov, B. Oldeman, R. Paffenroth, B. Sandstede, X. Wang, and C. Zhang. 2009. *AUTO-07P: Continuation and Bifurcation Software for Ordinary Differential Equations*. Concordia University, Montreal. Retrieved January 3, 2016 from http://cmvl.cs.concordia.ca/auto/.

J. F. Gibson. 2014. Private communications on computations using channelflow (www.channelflow.org).

Brian Gough. 2009. *GNU Scientific Library Reference Manual* (3rd ed.). Network Theory Ltd., Surrey, UK. http://www.network-theory.co.uk.

W. J. F. Govaerts. 2000. *Numerical Methods for Bifurcations of Dynamic Equilibria*. SIAM, Philadelphia, PA.

Genta Kawahara, Markus Uhlmann, and Lennaert van Veen. 2012. The significance of simple invariant solutions in turbulent flows. *Annual Review of Fluid Mechanics* 44, 203–225.

H. B. Keller. 1977. Numerical solution of bifurcation and nonlinear eigenvalue problems. In *Applications of Bifurcation Theory*, P. Rabinowitz (Ed.). Academic Press, New York, NY, 359–384.

C. T. Kelley. 2003. *Solving Nonlinear Equations with Newton's Method*. SIAM, Philadelphia, PA.

S. Kida. 1985. Three-dimensional periodic flows with high-symmetry. *Journal of the Physical Society of Japan* 54, 2132–2136.

D. A. Knoll and D. E. Keyes. 2004. Jacobian-free Newton-krylov methods: A survey of approaches and applications. *Journal of Computational Physics* 193, 1, 357–397.

Donald E. Knuth. 1997. *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (3rd ed.). Addison Wesley Longman Publishing Co., Inc., Redwood City, CA.

Y. Kuramoto and T. Tsuzuki. 1976. Persistent propagation of concentration waves in dissipative media far from thermal equilibrium. *Progress of Theoretical Physics* 55, 356–369.

Y. A. Kuznetsov. 1998. *Elements of Applied Bifurcation Theory*. Springer-Verlag, New York, NY.

A. S. Monin and A. M. Yaglom. 2007. *Statistical fluid mechanics, Volume 2*. Dover Publications, New York, NY.

Y. Saad and M. H. Schultz. 1986. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Science and Statistical Computing* 7, 856–869.

J. Sánchez, M. Net, B. García Archilla, and C. Simó. 2004. Newton-Krylov continuation of periodic orbits for Navier-Stokes flows. *Journal Computational Physics* 201, 1, 13–33.

G. Sivashinsky. 1977. Nonlinear analysis of hydrodynamic instability in laminar flames I. Derivation of basic equations. *Acta Astronautica* 4, 1177–1206.

L. van Veen, S. Kida, and G. Kawahara. 2006. Periodic motion representing isotropic turbulence. *Fluid Dynamics Research* 38, 19–46.

Zhong-Hua Yang and H. B. Keller. 1986. A direct method for computing higher order folds. *SIAM Journal of Science and Statistical Computing* 7, 2, 351–361.

Received September 2013; revised October 2014; accepted January 2015