An adaptive parallel arc-length method

Verhelst, H.M.; Den Besten, J.H.; Möller, M.

DOI
10.1016/j.compstruc.2024.107300

Publication date
2024

Document Version
Final published version

Published in
Computers & Structures

Computers and Structures 296 (2024) 107300

Contents lists available at ScienceDirect

Computers and Structures

journal homepage: www.elsevier.com/locate/compstruc

An adaptive parallel arc-length method

H.M. Verhelst [a,b,*], J.H. Den Besten [a], M. Möller [b]

[a] Delft University of Technology, Department of Maritime and Transport Technology, Mekelweg 2, Delft 2628 CD, the Netherlands
[b] Delft University of Technology, Department of Applied Mathematics, Van Mourik Broekmanweg 6, Delft 2628 XE, the Netherlands

ARTICLE INFO

Keywords:
Arc-length methods
Parallelisation
Isogeometric analysis
Kirchhoff-Love shell
Post-buckling

ABSTRACT

Parallel computing is omnipresent in today's scientific computer landscape, starting at multicore processors in desktop computers up to massively parallel clusters. While domain decomposition methods have a long tradition in computational mechanics to decompose spatial problems into multiple subproblems that can be solved in parallel, advancing solution schemes for dynamics or quasi-statics are inherently serial processes. For quasi-static simulations, however, there is no accumulating 'time' discretization error, hence an alternative approach is required. In this paper, we present an Adaptive Parallel Arc-Length Method (APALM). By using a domain parametrization of the arc-length instead of time, the multi-level error for the arc-length parametrization is formed by the load parameter and the solution norm. Given coarse approximations of arc-length intervals, finer corrections enable the parallelization of the presented method. This results in an arc-length method that is parallel within a branch and inherently adaptive. This concept is easily extended for bifurcation problems. The performance of the method is demonstrated using isogeometric Kirchhoff-Love shells on problems with snap-through and pitch-fork instabilities and applied to the problem of a snapping meta-material. These results show that parallel corrections are performed in a fraction of the time of the serial initialization, achievable on desktop scale.

1. Introduction

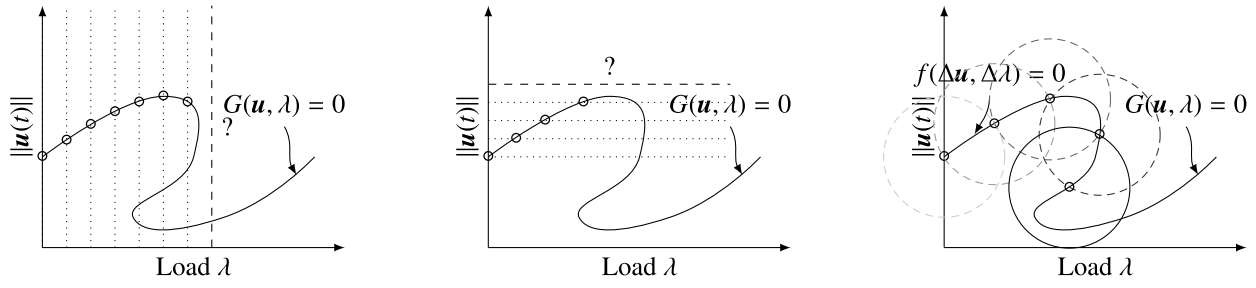
Over the last decades, computational power has increased exponentially. In the last year, most improvements were due to an increasing number of threads per processing unit rather than an increase in single-thread performance [1]. The trend of increasing logical cores with stagnating single-threaded performance calls for parallelization of existing codes to improve computational efficiency, amongst which numerical algorithms in computational mechanics. In the field of computational mechanics, parallelization in the spatial domain is common practice by using shared-memory assembly routines or distributed-memory parallelization using domain decomposition of meshes. Parallelization can also be achieved in linear solvers or in the temporal domain using parallel-in-time solvers [2] in the case of dynamic analyses or using parallel continuation for quasi-static or continuation problems - the latter two being sequential by nature.

For quasi-static problems, continuation methods can be used when the solution of an equation or a system of equations is desired, given the varying parameters of the system. Such methods, typically referred to as Arc-Length Methods (ALMs), are widely used for (but not limited to) the analysis of the stability of structures. The Riks and Crisfield methods [3,4] are commonly used and in combination with bifurcation algorithms [5], whereas ALMs provide a valuable tool in the analysis of the collapse and post-buckling behaviour of structures. Recent developments for ALMs include a new displacement-controlled formulation [6], an improved predictor scheme [7], and automatic exploration techniques [8,9]. Like time-stepping methods, ALMs are sequential by nature, meaning that the solution at a point is obtained from the solution at a previous point obtained previously.

Amongst many parallel time-integration schemes, Parareal is a parallel time-integration method proposed by [10] and works with a two-level parallel correction scheme of time intervals. The method starts with a series of solutions obtained in serial with a large time step, after which each sub-interval is computed with a finer time step such that a new solution is found at the end-point of the time interval. A multi-level extension of Parareal is proposed in [11] and is referred to as Multi-Grid Reduced in Time (MGRIT). This method is similar to Parareal but applies the two-level approach recursively. As a consequence, multi-grid-like cycles can be used to correct previously computed sub-intervals. This method has not only been applied to dynamic problems but also to

* Corresponding author at: Delft University of Technology, Department of Maritime and Transport Technology, Mekelweg 2, Delft 2628 CD, the Netherlands.
E-mail addresses: h.m.verhelst@tudelft.nl (H.M. Verhelst), henk.denbesten@tudelft.nl (J.H. Den Besten), m.moller@tudelft.nl (M. Möller).

https://doi.org/10.1016/j.compstruc.2024.107300
Received 20 October 2023; Accepted 24 January 2024
Available online 10 February 2024



(a) Load control (b) Displacement control (c) Arc-length control

Fig. 1. Load (left), displacement (middle), and arc-length control (right) for structural analysis problems. The question mark (?) indicates the iteration where load and displacement control encounter a limit point. In these situations, the next point obtained is typically difficult to find.

the training of neural networks [12] and [13]. Alternative methods for parallel time integration are reviewed in the work of [2].

Compared to temporal parallelization methods, parallelization of ALMs has received less attention in the academic community. As ALMs are typically used for explorations of solutions across branches, parallel evaluation of branches can be performed as soon as the starting point (and tangent) of each branch is known. The number of branches related to a problem, however, depends typically on the problem that is solved; hence, the parallel scalability of ALMs over branches is not guaranteed. Parallelization within a branch is enabled by the Parallel Adaptive Method for Pseudo-Arclength Continuation (PAMPAC) [14]. This method works with multiple predictors (with different step sizes) and consequently correctors to select an optimal step size, which can be performed in parallel. The PAMPAC method focuses on selecting a maximal step size for the ALM for which the method does not converge.

In this paper, a parallelization of the arc-length method is presented that is independent of the physical nature of the underlying problem. That is, the method is developed such that the parallelization can be performed within the branches. In addition to parallelization, the presented arc-length scheme also provides inherent adaptivity; therefore, the method is referred to as the Adaptive Parallel Arc Length Method (APALM). The working principle of the APALM is based on a multi-level approach – inspired by MGRIT methods – where a coarse serial approximation of the solution space is refined in parallel until a measure of convergence is achieved. Contrary to PAMPAC, the present method does not maximise the step size for convergence of the ALM iterations, but instead the parallelization is based on convergence of the solution sub-intervals. Without loss of generality, the method is developed given a constraint equation for the arc-length method; thus, it is generalised for the Riks and Crisfield methods, amongst other methods available.

The outline of this paper is as follows: Section 2 provides a background on arc-length methods. In section 3, the parallelization of arc-length methods is presented, referred to as the APALM. Thereafter, section 4 provides algorithms for non-intrusive implementation of the APALM, given an implementation of an existing ALM. Section 5 provides numerical benchmark problems and an application to the analysis of a snapping meta-material, inspired by [15]. Finally, section 6 provides conclusions on the presented method.

2. Arc-length methods

In this section, the concept of arc-length methods is presented for the sake of completeness. For a detailed overview, one can consult references [3,4,16,17]. Let $\boldsymbol{G}(\boldsymbol{u},\lambda)=\boldsymbol{0}$ be a non-linear system of equations to be solved, with $\boldsymbol{u}$ the solution to the system of equations given a parameter λ. For structural analyses, $\boldsymbol{u}$ is typically a vector containing discrete displacements of the degrees of freedom, and λ is a factor scaling the magnitude of an applied load $\boldsymbol{P}$, i.e.

$$\boldsymbol{G}(\boldsymbol{u},\lambda)=\boldsymbol{N}(\boldsymbol{u})-\lambda\boldsymbol{P}, \tag{1}$$

where $\boldsymbol{N}(\boldsymbol{u})$ is a vector of internal forces, depending on the deformation $\boldsymbol{u}$. For incremental analyses, i.e., quasi-static analyses, a series of solutions $\boldsymbol{w}_i=(\boldsymbol{u}_i,\lambda_i)$ is obtained by computing increments $\Delta\boldsymbol{w}_i=(\Delta\boldsymbol{u}_i,\Delta\lambda_i)$ such that $\boldsymbol{w}_{i+1}=\boldsymbol{w}_i+\Delta\boldsymbol{w}_i$ and equation (1) is satisfied for $\boldsymbol{w}_{i+1}$. These solutions can be obtained by Newton iterations: i) fixing λ and finding $\boldsymbol{u}$ (*load control*); ii) fixing some degrees of freedom in $\boldsymbol{u}$ and finding all $\boldsymbol{u}$ and λ (*displacement control*); or iii) constraining λ and $\boldsymbol{u}$ and solving for both (*arc-length control*); see Fig. 1. In the case of arc-length control, the increment $\Delta\boldsymbol{w}$ is measured by an increment length $d(\Delta\boldsymbol{w})$

$$d(\Delta\boldsymbol{w})=\Delta\boldsymbol{u}^\top\Delta\boldsymbol{u}+\Psi^2\Delta\lambda^2\boldsymbol{P}^\top\boldsymbol{P}, \tag{2}$$

where Ψ is a scaling parameter given in [18,19]. The increment $\Delta\boldsymbol{w}$ is constrained by the arc-length $\Delta\ell$ in the constraint equation

$$f(\Delta\boldsymbol{w})=d(\Delta\boldsymbol{w})-\Delta\ell=0. \tag{3}$$

Since $\boldsymbol{G}(\boldsymbol{u},\lambda)$ is non-linear, the increment $\Delta\boldsymbol{w}_i$ is obtained iteratively, i.e., $\Delta\boldsymbol{w}_{i,k+1}=\Delta\boldsymbol{w}_{i,k}+\delta\boldsymbol{w}_i$ with iteration count k. The constraint equation is solved together with equation (1) in every iteration, yielding the Riks and Crisfield methods [3,4]

$$f(\Delta\boldsymbol{w}_{i,k},\Delta l)=\Delta\boldsymbol{u}_{i,0}^\top\Delta\boldsymbol{u}_{i,k}+\Psi^2\Delta\lambda_{i,0}\Delta\lambda_{i,0}\boldsymbol{P}^\top\boldsymbol{P}-\Delta\ell^2=0,\quad\text{Riks,} \tag{4}$$

$$f(\Delta\boldsymbol{w}_{i,k},\Delta l)=\Delta\boldsymbol{u}_{i,k}^\top\Delta\boldsymbol{u}_{i,k}+\Psi^2\Delta\lambda_{i,k}\Delta\lambda_{i,k}\boldsymbol{P}^\top\boldsymbol{P}-\Delta\ell^2=0,\quad\text{Crisfield,} \tag{5}$$

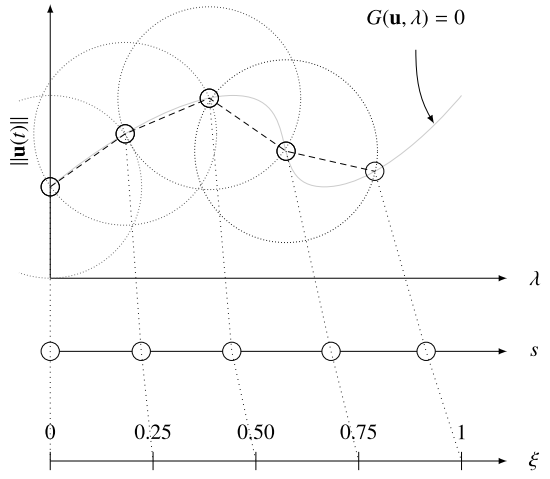
where $\Delta\boldsymbol{w}_0$ is the increment in the first iteration. The Crisfield method generally performs well with sharp snap-backs but has the disadvantage that the constraint equation has two intersections with the path formed by equation (1). Hence, a root has to be selected, which is elaborated in the works [4,16]. When multiple intersections are found, complex roots are found [20], which can be resolved using one of the methods proposed in [21,22]. It should be noted that any other arc-length method can be used within the scheme proposed in this paper, as long as the constraint equation is satisfied when the arc-length step is converged.

3. Adaptive parallel arc-length method

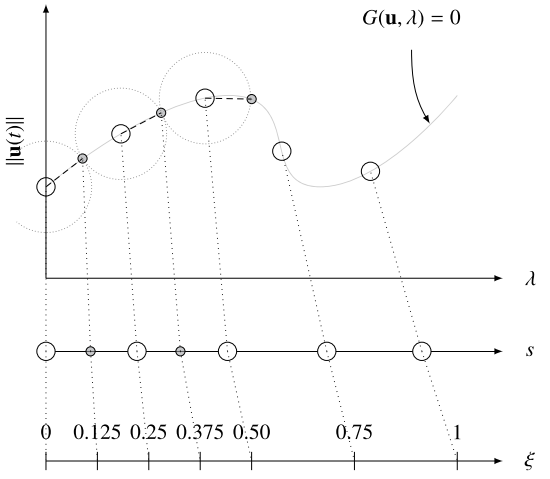
In this section, our new approach, the APALM, is presented. Firstly, the method is conceptualised along with some illustrative figures (section 3.1). Secondly, details are provided on the curve parameterization and the measurement of errors (section 3.2). Lastly, section 3.3 presents (re-)parameterization methods for the solution curve. These parameterizations will be essential to the data structure of the APALM. It should be noted that the method described in this section is presented only for one continuation parameter, λ.

3.1. Concept

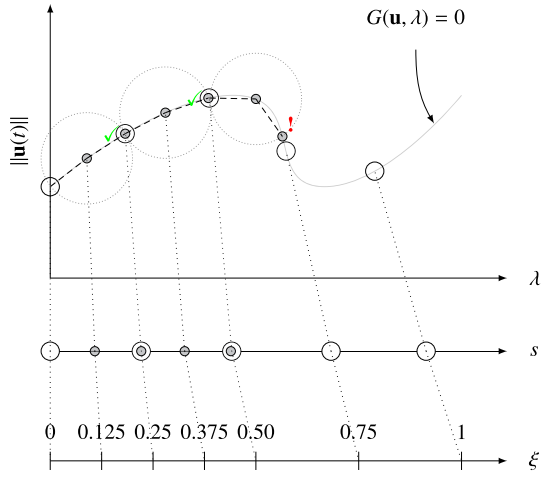
Learning from parallel-in-time methods like Parareal or MGRIT, parallelization in the APALM is achieved from a subdivision of the *curve*



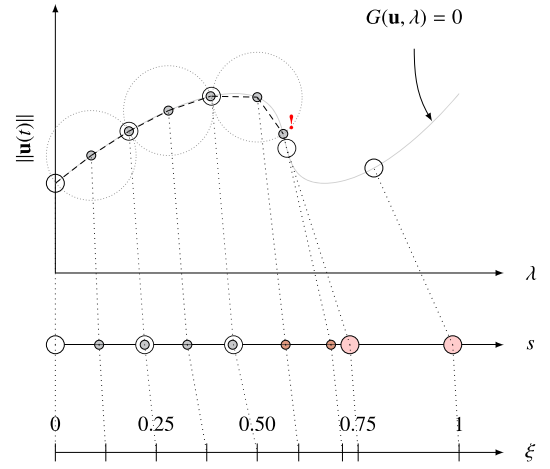
(a) **Initialization based on computed reference solutions.** Without losing generality, the solutions are separated by a fixed distance, Δs. Given their distances, an initial estimation of the *curve length* s can be produced, which can be mapped on the *parametric domain* ξ.



(b) **Parallel computation of intervals.** On each interval, a finer estimate can be performed by splitting the interval into n sub-intervals ($n = 2$ here).



(c) **Parallel verification of intervals.** When the last sub-interval is computed, the solution is verified with the next known reference solution (here, the solutions following from the initial simulations). When the distance is sufficiently small, the segment can be marked as convergent, and the in-between solutions on the interval can be written.



(d) **Curve-length reparameterization.** For sub-intervals where the length deviates too much from the previous length, i.e., where the distance between the last computed point and the reference is too large, all solutions of the sub-interval are added to the parameterization, and the known data points ahead of the newly computed points are shifted in the curve parametric coordinates.

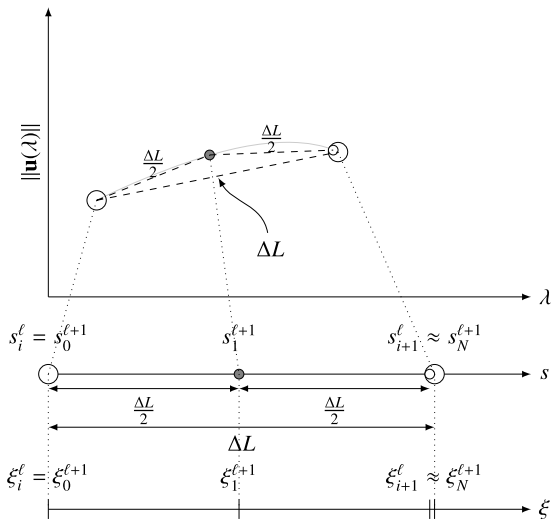
Fig. 2. Concept of the APALM. The large open circles represent *reference solutions* from a previously computed level. The small solid circles represent *new data* on the interval between two reference solutions, computed by the arc-length method (here the large dashed circle). The black dashed line indicates the curve estimation for which the sum is equal to the total curve length.

length domain. Contrary to MGRIT and Parareal, where the temporal domain $t \in [T_0, T_1]$ is fixed, the APALM will work with a changing *curve length domain* $s \in [S_0, S_1]$ depending on the length of the traversed path, with an underlying fixed parametric domain with parametric coordinate $\xi \in [0, 1]$. The APALM is initialised with an initial coarse grid approximation, in which the parametric and the curve length domains are subdivided into sub-domains $\xi \in [\xi_i, \xi_{i+1}]$ and $s \in [s_i, s_{i+1}]$, respectively, as illustrated in Fig. 2.

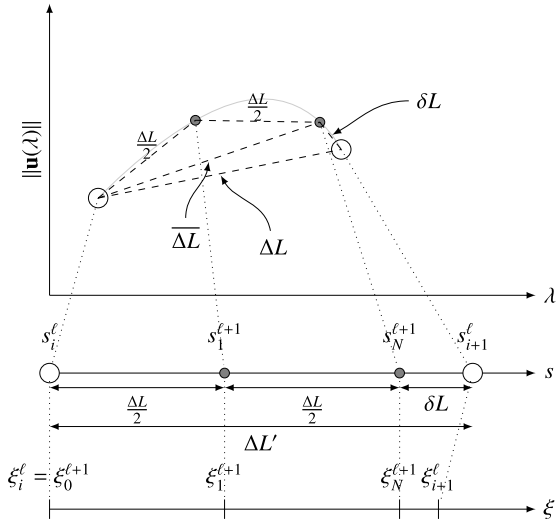
In the initialization phase of the APALM, the first subdivision into sub-intervals is made (see Fig. 2a). Here, the sizes of the sub-intervals $s \in [s_i^\ell, s_{i+1}^\ell]$ are determined based on the distance measure that is used by the corresponding ALM; see equation (2). Note that the superscript ℓ denotes the ℓ^{th} level. Based on the initial curve-length domain $s \in [0, S]$, where S is the total length of the initial curve, and the corresponding sub-intervals, the curve-length domain can be mapped accordingly onto a parametric domain; see section 4 for more details.

With an initialised computational domain, the number of sub-intervals determines the degree of parallelization. On any sub-interval, $[s_i^\ell, s_{i+1}^\ell]$ data at the start-point and end-point is known, which can be used to initiate an arc-length method to re-compute the sub-interval with N increments, i.e., with an arc-length of $\Delta L_i^{\ell+1} = \Delta L_i^\ell / N$ (see Fig. 2b).

After sub-interval $[s_i^{\ell+1}, s_{i+1}^{\ell+1}]$ has been finished, the distance of the end-point of the sub-interval can be compared to the previously known solution at s_{i+1}^ℓ, which is called *parallel verification of intervals* in Fig. 2c. Since the sub-interval is traversed in N increments with length $\Delta s_i^0 / N$, the triangle inequality with the arc-length measure implies that there must be a distance greater than or equal to zero between the newly found end-point and the reference end-point. The more 'curved' the domain in-between, the larger this distance. Based on an error measure (see section 3.2), intervals with a relatively large deviation between the coarse-level arc length and the fine-level arc length are to be marked for 'refinement'.



(a) Nearly straight interval where the small white dot is sufficiently close to the coarse reference. In this case, the solution of the second increment is not added to the parameterization.



(b) Curved interval where the solution of the second increment is ΔL_u away from the coarse reference. In this case, the distance between the coarse reference points δL is smaller than the actually traversed distance $\Delta L'$.

Fig. 3. Error measures on a nearly straight interval (a) and a curved interval (b). For the nearly straight interval, the distance δL (see b) is sufficiently small, whereas for the curved interval, it is too big. The measures ΔL and $\Delta L'$, $\overline{\Delta L}$, and δL are, respectively, the coarse arc length, the fine arc length, the lower distance, and the absolute error.

Lastly, the intervals with a too large deviation in the newly computed curve length need to be reparameterized (see section 3.3). This is because the total curve-length parameterization is elongated exactly by the distance between the newly computed endpoint and the previously known point. By this means, the reference interval is subdivided into $N+1$ sub-intervals, and the data corresponding to the N newly computed point is stored. For sub-intervals that have an error below the tolerance, only $N-1$ points are stored as references, and no reparameterization takes place. The process is sketched in Fig. 2d. After reparameterization, the marked interval can be re-computed, and the process can be repeated from Fig. 2b onward.

Remark 1 *(Difference with parallel-in-time methods)*. As mentioned, the multi-level approach that is employed in this method is derived from the idea of parallel-in-time methods. However, the fundamental difference between time integration and continuation comes from the fact that time integration methods typically compute the solution on the next time step with a certain time integration error $\mathcal{O}(\Delta t^p)$. Parallel-in-time methods rely on this time integration error to mark solution intervals as converged or not, and additionally, updated solutions contain smaller time integration errors, so sub-intervals need to be recomputed as soon as solutions previously in time have been updated.

For arc-length methods, Newton's method is applied to a system of equations that solves the arc-length constraint equation together with the discretized system $\boldsymbol{G}(\boldsymbol{u}, \lambda) = \boldsymbol{0}$. Therefore, the error of the solution that is found after an arc-length increment is independent of the arc-length increment size but solely depends on the convergence tolerance of Newton's method. Therefore, the end-point of an interval does not have to be updated, nor do intervals after the update be recomputed. This implies, in principle, that parallel corrections of the arc-length steps are not needed, since the intervals already capture the structural response at the equilibrium path. However, the parallel corrections are still meaningful to capture the equilibrium path in desired detail, in the case where the initial step size is chosen very coarse. As will follow from the results section, cf. section 5, the parallel performance increases for fewer (hence coarser) initial intervals.

Remark 2 *(Path-dependency)*. The concept presented in this paper assumes path-independence of the equation to be solved, in order to assume that from a given starting point on the equilibrium path, the same end-point could be reached irrespective of the computed intervals on the path in-between the points. Path-dependent problems are out of the scope of this paper.

3.2. Error measures

The refinement of computed sub-intervals depends on the distances between the points in the original (coarse) interval and the newly obtained solutions in this sub-interval. Here, error measures are presented, that can be used to mark an interval $[s_i^\ell, s_{i+1}^\ell]$ based on the obtained solutions $\{s_k^{\ell+1}\}_{k=0,\dots,N}$ at the finer level. Fig. 3 presents two possible situations: a nearly straight interval that would not be marked for refinement, and a curved interval that would be marked for refinement. Here, the interval is considered 'curved' in the discrete solution space if the hyperdimensional path between two solutions differs from the hyperplane between these solutions. The errors that determine the marking of an interval for refinement are illustrated in Fig. 3b and can be interpreted as follows: ΔL is the original arc length between two coarse solutions, $\Delta L'$ is the newly obtained length between two coarse solutions, the *lower distance* $\overline{\Delta L}$ is the distance between the start of the interval and the last solution at the fine level, and δL is the distance between the last obtained solution on the fine level and the final point on the coarse level. Using these distances, the *total error* (ε), the *lower error* (ε_l), and the *upper error* (ε_u) can be defined:

$$\varepsilon = (\Delta L' - \Delta L)/\Delta L, \qquad \text{total error,} \tag{6}$$

$$\varepsilon_l = (\Delta L - \overline{\Delta L})/\Delta L, \qquad \text{lower error,} \tag{7}$$

$$\varepsilon_u = (\varepsilon - \varepsilon_l)/\Delta L, \qquad \text{upper error.} \tag{8}$$

Here, the total error is the total difference between the coarse and fine intervals; the lower error is the contribution of the first N sub-intervals; and the upper error is the contribution of δL to the total error. Depending on these errors and specified tolerances, refinement rules can be set up, in particular:

$$\text{Refine the first } N \text{ intervals} \longleftrightarrow \varepsilon_l > \text{TOL}_l, \tag{9}$$

$$\text{Refine the last interval} \longleftrightarrow \varepsilon_u > \text{TOL}_u. \tag{10}$$

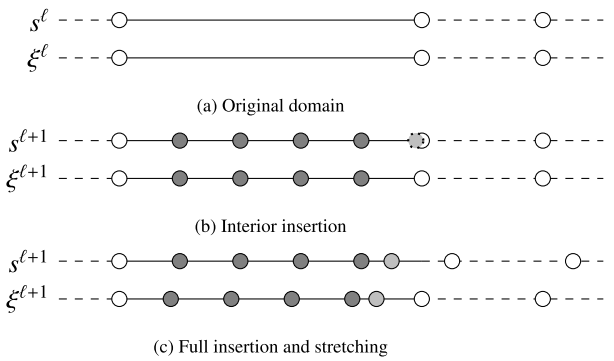


Fig. 4. Domain parameterizations on the curve-length domain s and the parameter domain ξ with levels ℓ and $\ell+1$. Fig. 4a illustrates the original domain, Fig. 4b illustrates the insertion of interior points in the case of a sufficiently close approximation of the end-point of the domain and Fig. 4c illustrates the full insertion of all sub-domain solutions combined with the stretching of the curve length domain.

3.3. Curve (re-)parameterization

As indicated in Fig. 2, the concept of the APALM is supported by the parameterization of the solutions of $G(\boldsymbol{u},\lambda)=\mathbf{0}$ by parameterizing the curve length using the increment length $d(\Delta\boldsymbol{w})$ embedded in the arc-length method. As illustrated in Fig. 2, the APALM maps solutions $\boldsymbol{w}$ to a point on the curve-length domain $[0,S]$, and points on the curve-length domain are mapped on a parametric domain $[0,1]$.

Provided a series of solutions from the initialization phase $\{\boldsymbol{w}_i^0\}_{i=0,\dots,I}$, with I denoting the total number of initial points, and defining solution intervals by $\Delta\boldsymbol{w}_i^\ell=\boldsymbol{w}_{i+1}^\ell-\boldsymbol{w}_i^\ell$, each solution $\boldsymbol{w}_i^0$ can recursively be assigned to the curve-length and parametric domains by

$$s_{i+1}^0=s_i^0+d(\Delta\boldsymbol{w}_i^0),\qquad i=1,\dots,I-1,\quad s_0=0, \tag{11}$$

$$\xi_i^0=\frac{s_i}{s_I},\qquad i=0,\dots,I, \tag{12}$$

where the superscript 0 represents the 0^{th} level. In addition, equation (11) guarantees that $S=s_I$ marks the total length of the curve that has been traversed, measured by the distance between each solution. Given the curve-length coordinates of each point as an increasing sequence, the parametric domain can simply be obtained by scaling the domain back to $[0,1]$; see equation (12). In the following, two ways of adding solutions to the parameterization are defined: i) interior insertion, and ii) full insertion and stretching. The operations are defined given a parent interval $[s_i^\ell,s_{i+1}^\ell)$ in which a set of new solutions $\{s_k^{\ell+1}\}_{k=0,\dots,N}$, where $s_0^{\ell+1}=s_i^\ell$, is computed, with N the total number of points in the interval; see Fig. 4a.

Firstly, the interior insertion operation inserts solutions *within* the sub-interval, see Fig. 4b, and is later used for intervals where the error is small. The idea behind this operation is that the solutions $\{s_k^{\ell+1}\}_{k=1,\dots,N-1}$ between s_i^ℓ and s_{i+1}^ℓ are inserted and that the solution $s_N^{\ell+1}$ is not added to the map. In the case of the interior insertion, the points $s_k^{\ell+1}$ and their parametric coordinates $\xi_k^{\ell+1}$ are added by:

$$\begin{aligned}&s_{k+1}^{\ell+1}=s_k^{\ell+1}+d(\Delta\boldsymbol{w}_k^{\ell+1}),\quad k=0,\dots,N-2,\\&s_0^{\ell+1}=s_i^\ell,\quad s_N^{\ell+1}=s_{i+1}^\ell,\end{aligned} \tag{13}$$

$$\xi_{k+1}^{\ell+1}=\xi_k^{\ell+1}+(\xi_{i+1}^\ell-\xi_i^\ell)\frac{s_{k+1}^{\ell+1}-s_k^{\ell+1}}{s_{i+1}^\ell-s_i^\ell},\quad j=0,\dots,N-2. \tag{14}$$

Note that $\boldsymbol{w}_k^{\ell+1}$ denotes the k^{th} solution on level $\ell+1$ on the computed sub-interval, here $[s_i^\ell,s_{i+1}^\ell]$.

The full insertion and stretching operation inserts the solutions of the sub-interval, including its end point, and also stretches the curve parameterization (see Fig. 4c), which is later used for intervals where the error is large, hence intervals that need refinement. The idea behind this operation is that the solutions $\{s_k^{\ell+1}\}_{k=1,\dots,N-1}$ between s_i^ℓ and s_{i+1}^ℓ are inserted and that the point s_{i+1}^ℓ is shifted such that $s_{i+1}^\ell=s_N^{\ell+1}$ and such that all points further than s_{i+1}^ℓ are updated to $\tilde{s}_j^\ell$ by the shift using the distance between the last computed solution and the reference solution, i.e. $d(\Delta\boldsymbol{w}_N^{\ell+1})$, $\Delta\boldsymbol{w}_N^{\ell+1}=\boldsymbol{w}_{i+1}^\ell-\boldsymbol{w}_N^{\ell+1}$:

$$\begin{aligned}&s_{k+1}^{\ell+1}=s_k^{\ell+1}+d(\Delta\boldsymbol{w}_k^{\ell+1}),\\&\quad k=0,\dots,N-1,\quad s_0^{\ell+1}=s_i^\ell,\quad s_{i+1}^\ell=s_N^{\ell+1},\end{aligned} \tag{15}$$

$$\xi_{k+1}^{\ell+1}=\xi_k^{\ell+1}+(\xi_{i+1}^\ell-\xi_i^\ell)\frac{s_{k+1}^{\ell+1}-s_k^{\ell+1}}{s_{i+1}^\ell-s_i^\ell},\qquad k=0,\dots,N-1, \tag{16}$$

$$\tilde{s}_j^\ell=s_j^\ell+d(\boldsymbol{w}_N^{\ell+1},\boldsymbol{w}_{i+1}^\ell),\qquad j=i+1,\dots,I. \tag{17}$$

As can be noticed in equation (16), the re-scaling of ξ is done using the parametric length of the original interval at level ℓ, $(\xi_{i+1}^\ell-\xi_i^\ell)$ and the curve coordinate s_{i+1} relative to the beginning point of the interval s_i with respect to the (updated) total curve length of the interval $s_{i+1}-s_i$, which is similar to the well-known *chord-length parameterization* in splines [23].

4. Implementation

In this section, data structures and algorithms for the implementation of the APALM are presented. In section 4.1, a data structure is provided for the implementation of the APALM. Thereafter, section 4.2 provides algorithms for the implementation of the APALM, and section 4.3 elaborates on the extension of these methods to multiple branches, hence enabling arc-length exploration.

4.1. Data structure

Since the APALM is based on a sub-interval approach where the start and end points of each sub-interval are known, a data structure referencing the sub-intervals is essential. Since the curve-length coordinate is subject to change after reparametrisation of the curve and since the curve parameter is fixed, the logical choice is to connect the data to parametric coordinates. That is, a series of discrete maps is constructed such that solutions, levels, and curve-length coordinates can be obtained via a parametric point ξ_k^ℓ.

Fig. 5 shows the data structure behind the APALM. Firstly, the data structure contains the map $S(\xi):[0,1]\to[0,S]$, which is the map that maps the parametric coordinate to the curve-length domain. Secondly, the maps $\mathcal{U}(\xi):[0,1]\to\mathbb{R}^{n+1}$ and $\mathcal{U}'(\xi):[0,1]\to\mathbb{R}^{n+1}$ map the solution and the previous solution from the parametric domain to the solution and previous solution domain, respectively. The mapper $\mathcal{U}'(\xi):[0,1]\to\mathbb{R}^{n+1}$ is constructed in order to construct the predictor of the ALM. Lastly, the map $\mathcal{L}(\xi):[0,1]\to\mathbb{N}$ is a map that can be used to obtain the level of a parametric point, which is optional but can be useful to limit the method to a certain depth.

4.2. Algorithms

Given the underlying data structure of the APALM (see section 4.1), algorithms are defined for its implementation. Firstly, it is assumed that the APALM is based on an ALM with possibly a black-box implementation, striving for the non-intrusiveness of the method. The required routines for the underlying ALM are:

- $\boldsymbol{w}_{i+1}^\ell\leftarrow$ `step` $(\boldsymbol{w}_i^\ell,\Delta\boldsymbol{w}_i^\ell\,\Delta L)$: Performs a step with length ΔL starting at point $\boldsymbol{w}_i^\ell$ and returns the new solution $\boldsymbol{w}_{i+1}^\ell$. Given the current solution and the previous solution, a predictor for the initial iteration that employs $\Delta\boldsymbol{w}_i^\ell=\boldsymbol{w}_i^\ell-\boldsymbol{w}_{i-1}^\ell$ could be available, as well as one for a cold start, i.e., $\Delta\boldsymbol{w}_i^\ell=\mathbf{0}$.

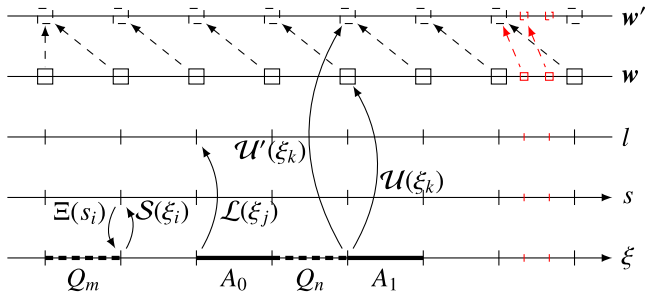


Fig. 5. The data structure behind the APALM. The axes represent data sets, which are monotonically increasing when the axis has an arrow. Solid arrows represent mappers from one axis to another, and dashed arrows represent data references. The mappers $\Xi(s_i)$ and $\mathcal{S}(\xi_i)$ map between the curve parametrisation and the curve length axes. The former takes a curve length s_i and returns the curve parameter ξ_i, and the latter maps the inverse. The mappers $\mathcal{U}(\xi_k)$ and $\mathcal{U}'(\xi_k)$ return the solution $\boldsymbol{w}_j$ and the previous solution $(\boldsymbol{w}')_j$, respectively, given a parametric coordinate ξ_j, and the mapper $\mathcal{L}(\xi_j)$ returns the level on which the coordinate ξ_j was computed. The guess is a data reference to the previous solution. The thick solid intervals represent running jobs assigned with an `ID`, and the thick dashed intervals represent queued intervals. Each interval is represented by a start-point and an end-point tuple (ξ_l, ξ_{l+1}). The red lines, squares, and arrows represent the `submit` operation when solutions are added to the data structure.

- $\Delta s \leftarrow$ `distance` $(\boldsymbol{w}_i^\ell, \boldsymbol{w}_j^\ell)$: gets the distance between two points $\boldsymbol{w}_i^\ell$ and $\boldsymbol{w}_j^\ell$, using equation (2) with $\Delta \boldsymbol{w}_i^\ell = \boldsymbol{w}_i^\ell - \boldsymbol{w}_j^\ell$.

In the following, three implementations are presented. The first implementation is a *serial implementation* without communication but with queueing, referred to as the Adaptive Serial Arc-Length Method (ASALM). The serial implementation provides the building blocks for the *hybrid implementation* and the *parallel implementation*. The *hybrid implementation*, referred to as the Adaptive Serial-Parallel Arc-Length Method (ASPALM), is a hybrid version of the APALM where parallel corrections are performed after a serial solve has finished. The *parallel implementation*, on the other hand, starts parallel corrections as soon as the first interval has been initialised; hence, there is no separation between a serial phase and a parallel phase. The parallel implementation is the final APALM.

4.2.1. Serial implementation

The global workflow for a serial APALM, i.e., the ASALM, is illustrated in Algorithm 1. As seen in this algorithm, the initialization is performed using a `serialSolve` routine, which defines the initial solution sequence $\{\boldsymbol{w}_i^0\}_{i=0}^I$ with I steps on level $\ell = 0$. In addition, this routine also provides a sequence of curve-length coordinates, $\{s_i^0\}_{i=0}^I$. Based on these sequences, the mappers from Fig. 5 and a queue Q are initialised in the `initializeMap` routine. Given the queue Q, the `correctQueueSerial` routine provides a sequence of solutions $\{\boldsymbol{w}_i\}$ and of curve parameters $\{s_i\}$ spanning multiple levels, hence the superscript ℓ is omitted.

Algorithm 1 Global ASALM routine (`ASALM`). The ASALM first consists of a serial solve of the whole curve length domain, followed by an evaluation of subintervals.

Input: ΔL, I
1: $\{\boldsymbol{w}_i^0\}_{i=0}^I, \{s_i^0\}_{i=0}^I \leftarrow$ `serialSolve`$(\Delta L, I)$
2: $Q \leftarrow$ `initializeMap`$(\{\boldsymbol{w}_i^0\}_{i=0}^I, \{s_i^0\}_{i=0}^I)$
3: $\{\boldsymbol{w}_i\}, \{s_i\} \leftarrow$ `correctQueueSerial`(Q)
Output: $\{\boldsymbol{w}_i\}$, $\{s_i\}$, $\{\xi_i\}$

Using basic ALM routines, Algorithm 2 defines an algorithm to obtain in serial a coarse approximation to initialise the ASALM. Optionally, a stability computation can be performed after the arc-length step, which could lead to a specialised solution towards a bifurcation point. This allows for automatic exploration of bifurcation diagrams [8,9] and is discussed more in detail in section 4.3.

Algorithm 2 Serial solve (`serialSolve`). This routine provides the initial step for the APALM/ASALM, i.e., the solution data $\{\boldsymbol{w}_i^\ell\}_{i=0,\dots,I}$ and the corresponding curve-length parameters $\{s_i^\ell\}_{i=0,\dots,I}$ on level $\ell = 0$.

Input: ΔL, I
1: Initialise $\boldsymbol{w}_0^\ell$, $s_0^\ell = 0$
2: $\boldsymbol{w}_1^\ell \leftarrow$ `step` $(\boldsymbol{w}_0^\ell, \boldsymbol{0}, \Delta L)$ ▷ Compute first solution
3: $s_1 \leftarrow \Delta L$
4: **for** $k = 1, \dots, I-1$ **do**
5: $\quad \boldsymbol{w}_{k+1}^\ell, \Delta s_{k+1}^\ell \leftarrow$ `initiate` $(\boldsymbol{w}_k^\ell, \Delta \boldsymbol{w}_{k-1}^\ell, \Delta L)$ ▷ Compute new solution
6: $\quad s_{k+1}^\ell = s_k^\ell + \Delta s_{k+1}^\ell$ ▷ Compute curve coordinate
7: **end for**
Output: $\{\boldsymbol{w}_i^\ell\}_{i=0}^I$, $\{s_i^\ell\}_{i=0}^I$

In Algorithm 2, the `initiate` routine (see Algorithm 3) computes an arc-length interval and returns a new point $\boldsymbol{w}_k^\ell$ and the traversed distance Δs_k^ℓ, provided the previous point $\boldsymbol{w}_{k-1}^\ell$, the previous solution interval $\Delta \boldsymbol{w}_k^\ell$ and the intended arc-length step size ΔL and using the `step` and `distance` functions. Typically, Δs_k^ℓ is equal to ΔL unless the arc-length step does not converge and needs to be bisected.

Algorithm 3 Initiation of an interval (`initiate`). Given a previous solution $\boldsymbol{w}_k^\ell$, the previous step size $\boldsymbol{w}_{k-1}^\ell$, and the desired arc-length ΔL, this routine returns a new solution $\boldsymbol{w}_{k+1}^\ell$ and a distance with respect to the previous solution $\boldsymbol{w}_k^\ell$, denoted by Δs_{k+1}^ℓ.

Input: $\boldsymbol{w}_k^\ell$, $\boldsymbol{w}_{k-1}^\ell$, ΔL
1: $\Delta \boldsymbol{w}_k^\ell = \boldsymbol{w}_k^\ell - \boldsymbol{w}_{k-1}^\ell$ ▷ Compute previous step
2: $\boldsymbol{w}_{k+1}^\ell \leftarrow$ `step` $(\boldsymbol{w}_{k-1}^\ell, \Delta \boldsymbol{w}_k^\ell, \Delta L)$ ▷ Compute new solution
3: $\Delta s_{k+1}^\ell \leftarrow$ `distance` $(\boldsymbol{w}_{k+1}^\ell, \boldsymbol{w}_k^\ell)$ ▷ Compute curve coordinate
Output: $\boldsymbol{w}_{k+1}^\ell$, Δs_{k+1}^ℓ

As soon as a set of solution data and curve parameters, $\{\boldsymbol{w}_i^\ell\}_{i=0,\dots,I}$ and $\{s_i^\ell\}_{i=0,\dots,I}$, respectively, are known, the initialization of the parallel computations can take place. In this initialization, the maps from Fig. 5 are constructed and a queue Q of jobs is created, cf. Algorithm 4.

Algorithm 4 Parallel initialization (`initializeMap`). Within this algorithm, the maps $\mathcal{U}$, $\mathcal{U}'$, $\mathcal{S}$, and Ξ are constructed from a series of solutions and corresponding curve length coordinates, $\{\boldsymbol{w}_i^\ell\}_{i=0}^I$ and $\{s_i^\ell\}_{i=0}^I$, respectively, both on level $\ell = 0$. Furthermore, the queue Q is initialised by adding all subintervals to the queue.

Input: $\{\boldsymbol{w}_i^\ell\}_{i=0}^I$, $\{s_i^\ell\}_{i=0}^I$
1: Add $\boldsymbol{w}_0^\ell$ to $\mathcal{U}$, s_0^ℓ to $\mathcal{S}$, and ξ_0^ℓ to Ξ solutions to the map $\mathcal{U}$ ▷ Add the start of the level 0
2: **for** $k = 1, \dots, I$ **do**
3: $\quad$ Add $\boldsymbol{w}_k^\ell$ to $\mathcal{U}$ and $\boldsymbol{w}_{k-1}^\ell$ to $\mathcal{U}'$.
4: $\quad$ Add s_k^ℓ to $\mathcal{S}$ and ξ_k^0 to Ξ.
5: $\quad$ Add $Q_{k-1} = [\xi_{k-1}^\ell, \xi_k^\ell)$ to Q. ▷ Construct elements of the queue Q
6: **end for**
Output: $Q = \{Q_k = [\xi_k^\ell, \xi_{k+1}^\ell]\}_{k=0,\dots,I}$

After initialization, the computation of the sub-intervals can take place. This requires the routine `correctQueueSerial` as defined in Algorithm 5 for fully serial computations. That is, no communication between manager and worker takes place since everything will be done on the same node.

The computation of the sub-interval takes place in the `correct` routine of Algorithm 6 and can be used both in a serial and a parallel

Algorithm 5 The routine that solves the queue (`correctQueueSerial`). Given a queue Q, this algorithm takes an entry from the queue using `pop` and solves the defined interval using `correct`. The new solution is added to the solution maps, and if required, new jobs are added to the queue Q using `submit`. The final solutions are collected from the maps using the `collectSolutions` routine.

Input: Q
1: **while** $Q \neq \emptyset$ **do**
2: Q, ID, ΔL, $\boldsymbol{w}_i^\ell$, $\boldsymbol{w}_{i-1}^\ell, \boldsymbol{w}_{i+1}^\ell \leftarrow$ pop(Q)
3: $\{d_k^{\ell+1}\}_{k=0,\dots,N-1}$, $\{\boldsymbol{w}_k^{\ell+1}\}_{k=0,\dots,N}$, $\overline{\Delta L}$, $\delta L \leftarrow$ correct($N, \Delta L_0, \boldsymbol{w}_i^\ell, \boldsymbol{w}_{i-1}^\ell, \boldsymbol{w}_{i+1}^\ell$)
4: $Q \leftarrow$ submit(ID, $\{d_k^{\ell+1}\}_{k=0,\dots,N-1}, \{\boldsymbol{w}_k^{\ell+1}\}_{k=0,\dots,N}, \overline{\Delta L}, \delta L, Q$) ▷ Submit the job; adds new jobs to Q if needed
5: **end while**
6: $\{\boldsymbol{w}_i\}$, $\{s_i\} \leftarrow$ collectSolutions
Output: $\{\boldsymbol{w}_i\}$, $\{s_i\}$

implementation. Given a number of sub-intervals N, the original distance between the end-points of the interval ΔL_0 and the start point, previous solutions, and reference solutions $\boldsymbol{w}_i^\ell$, $\boldsymbol{w}_{i-1}^\ell$ and $\boldsymbol{w}_{i+1}^\ell$, respectively, this routine computes a series of solutions of the sub-interval $\{\boldsymbol{w}_k^{\ell+1}\}_{k=0,\dots,N}$, their distances $\{d_k\}_{k=0,\dots,N-1}^{\ell+1}$ and the distances $\overline{\Delta L}$ and δL for error computation. Note that the distance $\Delta L'$ can be computed by taking the sum of the distances.

Algorithm 6 The routine that solves an interval (`correct`). This routine takes a number of subintervals N, the desired step length for the total interval, the start point $\boldsymbol{w}_i^\ell$, the previous point $\boldsymbol{w}_{i-1}^\ell$, and the next point $\boldsymbol{w}_{i+1}^\ell$. It returns the solutions on the subinterval and the distances between them, respectively $\{d_j^{\ell+1}\}_{j=0,\dots,N-1}$ and $\{\boldsymbol{w}_j^{\ell+1}\}_{j=0,\dots,N}$, as well as the distances $\overline{\Delta L}$ and δL. When the `step` does not converge, it is assumed that step size modification takes place and that the data points are adjusted accordingly.

Input: N, ΔL_0, $\boldsymbol{w}_i^\ell$, $\boldsymbol{w}_{i-1}^\ell$, $\boldsymbol{w}_{i+1}^\ell$
1: Initialise output vectors $\{d_k^{\ell+1}\}_{k=0,\dots,N-1}$, $\{\boldsymbol{w}_k^{\ell+1}\}_{k=0,\dots,N}$
2: $\Delta L = \Delta L_0 / N$ ▷ Defines the size of the sub-intervals
3: $\boldsymbol{w}_0^{\ell+1} = \boldsymbol{w}_i^\ell$
4: $\Delta \boldsymbol{w}^{\ell+1} = \boldsymbol{w}_i^\ell - \boldsymbol{w}_{i-1}^\ell$ ▷ Determine previous arc-length step, to be used to predict the step on $\ell + 1$
5: **for** $k = 0, \dots, N-1$ **do**
6: $\boldsymbol{w}_{k+1}^{\ell+1} \leftarrow$ step($\boldsymbol{w}_k^{\ell+1}, \Delta \boldsymbol{w}, \Delta L$) ▷ Perform the ALM iteration
7: $\Delta \boldsymbol{w} = \boldsymbol{w}_{k+1}^{\ell+1} - \boldsymbol{w}_k^{\ell+1}$ ▷ Update the solution step
8: $d_k \leftarrow$ distance($\boldsymbol{w}_{k+1}^{\ell+1}, \boldsymbol{w}_k^{\ell+1}$) ▷ Gets the distance
9: **end for**
10: $\delta L \leftarrow$ distance($\boldsymbol{w}_{i+1}^\ell, \boldsymbol{w}_N^{\ell+1}$)
11: $\overline{\Delta L} \leftarrow$ distance($\boldsymbol{w}_N, \boldsymbol{w}_0$)
Output: $\{d_j^{\ell+1}\}_{j=0,\dots,N-1}$, $\{\boldsymbol{w}_j^{\ell+1}\}_{j=0,\dots,N}$, $\overline{\Delta L}$, δL

The `correctQueueSerial` routine includes the `pop`, `submit`, and `collectSolutions` routines. These routines mainly involve read and write operations for the mappers defined in Fig. 5; hence, only a brief description is provided:

- Q, ID, ΔL, $\boldsymbol{w}_i^\ell$, $\boldsymbol{w}_{i-1}^\ell, \boldsymbol{w}_{i+1}^\ell \leftarrow$ pop(Q): Takes the first available interval from the queue Q and returns a job ID, an interval length ΔL, the start solution $\boldsymbol{w}_i^\ell$, the previous solution $\boldsymbol{w}_{i-1}^\ell$ and the next available solution $\boldsymbol{w}_{i+1}^\ell$. It also updates the queue Q internally by removing the current entry.
- $Q \leftarrow$ submit(ID, $\{d_k^{\ell+1}\}_{k=0,\dots,N-1}, \{\boldsymbol{w}_k^{\ell+1}\}_{k=0,\dots,N}, \overline{\Delta L}, \delta L, Q$): Takes a job ID, the series of solutions $\{\boldsymbol{w}_k^{\ell+1}\}_{k=0,\dots,N}$ and their distances $\{d_k^{\ell+1}\}_{k=0,\dots,N-1}$ and the distances $\overline{\Delta L}$ and δL. Using equations (6) to (8), the errors are computed and solution intervals are added to the queue Q if needed.
- $\{\boldsymbol{w}_i\}$, $\{s_i\} \leftarrow$ collectSolutions: Based on the underlying mappers, the solutions of all levels are collected into $\{\boldsymbol{w}_i\}$ and $\{s_i\}$.

4.2.2. Hybrid implementation

The hybrid implementation of the APALM is referred to as the Adaptive Serial-Parallel Arc-Length Method (ASPALM), since it is a two-stage method with a serial initialization and a parallel correction. This concept is similar to the concept presented for the ASALM, but communication between the manager and worker processes is added so that the correction stage can be performed in parallel. To this end, the `correctQueueSerial` routine is re-defined into `correctQueueParallel` and communications between workers and the manager are defined. In the following, the global solution algorithm for the ASPALM is defined in Algorithm 7. As for the ASALM (see Algorithm 1), the initialization is performed in serial by the `serialSolve` routine, and the maps are initialised, both by the manager process. As soon as queue Q is established, the queue can be processed in parallel. Indeed, this implies that the worker processes are idle until the queue Q is fully available.

Algorithm 7 Global ASPALM routine (`ASPALM`). The ASPALM first consists of a serial solve of the whole curve length domain, followed by a parallel evaluation of subintervals. This algorithm is specified for simple `manager-worker` parallelization; more advanced parallelization schemes, e.g., with multiple managers, are easily achieved.

Input: ΔL, I
1: **if** `manager` **then**
2: $\{\boldsymbol{w}_i^\ell\}_{i=0}^I, \{s_i^\ell\}_{i=0}^I \leftarrow$ serialSolve($\Delta L, I$) ▷ See Algorithm 2
3: $Q \leftarrow$ initializeMap($\{\boldsymbol{w}_i^\ell\}_{i=0}^I, \{s_i^\ell\}_{i=0}^I$)
4: $\{\boldsymbol{w}_i^\ell\}, \{s_i^\ell\} \leftarrow$ correctQueueParallel(Q)
5: **else**
6: Initialise stop=false
7: **while** stop = false **do**
8: stop $\leftarrow$ receiveStop
9: workerCorrect()
10: **end while**
11: **end if**
Output: $\{\boldsymbol{w}_i\}$, $\{s_i\}$

As seen in Algorithm 7, the manager process uses the `correctQueueParallel` routine; see Algorithm 9. This routine is called by the manager process and sends and receives data to and from the workers, updates the queue, and heavily relies on communication functions as defined in Table 1. The communication function `sendMetaData` can be omitted for the ASPALM since it is primarily used in the APALM to distinguish between initiation and correction jobs. Furthermore, as described in Table 1, the routine `sendJob` can be called with and without the reference solution $\boldsymbol{w}_{i+1}^\ell$, depending whether it is available (correction) or not (initiation). In the case of the ASPALM, all jobs that are popped from the queue Q in the `correctQueueParallel` routine are by definition correction jobs.

As shown in Algorithm 7, the worker processes will perform the `workerCorrect` from Algorithm 11. This algorithm contains the correction step for any interval that is received from the communications coming from the manager process. The `workerCorrect` is executed until a stop signal is received from the manager process. The latter is broadcast to all workers as soon as queue Q is empty. Table 1 gives an overview of the communication functions that are used for communications between the manager and worker processes in the hybrid and parallel implementations.

4.2.3. Parallel implementation

Contrary to the serial and hybrid implementations, the fully parallel implementation does not work with a two-staged procedure of serial initialization and parallel correction. Instead, the fully parallel solve consists of one stage with a single queue consisting of initialization and

Table 1
Required communications between manager and worker processes for the ASPALM and APALM.

Send/Receive	From	To	Data	Description
`sendMetaData` `receiveMetaData`	Manager	Worker	`ID`, ℓ	Communicates meta-data between the manager and the worker processes. In this case, only the `ID` and the level ℓ are needed.
`sendJob` `receiveJob`	Manager	Worker	`ID`, ΔL_0, $\boldsymbol{w}_i^\ell$, $\boldsymbol{w}_{i-1}^\ell$, $(\boldsymbol{w}_{i+1}^\ell)$	Communicates information to perform the computation of an interval between the manager and the worker processes. The reference solution $\boldsymbol{w}_{i+1}^\ell$ is optional since it is not available for initiation jobs.
`sendData` `receiveData`	Worker	Manager	W_j, `ID`, $\{d_j^{\ell+1}\}_{j=0,\dots,N-1}$, $\{\boldsymbol{w}_j^{\ell+1}\}_{j=0,\dots,N}$, $\overline{\Delta L}$, δL	Communicates the data resulting from a sub-interval computation between the manager and the worker processes. The receive communication also provides the worker from whom the data is received.
`sendStop` `receiveStop`	Manager	Worker (all)	Boolean	Communicates a stop signal between the manager and the worker processes.

correction jobs. For the Adaptive Parallel Arc-Length Method (APALM), the global routine is provided in Algorithm 8.

Algorithm 8 Global APALM routine (`APALM`). The APALM first consists of a serial solve of the whole curve length domain, followed by a parallel evaluation of subintervals. This algorithm is specified for simple `manager-worker` parallelization; more advanced parallelization schemes, e.g., with multiple managers, are easily achieved.

Input: ΔL, I
1: **if** `manager` **then**
2: $Q \leftarrow$ `initializeQueue`$(\Delta L, I)$
3: $\{\boldsymbol{w}_i^\ell\}, \{s_i^\ell\} \leftarrow$ `correctQueueParallel`(Q)
4: $\{\boldsymbol{w}_i\}, \{s_i\} \leftarrow$ `collectSolutions`
5: **else**
6: **while** `true` **do**
7: `stop` $\leftarrow$ `receiveStop`
8: **if** stop **then**
9: Break loop
10: **end if**
11: `ID`, $\ell \leftarrow$ `receiveMetaData`
12: **if** $\ell = 0$ **then**
13: `workerInitiate()`
14: **else**
15: `workerCorrect()`
16: **end if**
17: **end while**
18: **end if**
Output: $\{\boldsymbol{w}_i\}$, $\{s_i\}$, $\{\xi_i\}$

As can be seen in Algorithm 8, the manager process in the APALM only initialises the queue using the `initializeQueue` routine, and it contains the `correctQueueParallel` routine. The former is not specified explicitly since it only initialises a map with zero points and allocates the maximum number of intervals I as well as the interval length ΔL. The `correctQueueParallel` routine is given in Algorithm 9. It applies meta-data communication, see Table 1, to the worker processes so that the distinction based on the data level ℓ in Algorithm 8 can be made by the workers. On the side of the worker processes, the meta-data is received, and if the level is equal to 0, an interval is initiated using `workerInitiate` (see Algorithm 10), and if the level is larger than 0, an interval is corrected using `workerCorrect` (see Algorithm 11); see Algorithm 8. Inside the `workerInitiate` and `workerCorrect` routines, communications from the worker to the manager process (see Table 1) are included.

4.3. Arc-length exploration

To enable multi-branch parallelization of APALM, small modifications are required to the data structure and the algorithms presented in sections 4.1 and 4.2. The easiest multi-branch parallelization is achieved by identifying branch switches only in the serial solve, such that the initialization of the APALM can be done across different branches. In this case, the serial solve is performed on the main branch, and any bifurcation point is stored such that a restart can be performed from this point; see [9] for details. As soon as such a bifurcation point is identified, a branch switch can be performed, and a new serial solve can be started from that point. As a result, a series of solutions $\{\boldsymbol{w}_i\}_{i=0}^I$ is computed for each branch. Similar to the single-branch case, a data structure and a queue can be initialised using Algorithm 4 per branch. Depending on the parallel configuration, the queues Q^b, $b = 0, \dots, n_{\text{branches}}$ of all branches can be treated separately by multiple

Algorithm 9 The `correctQueueParallel` routine, accompanied by the `workerCorrect` routine from Algorithm 11 and communication functions defined in Table 1. This routine takes the queue Q and assigns jobs from the queue to the available workers. Then, while the queue Q is non-empty, data is communicated to and from the workers, and solutions are submitted. Note that the `pop` and `submit` routines are equivalent to the ones in Algorithm 5.

Input: Q
1: Initialise a pool of worker processes $W = \{W_j,\ j = 1, \dots, N_{\text{workers}}\}$
2: **while** $Q \neq \emptyset$ and $W \neq \emptyset$ **do**
3: Q, `ID`, ΔL, $\boldsymbol{w}_i^\ell$, $\boldsymbol{w}_{i-1}^\ell$, $\boldsymbol{w}_{i+1}^\ell \leftarrow$ `pop`(Q) ▷ See line 2 of Algorithm 5
4: `sendStop(false)`
5: `sentMetaData(ID,` ℓ`)`
6: `sendJob(ID,` $\Delta L_0, \boldsymbol{w}_i^\ell, \boldsymbol{w}_{i-1}^\ell, \boldsymbol{w}_{i+1}^\ell, W_j$`)`
7: Remove Q_i from Q and W_j from W
8: **end while**
9: ▷ Send jobs to workers when they are available
10: **while** $|W| \neq N_{\text{workers}}$ **do**
11: W_j, `ID`, $\{d_j^{\ell+1}\}_{j=0,\dots,N-1}$, $\{\boldsymbol{w}_j^{\ell+1}\}_{j=0,\dots,N}$, $\overline{\Delta L}, \delta L \leftarrow$ `receiveDataWorker2Manager`
12: $Q \leftarrow$ `submit(ID,` $\{d_j^{\ell+1}\}_{j=0,\dots,N-1}, \{\boldsymbol{w}_j^{\ell+1}\}_{j=0,\dots,N}, \overline{\Delta L}, \delta L, Q$`)` ▷ See line 4 of Algorithm 5
13: Add W_j to W
14: **while** $Q \neq \emptyset$ and $W \neq \emptyset$ **do**
15: Q, `ID`, ΔL, $\boldsymbol{w}_i^\ell$, $\boldsymbol{w}_{i-1}^\ell$, $\boldsymbol{w}_{i+1}^\ell \leftarrow$ `pop`(Q)
16: `sendStop(false)` ▷ The worker always expects a stop signal, now it is false.
17: `sentMetaData(ID,` ℓ`)`
18: `sendJob(ID,` $\Delta L_0, \boldsymbol{w}_i^\ell, \boldsymbol{w}_{i-1}^\ell, \boldsymbol{w}_{i+1}^\ell, W_j$`)`
19: Remove Q_i from Q and W_j from W
20: **end while**
21: **end while**
22: `sendStopManager2All(true)`
Output: $\{\boldsymbol{w}_i\}$, $\{s_i\}$

Algorithm 10 Solve routine for a worker (`workerInitiate`). This routine performs the initiation steps (see Algorithm 3) on jobs received from the manager, until a stop signal is received. More information on the communication functions can be found in Table 1.

Input:
1: $(\text{ID}, \Delta L_0, \boldsymbol{w}_i, \boldsymbol{w}_{i-1}, \boldsymbol{w}_{\text{ref}}) \leftarrow$ `receiveJob`
2: $\boldsymbol{w}_{k+1}^\ell, \Delta s_{k+1}^\ell \leftarrow$ `initiate` $(\boldsymbol{w}_k^\ell, \Delta \boldsymbol{w}_{k-1}^\ell, \Delta L)$
3: `sendDataWorker2Manager(ID,` $\{d_j\}_{j=0,\dots,N-1}^{\ell+1}, \{\boldsymbol{w}_j\}_{j=0,\dots,N}^{\ell+1}, \overline{\Delta L}$`)`
Output:

Algorithm 11 Solve routine for a worker (`workerCorrect`). This routine performs correction steps (see Algorithm 6) on jobs received from the manager, until a stop signal is received. More information on the communication functions can be found in Table 1.

Input:

1: $(\texttt{ID}, \Delta L_0, \boldsymbol{w}_i, \boldsymbol{w}_{i-1}, \boldsymbol{w}_{\text{ref}}) \leftarrow$ `receiveJob`

2: $\{d_k^{\ell+1}\}_{k=0,\dots,N-1}$, $\{\boldsymbol{w}_k^{\ell+1}\}_{k=0,\dots,N}$, $\overline{\Delta L}$, $\delta L \leftarrow$ `correct`$(N, \Delta L_0, \boldsymbol{w}_i^{\ell}, \boldsymbol{w}_{i-1}^{\ell}, \boldsymbol{w}_{i+1}^{\ell})$

3: `sendDataWorker2Manager`$(\texttt{ID}, \{d_j\}_{j=0,\dots,N-1}^{\ell+1}, \{\boldsymbol{w}_j\}_{j=0,\dots,N}^{\ell+1}, \overline{\Delta L})$

Output:

manager processes, or they can be combined into one large queue Q and handled by one single manager process. In the latter case, each job will also contain a branch identifier to refer to the corresponding data structure.

The advantages of the above approach combining multi-branch and within-branch parallelization using the APALM are that the extension from a single-branch APALM to a multi-branch APALM is straightforward. The disadvantage, however, is that the identification of bifurcations is only taken into account in the serial solve step; hence, any bifurcations that are identified in the parallel solve will not be taken into account. A remedy would be to rebuild the map and the data structure on the manager process as soon as a worker process finds a bifurcation point; this requires all active workers to terminate.

5. Numerical experiments

In this section, the APALM scheme is demonstrated on a series of benchmark problems. The first two benchmark problems are structural analysis problems with limit-point instabilities and complex collapsing mechanisms involving strongly curved solution paths. The third benchmark is a buckling problem containing a bifurcation with multiple branches to illustrate the concept of the APALM in a multi-branch setting. All benchmark problems are computed using isogeometric Kirchhoff-Love shell elements based on the works [24–26]. Furthermore, a scaling test with respect to the number of worker processes is performed for all benchmark problems. Here, a scaling analysis of the ASPALM is performed to demonstrate the relative computational costs of the serial initialization phase compared to the parallel correction phase. Furthermore, a scaling analysis of the APALM is performed to show the advantage of the fully parallel APALM scheme over the two-stage ASPALM scheme. For the scaling tests, the communications from Table 1 are performed using the Message Passing Interface (MPI), and they are performed on the Delft High Performance Computing Centre (DHPC) [27] with Intel XEON E5-6248R 24C 3.0 GHz nodes with 96 GB of memory per CPU. The code will be made available within the Geometry + Simulation modules [28] upon publication.

5.1. Collapse of a shallow roof

The first benchmark problem involves a shallow roof subject to a point load at the midpoint. The roof is discretized with 4×4 NURBS elements of degree 3. The roof is composed of a lay-up of composites with material properties as presented in Fig. 6, inspired by [29]. It is modelled using isogeometric Kirchhoff-Love shell elements [24] supporting composite laminates [30]. A Crisfield ALM is used with an initial arc length of 30 and a scaling parameter of $\Psi = 1$. The tolerance of the APALM is set to $\varepsilon_l = \varepsilon_u = 10^{-2}$. This tolerance implies that intervals are marked for refinement when the traversed length is deviates than 1% of the original interval length. A smaller tolerance would imply that more elements are marked for refinement and that the corrections are performed up to lower levels. The material, and load parameters for this benchmark can be found in Fig. 6. Reference solutions are obtained using a serial arc-length method with a sufficiently fine increment size.

Fig. 7 provides the results of the APALM applied to the collapse of the shallow roof. As can be seen from this figure, the serial computation provides a coarse estimate of the reference curve. Especially on the first limit point (between $\lambda \times w_A \in [-15, -8] \times [20, 25]$), the data is sparse, similar to the collapse itself (see inset in Fig. 7). The results of the APALM show that a lot of refinements are needed to represent the collapsing behaviour correctly in the region of the inset in Fig. 7. These regions do not necessarily involve extremely curved paths in the axes of Fig. 7, but the solutions $\boldsymbol{w}$ are most likely curved in the higher-dimensional solution space.

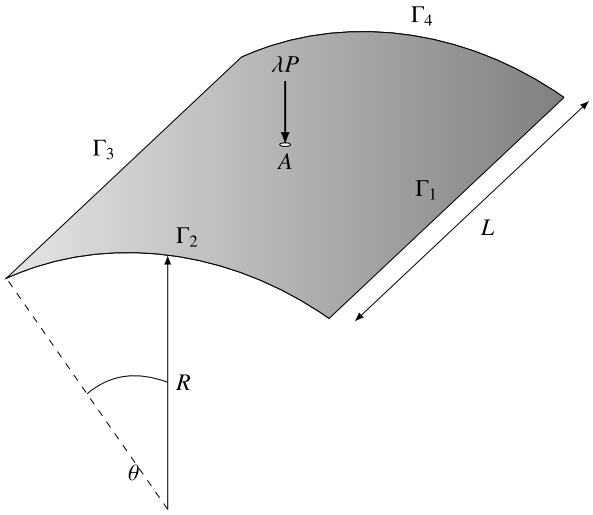


Fig. 6. The problem definition for the benchmark of the collapsing roof with length $L = 508$ [mm], with radius $R = 2540$ [mm], an angle $\theta = 10$ [rad] with a thickness of $t = 6.35$ [mm]. The boundaries Γ_1 and Γ_3 have fixed displacements, and the other sides are free. The material is modelled using a Saint-Venant Kirchhoff laminate with $E_{11} = 3300$ [N/mm^2], $E_{22} = E_{33} = 1100$ [N/mm^2], $G_{12} = G_{13} = 660$ [N/mm^2], $E_{23} = 440$ [N/mm^2] and $\nu_{12} = \nu_{13} = \nu_{23} = 0.25$ [−] and with lay-up angles $[0/90/0]°$. The load is variable with a magnitude $P = 10$ [N] and magnification factor λ.

In order to assess the parallelization of the APALM in this example, the test from Fig. 7 is performed with an increasing number of workers using the ASPALM and the APALM schemes. The results in Table 2a show that for the computation with arc-length parameter $\Delta L = 30$, the parallel correction step of the ASPALM scales optimally (i.e., with a factor 2) up to around 8 workers, after which the scalability decreases and the parallel correction phase takes around 15% of the total computational time. Using the APALM scheme, the total computational time is decreased compared to the ASPALM scheme, and parallel corrections can be started as soon as the first interval has been initialized. The computational time of the APALM stagnates around 4 workers, at a computational time similar to the serial initialization time for the ASPALM method, showing that adaptive parallel corrections can be performed without significantly more computational costs compared to a serial arc-length method. When the number of intervals is increased, e.g., by decreasing the arc-length parameter to $\Delta L = 2.5$ (Table 2b), it can be seen that the parallel stage of the ASPALM scales up to a higher number of workers, in this case 64, up to the point that it takes around 5% of the total computational time for 256 workers. The APALM again provides computational times similar to a serial ALM without corrections. The improved scalability is explained by the fact that the queue is in general longer; therefore, the time that workers are idle waiting for the last job to be finished is smaller relative to the total computational time.

5.2. Collapse of a truncated cone

The second benchmark example is based on [26] and involves the collapsing behaviour of a truncated cone with a hyperelastic Mooney-Rivlin material model. This benchmark is based on [31], but in the

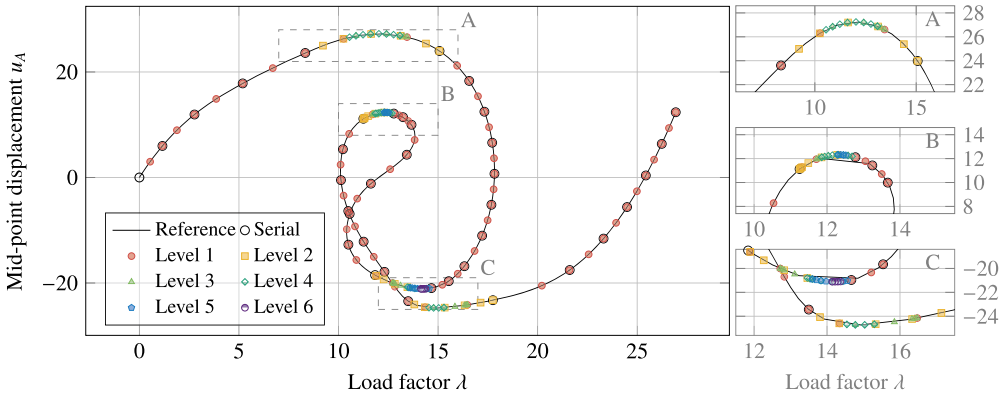


Fig. 7. Results of the collapsing roof. The figure on the left indicates the full solution path, and the figures on the right depict the insets indicated in the left figure. The reference and serial solutions are represented by the solid line and the black markers, respectively. The solutions computed by the APALM are indicated per level. The simulation is performed with a tolerance of $\epsilon_l = \epsilon_u = 10^{-2}$ and an increment length of $\Delta L = 30$.

Table 2
Computational time in [s] for the benchmark of the collapsing roof for the ASPALM and APALM for different numbers of worker processes. The times for the ASPALM are presented for the serial initialization and the parallel correction phases, and the sum of the two is given as the total computational time. The numbers in the *Serial* column should theoretically be the same, but they provide a representation of the variation in the time measurements. The results are presented for simulations with increment lengths $\Delta L = 30$ (a) and $\Delta L = 2.5$ (b), and the italic row with 0 workers denotes the ASALM method.

(a) $\Delta L = 30$

Workers	ASPALM			APALM
#	Serial	+ Parallel	= Total	Parallel
0	115.7	195.3	311.1	287.1
1	119.2	209.0	328.2	318.8
2	114.0	100.8	214.8	162.4
4	109.5	46.1	155.6	115.8
8	115.0	27.0	142.1	115.9
16	115.1	17.8	132.9	116.3
32	114.9	15.9	130.8	113.0
64	114.5	13.3	127.8	116.0

(b) $\Delta L = 2.5$

Workers	ASPALM			APALM
#	Serial	+ Parallel	= Total	Parallel
0	507.2	1,778.1	2,285.3	2,187.1
1	500.5	1,757.7	2,258.2	2,310.2
2	447.5	835.3	1,282.9	1,114.0
4	493.4	449.4	942.8	558.1
8	496.8	223.2	720.0	453.9
16	503.3	113.0	616.2	483.6
32	493.2	58.1	551.3	510.9
64	504.2	29.2	533.4	498.3
128	501.0	20.2	521.3	494.7
256	505.5	18.8	524.3	509.6

work of [26], the full collapsing behaviour was revealed using arc-length methods. The geometry, material, and load specifications can be found in Fig. 8.

The truncated cone is modelled using a quarter geometry using symmetry conditions to represent the axisymmetry as used in the original case of [31]. The geometry is modelled with 32 NURBS elements of degree 2 over the height. Further, an initial arc length of 0.5 is used, and the scaling factor is $\Psi = 0$. The top boundary Γ_2 is free, and on the bottom boundary Γ_4, all displacements are fixed. The other boundaries have symmetric boundary conditions. The governing material model is an incompressible Mooney-Rivlin material model with a strain energy density function (with a slight abuse of notation)

$$\Psi(\mathbf{C}) = \frac{c_1}{2}\left(I_1 - 3\right) + \frac{c_2}{2}\left(I_2 - 3\right), \tag{18}$$

with I_1 and I_2 the first and second invariants of the deformation tensor $\mathbf{C} = \boldsymbol{F}^{\top}\boldsymbol{F}$. More information on the problem set-up and the material models can be found in [26]. The reference results are obtained from a serial ALM computation with a sufficiently small increment size.

The results of the collapsing truncated cone problem are presented in Fig. 9. As seen in this picture, the serial initialization provides a coarse approximation of the path but leaves out details, e.g., the rotated "S"-shaped curve in the inset in Fig. 9. From the results, it is clear that the APALM focuses its refinement on the curved parts of the path and reveals the "S"-shaped curve among other features of the path.

Similar to the collapse of the roof, a scaling analysis of the parallel evaluations is performed. The results in Table 3a verify that, as for

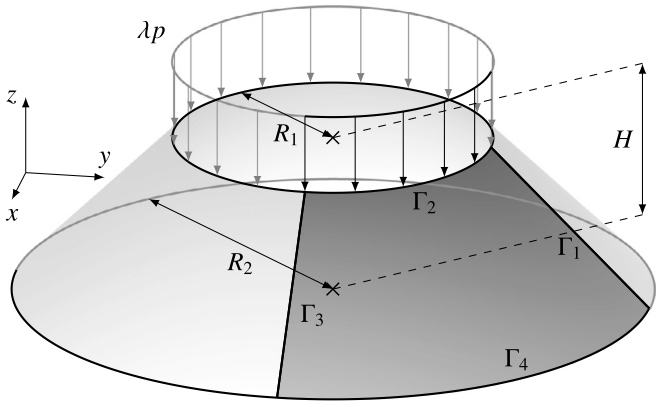


Fig. 8. The problem definition for the benchmark of the collapsing truncated cone with inner radii $R_1 = 1$ [m] and $R_2 = 2$ [m] and height $H = 1$ [m]. The thickness of the cone is $t = 0.1$ [m]. The cone is modelled by using a quarter of the geometry, using symmetry conditions on Γ_1 and Γ_3. The displacements at the bottom boundary (Γ_4) are fixed, and on the top boundary, a variable line load is applied and is variable with magnitude $p = 1$ [N/mm] and magnification factor λ. The material of the cone is modelled using an incompressible Mooney-Rivlin model with parameters $\mu = c_1 + c_2 = 4.225$ [N/mm^2], $c_1/c_2 = 7$.

the benchmark example with the collapsing roof, the scalability of the parallel correction phase scales optimally up to 8 workers, where the parallel correction phase takes around 15% of the total computational

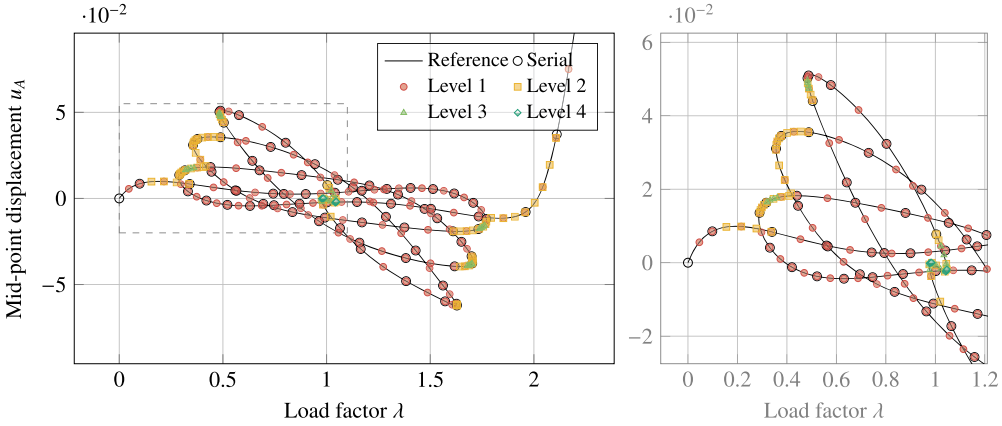


Fig. 9. Results of the collapsing truncated cone. The figure on the left depicts the full solution path, and the figure on the right depicts the inset indicated in the left figure. The reference and serial solutions are represented by the solid line and the black markers, respectively. The solutions computed by the APALM are indicated per level. The simulation is performed with a tolerance of $\varepsilon_l = \varepsilon_u = 10^{-2}$ and an increment length of $\Delta L = 0.5$.

Table 3
Computational time in [s] for the benchmark of the collapsing truncated cone for the ASPALM and APALM for different numbers of worker processes. The times for the ASPALM are presented for the serial initialization and the parallel correction phases, and the sum of the two is given as the total computational time. The numbers in the *Serial* column should theoretically be the same, but they provide a representation of the variation in the time measurements. The results are presented for simulations with increment lengths $\Delta L = 0.5$ (a) and $\Delta L = 0.0625$ (b), and the italic row with 0 workers denotes the ASALM method.

(a) $\Delta L = 0.5$

Workers	ASPALM			APALM
#	Serial	+ Parallel	= Total	Parallel
0	160.2	244.0	404.2	436.6
1	162.5	247.2	409.7	424.8
2	169.5	130.1	299.6	207.1
4	170.6	68.1	238.7	172.9
8	162.6	43.0	205.6	160.5
16	175.3	32.0	207.3	173.3
32	175.5	27.3	202.8	170.8
64	170.1	23.3	193.4	169.5

(b) $\Delta L = 0.0625$

Workers	ASPALM			APALM
#	Serial	+ Parallel	= Total	Parallel
0	499.7	2,575.9	3,075.6	3,055.3
1	467.5	2,232.5	2,700.0	2,783.8
2	496.3	1,337.0	1,833.2	1,573.4
4	467.8	654.4	1,122.2	789.5
8	490.1	322.6	812.7	489.4
16	467.6	167.6	635.1	496.0
32	494.1	97.1	591.1	483.9
64	491.4	55.7	547.1	493.6
128	485.0	41.5	526.5	494.5
256	493.8	32.9	526.7	491.4
512	491.8	25.8	517.6	488.5

time when using 64 workers. When using the APALM scheme, the collapsing cone also shows that the computation times of the APALM are similar to the times needed for serial initialization, in other words, a classical ALM without adaptive corrections. When the number of intervals is increased, i.e., when the arc-length parameter is decreased to $\Delta L = 0.0625$ (Table 3b), the scalability of the parallel phase of the ASPALM and of the full APALM reaches further, up to 64 workers.

5.3. Strip buckling

The third example involves a benchmark problem consisting of a bifurcation instability. The problem consists of a flat strip that is clamped on one edge and free on all the others, with an in-plane compressive load applied on the free end opposite to the clamped edge; see Fig. 10 for the problem set-up and [32] for the reference results. The ALM that is used is a Crisfield method with $\Psi = 0$, with a pre-buckling arc-length of $5 \cdot 10^{-5}$, a post-buckling arc-length of 5, and a tolerance of the APALM of $\varepsilon_l = \varepsilon_u = 10^{-3}$. The serial ALM is equipped with an extension for the computation of singular points (Wriggers 1988); see [33] for more details on this implementation. Using these methods, an initially flat strip is compressed until the bifurcation point has been computed. As soon as the strip becomes unstable, the bifurcation point is computed, and a branch switch is performed, marking the transition between the pre-buckling and post-buckling branches.

The results for the buckled strip are presented in Fig. 11. In this figure, the non-dimensional horizontal and vertical displacements of the end point are plotted with respect to the non-dimensional applied load. In the plots, the pre- and post-buckling branches are plotted separately for clarity, but the branches should obviously be connected at the bifurcation point. As can be seen from the results, a rather coarse serial approximation of the post-buckling branch gives a good starting point for a multi-level approximation of the curve, providing additional detail in the sharp corner in $W/L \in [0.7, 0.8]$. In addition, it can be seen that the pre-buckling branch requires no more levels than the first, as the behaviour there is just a linear axial compression, hence the solution path is straight.

As for the previous two benchmark examples, a scaling analysis of the parallel evaluations is performed. The main difference between the previous two examples is that the present example involves a bifurcation point. However, since the job queue includes the jobs from all branches together, there is no idle time to wait for a branch to finish before starting a new branch; hence, it is expected that the parallel scaling for a bifurcation problem should have the same scaling properties. Indeed, Table 4a shows that optimal scaling is achieved in the parallel correction phase of the ASPALM up to 8 nodes, after which the idle time to wait for the last job to finish significantly impacts the scaling, as observed in the other benchmarks. In addition, it is found that the APALM reaches efficient computation of the full adaptive load-

Table 4
Computational time in [s] for benchmark of the buckled strip using the ASPALM and APALM with different numbers of worker processes. The times for the ASPALM are presented for the serial initialization and the parallel correction phases, and the sum of the two is given as the total computational time. The numbers in the *Serial* column should theoretically be the same, but they provide a representation of the variation in the time measurements. The results are presented for simulations with increment lengths $\Delta L = 2.5$ (a) and $\Delta L = 0.025$ (b), and the italic row with 0 workers denotes the ASALM method.

(a) $\Delta L = 2.5$

Workers	ASPALM			APALM
#	Serial	+ Parallel	= Total	Parallel
0	50.7	171.8	222.0	244.0
1	58.7	198.1	257.0	244.0
2	58.5	102.6	161.0	112.0
4	59.0	51.1	110.0	77.0
8	59.0	27.9	87.0	67.0
16	59.3	24.9	84.0	68.0
32	60.3	23.0	83.0	68.0
64	58.6	24.4	83.0	66.0

(b) $\Delta L = 0.025$

Workers	ASPALM			APALM
#	Serial	+ Parallel	= Total	Parallel
0	963.3	1,963.3	2,926.6	3,017.6
1	1,022.3	2,065.7	3,088.0	3,020.6
2	1,025.0	1,053.7	2,078.7	1,509.7
4	943.9	464.0	1,407.9	1,034.1
8	1,019.5	256.0	1,275.5	1,028.2
16	1,006.2	129.3	1,135.5	1,027.7
32	1,026.6	68.8	1,095.4	1,028.3
64	1,032.7	33.6	1,066.4	1,028.0
128	935.1	18.9	954.0	1,023.5
256	1,012.6	12.0	1,024.6	1,026.8

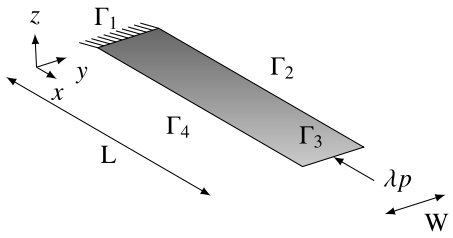


Fig. 10. The problem definition for the benchmark of the buckling of a strip with length $L = 1$ [m], width $W = 0.01$ [m] and thickness $t = 0.01$ [m] subject to a horizontal load with magnitude $p = 0.1$ [N] with magnification factor λ. The strip has fixed displacements and rotations at Γ_1 and fixed displacements in y-direction on Γ_2 and Γ_4. The material is modelled using a Saint-Venant Kirchhoff material model with Young's modulus $E = 75 \cdot 10^6$ [N/mm^2] and Poisson ratio $\nu = 0$ [−].

displacement curve within the time of a serial ALM computation, using 8 workers. When increasing the number of curve segments by decreasing the arc-length parameter to $\Delta L = 0.025$, it can again be seen that the parallel scalability increases. Optimal scaling of the parallel correction of the ASPALM is achieved up to 256 workers, with the correction phase taking only 1% of the total computational time. The APALM provides an adaptively refined solution curve in the time of a serial ALM computation, using only 4 to 8 workers.

5.4. Snapping meta-material

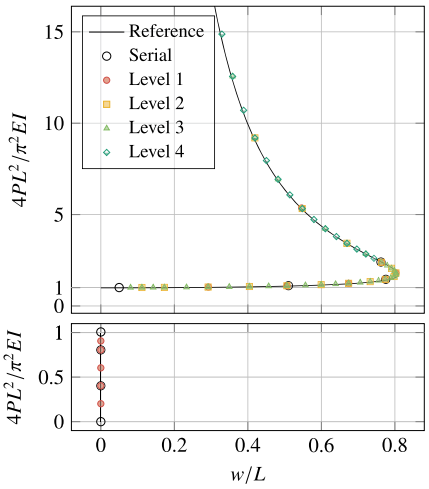
As a final example, the APALM is applied to a problem of larger scale. In particular, the snapping behaviour of a snapping meta-material is modelled, inspired by [15]. The meta-material consists of $N_x \times N_Y = 3 \times 2.5$ building blocks (see Fig. 12a) with a snapping and a bearing segment (see Fig. 12b), and the material is modelled with a compressible Neo-Hookean material model. The full problem details are provided in Fig. 12. The snapping behaviour of the meta-material is investigated by using arc-length methods on the varying load λP, with a step size of $\Delta L = 5 \cdot 10^{-2}$, until 1.5% strain. The simulation is modelled using 2D elasticity equations using the plane-stress assumption, which are discretised using B-splines with mixed degrees 2 and 3 and maximal regularity. The mesh is uniformly distributed, and the full system of equations has 16563 degrees of freedom. The simulations are performed using shared-memory parallelization of the assembly routines and distributed memory parallelization of the ASPALM and APALM. For reference, a displacement-controlled (DC) simulation is performed.

Fig. 13 depicts the stress-strain curve for the snapping meta-material depicted in Fig. 12. Firstly, it can be seen that the curve computed using a serial ALM coincides with the curves obtained from DC simulations, but the ALM shows additional snap-through behaviour on the points where the DC curve has kinks. These kinks coincide with the instability in the metamaterial. Furthermore, the figure shows that the initial course approximation at level 0 is refined up to level 4 in the adaptive arc-length method scheme proposed in this paper, provided the tolerance of $\varepsilon_l = \varepsilon_u = 10^{-3}$. The refinements of the adaptive scheme are mainly present in the highly curved segments of the load displacement curve. The reader is referred to Video 1 from the supplementary material for the deformations corresponding to the stress-strain curve in Fig. 13.

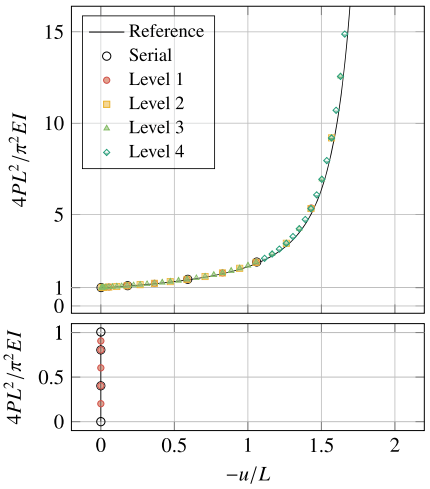
As for the other numerical experiments, the parallelization properties of the ASPALM and APALM schemes are investigated. For the snapping meta-material simulation, the computational times are presented in Table 5. The results in the table show high computational times for the ASALM (i.e., the ASPALM and APALM with 0 workers) compared to the DC simulation. However, the scalability observed in the previous benchmark problems can also be observed in the simulation of the snapping metamaterial. In fact, the APALM with 8 workers requires a factor of 4 less computational time, again equivalent to the computational time required for only the serial initialization phase of the ASPALM. Lastly, the case of the snapping meta-material shows that, compared to a naturally serial displacement-controlled method, the APALM achieves a speed-up of a factor of 2.5 while providing snapping behaviour with greater accuracy.

6. Conclusions and outlook

In this paper, an Adaptive Parallel Arc Length Method (APALM) is presented. Contrary to existing parallel implementations of the Arc-Length Method (ALM), the present method provides within-branch parallelization, hence providing scalable parallelization independent of the physics of the problem, i.e., the number of branches. The method employs a multi-level approach, where parallel corrections are performed on solution intervals that have been initialised before. Given the sub-intervals provided by the serial computation, computations with finer arc lengths can be performed and evaluated using suitable error measures, marking intervals for further refinement when needed. Employing the multi-level approach, their implementations are discussed: the Adaptive Serial ALM (ASALM), the Adaptive Serial-Parallel ALM (ASPALM), and the APALM. The ASALM is a serial implementation, employing only the inherent adaptivity of the concept provided in this paper. The ASPALM is a two-stage approach, separating a serial initialization of the full equilibrium path from the parallel corrections. The APALM is a fully parallel implementation, where parallel corrections

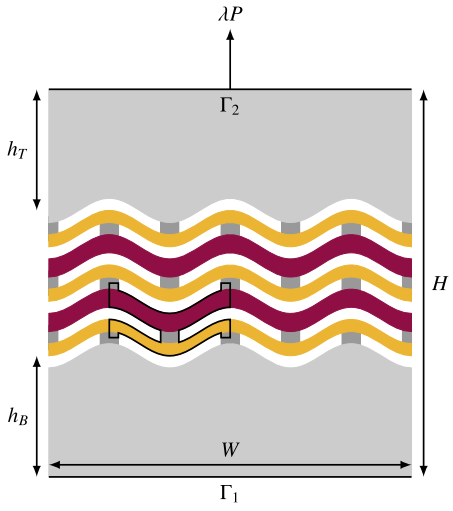


(a) Non-dimensional out-of-plane displacement of the beam with respect to the non-dimesional buckling load.

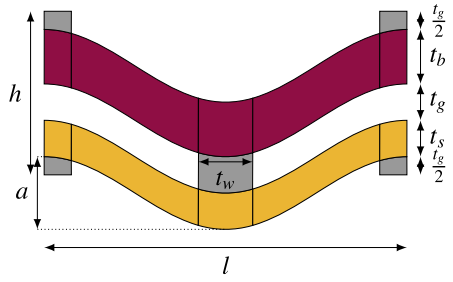


(b) Non-dimensional in-of-plane displacement (length direction) of the beam with respect to the non-dimensional buckling load.

Fig. 11. Results of the buckling of a clamped strip. The left figure provides the out-of-plane displacement of the free end with respect to the non-dimensional load $4PL^2/\pi^2EI$, and the right figure represents the horizontal displacement of the free end with respect to the same non-dimensional load. In both figures, buckling occurs when $4PL^2/\pi^2EI = 1$ and the axes are split on this point to make the pre- and post-buckling branches both visible. The simulation is performed with a tolerance of $\varepsilon_l = \varepsilon_u = 10^{-3}$ and a step length of $\Delta L = 5 \cdot 10^{-5}$ (pre-buckling) and $\Delta L = 5$ (post-buckling).



(a) A snapping meta-material with 3×2.5 building blocks, of which one is outlined. The total multi-patch consists of 132 patches.



(b) The snapping building block, composed of 15 patches outlines in black.

Fig. 12. The problem definition for the snapping meta material using a grid of 3×2.5 elements (a) with the element geometry as defined in (b). The element dimensions are defined using the thickness of the load-bearing part $t_b = 1.5$ [mm] and the thickness of the snapping part $t_s = 1.0$ [mm], the thickness of the gap $t_g = 1.0$ [mm] and the thickness of the connectors $t_w = 1.5$ [mm], such that the height $h = t_b + t_s + 2t_g$. The length of the element is $\ell = 10$ [mm], and the amplitude of the cosine wave defining the element shape is given by $a = 0.3l$. Since the meta-material has 3×2.5 elements, the total width is $W = 3\ell$. The height of the total metamaterial is given by $H = 3h + 2t_g + t_s + h_B + h_T$, where $h_B = h_T = 5t_g$ are the buffer zones on the top and the bottom. The thickness of the specimen (in out-of-plane direction) is $b = 3$ [mm]. The material is defined using a compressible Neo-Hookean material model with Young's modulus $E = 78$ [N/mm^2] and Poisson ratio $\nu = 0.4$ [−]. The bottom boundary Γ_1 is fixed using $u_x = u_y = 0$, and the top boundary Γ_2 is fixed in the horizontal direction ($u_x = 0$) and coupled in the vertical direction u_y. The load applied on the top boundary is a variable defined by λP.

are performed as soon as the first path segments have been initialized. Conceptually, the APALM has a higher degree of parallelization since the workers are not idle until the full solution curve is obtained. Given a basic step function and distance computation, the present paper provides all algorithms necessary for implementing the APALM with manager-worker parallelization.

The implementation of the APALM is evaluated using three benchmark problems and an application example. The first problem involves the collapse of a composite shallow cylindrical shell. The second problem involves the collapse of a truncated conical rubber shell, and the third example involves the bifurcation problem of a strip subject to an in-plane load. Moreover, the method is applied to the modelling of a snapping metamaterial to investigate its performance on a larger-scale problem. In all examples, it can be observed that the APALM provides an accurate description of the reference solution, given a (sufficiently) coarse serial initialization of the curve. Through refinement, the APALM provides refinements (hence details in the solution), typically on sharp corners in the load-displacement diagrams. In addition, the bifurcation example also shows that the APALM is able to work within an exploration scheme for bifurcations. In all benchmark problems, the ASPALM and APALM have been used to evaluate the parallelization of the schemes. The natural separation of the serial and parallel stages

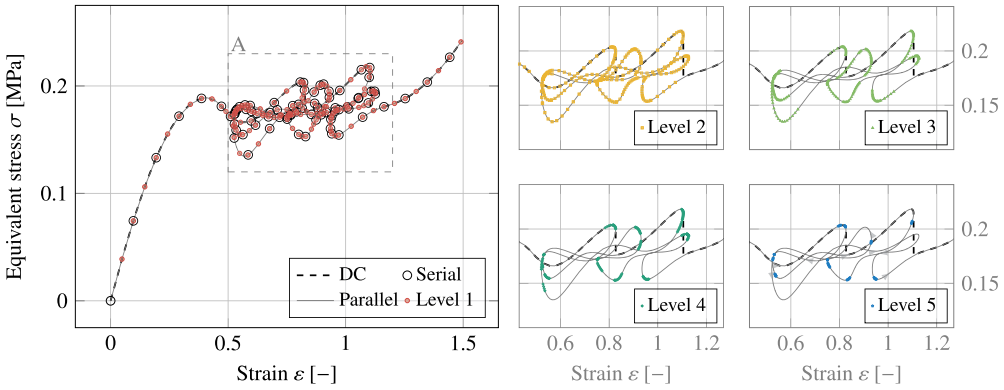


Fig. 13. Stress-strain diagram for the snapping meta-material from Fig. 12. The vertical axis depicts the equivalent stress $\sigma = \lambda P/(bW)$, and the horizontal axis represents the strain $\varepsilon = u_y/H$, where u_y is the displacement of the top boundary Γ_2. The complete curve with the displacement-controlled (DC) results, the points obtained in serial initialization, and the line obtained by parallel corrections are presented on the left. The figures on the right present the points from different hierarchical levels at the inset depicted in the left diagram. The simulation is performed with a tolerance of $\varepsilon_l = \varepsilon_u = 10^{-3}$ and an increment length of $\Delta L = 0.05$.

Table 5
Computational time in [s] for the example of the snapping meta-material for the ASPALM and APALM for different numbers of worker processes. The computational time for a displacement-controlled (DC) simulation with step $\Delta u_y = 0.0005\ [mm]$ is provided as a reference. The times for the ASPALM are presented for the serial initialization and the parallel correction phases, and the sum of the two is given as the total computational time. The numbers in the *Serial* column should theoretically be the same, but they provide a representation of the variation in the time measurements. The italic row with 0 workers denotes the ASALM method.

Workers	ASPALM			APALM	DC
#	Serial	+ Parallel	= Total	Parallel	Serial
0	1,571.6	5,204.8	6,776.4	7,022.9	4,400.8
1	1,686.9	4,593.2	6,280.1	5,319.1	
2	1,237.5	3,005.9	4,243.4	3,827.9	
4	1,742.7	1,548.2	3,290.9	2,137.3	
8	1,445.4	717.4	2,162.8	1,711.8	
16	1,931.1	352.2	2,283.3	1,632.9	
32	1,746.9	219.7	1,966.6	1,755.6	

of the ASPALM reveals the scalability of the parallel correction with respect to the number of workers, showing that the parallel correction can take only a fraction of the total computational time for a larger number of workers. Furthermore, the comparison between the ASPALM and the APALM shows that the full parallelization of the APALM provides a more efficient scheme than the two-stage approach of the ASPALM, as expected. The benchmarks and example also show that the APALM provides a full solution curve – including adaptive refinements – in the same computational time needed to compute only the initialisation of the ASPALM. This reveals the potential of the APALM: it can provide detailed solution paths without significantly increasing the computational time. The coarser the initial step size, the more arc-length intervals are computed during the parallel corrections of the method until a sufficient; hence, the higher the computational merit of the method to reach a desired level of detail. Moreover, the scaling analyses also show that the benefits of the APALM are already achieved with a small number of workers, e.g., 8 workers, making the APALM interesting on a desktop scale. For larger clusters, the APALM can be employed using dynamic load balancing within OpenMP.

As the APALM enables parallelization in the arc-length domain, future applications of this method include quasi-static computations for solid and fluid dynamics, among other problems, especially those with a large number of load steps. Therefore, future works with this method include automatic exploration of solution spaces, e.g., following the work of [8,9], or applications with large numbers of degrees of freedom, for instance with phase-field models for fracture mechanics [34]. Other future work includes combining the APALM with a spatial refinement scheme to enable space-quasi-time refinements. MPI scalability and distribution of cores per worker are topics to investigate for different applications. Another topic for further investigation is the convergence of the underlying arc-length method for large steps. Since a fewer number of initial intervals reduces the serial initialization time of the APALM, the parallel performance can be increased significantly when the initial step size is maximized. For example, the Mixed Integration Point (MIP) method increases the convergence of the ALM, allowing for larger step sizes. The performance of the MIP is demonstrated for isogeometric Kirchhoff–Love shells in [35–38]. Lastly, since the presented APALM is developed for path-independent problems, an extension to path-dependent problems is a natural direction for future research.

CRediT authorship contribution statement

H.M. Verhelst: Conceptualization, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **J.H. Den Besten:** Funding acquisition, Project administration, Supervision, Writing – original draft, Writing – review & editing. **M. Möller:** Funding acquisition, Project administration, Supervision, Writing – original draft, Writing – review & editing, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

The authors are grateful for the financial support from Delft University of Technology.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.compstruc.2024.107300.

References

[1] Rupp K. Microprocessor trend data. https://github.com/karlrupp/microprocessor-trend-data, 2022.
[2] Gander MJ. 50 years of time parallel time integration; 2015. p. 69–113.
[3] Riks E. The application of Newton's method to the problem of elastic stability. J Appl Mech 1972;39:1060.
[4] Crisfield MM. A fast incremental/iterative solution procedure that handles "snap-through". In: Computational methods in nonlinear structural and solid mechanics. Pergamon; 1981. p. 55–62.
[5] Wriggers P, Wagner W, Miehe C. A quadratically convergent procedure for the calculation of stability points in finite element analysis. Comput Methods Appl Mech Eng 1988;70:329–47.
[6] Pretti G, Coombs WM, Augarde CE. A displacement-controlled arc-length solution scheme. Comput Struct 2022;258:106674.
[7] Kadapa C. A simple extrapolated predictor for overcoming the starting and tracking issues in the arc-length method for nonlinear structural mechanics. Eng Struct 2021;234:111755.
[8] Thies J, Wouters M, Hennig RS, Vanroose W. Towards scalable automatic exploration of bifurcation diagrams for large-scale applications. Lect Notes Comput Sci Eng 2021;139:981–9.
[9] Wouters M, Vanroose W. Automatic exploration techniques of numerical bifurcation diagrams illustrated by the Ginzburg–Landau equation. SIAM J Appl Dyn Syst 2019;18:2047–98. https://doi.org/10.1137/19M1248467.
[10] Lions JL, Maday Y, Turinici G. Résolution d'EDP par un schéma en temps « pararéel ». C R Acad Sci, Sér 1 Math 2001;332:661–8.
[11] Falgout RD, Friedhoff S, Kolev TV, MacLachlan SP, Schroder JB. Parallel time integration with multigrid. SIAM J Sci Comput 2014;36:C635–61.
[12] Cyr EC, Günther S, Schroder JB. Multilevel initialization for layer-parallel deep neural network training. ArXiv preprint arXiv:1912.08974, 2019.
[13] Hessenthaler A, Falgout RD, Schroder JB, de Vecchi A, Nordsletten D, Röhrle O. Time-periodic steady-state solution of fluid-structure interaction and cardiac flow problems through multigrid-reduction-in-time. ArXiv preprint arXiv:2105.00305, 2021.
[14] Aruliah DA, Van Veen L, Dubitski A. Algorithm 956: PAMPAC, a parallel adaptive method for pseudo-arclength continuation. ACM Trans Math Softw 2016;42.
[15] Rafsanjani A, Akbarzadeh A, Pasini D. Snapping mechanical metamaterials under tension. Adv Mater 2015;27:5931–5.
[16] Ritto-Corrêa M, Camotim D. On the arc-length and other quadratic control methods: established, less known and new implementation procedures. Comput Struct 2008;86:1353–68.
[17] Ragon SA, Gürdal Z, Watson LT. A comparison of three algorithms for tracing nonlinear equilibrium paths of structural systems. Int J Solids Struct 2002;39:689–98.
[18] Schweizerhof K, Wriggers P. Consistent linearization for path following methods in nonlinear fe analysis. Comput Methods Appl Mech Eng 1986;59:261–79.
[19] Bellini P, Chulya A. An improved automatic incremental algorithm for the efficient solution of nonlinear finite element equations. Comput Struct 1987;26:99–110.
[20] Carrera E. A study on arc-length-type methods and their operation failures illustrated by a simple model. Comput Struct 1994;50:217–29.
[21] Lam WF, Morley CT. Arc-length method for passing limit points in structural calculation. J Struct Eng 1992;118:169–85.
[22] Zhou Z, Murray D. An incremental solution technique for unstable equilibrium paths of shell structures. Comput Struct 1995;55:749–59.
[23] Piegl L, Tiller W. The NURBS book, monographs in visual communications. Berlin, Heidelberg: Springer Berlin Heidelberg; 1995.
[24] Kiendl J, Bletzinger K-U, Linhard J, Wüchner R. Isogeometric shell analysis with Kirchhoff–Love elements. Comput Methods Appl Mech Eng 2009;198:3902–14.
[25] Kiendl J, Hsu M-C, Wu MC, Reali A. Isogeometric Kirchhoff–Love shell formulations for general hyperelastic materials. Comput Methods Appl Mech Eng 2015;291:280–303.
[26] Verhelst H, Möller M, Den Besten J, Mantzaflaris A, Kaminski M. Stretch-based hyperelastic material formulations for isogeometric Kirchhoff–Love shells with application to wrinkling. Comput Aided Des 2021;139:103075.
[27] Delft High Performance Computing Centre (DHPC). DelftBlue supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.
[28] Jüttler B, Langer U, Mantzaflaris A, Moore SE, Zulehner W. Geometry + simulation modules: implementing isogeometric analysis. PAMM 2014;14:961–2.
[29] Leonetti L, Magisano D, Madeo A, Garcea G, Kiendl J, Reali A. A simplified Kirchhoff–Love large deformation model for elastic shells and its effective isogeometric formulation. Comput Methods Appl Mech Eng 2019;354:369–96.
[30] Herrema AJ, Johnson EL, Proserpio D, Wu MC, Kiendl J, Hsu M-C. Penalty coupling of non-matching isogeometric Kirchhoff–Love shell patches with application to composite wind turbine blades. Comput Methods Appl Mech Eng 2019;346:810–40.
[31] Başar Y, Itskov M. Finite element formulation of the Ogden material model with application to rubber-like shells. Int J Numer Methods Eng 1998.
[32] Pagani A, Carrera E. Unified formulation of geometrically nonlinear refined beam theories. Mech Adv Mat Struct 2018;25:15–31.
[33] Verhelst HM, Moller M, Den Besten J, Vermolen F, Kaminski M. Equilibrium path analysis including bifurcations with an arc-length method avoiding a priori perturbations. In: Proceedings of ENUMATH2019 conference; 2020.
[34] Borden MJ, Hughes TJ, Landis CM, Verhoosel CV. A higher-order phase-field model for brittle fracture: formulation and analysis within the isogeometric analysis framework. Comput Methods Appl Mech Eng 2014;273:100–18.
[35] Magisano D, Leonetti L, Garcea G. How to improve efficiency and robustness of the Newton method in geometrically non-linear structural problem discretized via displacement-based finite elements. Comput Methods Appl Mech Eng 2017;313:986–1005.
[36] Magisano D, Leonetti L, Garcea G. Advantages of the mixed format in geometrically nonlinear analysis of beams and shells using solid finite elements. Int J Numer Methods Eng 2017;109:1237–62.
[37] Leonetti L, Kiendl J. A mixed integration point (MIP) formulation for hyperelastic Kirchhoff–Love shells for nonlinear static and dynamic analysis. Comput Methods Appl Mech Eng 2023;416:116325.
[38] Leonetti L, Magisano D, Liguori F, Garcea G. An isogeometric formulation of the Koiter's theory for buckling and initial post-buckling analysis of composite shells. Comput Methods Appl Mech Eng 2018;337:387–410.