Reducing time and memory requirements in topology optimization of transient problems

Theulings, M. J.B.; Maas, R.; Noël, L.; van Keulen, F.; Langelaar, M.

DOI
10.1002/nme.7461

Publication date
2024

Document Version
Final published version

Published in
International Journal for Numerical Methods in Engineering

Received: 7 September 2023 | Accepted: 4 February 2024

DOI: 10.1002/nme.7461

RESEARCH ARTICLE

WILEY

Reducing time and memory requirements in topology optimization of transient problems

M. J. B. Theulings[1,2] | **R. Maas**[2] | **L. Noël**[1] | **F. van Keulen**[1] | **M. Langelaar**[1]

[1]Computational Design and Mechanics, Delft University of Technology, Delft, The Netherlands

[2]Aerospace Vehicle Collaborative Engineering, Royal Netherlands Aerospace Centre, Amsterdam, The Netherlands

Correspondence
M. J. B. Theulings, Computational Design and Mechanics, Delft University of Technology, Delft, The Netherlands.
Email: m.j.b.theulings@tudelft.nl

Abstract

In topology optimization of transient problems, memory requirements and computational costs often become prohibitively large due to the backward-in-time adjoint equations. Common approaches such as the Checkpointing (CP) and Local-in-Time (LT) algorithms reduce memory requirements by dividing the temporal domain into intervals and by computing sensitivities on one interval at a time. The CP algorithm reduces memory by recomputing state solutions instead of storing them. This leads to a significant increase in computational cost. The LT algorithm introduces approximations in the adjoint solution to reduce memory requirements and leads to a minimal increase in computational effort. However, we show that convergence can be hampered using the LT algorithm due to errors in approximate adjoints. To reduce memory and/or computational time, we present two novel algorithms. The hybrid Checkpointing/Local-in-Time (CP/LT) algorithm improves the convergence behavior of the LT algorithm at the cost of an increased computational time but remains more efficient than the CP algorithm. The Parallel-Local-in-Time (PLT) algorithm reduces the computational time through a temporal parallelization in which state and adjoint equations are solved simultaneously on multiple intervals. State and adjoint fields converge concurrently with the design. The effectiveness of each approach is illustrated with two-dimensional density-based topology optimization problems involving transient thermal or flow physics. Compared to the other discussed algorithms, we found a significant decrease in computational time for the PLT algorithm. Moreover, we show that under certain conditions, due to the use of approximations in the LT and PLT algorithms, they exhibit a bias toward designs with short characteristic times. Finally, based on the required memory reduction, computational cost, and convergence behavior of optimization problems, guidelines are provided for selecting the appropriate algorithms.

KEYWORDS

adjoint sensitivity analysis, checkpointing, local-in-time, parallel-local-in-time, topology optimization of transient problems

1 | INTRODUCTION

Topology optimization is used to find the optimal material distribution for problems involving various types of physics such as mechanics, thermal, fluidics, magnetics, and many more. Within the vast literature on topology optimization, much attention is given to static problems. Few work focus on solving transient optimization problems,[1] although there is often a need for considering transient effects, for example, when designing thermally actuated compliant mechanisms,[2] transient heat conducting devices,[3] or fluid pumps.[4]

Gradient-based algorithms such as the method of moving asymptotes[5] (MMA) are often used to solve optimization problems such as topology optimization. To compute the required gradients, adjoint sensitivity calculations are commonly used due to their computational efficiency for problems with many design variables. For static problems, first the state equations pertaining to the optimization problems physics are solved. Secondly, the adjoint equations are solved resulting in the adjoint variables which are combined with the state solution to compute the sensitivities. A transient adjoint sensitivity computation follows the same procedure but has a larger computational cost and requires more memory because the state solutions are required to compute the sensitivities. During the *forward-in-time* solve of the state equations the solutions are thus stored for every discrete time step. Subsequently, the adjoint equations are solved and the adjoint solution is combined with the state solution to compute the sensitivities. The adjoint equations are however a terminal value problem and need to be solved *backward-in-time*.[6] A transient adjoint sensitivity analysis may lead to prohibitively large memory requirements as the state solution needs to be stored for every time step, and to large computational cost since solving the transient adjoint equations is often as expensive as solving the transient state equations. The described algorithm for transient adjoint sensitivity computation will be referred to as the Global-in-Time (GT) algorithm as it solves the adjoint and state equations in the whole (global) time domain at once. In this paper we investigate algorithms to reduce memory requirements while keeping computational cost low.

A relatively straightforward method to reduce memory requirements for transient optimization problems is the method of equivalent static loads (ESLs). Experienced designers are able to make heuristic approximations or create simple computations of dynamic peak loads and consequently optimize dynamic structures using ESLs representing these peak loads.[7] Advancements on these ad hoc design practices have been made by computing the ESLs of every displacement field computed in a transient analysis, where the ESLs at each time step is constructed such that it results in the exact displacement at that time step. Subsequently, these ESLs are used in a static optimization containing many loads.[8] Memory requirements may become substantial computing and storing an ESL of the same size as the state vector at each time step. Transient problems with quickly decaying transient state solutions, referred to as *stiff* problems, may be optimized using ESLs as the transient part of the state response may be neglected. An example of such a problem would be the dynamic optimization of a system which is subject to high stiffness and relatively low inertia effects. Although ESLs are useful for many simple linear design applications, transient effects are ignored and when a complex transient system is optimized the ESL approach may become inaccurate. In these optimization problems the transient response cannot be neglected but due to a regularity of the input and linearity of the state equations, model order reduction techniques may be used to reduce memory and/or computational cost. Methods based on model order reduction are able to circumvent the backward-in-time computation of the adjoint equations[9] or reduce the amount of required storage.[10] Frequency based modal reduction techniques have been used to reduce the amount of storage in transient topology optimization but are limited by the computational cost of solving the eigenvalue problem,[11] and have been shown to become computationally worthwhile only when when many time steps are considered.[12] However, when complex boundary conditions which change drastically over time or nonlinear systems are considered, constructing an accurate modal or proper orthogonal decomposition may prove cumbersome. For these systems, solving the complete state and adjoint solutions as in the GT algorithm is recommended. Characterizing the system to be optimized is crucial for the selection of the appropriate method for topology optimization of transient problems. In this work we focus on methods for topology optimization of transient problems which are nonlinear and/or subject to complex loads.

Several approaches tackling the large memory requirements of the GT algorithm have been proposed in literature. A well-known method to reduce data storage is the Checkpointing (CP) algorithm[13,14] in which the temporal domain is subdivided into multiple intervals. The algorithm consists of two steps. First, the forward state solution is computed but only stored at the interfaces between the intervals, that is, the so-called checkpoints. Second, the state solution is computed and stored only for the final interval and the adjoint equation is propagated backward on this interval. Subsequently, the state solution on the final interval is removed from memory, the state solution on the second-to-last interval is computed

again and stored on each time step and the adjoint variables are propagated further backward, and so on. Although the CP algorithm reduces memory requirements, it increases computational cost as the state equations are evaluated twice: for the computation of solutions on the checkpoints and for the backward propagation of the adjoint variables. The CP scheme was originally introduced to find the sensitivity of iterative functions using automatic differentiation,[13] where a binomial distribution of checkpoints was proposed which was proven optimal with respect to memory requirements.[15] Further developments of the CP algorithm have focused on problems where the number of time steps is not known a priori.[16,17]

Another method to reduce memory requirements is the Local-in-Time (LT) algorithm.[18] Similar to the CP algorithm, the LT algorithm divides the temporal domain into intervals but stores approximate adjoint solutions on the checkpoints instead of exact state solutions. The LT algorithm computes the sensitivity contribution of one interval at a time. To compute the sensitivity contribution of an interval, first the state equations on the interval are solved forward-in-time starting with an exact initial condition. Subsequently, the adjoint equations are solved backward-in-time starting from an approximate terminal adjoint solution for the interval. The sensitivity contribution is computed by combining adjoint and state solutions. Starting at the exact terminal state solution for the current interval the exact state solution can be computed in the next interval. Using an approximate terminal adjoint in the next interval, adjoint solutions can be evaluated and combined with the state solution to compute the sensitivities. To perform the sensitivity analysis, the full state solution needs to be stored for only one interval at a given time. Moreover, by updating the approximate adjoint solutions at the checkpoints in each design iteration, the LT algorithm is able to converge to an optimum while simultaneously converging the approximate adjoints to the exact adjoints. However, since approximate solutions are used for the adjoint equations, the LT algorithm computes *approximate* sensitivities in contrast to the GT and CP algorithms which compute *exact* sensitivities. The main advantage of the LT algorithm is its computational cost. As the LT algorithm only solves the state equations once on every interval, it has a computational cost comparable to the GT algorithm and lower than the CP algorithm. Three dimensional topology optimization problems have been successfully tackled using the LT algorithm[19,20] but further research on the effect of approximate adjoint fields on the convergence behavior of the optimization process and the obtained optimal solution is needed.

Simultaneously reducing the error on approximate adjoint variables and converging to the optimal design share similarities with multiple shooting algorithms for optimal control.[21] Such algorithms split the temporal domain into intervals to which are attached both approximate terminal adjoint and approximate initial state solutions. Subsequently, control problems are optimized by simultaneously solving for the control variables, approximate state, and approximate adjoint variables. Moreover, using approximate state and adjoint solutions, all temporal intervals can be decoupled and parallelized in time.[22] In comparison, common practice for parallel speedup in topology optimization is to parallelize via domain decomposition.[23,24] In domain decomposition, state equations are solved by splitting the spatial domain into several subdomains and performing computations on these domains in parallel.[25] A limitation in these methods is that adjacent domains share certain degrees of freedom (DOFs) and thus need to communicate with each other. If many subdomains and processors are used, a substantial amount of time is spent on communication.[25,26] However, when parallelization via domain decomposition saturates, parallel-in-time algorithms often offer opportunities for further parallelization.[27]

In this work we focus on methods which balance memory requirements and computational cost for topology optimization of transient problems which are nonlinear or subject to complex loads. We investigate the limitations of the CP and LT algorithms in terms of computational cost and convergence behavior, and propose two novel algorithms:

- the hybrid Checkpointing/Local-in-Time (CP/LT) algorithm which introduces an error measure and corrections for the LT algorithm,
- the Parallel-Local-in-Time (PLT) algorithm which parallelizes the optimization process by decomposing the time domain.

The new CP/LT algorithm addresses errors in the LT algorithm while keeping computational cost to a minimum. The PLT algorithm divides the temporal domain into parallel intervals and performs the computations on all these intervals at once while keeping computational overhead to a minimum by carrying out all parallel communication at once and only communicating between adjacent intervals. The computational cost, memory requirements, and limitations of the proposed algorithms are evaluated and compared to the state of the art on two-dimensional density-based topology

optimization problems: a transient heat conductor and a transient fluid pump. Both are discretized using the finite volume method.[28]

The remainder of this article is organized as follows. In Section 2, the exact methods for transient sensitivity analysis are introduced. A general approach for solving the transient state/adjoint equations and combining the solutions into the sensitivity computations is given and the CP algorithm is introduced. In Section 3, approximate algorithms for transient sensitivity analysis are introduced. Subsequently, in Section 3.1, we derive the LT algorithm from these general equations and identify some issues concerning stability and convergence. In Section 3.2, the novel CP/LT algorithm is proposed to address these stability and convergence issues. The final novel PLT algorithm which efficiently parallelizes the transient problem is developed in Section 3.3. The theoretical memory requirements and computational cost of the GT, CP, LT, CP/LT, and PLT algorithms are analyzed in Section 4, after which the algorithms are compared and evaluated using actual optimization examples in Section 5. Stability and convergence of the approximate algorithms are investigated in Section 5.1 using a thermal transient optimization problem and computational time is compared in Section 5.2 using a transient flow optimization problem. Finally, guidelines for algorithm selection are given in Section 6 and a discussion and conclusion on the results and algorithms for transient sensitivity analysis are provided in Section 7.

2 | EXACT METHODS FOR SENSITIVITY ANALYSIS OF TRANSIENT PROBLEMS

In this work we focus on methods for topology optimization of transient problems which are nonlinear or subject to complex transient loads. Optimization is performed using gradient-based algorithms which require the sensitivities of the objective and constraints with respect to the design variables. We are thus interested in methods which aim to compute discrete adjoint sensitivities such as the GT, CP, and LT algorithms. In this section we investigate the GT and CP algorithms which compute exact discrete sensitivities.

2.1 | Discrete transient sensitivities and the Global-in-Time algorithm

A generic topology optimization problem for transient physics on spatial domain $\vec{x} \in \Omega$ with boundary $\Gamma = \overline{\Omega} \setminus \Omega$, where $\overline{\Omega}$ is the closure of Ω, is defined as:

$$
\begin{array}{lll}
\underset{s(\vec{x})}{\text{minimize}} & F = \displaystyle\int_{t=0}^{t=t_t} \int_{\Omega} f(u(\vec{x},t), s(\vec{x})) d\Omega dt & \\
\text{subject to} & R(u(\vec{x},t), s(\vec{x}), t) = 0, & t \in [0, t_t],\ \vec{x} \in \Omega, \\
& R_\Gamma(u(\vec{x},t), \partial u(\vec{x},t)/\partial \vec{x}, s(\vec{x}), \vec{x}, t) = 0, & t \in [0, t_t],\ \vec{x} \in \Gamma, \\
& u(\vec{x}, t=0) = \hat{u}(\vec{x}), & \vec{x} \in \Omega, \\
& g_i(s(\vec{x})) \leq 0, & i \in I,\ \vec{x} \in \Omega,
\end{array}
\tag{1}
$$

where t_t is the terminal time, $u(\vec{x},t)$ is the time dependent state solution with $\hat{u}(\vec{x})$ as initial condition, $s(\vec{x})$ is the design field, $R(u(\vec{x},t), s(\vec{x}), t)$ is the Partial Differential Equation (PDE) constraint with boundary conditions $R_\Gamma(u, \partial u/\partial \vec{x}, s, \vec{x}, t)$ defined on boundary Γ, and $g_i(s(\vec{x}))$ is the ith inequality constraint in set I. Subsequently the continuous optimization problem is discretized in space and time as:

$$
\begin{array}{lll}
\underset{\boldsymbol{s}}{\text{minimize}} & F \approx \displaystyle\sum_{n=0}^{N} F_n(\boldsymbol{u}_n, \boldsymbol{s}) \Delta t & \\
\text{subject to} & R_n(\boldsymbol{u}_{n-1}, \boldsymbol{u}_n, \boldsymbol{s}, t_n) = 0, & n \in \{1, 2 \ \ldots \ N\}, \\
& R_0(\boldsymbol{u}_0, \boldsymbol{s}, t_0) = 0, & \\
& g_i(\boldsymbol{s}) \leq 0, & i \in I
\end{array}
\tag{2}
$$

where the full temporal domain $t \in [0, t_t]$ is divided into N time steps of length $\Delta t = t_t/N$, $\boldsymbol{u}_n$ is the column containing discretized state variables at time t_n, all discrete design variables are gathered in column $\boldsymbol{s}$, the continuous integral objective is discretized at time step n using $F_n \Delta t$, the column $R_n(\boldsymbol{u}_{n-1}, \boldsymbol{u}_n, \boldsymbol{s})$ contains all discretized PDE constraints and boundary conditions at time t_n, $R_0(\boldsymbol{u}_0, \boldsymbol{s}, t_0)$ contains the initial conditions, and $g_i(\boldsymbol{s})$ is the i^{th} inequality constraints on $\boldsymbol{s}$. Furthermore, the discretized time step $R_n(\boldsymbol{u}_{n-1}, \boldsymbol{u}_n, \boldsymbol{s}, t_n)$ accounts for several discrete integration techniques such as backward/forward Euler, Crank–Nicolson, and several of the Runge–Kutta methods. The equations derived in this work are thus valid for these methods which can be described using $R_n(\boldsymbol{u}_{n-1}, \boldsymbol{u}_n, \boldsymbol{s}, t_n)$. To compute the sensitivities of the PDE constrained objective, an augmented objective F^* is constructed,

$$F^* = F_0(\boldsymbol{u}_0, \boldsymbol{s})\Delta t + \boldsymbol{\lambda}_0^\intercal R_0(\boldsymbol{u}_0, \boldsymbol{s}, t_0) + \sum_{n=1}^{N} \left(F_n(\boldsymbol{u}_n, \boldsymbol{s})\Delta t + \boldsymbol{\lambda}_n^\intercal R_n(\boldsymbol{u}_{n-1}, \boldsymbol{u}_n, \boldsymbol{s}, t_n) \right), \tag{3}$$

where the constraints are introduced using adjoint variables $\boldsymbol{\lambda}_n$ at time t_n. Subsequently, to construct the adjoint equations, the sensitivities are derived as:

$$\begin{aligned} \frac{dF^*}{d\boldsymbol{s}} = &\sum_{n=0}^{N-1} \left(\frac{\partial F_n}{\partial \boldsymbol{s}} \Delta t + \boldsymbol{\lambda}_n^\intercal \frac{\partial R_n}{\partial \boldsymbol{s}} + \left(\frac{\partial F_n}{\partial \boldsymbol{u}_n} \Delta t + \boldsymbol{\lambda}_n^\intercal \frac{\partial R_n}{\partial \boldsymbol{u}_n} + \boldsymbol{\lambda}_{n+1}^\intercal \frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n} \right) \frac{\partial \boldsymbol{u}_n}{\partial \boldsymbol{s}} \right) \\ &+ \frac{\partial F_N}{\partial \boldsymbol{s}} \Delta t + \boldsymbol{\lambda}_N^\intercal \frac{\partial R_N}{\partial \boldsymbol{s}} + \left(\frac{\partial F_N}{\partial \boldsymbol{u}_N} \Delta t + \boldsymbol{\lambda}_N^\intercal \frac{\partial R_N}{\partial \boldsymbol{u}_N} \right) \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{s}}, \end{aligned} \tag{4}$$

where we drop the dependencies of R_n on $\boldsymbol{u}_{n-1}$, $\boldsymbol{u}_n$, $\boldsymbol{s}$, and t_n for brevity. To avoid the computation of matrix $\partial \boldsymbol{u}_n / \partial \boldsymbol{s}$, we set all sums multiplied by $\partial \boldsymbol{u}_n / \partial \boldsymbol{s}$ to zero, resulting in the adjoint equations:

$$n = N : \boldsymbol{\lambda}_N = -\frac{\partial R_N}{\partial \boldsymbol{u}_N}^{-\intercal} \frac{\partial F_N}{\partial \boldsymbol{u}_N}^{\intercal} \Delta t, \tag{5a}$$

$$n \in \{N-1, N-2, \dots 0\} : \boldsymbol{\lambda}_n = -\frac{\partial R_n}{\partial \boldsymbol{u}_n}^{-\intercal} \left(\frac{\partial F_n}{\partial \boldsymbol{u}_n}^{\intercal} \Delta t + \frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n}^{\intercal} \boldsymbol{\lambda}_{n+1} \right). \tag{5b}$$

Since the adjoint equations only have a terminal condition for $\boldsymbol{\lambda}_N$, they can only be solved backward-in-time starting at $t_N = t_t$ taking backward steps until $t_0 = 0$. A notable property of the backward-in-time adjoint equations is the fact that they resemble the forward-in-time state equations. If we assume that R_n is a linear equation with respect to $\boldsymbol{u}_{n-1}$ and $\boldsymbol{u}_n$, the forward-in-time state equations may be written as the solution of:

$$n = 0 : \boldsymbol{u}_0 = \hat{\boldsymbol{u}}_0, \tag{6a}$$

$$n \in \{1, 2, \dots N\} : \boldsymbol{u}_n = -\frac{\partial R_n}{\partial \boldsymbol{u}_n}^{-1} \left(\boldsymbol{q}_n + \frac{\partial R_n}{\partial \boldsymbol{u}_{n-1}} \boldsymbol{u}_{n-1} \right), \tag{6b}$$

where $\hat{\boldsymbol{u}}_0$ are the discretized initial conditions and $\boldsymbol{q}_n$ is the part of R_n independent of the state variables. Since the backward-in-time adjoint equations mirror the forward-in-time state equations, we can assume that the same stability and convergence criteria hold for both equations. However, for nonlinear $R_n(\boldsymbol{u}_{n-1}, \boldsymbol{u}_n, \boldsymbol{s}, t_n)$ there is an important difference between the state and adjoint equations. As the adjoint equations are a linearized version of the state equations, solving the adjoint equations may take significantly less computational work and time than solving nonlinear state equations. At best both the state and adjoint equations are linear and require the same amount of computational work.

After solving the adjoint equations backward-in-time, the adjoint variables can be combined with the stored state equations to compute the sensitivities:

$$\frac{dF}{d\boldsymbol{s}} = \sum_{n=0}^{N} \left(\frac{\partial F_n}{\partial \boldsymbol{s}} \Delta t + \boldsymbol{\lambda}_n^\intercal \frac{\partial R_n}{\partial \boldsymbol{s}} \right). \tag{7}$$

To compute the sensitivities, two types of equations need to be solved; the state equations pertaining to the physics, and the adjoint equations. The most straightforward method to solve these equations is the GT algorithm which first solves

the state equations forward-in-time and stores the state solution at every time step. Subsequently, the adjoint equations are solved backward-in-time using Equation (5) while simultaneously updating the sensitivities using Equation (7). As the sensitivities depend on $\partial R_n/\partial \boldsymbol{s}$ which in turn is dependent on $\boldsymbol{u}_n$ and $\boldsymbol{u}_{n-1}$, the state solution is thus required while propagating the adjoint solution backward to computed the sensitivities. The full state solution is thus stored during the forward solve and the main restriction in using the GT algorithm is the large memory requirement. For a problem with a state solution $\boldsymbol{u}_n$ of size m and involving N time steps, memory requirements M scale proportionally as $M \propto mN$. To compare the computational cost of the following algorithms in Sections 2.2 and 3 we evaluate the cost of the GT algorithm. We define the cost of solving one state step $R_n(\boldsymbol{u}_{n-1}, \boldsymbol{u}_n, \boldsymbol{s}, t_n)$ as c_s and the cost of solving one adjoint step using Equation (5) and consequently updating sensitivities using Equation (7) as c_a. The cost of the adjoint step and updating sensitivities is combined as this is a practical implementation in code. For highly nonlinear systems the cost of solving an adjoint step is much cheaper and we define ratio $r_c = c_a/c_s$. As the GT algorithm solves a state/adjoint step only once for all N time steps, the computational cost C scales proportionally as $C \propto Nc_s(1 + r_c)$.

2.2 | The Checkpointing algorithm

To reduce memory requirements the CP algorithm[13,14] may be used. The algorithm subdivides the temporal domain into K discrete intervals of length $\Delta\theta = t_t/K$, resulting in $K + 1$ checkpoints θ_k, where each θ_k is assumed to correspond to one of the discrete times t_n but not all t_n have a corresponding θ_k. The K intervals are thus defined as $\Theta_k = [\theta_k, \theta_{k+1}]$, and contain all discrete times $\{n|t_n \in [\theta_k, \theta_{k+1}]\}$, as illustrated in Figure 1. The subscript k is used to denote checkpoints and intervals. Furthermore, we use state/adjoint variables $\boldsymbol{U}_k/\boldsymbol{\Lambda}_k$ as the variables at checkpoint k where $\boldsymbol{U}_k = \boldsymbol{u}_n$ and $\boldsymbol{\Lambda}_k = \lambda_n$ at $\theta_k = t_n$, respectively. These variables and subscript k are introduced to clearly describe and visualize the algorithms presented in this paper as will be shown for the CP algorithm in Figure 2. To compute sensitivities, the CP algorithm first computes the full forward state solution and stores only $\boldsymbol{U}_k$ at checkpoints θ_k. The full state solution is only stored for the last interval Θ_{K-1} and adjoint equations are subsequently solved and used to update the sensitivities, after which the state solution on the terminal interval can be removed from memory. Next, the adjoint solution is propagated further backward by recomputing and storing the state solution at the final to last interval Θ_{K-2} from the stored state solution $\boldsymbol{U}_{K-2}$ at checkpoint θ_{K-2}. Thus by continuously recomputing the state solution on the previous interval, and removing used state solutions from memory, exact sensitivities can be computed while reducing the memory requirements. A schematic of the CP algorithm can be found in Figure 2.

Depending on the number of intervals K, memory may be greatly reduced. If an initial state solution $\boldsymbol{U}_k$ of size m is stored for each of the K intervals except the first where it is found by solving $R_0(\boldsymbol{u}_0, \boldsymbol{s}, t_0)$, this requires the storage of $m(K-1)$ discrete state variables. Additionally, on each of the intervals we require the storage of the full state solution which requires a memory of mN/K discrete state variables. Memory requirements thus scale as $M \propto m(K-1) + mN/K$, and are reduced with respect to the GT algorithm. To reduce the computational overhead, we recommend to use the minimal number of intervals allowed by the memory limitations. Other approaches for optimal memory reduction such as the binomial distribution of checkpoints are also proposed in the literature.[15]

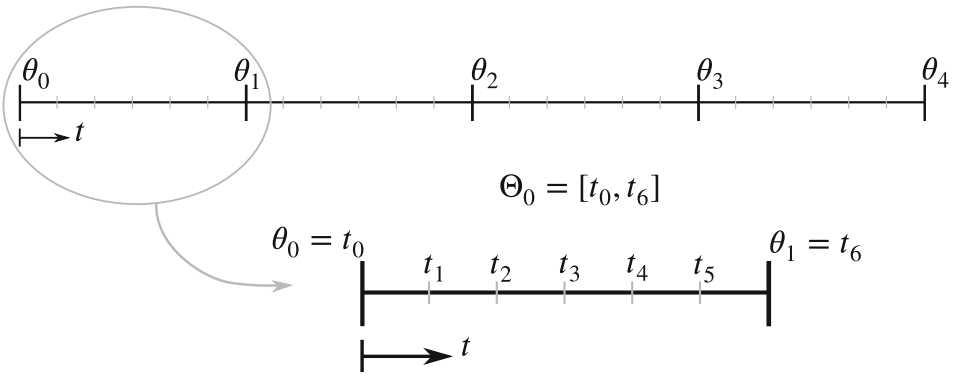


FIGURE 1 A temporal domain discretized using $N = 24$ time steps, and subdivided into $K = 4$ intervals $\Theta_k = [\theta_k, \theta_{k+1}]$, where θ_k are the temporal checkpoints. Each interval Θ_k contains time steps $\{n|t_n \in [\theta_k, \theta_{k+1}]\}$. For instance, the first interval Θ_0 is defined as $\Theta_0 = [t_0, t_6]$, and contains discrete time steps $n \in \{0, 1, 2, 3, 4, 5, 6\}$.

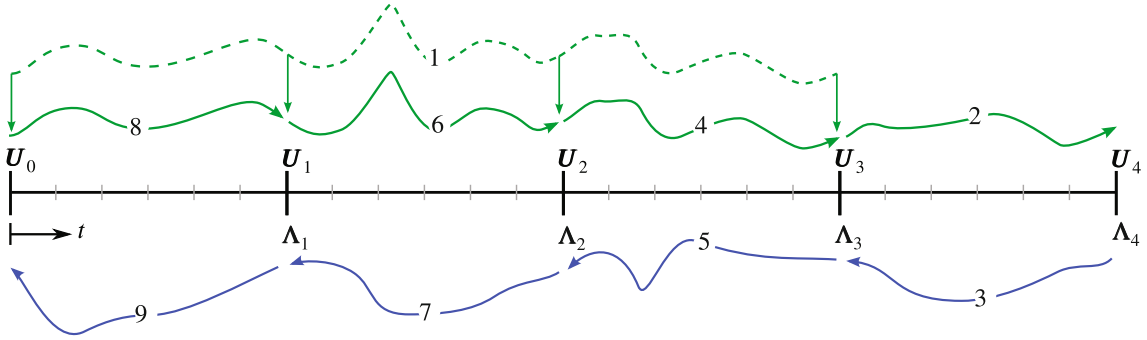


FIGURE 2 A schematic of the CP algorithm. The numbers represent the order of operations. The dashed green line represents the solving of the full forward state solution while only storing it at the checkpoints. A green arrow represents the computation and storage of state solutions on a complete interval. A blue arrow represents the backward computation of adjoint variables and update of sensitivities using the adjoint variables and stored state variables.

The reduced memory in the CP algorithm comes at a higher computational cost as state solutions are recomputed on the first $K-1$ intervals of length N/K. Recomputation of the state solutions is associated to an additional cost proportional to $c_s(N/K)(K-1) = Nc_s(1-1/K)$. The computational cost of the CP algorithm is thus the cost of the GT algorithm ($C \propto Nc_s(1+r_c)$) with the addition of the recomputed state solutions, that is, $C \propto Nc_s(2+r_c-1/K)$.

3 | APPROXIMATE METHODS FOR SENSITIVITY ANALYSIS OF TRANSIENT PROBLEMS

Other approaches for the reduction of memory requirements and computational time are based on approximations of the state and/or adjoint variables. Generally, these methods rely on an iterative procedure to update approximate state and/or adjoint variables and the design until convergence. In a standard optimization procedure, design variables $\boldsymbol{s}^j$ at optimization iteration j are iteratively improved by computing sensitivities $dF^*/d\boldsymbol{s}^j$ and using a gradient-based optimizer. Exact solutions for all state and adjoint variables are computed. In the proposed approximate methods, state and/or adjoint equations are not satisfied at every iteration j. Approximate states $\widetilde{\boldsymbol{U}}_k^j \approx \boldsymbol{U}_k^j$ and/or adjoint variables $\widetilde{\boldsymbol{\Lambda}}_k^j \approx \boldsymbol{\Lambda}_k^j$ at checkpoints θ_k are updated using fixed point iterations. The idea is to let these variables converge to exact solutions simultaneously with the convergence of the design to the optimum, with the purpose of reducing memory requirements and/or computational time. Examples of such algorithms are the LT and multiple shooting type algorithms. In this section, we first introduce the LT algorithm and discuss its limitations. Subsequently, we propose some modifications to the LT algorithm to increase stability and introduce a novel multiple shooting type algorithm for topology optimization of transient problems.

3.1 | The Local-in-Time algorithm

To reduce memory requirements, the LT[18] algorithm can be used. In the LT algorithm, adjoint variables are approximated and solved for iteratively. The algorithm computes the sensitivities successively on each interval Θ_k moving forward-in-time. Memory is reduced as the state solution is only stored on a single interval Θ_k at a time. However, errors are introduced as we approximate $\widetilde{\boldsymbol{\Lambda}}_k^j \approx \boldsymbol{\Lambda}_k^j$ and solve for the exact adjoint over design iterations j. Furthermore, to initialize the LT algorithm approximate adjoint variables $\widetilde{\boldsymbol{\Lambda}}_k^{j=0}$ are required and are set to $\widetilde{\boldsymbol{\Lambda}}_k^{j=0} = \boldsymbol{0}$ as suggested by Yamaleev et al.[18] The process to perform the complete sensitivity analysis is illustrated in Figure 3. The contribution of one interval Θ_k is computed following three steps:

1. **Computation and storage of the state solution** Firstly, the initial state solution $\boldsymbol{u}_n^j = \boldsymbol{U}_k^j$ at time $t_n = \theta_k$ is retrieved, which is either the initial value from $R_0(\boldsymbol{U}_0^j, \boldsymbol{s}, t_0) = \boldsymbol{0}$ at $k = 0$ or the terminal value $\boldsymbol{U}_k^j$ at time θ_k on the previous interval

Θ_{k-1}. Starting from this initial state solution, the complete state solution is evaluated and stored forward-in-time until $\boldsymbol{U}^j_{k+1}$ at time θ_{k+1} is reached.

2. **Computation of approximate adjoint solution and sensitivities** We retrieve an approximate terminal adjoint variable $\widetilde{\lambda}^j_n = \widetilde{\boldsymbol{\Lambda}}^j_{k+1}$ at time $t_n = \theta_{k+1}$ from memory, or when the terminal interval Θ_{K-1} is being investigated we compute the exact terminal adjoint $\lambda^j_N = \boldsymbol{\Lambda}^j_K$ by solving Equation 5a. Subsequently, Equation (5b) and the stored state solution $\boldsymbol{u}^j_n$ are used to propagate the approximate adjoint backward as:

$$\widetilde{\lambda}^j_n = -\left(\frac{\partial R_n}{\partial \boldsymbol{u}_n}\right)^{-\mathsf{T}}\left(\frac{\partial F_n}{\partial \boldsymbol{u}_n}\Delta t + \frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n}^{\mathsf{T}}\widetilde{\lambda}^j_{n+1}\right), \text{for } n \in \{n|\theta_k + \Delta t \leq t_n < \theta_{k+1}\}, \tag{8}$$

while simultaneously updating the sensitivities using Equation 7.

3. **Update and storage of approximate adjoint variables** After computing the sensitivities on the interval, we solve Equation 5b one final time to compute the approximate adjoint at the checkpoint at time $t_n = \theta_k$ for the next design iteration $j+1$, as:

$$\widetilde{\lambda}^{j+1}_n \approx -\left(\frac{\partial R_n}{\partial \boldsymbol{u}_n}\right)^{-\mathsf{T}}\left(\frac{\partial F_n}{\partial \boldsymbol{u}_n}\Delta t + \frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n}^{\mathsf{T}}\widetilde{\lambda}^j_{n+1}\right), \text{for } n|t_n = \theta_k. \tag{9}$$

The approximate adjoint $\widetilde{\boldsymbol{\Lambda}}^{j+1}_k = \widetilde{\lambda}^{j+1}_n$ from Equation 9 is stored to be used as a terminal value at $t_n = \theta_k$ for interval Θ_{k-1} in the next design iteration $j+1$. We emphasize that within iteration j, adjoint vector $\widetilde{\lambda}^j_n = \widetilde{\boldsymbol{\Lambda}}^j_k$ and not $\widetilde{\lambda}^{j+1}_n = \widetilde{\boldsymbol{\Lambda}}^{j+1}_k$ at time $t_n = \theta_k$ is used to update the sensitivities using Equation (7).

After the evaluation of sensitivity contributions on interval Θ_k, the state solutions on the interval are removed from memory. Starting from the solution $\boldsymbol{u}^j_n = \boldsymbol{U}^j_{k+1}$, the sensitivities in domain Θ_{k+1} are computed as shown in the LT schematic in Figure 3. In the LT algorithm, approximate adjoint variables $\widetilde{\lambda}^j_n = \widetilde{\boldsymbol{\Lambda}}^j_{k+1}$ are used to compute sensitivities on interval Θ_k. They are propagated backward-in-time and used to update $\widetilde{\boldsymbol{\Lambda}}^{j+1}_k = G(\widetilde{\boldsymbol{\Lambda}}^j_{k+1})$ using Equations (8) and (9). Here, we define $G(\widetilde{\boldsymbol{\Lambda}}^j_{k+1})$ as an operator which propagates adjoint variables backward-in-time over an interval as used in Figure 3, which we find to be a linear operator by inspecting Equations (8) and (9).

It should be noted that the adjoint variables at time $t_n = t_N$ are always exact as they are computed using Equation 5a, and all adjoint solutions on terminal interval Θ_{K-1} are thus exact. Consequently, when adjoint variables have stabilized ($\widetilde{\boldsymbol{\Lambda}}^j_k = \widetilde{\boldsymbol{\Lambda}}^{j+1}_k = G(\widetilde{\boldsymbol{\Lambda}}^j_{k+1})$), the exact variables from the terminal interval propagate backward and the approximate adjoint variables have converged to the exact variables found in the GT algorithm ($\widetilde{\boldsymbol{\Lambda}}^j_k = \boldsymbol{\Lambda}^j_k$). The idea behind the LT algorithm is thus that approximate adjoint variables are improved by $\widetilde{\boldsymbol{\Lambda}}^{j+1}_k = G(\widetilde{\boldsymbol{\Lambda}}^j_{k+1})$ and converge to the exact values as the design converges to the optimum and stabilizes.

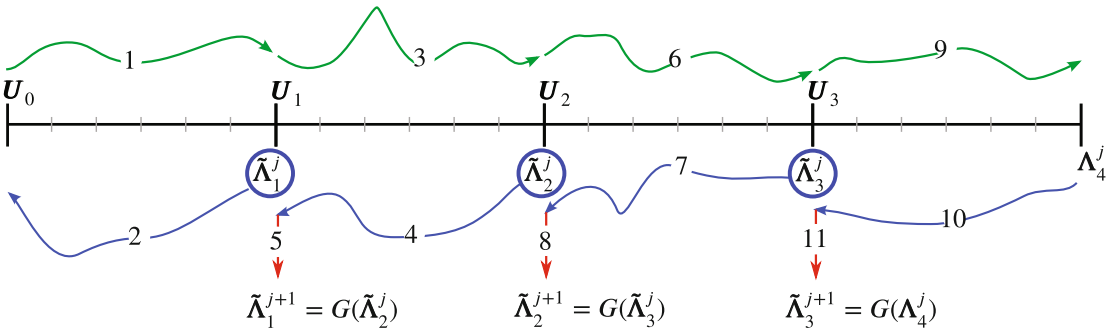


FIGURE 3 A schematic of the LT algorithm. The numbers represent the order of operations. A green arrow represents the computation and storage of state solutions on a certain interval. The blue arrow represents the computation of adjoint variables and update of sensitivities using the stored state and computed adjoint variables. Finally, the red arrow represents the update of the approximate adjoint variable.

Memory requirements may be significantly reduced using the LT algorithm. Since the adjoint vector $\widetilde{\boldsymbol{\Lambda}}_k$ and state vector $\boldsymbol{U}_k$ both have the same size m and the LT algorithm stores the approximate terminal adjoint variables on each of the K intervals except the last, $m(K-1)$ discrete approximate adjoint variables need to be stored. In addition, the complete state solution on one of the intervals at a time is stored which requires the storage of mN/K discrete state variables. The memory requirement of the LT algorithm thus scales as $M \propto m(K-1) + mN/K$, which is identical to the CP algorithm.

Whereas the CP algorithm reduces memory at the cost of increased computational time compared to the GT algorithm, the LT algorithm decreases memory requirements with no significant increase in computational time. As the state and adjoint solutions are solved only once per interval, the computational cost of the LT algorithm is exactly the same as the cost of the GT algorithm: $C \propto Nc_s(1+r_c)$. However, this does assume that the convergence of the optimization process using the LT algorithm is not negatively affected by the use of approximate adjoint solutions and sensitivity information.

3.1.1 | Stability and convergence of the LT algorithm

In the LT algorithm, it is assumed that as a design converges over multiple design iterations, the adjoint field will also converge to the exact adjoint field resulting in accurate sensitivities. Using these sensitivities, we ensure an accurate local minimum is found. The assumption is thus that upon stabilization of the design, the adjoint field stabilizes and $\boldsymbol{\Lambda}_k^j = \widetilde{\boldsymbol{\Lambda}}_k^j = G(\widetilde{\boldsymbol{\Lambda}}_{k+1}^j)$. To allow for both the design and adjoint fields to stabilize, two types of convergence are necessary: local convergence and global convergence. Local convergence relates to the decrease in the error of the approximate adjoint field on subdomain Θ_k. Assuming that, by backward-in-time propagation of $\widetilde{\boldsymbol{\Lambda}}_k^{j+1} = G(\widetilde{\boldsymbol{\Lambda}}_{k+1}^j)$, the error in $\widetilde{\boldsymbol{\Lambda}}_k^{j+1}$ decreases with respect to the error in $\widetilde{\boldsymbol{\Lambda}}_{k+1}^j$ on every interval Θ_k, we are able to deduce that upon a stable nonchanging design $\boldsymbol{s}^j$ all approximation errors will reduce to zero. Global convergence relates to the simultaneous convergence of the design and approximate adjoint field.

Local convergence

Using a modal analysis of the backward-in-time adjoint equations, local convergence is investigated. We assume that the exact adjoint variables λ_n at time t_n can be expressed as the sum of the approximate adjoint variables and a correction $\Delta\lambda_n$:

$$\lambda_n = \widetilde{\lambda}_n + \Delta\lambda_n. \tag{10}$$

Furthermore, we assume that the approximate adjoint variables approach the exact adjoint variables as found in the GT algorithm in a stable optimum such that $\widetilde{\lambda}_n \approx \lambda_n$ and the adjoint correction approaches zero $\Delta\lambda_n \to \mathbf{0}$. We thus investigate the evolution of the corrections by constructing a time stepping scheme for the corrections. In Equation (11a), the exact adjoint equation is repeated from Equation (5). The improved approximation $\lambda_n^j = \widetilde{\lambda}_n^j + \Delta\lambda_n^j$ is substituted into Equation (11a) resulting in Equation (11b):

$$\lambda_n^j = -\left(\frac{\partial R_n}{\partial \boldsymbol{u}_n^j}\right)^{-\mathsf{T}}\left(\frac{\partial F_n}{\partial \boldsymbol{u}_n^j}\Delta t + \frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n^j}^{\mathsf{T}}\lambda_{n+1}^j\right), \tag{11a}$$

$$\widetilde{\lambda}_n^j + \Delta\lambda_n^j = -\left(\frac{\partial R_n}{\partial \boldsymbol{u}_n^j}\right)^{-\mathsf{T}}\left(\frac{\partial F_n}{\partial \boldsymbol{u}_n^j}\Delta t + \frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n^j}^{\mathsf{T}}\widetilde{\lambda}_{n+1}^j + \frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n^j}^{\mathsf{T}}\Delta\lambda_{n+1}^j\right). \tag{11b}$$

To form the update of the adjoint correction over one time step we subtract the update of the approximate adjoint in Equation (8) from Equation (11b), resulting in:

$$\Delta\lambda_n^j = -\left(\frac{\partial R_n}{\partial \boldsymbol{u}_n^j}\right)^{-\mathsf{T}}\frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n^j}^{\mathsf{T}}\Delta\lambda_{n+1}^j, \tag{12}$$

which propagates the correction one step backward. Subsequently, the adjoint correction can be propagated over an entire interval from time step $n = n_{k+1}$ at terminal time $t_{n_{k+1}} = \theta_{k+1}$ until time step $n = n_k$ at initial time $t_{n_k} = \theta_k$:

$$\Delta\boldsymbol{\Lambda}_k = \left(\prod_{n=n_k+1}^{n_{k+1}} - \left(\frac{\partial R_{n-1}}{\partial \boldsymbol{u}_{n-1}^j} \right)^{-\mathsf{T}} \frac{\partial R_n}{\partial \boldsymbol{u}_{n-1}^j}^{\mathsf{T}} \right) \Delta\boldsymbol{\Lambda}_{k+1} = G_c(\Delta\boldsymbol{\Lambda}_{k+1}), \tag{13}$$

for which we used $\Delta\boldsymbol{\Lambda}_k = \Delta\lambda_n$ at $\theta_k = t_n$ and we define operator G_c which propagates the adjoint correction backward over an interval as in Equation (13). Operator G_c thus differs from operator G in the fact that G propagates an adjoint variable using Equations (8) and (9), and G_c propagates an adjoint correction using Equations (12) and (13). Further analysis of local stability is performed by assuming the state equations $R_n(\boldsymbol{u}_n, \boldsymbol{u}_{n-1}, \boldsymbol{s}, t_n)$ to be linear with respect to $\boldsymbol{u}_n$ and $\boldsymbol{u}_{n-1}$. Although stability criteria can be derived for nonlinear state equations, proving local stability for nonlinear systems is out of the scope of this work. We assume linear state equations R^{lin}, which are used to represent any linear time stepping scheme:

$$R_n^{lin}(\boldsymbol{u}_n, \boldsymbol{u}_{n-1}, \boldsymbol{s}, t_n) = \boldsymbol{u}_n - \boldsymbol{A}_n(\boldsymbol{s})\boldsymbol{u}_{n-1} - \boldsymbol{q}_n(\boldsymbol{s}, t_n) = \boldsymbol{0}, \tag{14}$$

where the state update matrix $\boldsymbol{A}_n(\boldsymbol{s})$ and the external load vector $\boldsymbol{q}_n(\boldsymbol{s}, t_n)$ are both independent of $\boldsymbol{u}_n$ and $\boldsymbol{u}_{n-1}$. We note that any linear time stepping scheme (such as Forward Euler, backward Euler or Crank–Nicolson) can be transformed into R^{lin} by simple matrix manipulations. Consequently, the state variables are updated following the linear time stepping scheme:

$$\boldsymbol{u}_n = \boldsymbol{A}_n(\boldsymbol{s})\boldsymbol{u}_{n-1} + \boldsymbol{q}_n(\boldsymbol{s}, t_n). \tag{15}$$

This linear time stepping scheme is only stable if all eigenvalues ϕ_i of matrix $\boldsymbol{A}_n(\boldsymbol{s})$ are bounded as $|\phi_i(\boldsymbol{K}_n)| < 1$. Returning to the propagation of the adjoint correction we substitute derivatives:

$$\frac{\partial R_{n-1}^{\text{lin}}}{\partial \boldsymbol{u}_{n-1}^j} = \boldsymbol{I}, \quad \frac{\partial R_n^{\text{lin}}}{\partial \boldsymbol{u}_{n-1}^j} = -\boldsymbol{A}_n, \tag{16}$$

into the correction update in Equation (13), and find:

$$\Delta\boldsymbol{\Lambda}_k = \left(\prod_{n=n_k+1}^{n_{k+1}} \boldsymbol{A}_n^{\mathsf{T}} \right) \Delta\boldsymbol{\Lambda}_{k+1}. \tag{17}$$

Consequently, the eigenvalues with which the adjoint errors are propagated backward are also smaller than one $|\phi_i(\boldsymbol{A}_n^{\mathsf{T}})| = |\phi_i(\boldsymbol{A}_n)| < 1$. We may conclude that if the discrete state equations for a linear transient problem are stable then the discrete adjoint equations will also be stable and the adjoint error $\Delta\boldsymbol{\Lambda}_k$ will decrease over an interval. Furthermore, for a larger decrease in error, the number of steps in an interval $n_{k+1} - n_k \gg 1$ should be large or the absolute eigenvalues $|\phi_i(\boldsymbol{A}_n)| \ll 1$ should be smaller. However, a fine spatial mesh and thus a large size of the state solution m may cause large memory usage and thus require a user with limited memory to use many short intervals $K > 1$, which is prohibitive for the local convergence of the LT algorithm.

Global convergence

One of the most important assumptions of the LT method is that the design stabilizes and converges, which allows for the adjoint field to converge to the exact adjoint solution over multiple design iterations. This is caused by local convergence and the terminal adjoint $\boldsymbol{\Lambda}_K^j = \boldsymbol{\lambda}_N^j$ which is exact by definition. When the design $\boldsymbol{s}^j$ and the approximate adjoints $\widetilde{\boldsymbol{\Lambda}}_k^j$ stabilize, the correct terminal adjoint $\boldsymbol{\Lambda}_K^j$ propagates backward-in-time to all other $\widetilde{\boldsymbol{\Lambda}}_k^j$, which must also be exact due to local convergence. Stable design and adjoints thus lead to exact adjoints and consequently an exact optimum where the objective is stationary $dF^*/d\boldsymbol{s} = 0$.

If the design does not stabilize, the adjoint field will not converge to the exact field resulting in erroneous sensitivities, which in turn, can cause the design to destabilize. This type of stability is referred to as global stability in this paper. By changing the adjoint equations, state equations and resulting state solutions, a design update changes the adjoint solutions. The update of the design as $\boldsymbol{s}^{j+1} = \boldsymbol{s}^j + \delta\boldsymbol{s}^j$ is thus associated with a change in exact adjoint variables $\boldsymbol{\Lambda}_k^{j+1} = \boldsymbol{\Lambda}_k^j + \delta\boldsymbol{\Lambda}_k^j$. If we assume local convergence is satisfied, we expect the backward propagation of the approximate adjoint over

an interval to result in the exact adjoint $G(\widetilde{\boldsymbol{\Lambda}}_{k+1}^{j}) = \widetilde{\boldsymbol{\Lambda}}_{k}^{j+1} \approx \boldsymbol{\Lambda}_{k}^{j}$, but only for the current design $\boldsymbol{s}^{j}$. By updating the design, we change the exact adjoint solution by $\delta\boldsymbol{\Lambda}_{k}^{j}$, which in turn, introduces an error in the approximate adjoints estimated using the exact adjoints in the previous design iteration:

$$\widetilde{\boldsymbol{\Lambda}}_{k}^{j+1} \approx \boldsymbol{\Lambda}_{k}^{j} = \boldsymbol{\Lambda}_{k}^{j+1} - \delta\boldsymbol{\Lambda}_{k}^{j}. \tag{18}$$

Conversely, as the sensitivities are dependent on the approximate adjoint variables and as we aim to use gradient-based design updates, $\delta\boldsymbol{s}^{j+1}$ is dependent on $\widetilde{\boldsymbol{\Lambda}}_{k}^{j+1} \approx \boldsymbol{\Lambda}_{k}^{j+1} - \delta\boldsymbol{\Lambda}_{k}^{j}$ and thus on $\delta\boldsymbol{\Lambda}_{k}^{j}$. For the optimization procedure to stabilize, we require the effect of $\delta\boldsymbol{\Lambda}_{k}^{j}$ on $\delta\boldsymbol{s}^{j+1}$ and subsequently the effect of $\delta\boldsymbol{s}^{j+1}$ on $\delta\boldsymbol{\Lambda}_{k}^{j+1}$ to decrease the change in adjoint variables, such that $|\delta\boldsymbol{\Lambda}_{k}^{j+1}| < |\delta\boldsymbol{\Lambda}_{k}^{j}|$. We call instabilities due to this feedback loop of adjoint and design changes *strong global convergence* issues. A formal proof of strong global convergence is out of the scope of this paper and is not further discussed.

Beside strong global convergence, we can make some observations using the similarity between state and adjoint equations leading to *weak global convergence* issues. As shown in Section 2.1, the state equations resemble the adjoint equations and we expect features with large characteristic times in the state equations to also be associated with large characteristic times in the adjoint equations. Large approximate characteristic times[29] are associated with large eigenvalues and relatively large settling times[30] over which state solutions stabilize. Consequently, adjoint errors decay relatively slow in these features. Further examples of these features and the computation of characteristic times will be given in Sections 5.1.1 and 5.2.1. Features with short characteristic times will have faster settling times and eigenvalues leading to a stronger local convergence. Essentially, we expect that the correct sensitivities for features with short characteristic times will be quickly found, while features with long characteristic times will have inaccurate sensitivities and may not be found by the optimizer. This introduces a bias toward features associated with short characteristic times. The LT algorithm may converge to inferior local optima containing features with shorter characteristic times than the local optima found by the GT algorithm which contains features with longer characteristic times.

We thus identified two types of stability which can be enforced in the following ways. For local stability we require the absolute eigenvalues to be smaller than one. To promote local convergence, we either require the maximum absolute eigenvalues to be *much* smaller than one, or the interval lengths to be relatively long. A global stability criterion is harder to define. In the authors experience, by enforcing local stability and using optimization methods which limit large changes in design such as MMA,[5] global stability issues can be overcome. Since the MMA employs adaptive move limits, design changes are restrained which restricts the associated changes in adjoint $\delta\boldsymbol{\Lambda}_{k}^{j}$. We will illustrate this fact using the examples in Section 5 and particularly the example in Figure 10B where we will find global stability issues which resolve themselves after additional design iterations. Furthermore, in this section, we only discuss the characteristic time in broad terms, and do not give strict criteria for the adjoint equations to be stable leading to consistent sensitivities. However, in Section 5, we examine the stability requirements using practical examples. In Sections 5.1.1 and 5.2.1 we study the characteristic times in more detail for a thermal and a flow problem. Subsequently, in Section 5.1.4 we will find a weak global convergence criterion which is verified in Section 5.2.4.

3.2 | The hybrid Checkpointing/Local-in-Time algorithm

As the LT algorithm is faster than the CP algorithm but contains errors due to the adjoint approximations, we would like to combine these two algorithms to attain a more accurate and computationally efficient algorithm. We call this novel algorithm the hybrid Checkpointing/Local-in-Time (CP/LT) algorithm. In principle, the CP/LT algorithm is based on the LT algorithm. However, when errors in the approximate adjoints become too large, corrections are performed using the CP algorithm. In Figure 4, we illustrate that the CP/LT algorithm is based on the LT algorithms in phase A, and adaptively performs corrections in phase B.

To measure the accuracy of the adjoint approximation in the LT algorithm in phase A, an approximate adjoint error in design iteration j at checkpoint k is defined as:

$$\varepsilon_{\lambda_k}^{j} = \frac{\|\widetilde{\boldsymbol{\Lambda}}_{k}^{j+1} - \widetilde{\boldsymbol{\Lambda}}_{k}^{j}\|_2}{\|\widetilde{\boldsymbol{\Lambda}}_{k}^{j+1}\|_2} = \frac{\|\Delta\widetilde{\boldsymbol{\Lambda}}_{k}^{j}\|_2}{\|\widetilde{\boldsymbol{\Lambda}}_{k}^{j+1}\|_2}, \tag{19}$$

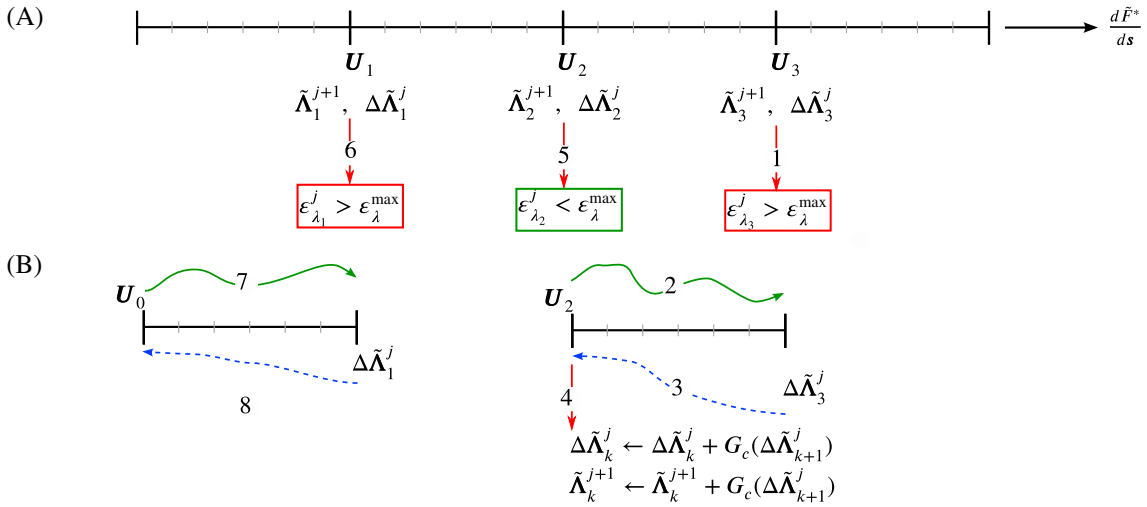


FIGURE 4 A schematic of the CP/LT algorithm. The numbers represent the order of operations. First the LT algorithm is executed in phase A resulting in approximate sensitivities $d\tilde{F}^*/d\boldsymbol{s}$. Additionally, $\boldsymbol{U}_k$ and $\Delta\tilde{\boldsymbol{\Lambda}}_k^j$ are stored on the checkpoints besides $\tilde{\boldsymbol{\Lambda}}_k^{j+1}$. In phase B, starting at the final to last interval and moving one interval backward at a time, the adjoint error is evaluated and a correction on the interval is performed if necessary. The dashed blue arrows represent the solving of the adjoint correction using Equation (20) and update of sensitivities using Equation (22). After propagating the adjoint correction backward from $k+1$ to k, the adjoint correction and adjoint approximation are both updated.

where we denote the L^2-norm as $\|\square\|_2$. Moreover, the correction in adjoint at checkpoint k is approximated as $\Delta\tilde{\boldsymbol{\Lambda}}_k^j = \tilde{\boldsymbol{\Lambda}}_k^{j+1} - \tilde{\boldsymbol{\Lambda}}_k^j \approx \boldsymbol{\Lambda}_k^j - \tilde{\boldsymbol{\Lambda}}_k^j = \Delta\boldsymbol{\Lambda}_k^j$, where we assume $\tilde{\boldsymbol{\Lambda}}_k^{j+1} \approx \boldsymbol{\Lambda}_k^j$ due to local convergence. Because the sensitivity in Equation 7 is dependent on the adjoint, the error is used as a measure of accuracy of the sensitivity. If the error $\varepsilon^j_{\lambda_k}$ is higher than a user-defined threshold $\varepsilon^{\max}_\lambda$, the adjoint solution and sensitivity update on domain Θ_{k-1} are deemed inaccurate and need to be corrected. It should be noted that the error $\varepsilon^j_{\lambda_k}$ is dependent on $\tilde{\boldsymbol{\Lambda}}_k^{j+1} = G(\tilde{\boldsymbol{\Lambda}}_{k+1}^j)$ which is only known after evaluating the adjoint solution on domain Θ_k in the LT algorithm. As domain Θ_k needs to be evaluated before considering correcting domain Θ_{k-1}, we do not have the state solution on domain Θ_{k-1} in memory to perform a correction. The state solution thus needs to be recomputed on domain Θ_{k-1} starting from a stored state $\boldsymbol{U}^j_{k-1}$.

Making use of the linearity of both the adjoint and sensitivity equations, the sensitivity correction may be simplified to reduce additional computational effort. In Section 3.1.1, we found that the adjoint correction can be propagated backward over an interval using $\Delta\boldsymbol{\Lambda}_k = G_c(\Delta\boldsymbol{\Lambda}_{k+1})$ as defined in Equations (12) and (13). This propagation can also be performed for approximate $\Delta\tilde{\boldsymbol{\lambda}}_n^j$, which is propagated over one time step as:

$$\Delta\tilde{\boldsymbol{\lambda}}_n^j = -\left(\frac{\partial R_n}{\partial \boldsymbol{u}_n^j}\right)^{-\intercal} \frac{\partial R_{n+1}}{\partial \boldsymbol{u}_n^j}^{\intercal} \Delta\tilde{\boldsymbol{\lambda}}_{n+1}^j. \tag{20}$$

Moreover, a correction for the sensitivities is defined by substituting the improved approximation $\boldsymbol{\lambda}_n^j \approx \tilde{\boldsymbol{\lambda}}_n^j + \Delta\tilde{\boldsymbol{\lambda}}_n^j$ into the exact sensitivity formulation (Equation 7, for clarity repeated in Equation 21a) resulting in Equation (21b):

$$\frac{dF}{d\boldsymbol{s}} = \sum_{n=0}^{N}\left(\frac{\partial F_n}{\partial \boldsymbol{s}}\Delta t + \boldsymbol{\lambda}_n^{\intercal}\frac{\partial R_n}{\partial \boldsymbol{s}}\right), \tag{21a}$$

$$\frac{dF}{d\boldsymbol{s}} = \sum_{n=0}^{N}\left(\frac{\partial F_n}{\partial \boldsymbol{s}}\Delta t + \tilde{\boldsymbol{\lambda}}_n^{j^{\intercal}}\frac{\partial R_n}{\partial \boldsymbol{s}} + \Delta\tilde{\boldsymbol{\lambda}}_n^{j^{\intercal}}\frac{\partial R_n}{\partial \boldsymbol{s}}\right), \tag{21b}$$

for which the first two terms have already been computed in the LT algorithm and a correction for the sensitivities can be build as:

$$\Delta\frac{dF}{d\boldsymbol{s}} = \sum_{n=0}^{N}\Delta\tilde{\boldsymbol{\lambda}}_n^{j^{\intercal}}\frac{\partial R_n}{\partial \boldsymbol{s}}. \tag{22}$$

Consequently, when adjoint errors for interval Θ_{k-1} are too high ($\varepsilon^j_{\lambda_k} > \varepsilon^{\max}_\lambda$), we propagate the approximate adjoint correction backward using Equations (20) while simultaneously updating the sensitivities using Equation (22).

Adjoint correction $\Delta\widetilde{\boldsymbol{\lambda}}^j_n$ is used to correct the sensitivities as it is an easier operation than updating the sensitivities using a new and improved adjoint field. Generally, we immediately write contributions to the sensitivity vector $dF/d\boldsymbol{s}$ to memory. To update the sensitivity using an improved adjoint field, we would have to remove the contributions added using the old adjoint field first. Using the adjoint correction we can update the sensitivity vector when necessary. Moreover, we found that updates for the adjoint correction using Equation (20) are computationally cheaper than updates in the full adjoint using Equation (8). Inspecting these equations, we find that propagating the adjoint involves a matrix vector multiplication, a vector addition, and a linear system solve, while propagating the adjoint correction only requires the matrix vector multiplication and the linear system solve. Nonetheless, the computational cost of the full process is increased compared to the LT algorithm as we need to recompute the full state solution on a domain which is being corrected to compute the sensitivity corrections in Equation (22). Furthermore, we note that after propagating the adjoint correction backward over an interval, the propagated correction may be used to improve the approximate adjoint at the checkpoint, that is, $\widetilde{\boldsymbol{\Lambda}}^j_k = G(\widetilde{\boldsymbol{\Lambda}}^j_{k+1}) + G_c(\Delta\widetilde{\boldsymbol{\Lambda}}^j_{k+1})$. Finally, whether or not to apply this correction on an interval depends on the chosen adjoint error threshold $\varepsilon^{\max}_\lambda$. Based on the experience of the authors, a relatively high threshold of $\varepsilon^{\max}_\lambda = 0.1$ can be chosen. Using this threshold we will find errors in sensitivity to remain below 10%.

Based on these developments, the CP/LT algorithm as shown in Figure 4 consists of two phases, and can be described as follows.

(A) **LT algorithm**: The sensitivities are computed using the LT algorithm. However, besides the approximate adjoint variables $\widetilde{\boldsymbol{\Lambda}}^{j+1}_k$, the adjoint correction $\Delta\widetilde{\boldsymbol{\Lambda}}^j_k$, and state solution $\boldsymbol{U}_k$ are also stored at the checkpoints as shown in Figure 4.

(B) **Checkpointing corrections**: Starting at the second to last interval $\Theta_k = \Theta_{K-2}$ using the stored $\widetilde{\boldsymbol{\Lambda}}^{j+1}_{k+1}$ and $\Delta\widetilde{\boldsymbol{\Lambda}}^j_{k+1}$, the error $\varepsilon^j_{\lambda_{k+1}}$ is computed. If the error is larger than the fixed threshold $\varepsilon^{\max}_\lambda$, the state solution is recomputed and correction $\Delta\widetilde{\boldsymbol{\Lambda}}^j_{k+1}$ is propagated backwards over the interval using Equation 20, while simultaneously updating the sensitivity using Equation (22). The backward propagation of the correction from $k+1$ to k is used to update the correction and adjoint variables at k as $\Delta\widetilde{\boldsymbol{\Lambda}}^j_k \leftarrow \Delta\widetilde{\boldsymbol{\Lambda}}^j_k + G_c(\Delta\widetilde{\boldsymbol{\Lambda}}^j_{k+1})$ and $\widetilde{\boldsymbol{\Lambda}}^{j+1}_k \leftarrow \widetilde{\boldsymbol{\Lambda}}^{j+1}_k + G_c(\Delta\widetilde{\boldsymbol{\Lambda}}^j_{k+1})$. Subsequently, a sensitivity correction is performed at interval Θ_{k-1} if necessary, and so on.

The CP/LT algorithm requires more memory than the LT and CP algorithms since the state solution $\boldsymbol{U}_k$, the adjoint field $\widetilde{\boldsymbol{\Lambda}}^{j+1}_{k+1}$, and the adjoint correction $\Delta\widetilde{\boldsymbol{\Lambda}}^{j+1}_{k+1}$ are stored on the checkpoints. On each interval except the first, the initial states are stored $m(K-1)$ discrete state variables, and on each interval except the last, the terminal adjoint and the terminal adjoint correction are stored $2m(K-1)$ discrete adjoint variables, resulting in the storage of $3m(K-1)$ discrete variables. Combining this with the storage of the state solution on an interval of mN/K discrete state variables, the CP/LT algorithm has a memory requirement which scales as $M \propto 3m(K-1) + mN/K$.

The computational cost of the CP/LT algorithm is not known a priori but a lower and upper bound can be derived. If the adjoint errors are negligible at all checkpoints and the sensitivities are thus reliable, no corrections are performed and the CP/LT algorithm performs as the LT algorithm in phase A. The lower bound on the computational cost is thus the cost of the LT algorithm, that is, $\underline{C} \propto Nc_s(1+r_c)$. However, if the adjoint errors are large at all checkpoints, a correction is performed at the first $K-1$ intervals which includes the solution of the state equations at a cost proportional to c_sN/K. Moreover, on each corrected domain, an adjoint correction is performed for which we estimate the computational cost as similar to the cost of the normal adjoint propagation c_sr_cN/K, although in practice the adjoint correction is a cheaper operation. The cost of the adjoint corrections is thus proportional to $(K-1)(c_sN/K + c_sr_cN/K) = Nc_s(1+r_c)(1-1/K)$, and an upper bound of the computational cost is derived as $\overline{C} \propto \underline{C} + Nc_s(1+r_c)(1-1/K) = Nc_s(1+r_c)(2-1/K)$.

3.3 | The Parallel-Local-in-Time algorithm

Parallelization is a useful method to speed up computations. To parallelize the computations, spatial domain decomposition is often used. However, this technique may suffer from communication overhead, limiting the maximum speedup. To efficiently reduce such communication, a novel Parallel-Local-in-Time (PLT) algorithm is proposed in this work. The PLT algorithm is an extension of the LT algorithm and is similar to the direct multiple shooting algorithm.[21] For the temporal parallelization, we discretize the temporal domain into intervals Θ_k. These intervals are however decoupled and

we will solve iteratively for both the approximate state and adjoint solutions. Beside the approximation of the adjoint variables as in the LT algorithm we approximate and update the initial state variables on an interval as:

$$\boldsymbol{U}_k^j \approx \widetilde{\boldsymbol{U}}_k^j = U\left(\widetilde{\boldsymbol{U}}_{k-1}^{j-1}\right), \tag{23}$$

where $U\left(\widetilde{\boldsymbol{U}}_{k-1}^{j-1}\right)$ is an operator which propagates initial state $\widetilde{\boldsymbol{U}}_{k-1}^{j-1}$ over interval Θ_{k-1} in design iteration $j-1$ to terminal state $\widetilde{\boldsymbol{U}}_k^{j-1}$. Terminal state solutions on interval $k-1$ at design iteration $j-1$ are thus used as initial states for interval Θ_k at design iteration j. Starting from the approximate initial state, the state equations can be solved on each interval in parallel. Subsequently, using approximate terminal adjoints $\widetilde{\boldsymbol{\Lambda}}_{k+1}^j$, the complete sensitivity computation can be done in parallel for each interval Θ_k as illustrated in Figure 5. We extract the sensitivity contribution for interval Θ_k from Equation (7) as all contributions from time step $n = n_k + 1$ at initial time $t_{n_k} = \theta_k$ until time step $n = n_{k+1}$ at terminal time $t_{n_{k+1}} = \theta_{k+1}$:

$$\frac{dF_k^*}{d\boldsymbol{s}} = \sum_{n=n_k+1}^{n_{k+1}} \left(\widetilde{\boldsymbol{\lambda}}_n^{j\intercal} \frac{\partial R_n(\widetilde{\boldsymbol{u}}_{n-1}^j, \widetilde{\boldsymbol{u}}_n^j, \boldsymbol{s}, t_n)}{\partial \boldsymbol{s}} + \frac{\partial F_n(\widetilde{\boldsymbol{u}}_n^j, \boldsymbol{s})}{\partial \boldsymbol{s}} \Delta t \right), \tag{24}$$

where we approximated all state variables as $\widetilde{\boldsymbol{u}}_n^j \approx \boldsymbol{u}_n^j$ and used approximate adjoint variables $\widetilde{\boldsymbol{\lambda}}_n^j$. To initialize the PLT algorithm, approximate adjoints and an acceptable guess for the state variables are required at the checkpoints. In the first design iteration, we compute the sensitivities using the LT algorithm and store the resulting state and adjoint variables $\widetilde{\boldsymbol{U}}_k^{j=0}/\widetilde{\boldsymbol{\Lambda}}_k^{j=0}$ at the checkpoints θ_k. Subsequently, the algorithm consists of the following three steps:

1. **Send information to computational nodes**: To initialize the design evaluation, we assume there are as much computational nodes as there are intervals K, though this is not essential to the algorithm. Furthermore, we associate interval Θ_k to node k and send it an approximate initial state condition $\tilde{\boldsymbol{U}}_k^j$ (exact state solution $\boldsymbol{u}_0^j$ for the first interval), an approximate terminal adjoint condition $\widetilde{\boldsymbol{\Lambda}}_{k+1}^j$ (exact adjoint solution $\boldsymbol{\lambda}_N^j$ for the final interval), and the design $\boldsymbol{s}^j$.
2. **Design evaluation**: On each of the intervals Θ_k, the design is evaluated. Starting at approximate $\tilde{\boldsymbol{U}}_k^j$, the state solution is propagated forward to $\tilde{\boldsymbol{U}}_{k+1}^{j+1}$ by solving the state equations. Moreover, the state solution is stored for each time step within the interval. Subsequently, adjoint variable $\widetilde{\boldsymbol{\Lambda}}_{k+1}^j$ is propagated backward to $\widetilde{\boldsymbol{\Lambda}}_k^j$ using Equations 8 and 9 and used to update the interval sensitivities using Equation (24).

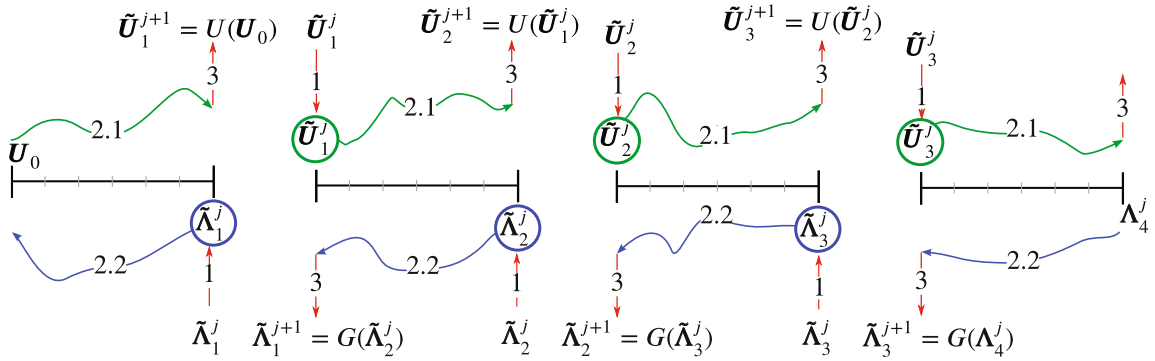


FIGURE 5 A schematic of the PLT algorithm. The numbers represent the order of operations. In the first step, we distribute all required state and adjoint fields to the intervals. In the second step, for each interval Θ_k, we evaluate and store the state solution starting from the distributed $\widetilde{\boldsymbol{U}}_k^j$, and subsequently we evaluate the adjoint solution starting from distributed $\widetilde{\boldsymbol{\Lambda}}_{k+1}^j$ while simultaneously computing domain sensitivity $dF_k^*/d\boldsymbol{s}$ as defined in Equation (24). Solving the state solutions and the adjoint solutions are labeled steps 2.1 and 2.2, respectively. This does not imply that all intervals wait until all state solutions are solved. Some intervals might in fact be working on the adjoint solutions while others are working on the state solution but in general all intervals will be working on the state or adjoint solution at the same time. Finally, in the third step, all state and adjoint updates are gathered in addition to the sensitivity contributions $dF_k^*/d\boldsymbol{s}$ which are combined into the total sensitivity using Equation (25).

3. **Receive information from computational nodes**: From each computational node k, we receive the terminal state solution $\tilde{\boldsymbol{U}}^j_{k+1}$, the initial adjoint solution $\tilde{\boldsymbol{\Lambda}}^j_k$, and domain sensitivities $dF^*_k/d\boldsymbol{s}$. Subsequently, the global sensitivity is computed by gathering the contributions of all intervals:

$$\frac{dF^*}{d\boldsymbol{s}} = \sum_{k=1}^{K} \frac{dF^*_k}{d\boldsymbol{s}}. \tag{25}$$

To improve approximate state and adjoint solutions, we update them using the propagated solutions on the neighboring intervals as:

$$\begin{aligned} \boldsymbol{U}^{j+1}_{k+1} &\approx U(\tilde{\boldsymbol{U}}^j_k), \\ \boldsymbol{\Lambda}^{j+1}_k &\approx G(\tilde{\boldsymbol{\Lambda}}^j_{k+1}), \end{aligned} \tag{26}$$

where we assume that $\boldsymbol{U}^{j+1}_{k+1} \approx U(\tilde{\boldsymbol{U}}^j_k)$ decreases the error in state, similar to the decreased error in adjoint by $\boldsymbol{\Lambda}^{j+1}_k \approx G(\tilde{\boldsymbol{\Lambda}}^j_{k+1})$ as discussed in Section 3.1.1.

As was the case in the LT algorithm, we use approximate adjoint variables and therefore expect some inaccuracies due to local and global convergence issues in the PLT algorithm. Moreover, in the PLT algorithm we have also introduced approximate state solutions which may cause additional inaccuracies. However, following a similar reasoning as in Section 3.1.1 for the stability of the approximate adjoints, the errors in approximate state solutions are expected to reduce over an interval if local convergence is satisfied. If we describe the state solution as $\boldsymbol{U}_n = \tilde{\boldsymbol{U}}_n + \Delta\boldsymbol{U}_n$, where $\Delta\boldsymbol{U}_n$ is the error in state solution, we expect the error to behave the same as a disturbance of an initial state solution. For stable transient systems, a disturbance of the initial state solution is known to dampen out over time. These disturbances can generally be said to dampen out on a similar timescale as it takes the system to reach a stable steady-state solution. To enforce local stability, we thus require relatively long intervals with respect to the characteristic time of the system. Consequently, as discussed for the LT algorithm in Section 3.1.1, we expect weak global convergence issues. The PLT algorithm is expected to favor convergence to local optima with features of shorter characteristic times as for these features accurate adjoints and states, and thus sensitivities, are found much quicker. Additionally, this bias may be enhanced by the PLT algorithm due to the approximate state solutions $\tilde{\boldsymbol{U}}_k$. Strong global convergence issues might also pose a problem for the PLT algorithm. However, in the authors experience the algorithm behaves similar to the LT algorithm. By enforcing local stability and using optimization methods which limit large changes in design, the PLT algorithm is found to avoid strong global convergence issues, and converges to stable optima with accurate sensitivities as will be shown in Section 5.

For the PLT algorithm to converge, we thus require errors in state solution to dampen out over successive design iterations. This may not always be the case. If the physics show chaotic behavior, small differences in initial state may cause large differences in later states. It is clear that for these types of systems, errors in state solution do not dampen out over successive design iterations. Moreover, under certain conditions the adjoint solutions of chaotic problems may grow exponentially, and the adjoint sensitivities may be inaccurate.[31,32] For chaotic systems even the CP and GT algorithms which compute exact sensitivities may also be inaccurate. Algorithms presented in this paper should therefore not be applied to chaotic systems without a careful consideration.

Since the PLT algorithm performs all computations simultaneously, the full state solution at all intervals (and thus the whole temporal domain) needs to be stored. Memory requirements thus scale as $M \propto Nm$. However, an increase in computational nodes often comes with an increase in memory alleviating this problem. When memory becomes a limiting factor, the parallel intervals can be subdivided once more into local subintervals where the LT or CP algorithm is employed. On these subintervals, we thus use different approximations for initial state and terminal adjoint equations. Another solution is to divide the domain into shorter intervals and evaluate some intervals sequentially instead of in parallel, but still use approximate initial state and terminal adjoint solutions on each of these shorter intervals. Doing so, the full state solution does not need to be stored but only the state solutions on the intervals being evaluated.

As all state and adjoint solutions are solved once per interval, the computational cost of the PLT algorithm is the same as for GT algorithm, $C \propto Nc_s(1 + r_c)$. Due to the parallelization, the computational time may be reduced. We investigate a theoretical two-dimensional (2D) optimization problem for illustration purpose, though a similar investigation holds for three-dimensional (3D) problems. Solving a transient 2D problem we might conceptualize as solving the discrete solution

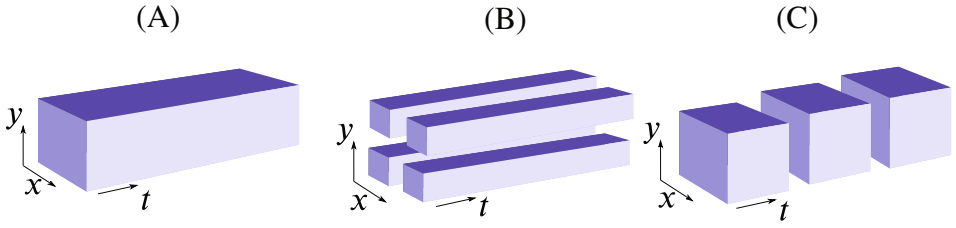


FIGURE 6 An example of the two options for paralellizing a two-dimensional (2D) transient computation, where A is the nonparallelized domain, B is a parallelization via spatial domain decomposition, and C is a parallelization via temporal domain decomposition.

on a 3D cuboid where two of its axis represent the spatial axis and one represents the temporal axis as can be seen in Figure 6. Implementing a spatial domain decomposition, we cut the domain along its length and generate interface area across which the nodes have to communicate. If a time stepping algorithm is used this communication happens every time step leading to increased idle time. Moreover, the communication overhead might become even worse when nonlinear systems are solved as they may require many subsolves per time step. If a temporal parallelization can be used, the cuboid can be cut across its width generating less interface area and less communication overhead. We note that for this analysis we used a problem with more time steps than elements in the x or y direction. A problem with more elements in x or y direction than time steps may have less interface area and be more efficiently parallelized when using spatial domain decomposition techniques.

4 | MEMORY AND COMPUTATIONAL COST

All presented sensitivity computation algorithms have their advantages and limitations and are suited for different kind of optimization problems. Based on a combination of memory requirements, computational cost, and convergence, different algorithms are recommended for different problems.

4.1 | Memory requirements

When memory requirements are not a limiting factor, the full state solution may be stored and the GT or PLT algorithms can be used. Moreover, if the parallel algorithms are preferred for their speedup but memory is a limiting factor, a hierarchical structure of the algorithms may be used. In this paper, the CP, LT, and CP/LT algorithms were used to reduce memory requirements as summarized in Table 1. However, for local stability, we require the intervals to be as long as possible and we thus use the least amount of intervals K as possible. Subsequently, it can be argued that for optimization problems where $N \gg K$ the differences in memory reduction between the algorithms are negligible, as the largest memory burden is spent on storing the complete state solution on an interval and not on storing the additional information at the checkpoints.

Under this assumption all sequential algorithms thus perform similarly with respect to memory reduction. However, if a small number of time steps is used and memory limits are reached due to m, the size of the solution at one time step, these assumptions do not hold anymore and the CP or LT algorithms should be used for the largest memory reduction.

TABLE 1 Memory requirements M of the sequential algorithms for a problem containing K intervals, N time steps, and adjoint/state fields of size m.

	GT, PLT	CP	LT	CP/LT
Memory requirements	mN	$m\left(\frac{N}{K}+K-1\right)$	$m\left(\frac{N}{K}+K-1\right)$	$m\left(\frac{N}{K}+3(K-1)\right)$
$K \ll N$	mN	$m\frac{N}{K}$	$m\frac{N}{K}$	$m\frac{N}{K}$

Note: For problems where $K \ll N$, memory requirements are identical for all algorithms.
Abbreviations: CP, Checkpointing; GT, Global-in-Time; LT, Local-in-Time; PLT, Parallel-Local-in-Time; CP/LT, hybrid Checkpointing/Local-in-Time.

4.2 | Computational cost

To make a comparison in terms of computational cost, an overview is given in Table 2. The CP/LT algorithm does not have a fixed computational cost but rather a range of possible costs with a lower bound at the cost of the GT, PLT, and LT algorithms and an upper bound above all other algorithms. However, when the state equations are highly nonlinear and $r_c \to 0$, we find the upper bound of the cost of the CP/LT algorithm in Table 2 to be the same as the cost of the CP algorithm. Another simplification can be made for linear implicit state equations. When the state equations are linear, adjoint equations are linear and have a similar computational cost. The resulting *implicit* adjoint equations involve a relatively expensive inverse matrix problem, while the sensitivity update only involves cheap matrix/vector multiplications. The computational cost c_a consisting of the adjoint and sensitivity update can thus be assumed to be dominated by the adjoint update which is similar to the state update: $c_a = c_s = c$ and $r_c = 1$. This particular case is shown in the bottom row of Table 2. However, when the state and consequently adjoint equations are linear *explicit*, both the adjoint and state equations consist only of matrix vector multiplications. In this case adjoint, state, and sensitivity updates have a similar computational cost. The cost of solving adjoint and updating sensitivity c_a will thus be larger than c_s and $r_c > 1$. This case is further investigated in Section 5.1.5.

Selecting an appropriate algorithm for a problem may depend on the computational cost and we further analyze the cost of the CP and CP/LT algorithms. Firstly, we examine the relative computational cost of the CP and GT algorithms by dividing the computational cost of the CP algorithm C^{CP} by the cost of the GT algorithm C^{GT}:

$$\frac{C^{\mathrm{CP}}}{C^{\mathrm{GT}}} = \frac{c_s N(2 + r_c - 1/K)}{c_s N(1 + r_c)} = \frac{2 + r_c - 1/K}{1 + r_c}. \tag{27}$$

As expected, we find the computational cost of the GT and CP algorithms with $K = 1$ to be equal. Moreover, smaller K make the ratio drop and have a diminishing effect on the increased computational cost of the CP algorithm. We thus advise for the CP algorithm to keep K as low as possible. For $K \gg 1$, the relative computational cost only depends on r_c. The biggest increase in computational cost is found for nonlinear systems where $r_c \to 0$ and $C^{\mathrm{CP}}/C^{\mathrm{GT}} \to 2$. Therefore, we advise to disregard the CP algorithm for nonlinear systems based on the required computational cost.

As corrections, even if only a few, are always performed for the CP/LT algorithm, not only the CP but also the CP/LT algorithm turns out to be more expensive than the GT, LT, and PLT algorithms. An informed choice between the CP/LT and CP algorithms thus depends on the number of corrections required by the CP/LT algorithm and the resulting relative computational cost. Comparing computational cost is not straightforward as the cost of the CP/LT algorithm is undefined *a priori*. We adaptively correct intervals only when errors are high and do not correct all intervals, as shown in Figure 4. Relative computational cost thus depends on the fraction of corrected intervals, and the relative cost of CP/LT and CP intervals. The computational cost of the terminal interval Θ_{K-1} is the same for both the CP/LT and CP algorithms as the state and adjoint equations are evaluated only once on this interval. On the other $K - 1$ intervals which contain N/K time steps each, computational costs differ. In the CP algorithm, the state equations on these intervals are solved twice and the adjoint equations once, at a cost $\propto c_s(N/K)(2 + r_c)$ per interval. On a corrected CP/LT interval, the state equations are solved twice and both the adjoint and adjoint correction are solved once at a cost $\propto c_s(N/K)(2 + 2r_c)$, and on an uncorrected interval state and adjoint are solved only

TABLE 2 Computational costs C of the sequential algorithms for problems containing K intervals and N time steps.

	GT, LT, PLT	CP/LT	CP
C	$c_s N(1 + r_c)$	$\left[c_s N(1 + r_c), c_s N(1 + r_c)(2 - 1/K)\right]$	$c_s N(2 + r_c - 1/K)$
$C, r_c \to 0$	$c_s N$	$\left[c_s N, c_s N(2 - 1/K)\right]$	$c_s N(2 - 1/K)$
$C, r_c = 1$	$2cN$	$[2cN, cN(4 - 2/K)]$	$cN(3 - 1/K)$

Notes: Solving one state step has a computational cost of c_s while solving one adjoint step while simultaneously updating the sensitivities has a cost of c_a, relative cost is defined as $r_c = c_a/c_s$. Contrary to the GT, PLT, LT, and CP algorithms, the CP/LT algorithm has a lower and upper bound on its computational cost. The lower bound is equal to the cost of the GT, PLT, and LT algorithms while the upper bound is higher than the cost of the CP algorithm as in the CP/LT algorithm state and adjoint solutions may be recomputed whereas in the CP algorithm only state solutions are recomputed. A simplification is shown for problems with highly nonlinear state equations where $r_c \to 0$ and for problems with linear implicit state equations where $r_c = 1$ and $c_a = c_s = c$.
Abbreviations: CP, Checkpointing; GT, Global-in-Time; LT, Local-in-Time; PLT, Parallel-Local-in-Time; CP/LT, hybrid Checkpointing/Local-in-Time.

once at a cost $\propto c_s(N/K)(1+r_c)$. A corrected and uncorrected interval thus have a cost relative to a CP interval, respectively, as:

$$\frac{c_s(N/K)(2+2r_c)}{c_s(N/K)(2+r_c)} = 2\frac{1+r_c}{2+r_c}, \qquad \frac{c_s(N/K)(1+r_c)}{c_s(N/K)(2+r_c)} = \frac{1+r_c}{2+r_c}. \tag{28}$$

To compare the overall computational cost, we define the fraction of corrected intervals f_c. Since the terminal interval has the same computational cost for both CP and CP/LT, the fraction only considered the first $K-1$ intervals. Subsequently, f_c is defined as the number of corrected intervals in the whole optimization process, divided by the product of $K-1$ and the number of design iterations. It is thus the fraction of corrected intervals, averaged over all design iterations. The relative computational cost of the CP/LT algorithm to the CP algorithm can be computed as:

$$\frac{C^{CP/LT}}{C^{\mathrm{CP}}} = \frac{K-1}{K}\left(f_c 2\frac{1+r_c}{2+r_c} + (1-f_c)\frac{1+r_c}{2+r_c}\right) + \frac{1}{K} = \frac{K-1}{K}(f_c+1)\frac{1+r_c}{2+r_c} + \frac{1}{K}, \tag{29}$$

where for the first $K-1$ intervals, we have differing computational cost at $(K-1)/K\%$ of the computations, and at the last interval we have the same computational cost at $1/K\%$ of the computations.

In practice we need to approximate f_c to choose whether the CP/LT or the CP algorithm is cheaper. However, in Sections 5.1.5 and 5.2.3 we find that corrections are only performed during the first part of the optimization process where large design changes are present. Experienced designers can thus use their knowledge of the convergence behavior of the problem to estimate f_c and select the computationally most advantageous algorithm, as further discussed in the guidelines in Section 6.

5 | RESULTS

In this section we investigate and compare the GT, CP, LT, CP/LT, and PLT algorithms. We investigate stability by optimizing a transient thermal problem in Section 5.1 and computational cost by optimizing a flow problem in Section 5.2. The thermal problem will minimize the temperature in heat generating components and the flow problem will optimize a piston pump. Both problems are optimized using density-based topology optimization.[33] In density-based topology optimization, we aim to find an optimal material distribution in a given design domain. The design domain is divided into grid cells to which design variables are attached. Subsequently, the design variables are used to interpolate continuously between phases, solid and void for the thermal problem, solid and fluid for the flow problem. As we interpolate continuously between phases, the gradient-based MMA[5] algorithm can be used. To compute the gradients in our transient thermal and flow problems, the algorithms presented in this paper will be used.

For the gradient computation, the adjoint equations need to be solved. To solve the adjoint equations, Jacobians $\partial \boldsymbol{R}_n/\partial \boldsymbol{u}_n$ and $\partial \boldsymbol{R}_n/\partial \boldsymbol{u}_{n-1}$ need to be constructed. We assume that $\boldsymbol{u}_n$ either contains discrete temperatures or velocities and pressures. The Jacobians will be constructed using the same approach as in Theulings et al.[34] For completeness, it is summarized here. The finite volume method will be used to discretize the PDE equations into a column $\boldsymbol{R}_n$. Each i^{th} component R_n^i of the column may be associated with small subsets $\boldsymbol{u}_n^i$ and $\boldsymbol{u}_{n-1}^i$ of the complete sets of DOFs $\boldsymbol{u}_n$ and $\boldsymbol{u}_{n-1}$. To construct the Jacobians we use the MATLAB[35] symbolic toolbox and construct symbolic equations for $R_n^i(\boldsymbol{u}_n^i, \boldsymbol{u}_{n-1}^i)$ in terms of the *symbolic* variables in $\boldsymbol{u}_n^i$ and $\boldsymbol{u}_{n-1}^i$. Subsequently, we use symbolic differentiation to compute $\partial R_n^i/\partial \boldsymbol{u}_n^i$, $\partial R_n^i/\partial \boldsymbol{u}_{n-1}^i$ and use MATLAB to automatically construct a function which takes DOFs $\boldsymbol{u}_n^i$, $\boldsymbol{u}_{n-1}^i$ and returns $\partial R_n^i/\partial \boldsymbol{u}_n^i$, $\partial R_n^i/\partial \boldsymbol{u}_{n-1}^i$. This function is used to compute all derivatives after which they are assembled into $\partial \boldsymbol{R}_n/\partial \boldsymbol{u}_n$ and $\partial \boldsymbol{R}_n/\partial \boldsymbol{u}_{n-1}$. Moreover, derivatives $\partial \boldsymbol{R}_n/\partial \boldsymbol{s}$ are computed following a similar approach.

5.1 | Stability investigation through thermal optimization

In this section we investigate the stability of the approximate LT, PLT, and CP/LT algorithms by optimizing a thermal problem. Weak global convergence is investigated by examining characteristic times and comparing it against the interval lengths. As discussed in Section 3.1.1, we expect the optimizer to be biased toward features with short characteristic times. Specifically, we expect a stronger bias for features with a long characteristic time relative to interval length $\Delta\theta$.

Additionally, we examine the predicted and measured computational cost of the algorithms for problems with linear explicit state equations. The thermal problem is chosen for its simplicity as we can estimate relatively easily how fast errors in approximate adjoint decay and thus how fast we converge to accurate solutions.

5.1.1 | Transient thermal model

Before describing the optimization problem, we introduce the thermal physic and numerical model. The characteristic timescales related to the physics and discretization are investigated as they play an important role in the stability of the algorithms and set up of the optimization problem. First, we discretize the transient equations in space and time. A two-dimensional transient thermal problem is considered and is defined on temporal domain $t \in [0, t_t]$ and spatial domain $\vec{x} \in \Omega$ with boundary $\Gamma = \Gamma_d \cup \Gamma_n$. The thermal problem is governed by:

$$\begin{aligned} \alpha_s \rho_s c_{ps} \dot{T} - \nabla \cdot (\alpha_s \kappa_s \nabla T) - \alpha_s Q &= 0, && \text{on } \Omega, \\ T &= T_\Gamma, && \text{on } \Gamma_d, \\ \alpha_s \kappa_s \nabla T \cdot \boldsymbol{n} &= \boldsymbol{q}_T \cdot \boldsymbol{n}, && \text{on } \Gamma_n, \\ T &= \hat{T}, && \text{on } \Omega \text{ at } t = 0, \end{aligned} \tag{30}$$

where the subscript $\square_s$ denotes solid material properties, $T(\vec{x}, t)$ is the temperature field with time derivative $\dot{T} = \partial T / \partial t$, the solid density, specific heat capacity, and thermal conductivity are ρ_s, c_{ps}, and κ_s, respectively, $Q(\vec{x}, t)$ is an externally applied heat load, $T_\Gamma(\vec{x}, t)$ is the fixed temperature on boundary Γ_d, $\boldsymbol{q}_T(\vec{x}, t)$ is an applied heat flux on boundary Γ_n with outward normal $\boldsymbol{n}$, and $\hat{T}(\vec{x})$ is the initial temperature distribution. The solid volume fraction $\alpha_s(\vec{x})$ is used as design variable. The solid domain is defined by $\alpha_s(\vec{x}) = 1$ and the void domain by $\alpha_s(\vec{x}) = \underline{\alpha}_s$ which is a lower bound on the volume fraction for which $0 < \underline{\alpha}_s \ll 1$. The transient thermal equations are discretized on a Cartesian mesh illustrated in Figure 7 using the finite volume techniques as described by Versteeg and Malalasekera.[28] Specifically, we use the same techniques as used by Gersborg-Hansen et al.,[36] where the arithmetic average is used to interpolate discrete design variables at the interface between mesh elements. Moreover, we discretize in time using a forward Euler scheme such that we find discretized equations of the form:

$$\begin{aligned} R_n(\boldsymbol{T}_{n-1}, \boldsymbol{T}_n, \boldsymbol{s}, t_n) &= \boldsymbol{M}(\boldsymbol{s}) \frac{\boldsymbol{T}_n - \boldsymbol{T}_{n-1}}{\Delta t} - \boldsymbol{K}(\boldsymbol{s}) \boldsymbol{T}_{n-1} - \boldsymbol{Q}_n(\boldsymbol{s}), \\ R_0(\boldsymbol{T}_0) &= \boldsymbol{T}_0 - \hat{\boldsymbol{T}}, \end{aligned} \tag{31}$$

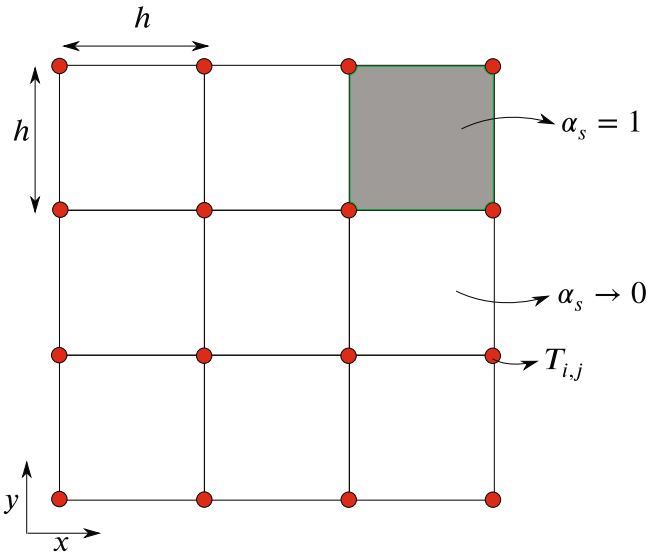


FIGURE 7 The equidistant uniform mesh used to discretize the thermal problem. Red dots are the temperature nodes with degrees of freedom $T_{i,j}$ and densities are defined per element where $\alpha_s = 1$ is a solid element and $\alpha_s \to 0$ a void element.

where $\boldsymbol{T}_n$ and $\boldsymbol{Q}_n$ are the vectors of discrete nodal temperatures and heat loads at time t_n, $\hat{\boldsymbol{T}}$ is the vector of discrete initial temperatures, and $\boldsymbol{s}$ is the vector of discrete design variables. The advantage of using the explicit forward Euler updating scheme is the relatively small computational cost attributed to performing one time step.

To inspect stability, we approximate the characteristic times of the physics and discretization scheme. Although performing a forward Euler update is inexpensive, the scheme for transient thermal equations has a relatively strict stability constraint on the time step defined as:

$$\Delta t < \frac{h^2}{2D}, \tag{32}$$

where h is the mesh size and $D = \kappa_s/(\rho_s c_{ps})$ is the thermal diffusivity. Thermal diffusivity in both the solid ($\alpha_s = 1$) and void domains ($\alpha_s \to 0$) will stay the same as in Equation (30) both the conductivity κ_s and the density ρ_s are multiplied with α_s and $D = \alpha_s\kappa_s/(\alpha_s\rho_s c_{ps}) = \kappa_s/(\rho_s c_{ps})$. The strict time step constraint leads to the need for many updates to be performed which increases computational time. However, we find that this small time step may be required to accurately represent the physics. The characteristic timescale for a thermal diffusion problem can be defined as[29]:

$$\tau = \frac{L^2}{D}, \tag{33}$$

where L is the characteristic lengthscale of the problem. The characteristic timescale indicates a time over which a transient response settles. If we are interested in a design with a feature size of a few elements ($L \approx h$) we find that these features have a characteristic timescale of $\tau_h \approx h^2/D$, and thus need the relatively small time step defined in Equation 32 to accurately model such features. A disadvantage of using a small time step is that it will cause large memory requirements. However, the methods presented in this paper can be used to keep these memory requirements to a minimum. Since this is a linear system, the local stability analysis in Section 3.1.1 holds. The adjoint equations are stable if the state equations are stable when we satisfy Equation (32). Moreover, as discussed in Section 3.1.1, the adjoint equations react and stabilize over similar timescales as the state equations. We can thus use the estimate for characteristic time in Equation (33) to estimate over which timescales adjoint errors converge to zero.

5.1.2 | Transient thermal optimization problem

The problem in Figure 8 is designed to investigate the limitations of the approximate algorithms. The problem consists of a plate on which three heat generating components are attached. Material and problem specific parameters are given in Table 3, and result in a diffusivity of $D = 10^{-3}$. The characteristic timescale of the plate is estimated as $\tau = 10$, which is

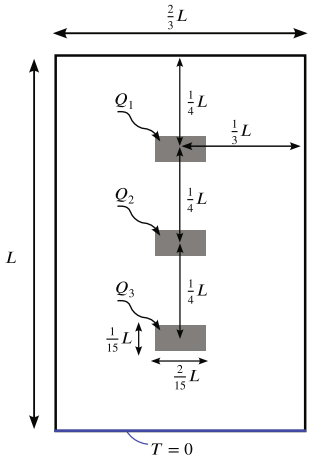


FIGURE 8 The thermal optimization problem with parameters in Tables 3 and 4. On the gray areas, time dependent heat loads are applied as defined in Equation (34). The heat loads are defined such that Q_2 delivers half as much and Q_3 twice as much energy to the system as Q_1. On the bottom boundary, a heat sink is applied which fixes the temperature to $T = 0$ and all other boundaries are isolated.

TABLE 3 The parameters used for the thermal optimization scenario as shown in Figure 8 with results shown in Section 5.1.3.

L	h	t_t	Δt	ρ_s	c_{ps}	κ_s
0.1	$\frac{10^{-2}}{12}$	10	$\frac{1}{5760}$	100	10	1

used as final time $t_t = \tau = 10$. The bottom boundary of the plate is cooled and has fixed temperature $T = 0$ while all other boundaries are isolated. Three time-dependent heat sources Q_1, Q_2, and Q_3 are added to the plate:

$$Q_1 = 10^3, \qquad Q_2 = \begin{cases} \frac{3}{4} \cdot 10^3, & \text{if } t > \frac{t_t}{3} \\ 0, & \text{otherwise} \end{cases}, \qquad Q_3 = \begin{cases} 6 \cdot 10^3, & \text{if } t > \frac{2t_t}{3} \\ 0, & \text{otherwise} \end{cases}. \tag{34}$$

The objective of the optimization procedure is to minimize the average temperature at the heat sources:

$$F = \int_0^{t_t} \int_{\Omega_Q} T d\Omega \, dt, \tag{35}$$

where Ω_Q is the gray domain in Figure 8 where heat loads are applied.

Furthermore, to prevent checkerboarding and regularize the optimization procedure, a smoothing and continuous Heaviside projection filter are needed. First all ith design variables s_i of $\boldsymbol{s}$ are smoothed using the filter as presented by Bruns et al.[37] using the design variables within a distance r to the center of s_i, resulting in smoothed variables $\bar{s}_i$. Subsequently, the continuous Heaviside function as presented by Wang et al.[38] is applied to the smoothed design variables $\bar{s}_i$ which projects them to a 0/1 solution as:

$$\tilde{s}_i = \frac{\tanh(\beta\eta) + \tanh(\beta(\bar{s}_i - \eta))}{\tanh(\beta\eta) + \tanh(\beta(1 - \eta))}, \tag{36}$$

where $\tilde{s}_i$ is the design variable after the projection filter is applied, η is the threshold value, and β indicates the slope and sharpness of the projection filter. Generally, β is increased during an optimization procedure through a continuation scheme such that in the earlier design iteration the continuous Heaviside projection does not restrict the convergence and in the later iterations the design is pushed to a 0/1 solution. In this work we set the slope to $\underline{\beta}$ in the first 10 design iterations after which we increase it with $\Delta\beta$ each iteration until it reaches its upper bound $\beta = \overline{\beta}$. Subsequently, a volume constraint is applied on the smoothed and projected design variables $\tilde{\boldsymbol{s}}$. The constraint imposes an upper limit V_f on the solid volume fraction:

$$g_v(\tilde{\boldsymbol{s}}) = \frac{\sum_{i=1}^{N_d} \tilde{s}_i}{N_d} - V_f \leq 0, \tag{37}$$

where N_d is the number of discrete design variables $\tilde{s}_i$ in $\tilde{\boldsymbol{s}}$. Moreover, the Solid Isotropic Material with Penalization (SIMP) material interpolation[39] is used which defines a relation between design variables and material properties. We use SIMP to interpolate the solid volume fraction as:

$$\alpha_s(\tilde{s}_i) = \underline{\alpha}_s + (1 - \underline{\alpha}_s)\tilde{s}_i^p, \tag{38}$$

where $\underline{\alpha}_s$ is the lower bound on the solid volume fraction and p regulates the convexity of the interpolation function. All optimization parameters can be found in Table 4. Finally, using the discrete thermal model in Equation (31), the discrete optimization problem is defined as:

$$\begin{aligned} \underset{\boldsymbol{s}}{\text{minimize}} \quad & F \approx \sum_{n=0}^{N-1} F_n(\boldsymbol{T}_n, \boldsymbol{s}) \Delta t, \\ \text{subject to} \quad & R_n(\boldsymbol{T}_{n-1}, \boldsymbol{T}_n, \boldsymbol{s}, t_n) = 0 \quad n \in \{1, 2, \ldots N\}, \\ & R_0(\boldsymbol{T}_0) = 0, \\ & g_v(\tilde{\boldsymbol{s}}) \leq 0, \end{aligned} \tag{39}$$

TABLE 4 The optimization parameters for the thermal problem in Figure 8.

r	$\underline{\beta}$	$\Delta\beta$	$\overline{\beta}$	p	V_f	$\underline{\alpha}_s$	$\epsilon_\lambda^{\max}$
$3h$	2	0.1	8	3	0.3	10^{-3}	0.1

Notes: We smooth the design over a radius of three elements ($r = 3h$), increase the Heaviside projection slope from $\underline{\beta}$ to $\overline{\beta}$ by $\Delta\beta$ over iterations 10 to 70 and apply the SIMP interpolation to the volume fraction α_s using factor p. Furthermore, a maximum allowable adjoint error of $\epsilon_\lambda^{\max}$ is set for the CP/LT algorithm.

We examine the characteristic times of the heat generating components to compare them to the interval lengths and make predictions about the designs found in Section 5.1.4. We expect the optimal design to connect all heated domains to the heat sink at the bottom boundary. The heat sources are defined such that Q_2 delivers half as much and Q_3 twice as much heat to the system as Q_1. Connecting the top domain directly to the heat sink might be disadvantageous as this allows for a direct flow of thermal energy from the top domain through the center and bottom domains. As the top source is located further away from the heat sink, we expect it to have a larger characteristic time, leading to a slower decay of errors in adjoint. We estimate the characteristic timescales of the heat sources using their respective distances to the heat sink: $L_1 = \frac{3}{4}L$, $L_2 = \frac{1}{2}L$, and $L_3 = \frac{1}{4}L$. Subsequently, the characteristic timescales are estimated following Equation (33):

$$\tau_1 = \frac{L_1^2}{D} = \frac{\left(\frac{3}{4}L\right)^2}{D} = \frac{9}{16}\frac{L^2}{D} = \frac{9}{16}\tau, \qquad \tau_2 = \frac{1}{4}\tau, \qquad \tau_3 = \frac{1}{16}\tau. \tag{40}$$

The characteristic timescale of the bottom source is thus almost an order of magnitude smaller than the characteristic timescale of the top source. Moreover, the distance between the top two heat sources is $L_{1,2} = \frac{1}{4}L = L_3$, and the characteristic timescale of interaction between these two sources is thus also $\tau_{1,2} = \tau_3 = \frac{1}{16}\tau$. Erroneous adjoints carrying information about the interaction between the two top domains may thus converge to the correct adjoint much quicker than erroneous adjoints carrying information about the interaction between the top influx and heat sink. We thus expect the approximate algorithms to experience problems around the top influx domain.

To estimate the memory requirements of this optimization problem we compute the number of DOFs in x and y-direction, respectively, as $N_x = 81$ and $N_y = 121$. Subsequently, by using the fact that each DOF is stored as a double containing 8 bytes the size of one state solution is computed as $m = 8N_xN_y \approx 78$ kB. Since the number of time steps is $N = t_t/\Delta t = 57.6 \cdot 10^3$, we estimate the memory requirements assuming that $K \ll N$. Following Table 1, the GT and PLT algorithms require a memory of $M = mN \approx 4.5$ GB and the CP, LT, and CP/LT algorithms require a memory of $M \approx mN/K \approx 4.5/K$ GB. With these reasonable memory requirements, no large K is needed. However, if a similar problem is solved in 3D with a resolution of $N_z = N_x = 81$ in z-direction, this would increase the size of the state solution to $m = 8N_xN_yN_z \approx 6.4$ MB. The memory requirements would thus increase as $M \approx mN \approx 368$ GB for the GT and PLT algorithms. We thus argue that the relatively high number of intervals $K = 20$ should be used to reduce the theoretical memory in 3D to $M \approx mN/K \approx 17.8$ GB for the CP, LT, and CP/LT algorithms.

5.1.3 | Reference results

Using the problem as described in Section 5.1.2, we analyze the stability of the approximate LT, PLT, and CP/LT algorithms. We expect the adjoint errors in the approximate algorithms to dampen out over multiple design iterations. However, due to the large errors in the first few design iterations, the optimization procedure may experience weak global convergence issues discussed in Section 3.1.1. The approximate LT, CP/LT, and PLT algorithms might take a different convergence path than the exact GT and CP algorithms and end up in different local optima with a bias toward design features with short characteristic times.

As a reference we optimize the problem using the GT and CP ($K = 20$) algorithms which both lead to the same optimum shown in Figure 9. The optimal *bowling pin* like design connects all source domains to the bottom heat sink. We expect to find weak global convergence issues in the designs around the top source using the approximate algorithms. To analyze these errors we measure the characteristic time of the top influx domain in the GT and CP design in Figure 9. To measure the characteristic time, a constant heat flux is applied only to the top interval ($Q_1(t) = 10^3$, $Q_2(t) = Q_3(t) = 0$).

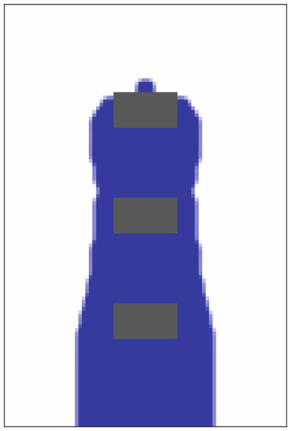


FIGURE 9 The bowling pin like design found by the GT and CP ($K = 20$) algorithms characterized by an objective value $f^*_{\text{GT}} = 1.0263$. The design connects all influx domains to the lower heat sink. The hat on top of the pin is an artifact of the optimization procedure.

Under these conditions, the characteristic time is defined as the time it takes for the transient response to reach a solution with 5% error compared to the steady-state solution. The characteristic time of the upper domain in the GT and CP design is measured as $\tau_1 = 6.12$.

Moreover, to measure the accuracy of the approximate sensitivities computed using the LT, CP/LT, and PLT algorithms, we compare them with sensitivities obtained with the GT algorithm. For each design and sensitivity $d\widetilde{F}/d\boldsymbol{s}$ found by the approximate algorithms, an error is defined by comparing with exact sensitivities $dF/d\boldsymbol{s}$ computed by the GT algorithm on the same design:

$$\varepsilon' = \frac{\left\| \frac{d\widetilde{F}}{d\boldsymbol{s}} - \frac{dF}{d\boldsymbol{s}} \right\|_2}{\left\| \frac{dF}{d\boldsymbol{s}} \right\|_2}. \tag{41}$$

Furthermore, a deviation in objective value Δf^* from the GT optimum characterized by f^*_{GT} is computed as:

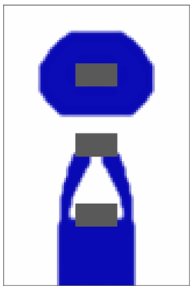
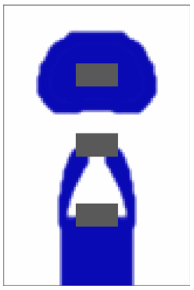
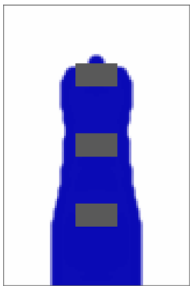
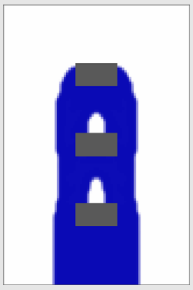
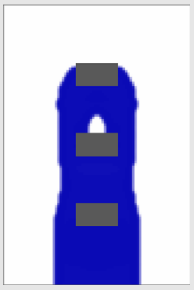
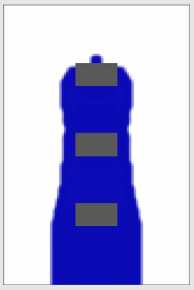
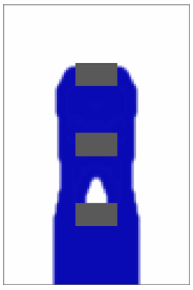
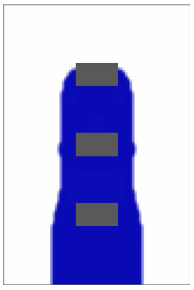
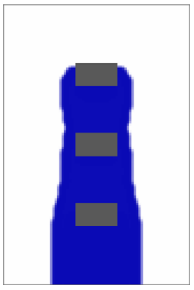
$$\Delta f^* = \frac{f^* - f^*_{\text{GT}}}{f^*_{\text{GT}}}, \tag{42}$$

where a positive Δf^* is a deteriorated objective and a negative Δf^* is an improved objective.

5.1.4 | Approximate algorithm designs and stability

We compare the baseline design to the designs found by the approximate algorithms. To investigate the influence of the number of intervals K, we optimize for $K = 2$, 5, and 20. Optimal designs for the LT, CP/LT, and PLT algorithms can be found in Table 5, objective histories in Figure 10, and sensitivity error histories in Figure 11. The most striking difference in design is found for $K = 20$ using the LT and PLT algorithms in Table 5A,B. These designs contain a large island around the upper influx domain which is disconnected from the heat sink. As can be seen by the increased objective values, this is an inferior local optimum with $\Delta f^* = 14\%$ for both the LT and PLT algorithms. The optimizer finds this inferior optimum due to the weak global convergence issues as discussed in Section 3.1.1. While the intervals with $K = 20$ have a length of $\Delta\theta = t_t/K = 0.5$, the characteristic time of the upper domain in the GT design was found as $\tau_1 = 6.12$. The interval length is thus an order of magnitude smaller than the characteristic time. In fact, if we assume a static unchanging design, the adjoint error should propagate over $\tau_1/\Delta\theta \approx 12$ intervals. As adjoint errors can only reduce over one interval each design iteration, it would take 12 design iterations for adjoint errors around the top influx to reduce sufficiently. Moreover, as can be seen in the convergence plots of the objective in Figure 10, it takes around 10 iterations for the design to stabilize.

TABLE 5 The designs, objective values f^*, and relative changes in objective Δf^* found using the approximate sensitivity computation algorithms.

	LT	PLT	CP/LT
$K = 20$	 (a) $f^* = 1.1671$, $\Delta f^* = 14\%$.	 (b) $f^* = 1.1727$, $\Delta f^* = 14\%$.	 (c) $f^* = 1.0266$, $\Delta f^* = -0.03\%$.
$K = 5$	 (d) $f^* = 1.0290$, $\Delta f^* = 0.26\%$.	 (e) $f^* = 1.0303$, $\Delta f^* = 0.40\%$.	 (f) $f^* = 1.0266$, $\Delta f^* = -0.03\%$.
$K = 2$	 (g) $f^* = 1.0299$, $\Delta f^* = 0.35\%$.	 (h) $f^* = 1.0247$, $\Delta f^* = -0.16\%$.	 (i) $f^* = 1.0250$, $\Delta f^* = -0.13\%$.

Notes: The change in objective Δf^* is relative to the objective found using the GT and CP algorithms, where a positive Δf^* increases and a negative Δf^* deteriorates the objective. Both the LT and PLT algorithms disconnect the upper heat source from the heat sink for $K = 20$ leading to an inferior optimized design. These errors emerge as the adjoint errors do not converge quickly enough and the design gets stuck in an inferior local optimum. The CP/LT designs do not suffer from this problem as they correct the largest errors in adjoint.

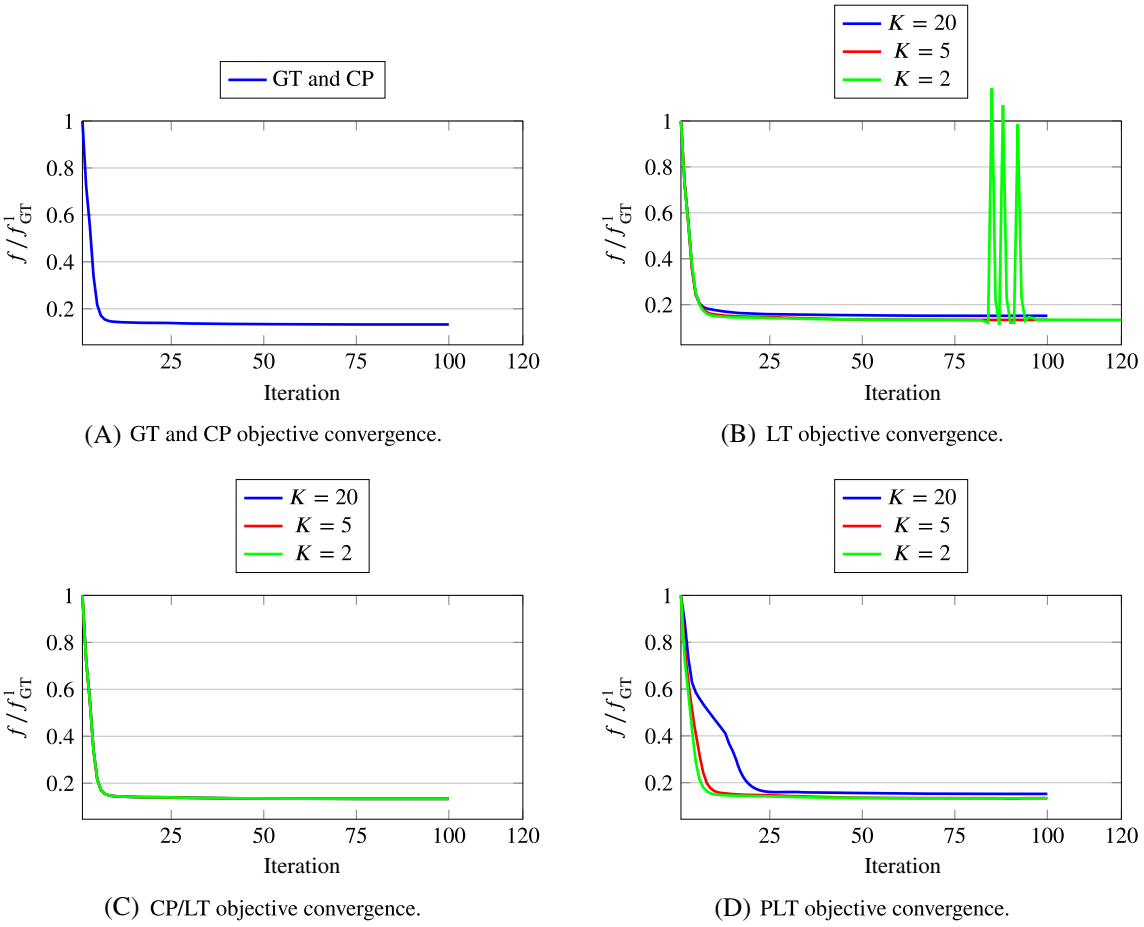
Abbreviations: LT, Local-in-Time; PLT, Parallel-Local-in-Time; CP/LT, hybrid Checkpointing/Local-in-Time.

The LT and PLT designs thus stabilize into an inferior local optimum before adjoint errors sufficiently reduce. These results suggest that to prevent weak global convergence, the length of an interval should not be an order smaller than the characteristic time:

$$\Delta\theta = \frac{t_t}{K} > \frac{\tau}{10}, \tag{43}$$

which we defined as the weak global convergence requirement.

Other significant deviations from the reference are found in the $K = 5$ and $K = 2$ LT designs in Table 5D,G and convergence in Figure 10B. Firstly, the LT $K = 2$ convergence plot shows some large instabilities during iterations 80 to 100. These errors are caused by global instabilities as discussed in Section 3.1.1. Small changes in design lead to large



(A) GT and CP objective convergence.

(B) LT objective convergence.

(C) CP/LT objective convergence.

(D) PLT objective convergence.

FIGURE 10 The objective convergence for designs in Figure 9 and Table 5. Objective values are normalized using the objective in the first design iteration computed using the GT algorithm f^1_{GT}. The GT and CP algorithms both compute exact sensitivities and show the same convergence behavior. Additionally, as the CP/LT algorithm corrects erroneous sensitivities and computes fairly accurate sensitivities as can be seen in Figure 11, it also shows similar convergence behavior to the GT and CP algorithms. The objective convergence of the LT algorithm using $K = 2$ shows large fluctuations toward the end. These fluctuations are caused by global instabilities in the design. However, the global instabilities dampen out and the design again converges to a stable optimum. The PLT convergence plot shows that when more intervals are used, the objective converges slower over more design iterations. This is caused by the approximations in state variables which leads to larger changes in objective.

changes in adjoints and consequently in sensitivities. Subsequently, these sensitivity changes lead to larger changes in design and so forth. In Figure 11A, we compare the LT sensitivities to the ones computed using the GT algorithm on the same design. We verify the increase in sensitivity error over design iterations 80–100. Moreover, we find that sensitivity errors do not reduce monotonically, and increase in parts of the convergence history. This is also the case for the sensitivity error in the PLT and CP/LT algorithms in Figure 11. Due to the stability of MMA, the design is able to revert back to a stable design and converges after 20 additional iterations. We can conclude that an increase in sensitivity errors leading to global instabilities is difficult to predict but may happen and is a risk inherent to the LT and PLT algorithms.

The aim of the CP/LT algorithm is to correct the errors in the LT algorithm, and we expect it to be less prone to global stability issues. Table 5C shows that the CP/LT algorithm is able to find the bowling pin design for $K = 20$ and did not suffer from weak global stability issues. Moreover, no global convergence issues are found for the CP/LT algorithms in Figure 10C. In Figure 11B, the sensitivity error ε' for the CP/LT algorithm is shown. During the first seven iterations, all domains are corrected and the error in sensitivity is close to numerical accuracy. When corrections stop in the eighth design iteration, an error in sensitivities is introduced and reduces over the subsequent iterations. Furthermore, the CP/LT results illustrate that the approximate algorithms find optima which are slightly different but not necessarily inferior to the ones found using exact algorithms. The designs found by the CP/LT algorithm for $K = 5$ and $K = 2$ perform slightly

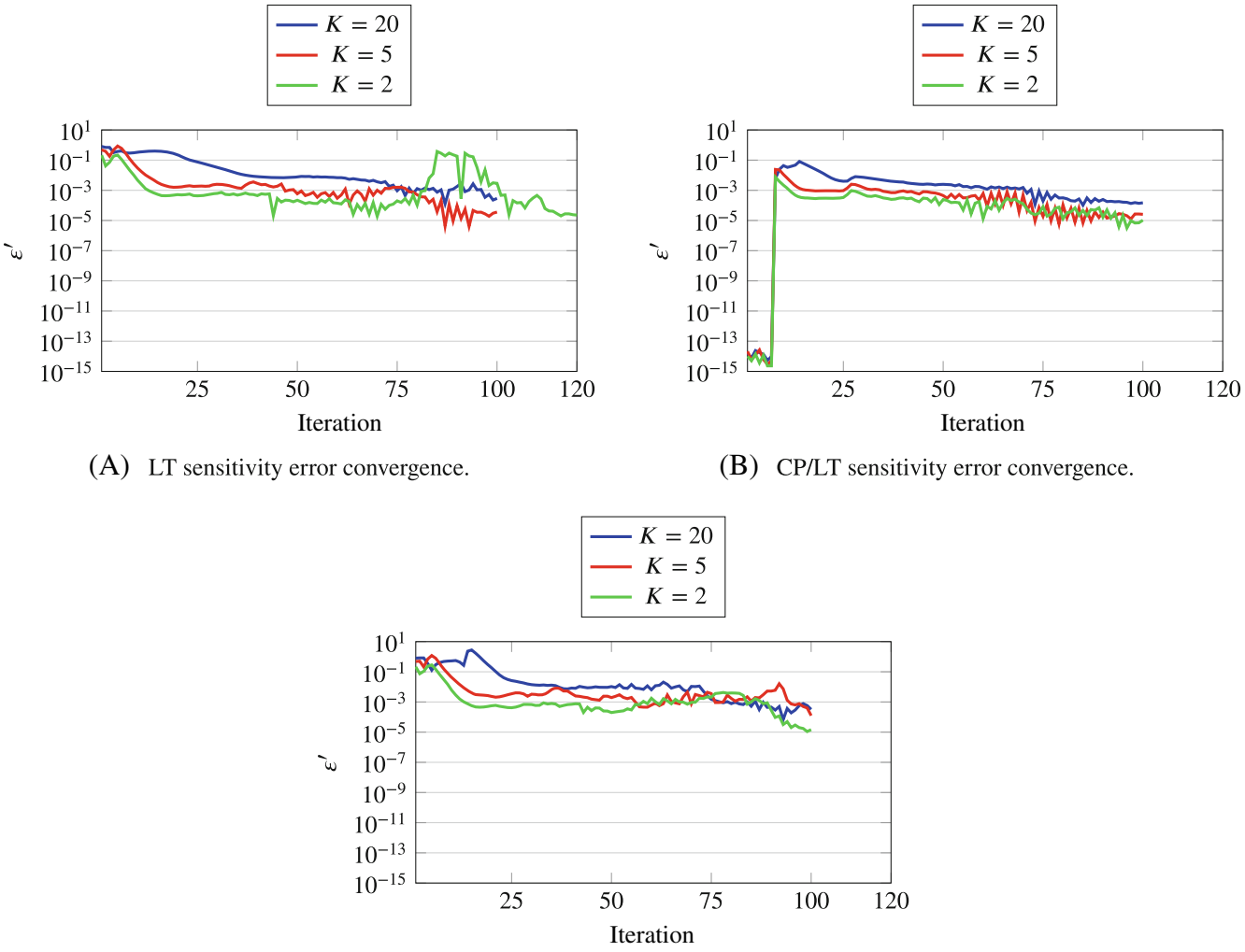


FIGURE 11 The convergence of the sensitivity error ϵ' for all design iterations. In Figure 11A clear correlation can be found between increasing sensitivity errors for LT, $K = 2$ and the global instabilities found in Figure 10B. Moreover, none of the figures show a monotonously decreasing sensitivity error as they all show some error fluctuations. These fluctuations are caused by global instabilities, but when they remain small have no noticeable effect on the objective convergence and design updates. The sensitivity errors in the CP/LT algorithm are of numerical precision in the first few design iterations where all intervals are corrected. However, as the algorithm stops correcting all intervals, errors shoot up to about $\epsilon' \approx 10^{-2}$.

better than the GT and CP design, as can be seen by the negative Δf^*. To summarize, global convergence problems are alleviated by the CP/LT algorithm, and the algorithm generally shows a more stable convergence behavior than the LT algorithm.

Finally, we compare the designs and convergence behavior of the PLT algorithm to the other algorithms. As previously discussed, the PLT algorithm suffers from weak global convergence issues for $K = 20$. Moreover, the convergence of the objective for the PLT algorithm is much slower than for the other algorithms, see Figure 10D, even when few intervals are used. This issue is caused by the errors in state variable which require more iterations to converge together with the adjoint variables. It should be noted that the PLT algorithm only requires more time to reach a stable topology and objective value during the start of the optimization procedure. Subsequently, the PLT algorithm only slightly adapts the topology and improves the objective value. While other algorithms take about ten iterations, the PLT algorithm with $K = 20$ takes 20 to find an adequate topology. As the optimizer runs for at least 100 design iterations, this does not have a large influence on the overall design convergence.

5.1.5 | Computational cost of the thermal results

Although the thermal problem is specifically designed to investigate stability, we also analyze the computational wall times found in Table 6. We show the wall time to compute sensitivities summed over all design iterations $t_{\text{solv}} = t_{\text{state}} + t_{\text{adj}}$, which consists of the time required to solve the state solution t_{state} and the time to update the adjoint and sensitivity t_{adj}. We note that for the measured wall time we summed the times for all design iterations in the optimization process. The serial algorithms were all run on one CPU, while the PLT algorithm was run on the same number of CPUs as intervals K.

Among the serial algorithms, the GT and LT $K = 20$ and $K = 5$ are the fastest. However, as the LT algorithm using $K = 2$ required 20 additional design iterations due to the strong global stability issues, it required more computational time. Compared to the increased computational time in the CP/LT algorithms, the LT algorithm with $K = 2$ performed worse due to the additional iterations, even though computational time for the CP/LT algorithm is increased by performing the adjoint corrections. The worst performing algorithm was the CP algorithm due to the recomputation of the state solutions. The parallel PLT algorithm was run using the same number of CPUs as there were intervals K. Each interval was thus attributed to one CPU. We observed a large decrease in computational time using more intervals as for $K = 20$ the PLT algorithm took about one fifteenth of the time of the GT algorithm. A perfect scaling of one twentieth of the time is not achieved here and will be further investigated in Section 5.2.

We verify the relative computational costs predicted in Section 4.2. By assuming that the measured wall times are proportional to the computational cost ($t_{\text{state}} \propto c_s$, $t_{\text{adj}} \propto c_a$), relative computational cost $r_c = c_a/c_s = t_{\text{state}}/t_{\text{adj}}$ is approximated using the computational times of GT: $r_c = t^{\text{GT}}_{\text{adj}}/t^{\text{GT}}_{\text{state}} = 84.19/32.83 \approx 2.56$. Since both state and adjoint updates are simple matrix vector multiplications, they have the same cost. Because the sensitivity computation is also a matrix vector multiplication, it has a similar cost. This leads to a larger c_a which contains both adjoint solve and sensitivity computation, compared to c_s which contains only the state solve. Subsequently, we compare the analytical relative computational cost of the CP algorithm in Equation (27), to the measured computational cost, respectively, as:

$$\frac{C^{\text{CP}}}{C^{\text{GT}}} \approx \frac{2 + r_c - 1/K}{1 + r_c} = 1.27, \qquad \frac{C^{\text{CP}}}{C^{\text{GT}}} = \frac{t^{\text{CP}}_{solv}}{t^{\text{GT}}_{solv}} = \frac{150.3}{117.0} = 1.28, \tag{44}$$

which is in agreement with our predictions in Equation (27).

A disadvantage of the corrections performed by the CP/LT algorithm is that they increase computational cost. In Table 6, we find that the GT algorithm takes 117.0 min to compute sensitivities, while the CP/LT algorithm takes at least 10 min more. The more intervals are used, the more computational time is increased. The increase in computational time for more intervals is caused by the additional time spent in adjoint corrections. This remark is supported by the number of intervals in each design iteration for which a correction is carried out as illustrated in Figure 12. For $K = 20$, 5, and 2, corrections are only performed in the first 12, 9, and 8 design iterations, respectively. All intervals except the last are corrected the first seven iterations for each K. At the eight iteration, the design stabilizes and the number of corrected intervals reduces. As the correct terminal adjoint propagates backward-in-time, fewer intervals need to be corrected. Intervals which are not corrected start at the terminal interval and propagate backward-in-time. Since the correct terminal adjoint can only propagate one interval at a time, using more intervals leads to a slower decrease in adjoint error, a larger number of performed corrections, and an increased computational time.

The observation that corrections are performed only in the first iterations can be used to analyze the relative computational cost of CP/LT a priori. A designer with some knowledge of their optimization problem can estimate how many iterations are required for the design to stabilize. Subsequently, the percentage of corrected intervals f_c, as defined in

TABLE 6 The computational time $t_{\text{solv}} = t_{\text{state}} + t_{\text{adj}}$ is the sum over all design iteration and consists of the time to solve the state equations t_{state} and the time to solve adjoint equations while simultaneously updating sensitivities t_{adj}.

	GT	CP $K = 20$	LT $K = 20$	LT $K = 5$	LT $K = 2$	CP/LT $K = 20$	CP/LT $K = 5$	CP/LT $K = 2$	PLT $K = 20$	PLT $K = 5$	PLT $K = 2$
t_{solv} (min)	117.0	150.3	118.9	114.6	138.0	132.8	127.3	127.0	7.698	25.66	64.99

Abbreviations: CP, Checkpointing; GT, Global-in-Time; LT, Local-in-Time; PLT, Parallel-Local-in-Time; CP/LT, hybrid Checkpointing/Local-in-Time.

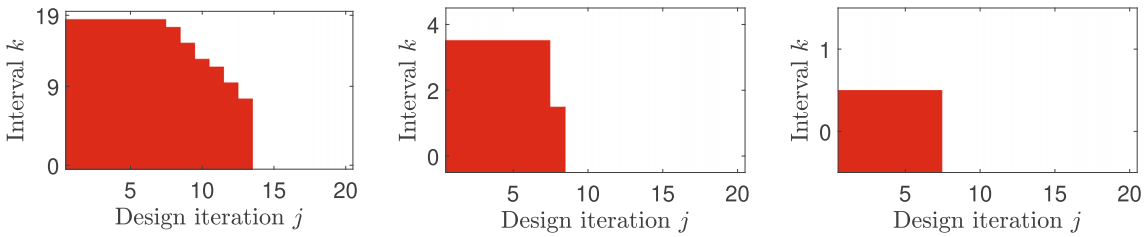


FIGURE 12 The intervals Θ_k, that require corrections for the CP/LT algorithm are flagged in red. The sensitivities are corrected only during the first part of the procedure. when more intervals are used, we require corrections in more design iterations, leading to increased computational times as shown in Table 6. Moreover, we find that starting from the terminal interval less and less intervals are corrected. This is caused by the fact that the terminal adjoint value is exact and this correction slowly propagates throughout the system.

Section 4.2, can be estimated as the percentage of unstable to stable design iterations. Equation (29) is then used to compare the CP/LT and CP performance. We verify Equation (29) for CP/LT $K = 20$. First, we compute the average percentage of corrected intervals over all design iterations, as shown in Figure 12A, as $f_c = 0.112$. Using Equation (29) and comparing it to the measured relative computational cost, we find respectively:

$$\frac{C^{\text{CP/LT}}}{C^{\text{CP}}} \approx \frac{K-1}{K}(f_c+1)\frac{1+r_c}{2+r_c}+\frac{1}{K} = 0.875, \qquad \frac{C^{\text{CP/LT}}_{\text{comp}}}{C^{\text{CP}}_{\text{comp}}} = \frac{t^{\text{CP/LT}}_{\text{solv}}}{t^{\text{CP}}_{\text{solv}}} = \frac{132.8}{150.3} = 0.884, \tag{45}$$

which is in agreement with our predictions in Equation (29).

5.2 | Computational cost investigation through flow optimization

In this section, we compare the computational cost and the computational wall time of the presented algorithms. A transient flow problem is optimized as the nonlinear flow physics increase the complexity and consequently the computational effort. Furthermore, the problem is used to emphasize the difference between adjoint updates which are linear by nature and state updates which may be linear or nonlinear. Subsequently, the speedup of the PLT algorithm using multiple computational nodes is investigated. Additionally, we examine local/global stability and design convergence. In the PLT speedup investigation, we verify the weak global convergence requirement found in Equation (43).

5.2.1 | Transient flow model

To ensure stability of the approximate algorithms, we investigate the characteristic timescale of the flow model, its discretization, and solution scheme. We discretize the transient equations in space and time. A two-dimensional transient flow problem is examined and is defined on temporal domain $t \in [0, t_t]$ and spatial domain $\vec{x} \in \Omega$ with boundary Γ. The flow problem is governed by the Navier–Stokes equations:

$$\begin{aligned}
\rho_f \dot{\mathbf{v}} + \rho_f \nabla \cdot (\mathbf{v}\mathbf{v}^\intercal) &= -\nabla p + \mu_f \nabla^2 \mathbf{v} - \frac{k_f(\alpha_f)}{\alpha_f}\mathbf{v}, && \text{on } \Omega,\\
\nabla \cdot \mathbf{v} &= 0, && \text{on } \Omega,\\
\mathbf{v} &= \mathbf{v}_\Gamma, && \text{on } \Gamma^{\text{v}}_d,\\
p &= p_\Gamma, && \text{on } \Gamma^{p}_d,\\
\mathbf{v} &= 0, && \text{on } \Gamma_w,\\
\mathbf{v} = \hat{\mathbf{v}},\ p, &= \hat{p}, && \text{at } t = 0,
\end{aligned} \tag{46}$$

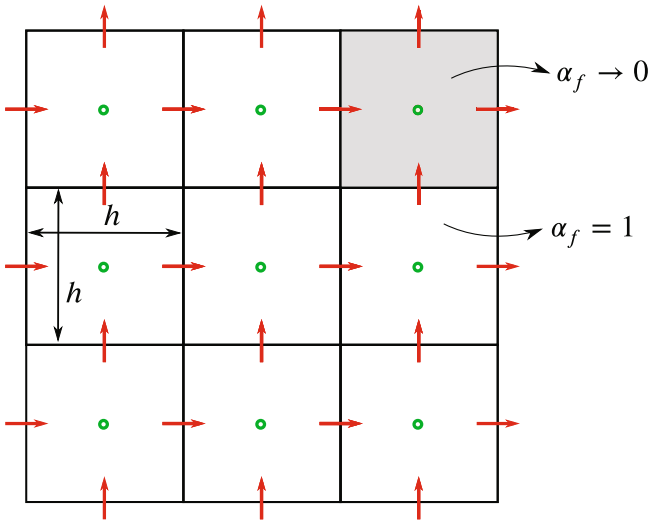


FIGURE 13 The equidistant uniform mesh used to discretize the flow problem. Green circles are the pressure nodes and densities are defined per element where $\alpha_f = 1$ is a fluid element and $\alpha_f \to 0$ is a solid element. Horizontal red arrows are the u-velocity degrees of freedom (DOFs) while vertical red arrows represent the v-velocity DOFs.

where the subscript $\square_f$ denotes fluid material properties, $\mathbf{v}(\vec{x}, t) = [u, v]^\intercal$ is the velocity field with time derivative $\dot{\mathbf{v}} = \partial \mathbf{v}/\partial t$ and component u in x-direction and v in y-direction, $p(\vec{x}, t)$ is the pressure field, $\alpha_f(\vec{x})$ is the design field which represents the fluid volume fraction and is thus set to $\alpha_f = 1$ in the fluid domain. Furthermore, ρ_f and μ_f are the fluid density and dynamic viscosity, respectively, and $k_f(\alpha_f)$ is the Darcy impermeability used to inhibit flow in the solid domain where the fluid volume fraction tends to zero $\alpha_f \to 0$. Furthermore, $\mathbf{v}_\Gamma(\vec{x})$ and $p_\Gamma(\vec{x})$ are the fixed flow and pressure on boundary Γ_d^{v} and Γ_d^p, respectively, the flow at the solid/fluid interface Γ_w is prescribed as $\mathbf{v} = \mathbf{0}$, and $\hat{\mathbf{v}}(\vec{x})$ and $\hat{p}(\vec{x})$ are the initial flow and pressure distributions. The Navier–Stokes equations are discretized using the finite volume method as described in Theulings et al.[34] on a staggered grid as shown in Figure 13. Moreover, they are solved using the SIMPLE algorithm as described by Versteeg and Malalasekera.[28] As the SIMPLE algorithm is an implicit algorithm, the equations take the form:

$$\begin{aligned} R_n^m(\mathbf{v}_{n-1}, \mathbf{v}_n, \boldsymbol{p}_n, \boldsymbol{s}, t_n) &= \boldsymbol{M}^t \frac{\mathbf{v}_n - \mathbf{v}_{n-1}}{\Delta t} + \boldsymbol{M}^c(\mathbf{v}_n)\mathbf{v}_n + \boldsymbol{C}^p \boldsymbol{p}_n - \boldsymbol{D}\mathbf{v}_n + \boldsymbol{K}(\boldsymbol{s})\mathbf{v}_n, \\ R_n^c(\mathbf{v}_n) &= \boldsymbol{C}^c \mathbf{v}_n, \\ R_0^{\mathrm{v}}(\mathbf{v}_0) &= \mathbf{v}_0 - \hat{\mathbf{v}}, \\ R_0^p(\boldsymbol{p}_0) &= \boldsymbol{p}_0 - \hat{\boldsymbol{p}}, \end{aligned} \tag{47}$$

where R_n^m and R_n^c are the discretized momentum and continuity equations, respectively, and $\mathbf{v}_n$, $\boldsymbol{p}_n$ are vectors of discrete nodal velocities and pressures at time step t_n with initial condition $\hat{\mathbf{v}}$ and $\hat{\boldsymbol{p}}$, respectively. As can be seen by the dependence of the convective matrix $\boldsymbol{M}^c$ on $\mathbf{v}_n$, the momentum equations are nonlinear and several subsolves are required to advance one time step in the SIMPLE algorithm. Each subsolve includes the assembly of the *pressure correction matrix* and subsequent solution of the pressure correction which involves a linear matrix solve. However, a backward adjoint step only involves one matrix assembly and one solve as the adjoint equations are linear by nature. Solving a backward adjoint step is thus cheaper than solving a forward state step.

To ensure stability of the approximate algorithms and the time stepping scheme, we estimate the characteristic timescale of the physics and discretization scheme. For the SIMPLE algorithm to converge, the time step is constrained by the Courant–Friedrichs–Lewy (CFL) condition as:

$$\Delta t < \frac{h}{\overline{u}}, \tag{48}$$

where $\overline{u}$ is the maximum flow magnitude. Furthermore, the bound on the time step is related to a characteristic timescale. If we imagine a parcel of fluid flowing over a distance L with velocity $\overline{u}$, it travels this distance in:

$$\tau^{\rho} = \frac{L}{u}. \tag{49}$$

The characteristic timescale for flow can be estimated by τ^{ρ}, which is in fact the characteristic timescale of inertia.[29] However, in the Navier–Stokes equations, another characteristic time is defined by the viscous dissipation[29] in Equation (46):

$$\tau^{\mu} = \frac{L^2}{\nu_f}, \tag{50}$$

where $\nu_f = \mu_f / \rho_f$ is the diffusivity of momentum, more commonly called the kinematic viscosity. While τ^{ρ} is related to the characteristics of the flow, τ^{μ} is related to the characteristics of the viscous energy dissipation. Although no local stability criterion for nonlinear state equations was derived in Section 3.1.1, we generally found the adjoint equations to be stable when the constraint in Equation (48) was applied. Furthermore, for strong/weak global stability, the characteristic timescale in Equations (49) and (50) will be considered in the problem setup in Section 5.2.2.

For the thermal problem in Section 5.1.1, the time step constraint and characteristic times at the element scale were similar. However, they are not necessarily directly related for the flow problem. We thus investigate the possible length scales of features in the design, and use it to find the characteristic times of these features. In topology optimization, we may design features of a few elements $L \approx h$ and are interested in characteristic times of:

$$\tau_h^{\rho} \approx \frac{h}{u}, \qquad \tau_h^{\mu} \approx \frac{h^2}{\nu_f}, \tag{51}$$

where τ_h^{ρ} is already satisfied by the stability condition in Equation (48). However, τ_h^{μ} may be smaller than τ_h^{ρ}, and in design components of a few elements long viscous effects may dominate. A smaller time step may thus be needed to accurately represent the transient effects in these design features. One could think of a design with many narrow channels with low flow speeds and low Reynolds numbers. It is left to the user to make an a priori estimation of the type of design resulting from the optimization procedure and select the appropriate time steps.

5.2.2 | Transient flow optimization problem

The problem in Figure 14 with parameters in Table 7 is designed such that nonlinear flow effects are of importance for the optimized designs. We design a piston pump without moving parts. A piston pumps fluid up and down at the top boundary Γ_{pump}, where an oscillating flow is applied as:

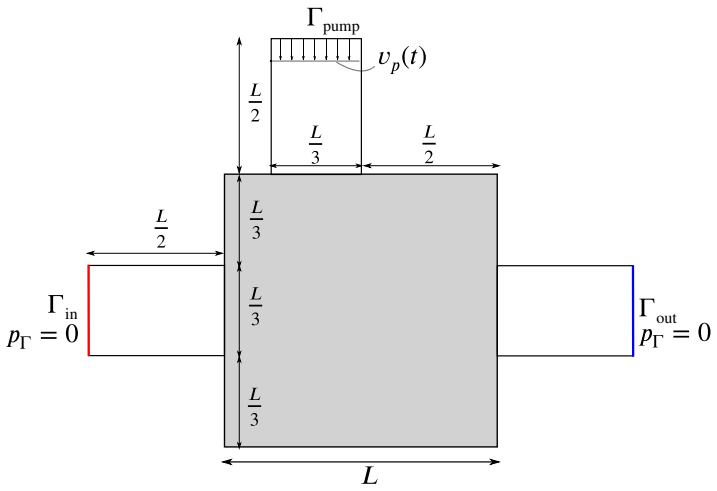


FIGURE 14 The piston pump optimization problem. At the top inlet/outlet, a fluctuating velocity is prescribed and at the red inlet and blue outlet only reference pressures are prescribed. The white domains are non-design areas, and in the gray domain, the material layout is optimized.

TABLE 7 The discretization and material parameters used for the flow optimization problem in Figure 14.

L	h	t_t	Δt	ρ_f	μ_f	$\bar{v}$
1	$\frac{1}{42}$	5	$\frac{1}{42}$	1	$\frac{1}{60}$	1

$$\begin{aligned} v_p(t) &= \bar{v}\sin(4\pi t), \\ u_p(t) &= 0, \end{aligned} \tag{52}$$

with $\bar{v}$ the maximum inlet velocity in y-direction over time as found in Table 7. We optimize the mass flow to the right from inlet Γ_{in} to outlet Γ_{out}, and apply a constant pressure on both the inlet and outlet $p_\Gamma = 0$. During the downstroke of the piston, we expect fluid to be pushed out of the system through Γ_{out}; while during the upstroke, we expect fluid to be pulled into the systems through Γ_{in}. Subsequently, we maximize the volumetric flow through the system from inlet Γ_{in} to outlet Γ_{out}:

$$F = \int_0^{t_t} \left(-\int_{\Gamma_{\text{in}}} \mathbf{v}\cdot\boldsymbol{n} d\Gamma + \int_{\Gamma_{\text{out}}} \mathbf{v}\cdot\boldsymbol{n} d\Gamma \right) dt, \tag{53}$$

where $\boldsymbol{n}$ is the outward normal on the boundaries. To normalize the problem, we define a theoretical maximum of a piston pump with moving parts which closes off the inlet during the downstroke and closes off the outlet during the upstroke. In such a pump, the volumetric flow to the right through the in- and outlet is the same as the absolute volumetric flow through Γ_{pump}:

$$\overline{F} = \int_0^{t_t} \int_{\Gamma_{\text{pump}}} |\bar{v}\sin(4\pi t)| d\Gamma dt = \bar{v}\frac{L}{3}\frac{10}{\pi}. \tag{54}$$

Subsequently, we use the theoretical optimum and minimize for the normalized objective $F_p = (\overline{F} - F)/\overline{F} = 1 - F/\overline{F}$. The normalized objective can be interpreted as the decrease in efficiency of the pump without moving parts compared to a pump with moving parts. Furthermore, using the material parameters in Table 7, the Reynolds number is estimated as:

$$Re = \frac{\rho_f \bar{v} L}{\mu_f} = 60. \tag{55}$$

The optimal design is expected to use the nonlinearity of moderate Reynolds flow.

To regularize the optimization procedure, filters and interpolation functions are used. Firstly, design variables s_i are smoothed[37] and projected[38] following the approach described in Section 5.1.2 resulting in $\widetilde{s}_i$. The volume constraint $g_v(\widetilde{\boldsymbol{s}})$ is applied using Equation (37). The interpolation of the fluid volume fraction α_f, the Darcy impermeability $k_f(\alpha_f)$ and the maximum impermeability $\overline{k}_f$ are defined following Theulings et al.[34] For both α_f and k_f, a linear interpolation is used:

$$\begin{aligned} \alpha_f(\widetilde{s}_i) &= \underline{\alpha}_f + (1 - \underline{\alpha}_f)\widetilde{s}_i, \\ k_f(\alpha_f) &= \overline{k}_f(1 - \alpha_f), \end{aligned} \tag{56}$$

such that in the fluid domain ($\alpha_f = 1$), no penalization ($k_f = 0$) is imposed and in the solid domain ($\alpha_f = \underline{\alpha}_f \to 0$), the maximum penalization ($\overline{k}_f(1 - \underline{\alpha}_f) \approx \overline{k}_f$) is prescribed. Moreover, the maximum penalization $\overline{k}_f$ is set using parameter $H^e = h^{-2}$ and the elemental Reynolds number $Re^e = \rho_f \bar{v} h/\mu$:

$$\overline{k}_f = 10^q \begin{cases} \mu H^e, & Re^e \leq 1, \\ \mu H^e Re^e, & Re^e > 1. \end{cases} \tag{57}$$

Parameter q is used to set the magnitude of the flow penalization. All optimization parameters can be found in Table 8. Furthermore, using the material parameters in Table 7, the elemental Reynolds number is computed as:

$$Re^e = \frac{60}{42}, \tag{58}$$

TABLE 8 The optimization parameters used for the flow problem in Figure 14.

r	$\underline{\beta}$	$\Delta\beta$	$\overline{\beta}$	V_f	q	$\underline{\alpha}_f$	$\epsilon_\lambda^{\max}$
$3h$	1	0.14	8	0.6	2	10^{-1}	0.1

Notes: We smooth the design over a radius of three elements ($r = 3h$) and increase the Heaviside projection slope from $\underline{\beta}$ to $\overline{\beta}$ by $\Delta\beta$ over iterations 10–60. Furthermore, a maximum allowable adjoint error of $\epsilon_\lambda^{\max}$ is set for the CP/LT algorithm.

resulting in a maximum penalization of:

$$\overline{k}_f = 10^q \mu_f H^e Re^e = 4200. \tag{59}$$

Finally, using the discrete model in Equation 47, the discrete optimization problem is stated as:

$$\begin{aligned}
&\underset{\boldsymbol{s}}{\text{minimize}} && F_p \approx \sum_{n=0}^{N-1} F_n(\mathbf{v}_n, \boldsymbol{s})\Delta t && \\
&\text{subject to} && R_n^m(\mathbf{v}_{n-1}, \mathbf{v}_n, \boldsymbol{p}_n, \boldsymbol{s}) = 0 && n \in \{1, 2, \dots N\}, \\
& && R_n^c(\mathbf{v}_n) = 0 && n \in \{1, 2, \dots N\}, \\
& && R_0^v(\mathbf{v}_0), && \\
& && R_0^p(\boldsymbol{p}_0), && \\
& && g_v(\boldsymbol{s}) \le 0, &&
\end{aligned} \tag{60}$$

Characteristic times are inspected to investigate the weak global convergence requirement in Equation (43). First, we compare the characteristic time of the flow and of the viscous diffusion:

$$\begin{aligned}
\tau^\rho &= \frac{L}{\overline{v}} = 1, \\
\tau^\mu &= \frac{L^2}{\nu_f} = 60,
\end{aligned} \tag{61}$$

where we estimated the maximum flow speed as the maximum flow at the inlet $\overline{v}$, and substituted $\nu_f = \mu_f/\rho_f$ using the values in Table 7. Since the characteristic time of the viscous diffusion is an order of magnitude higher than that of the inertia, and we examine a problem with terminal time $t_t = 5\tau^\rho$, we expect transient effects of inertia to be dominant. Furthermore, weak global convergence issues emerged in Section 5.1.3 when Equation (43) was not satisfied. We thus expect that they may occur in this problem when the interval length $\Delta\theta = t_t/K$ is an order of magnitude smaller than τ^ρ and thus:

$$K > \frac{10 t_t}{\tau^\rho} = 50. \tag{62}$$

In Section 5.2.4, we will use PLT with $K = 48$ and may thus find weak global convergence issues. Strong global convergence issues are harder to predict and may occur. In the authors experience, convergence is generally achieved when sufficient design iterations are used.

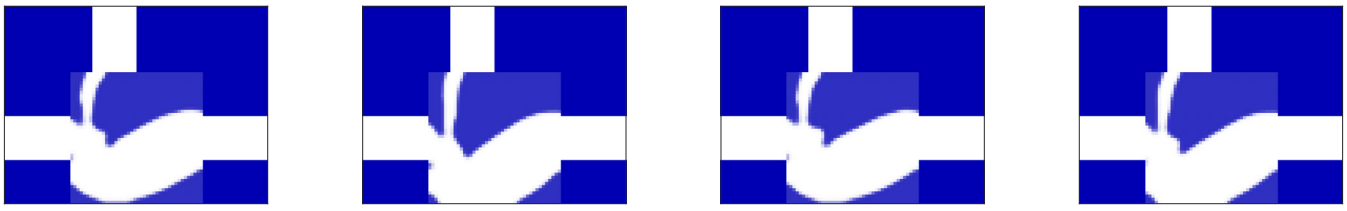
Similar to the analysis at the end of Section 5.1.2, we estimate the memory requirements of the problem and select the appropriate number of intervals K. As the CFL condition in Equation (48) defines an upper bound on the time step dependent on h, the number of time steps and memory requirements are dependent on the spatial resolution. The number of flow and pressure DOFs in this problem is approximately $N_d \approx 7500$. As each DOF is stored as a double, the size of the state solution at a time step is $m = 8N_d \approx 60$ kB. Furthermore, using the parameters in Table 7, we store the state solution at $N = t_t/\Delta t = 210$ time steps. Assuming that $K \ll N$, memory requirements following Table 1 scale as $M = mN \approx 0.13$ GB for the GT and PLT algorithms and as $M = mN/K \approx 0.13/K$ GB for the CP, LT, and CP/LT algorithms. Although for this relatively small 2D problem, there is no need for memory reducing algorithms, memory requirements scale up quickly when 3D problems are investigated and the mesh is refined. For the analysis of the computational cost, we will use $K = 8$ intervals.

5.2.3 | Computational cost comparison

In Section 4.2, Table 2, we made a theoretical approximation of the computational cost of the algorithms. In this section, we compare these theoretical approximations with measured computational costs. All computations of the serial algorithms are carried out on the same type of CPU. Moreover, to quantify computational cost, we measure wall time. Since the same type of CPUs with similar computational power are used, we expect wall time to be a sufficiently accurate measure of computational cost. However, for the PLT algorithm the same number of CPUs as there are intervals is used, contrary to the serial algorithms which use only one CPU. As the PLT algorithm uses more computational power, we expect a significant speedup.

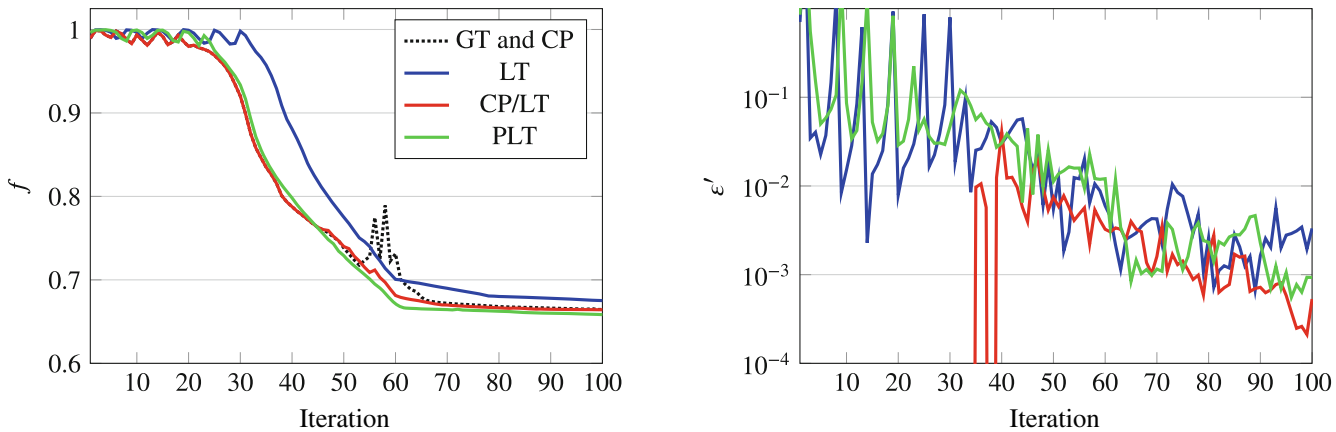
Different designs lead to different flow solutions which require a different computational effort to solve. To compare speedup, we would like the resulting designs to be similar. As $K = 8$ intervals are used and since we expect global convergence issues to start at $K = 50$ intervals as shown in Equation (62), we expect similar designs to be found by all algorithms. This is confirmed by the designs in Figure 15. However, as shown in Figure 16A, the different algorithms take a different convergence path to their local optima. During the design convergence, the computational cost of solving the nonlinear state equations may differ at any given design iteration. This effect is responsible for some small deviations between expected and measured wall times.

To verify stability and convergence, we further analyze the objective and sensitivity error convergence in Figure 16. We note that the objective is increased over iterations 55–60 for the GT and CP algorithms. This is caused by the nonlinearity of the flow solution leading to an increase in the objective. Moreover, as the CP/LT algorithm corrects erroneous



(A) GT and CP, $f^* = 0.6645$, $\Delta f^* = 0\%$.
(B) LT, $f^* = 0.6752$, $\Delta f^* = 1.61\%$.
(C) CP/LT, $f^* = 0.6643$, $\Delta f^* = -0.03\%$.
(D) PLT, $f^* = 0.6584$, $\Delta f^* = -0.92\%$.

FIGURE 15 The optimal designs, objectives f^*, and relative changes in objective Δf^* for the flow problem found using the GT, CP, LT, CP/LT, PLT algorithms. For $K = 8$ intervals, no weak global stability problems are encountered and all designs and objective values are similar. GT and CP.



(A) The objective convergence (B) The sensitivity error convergence

FIGURE 16 The objective and sensitivity error convergence of the flow designs in Figure 15. The GT and CP algorithms both compute exact sensitivities and show the same convergence behavior. After the initial fluctuations in design and objective during the first 30 iterations, the design stabilizes and the sensitivity errors reduce.

sensitivities and computes exact sensitivities in the first 34 iterations as can be seen in Figure 16B, it follows the same convergence path as the GT and CP algorithms during these iterations. Furthermore, during the first 20–30 iterations, the design is subject to significant change and sensitivity errors remain high as shown in Figure 16B. This coincides with the relatively high and fluctuating objective values found in Figure 16A. Once the topology stabilizes and only the shape of the design changes, the objective and sensitivity errors start to decrease. Due to the nonlinearity of the equations, the objective and sensitivity errors do not decrease monotonically and may fluctuate significantly for both exact and approximate algorithms. However, given sufficient design iterations, the objective generally stabilizes and sensitivity errors reduce.

Subsequently, we examine and compare the computational cost of the algorithms. Firstly, the relative computational cost of the CP algorithm in Equation (27) is investigated. In Figure 17, we show three wall times: the time to solve the state equations t_{state}, the time to update the adjoint while simultaneously updating sensitivities t_{adj}, and $t_{\text{solv}} = t_{\text{state}} + t_{\text{adj}}$. We note that for t_{state} and t_{adj} we summed the wall times for each of the 100 design iterations in the optimization process. Wall times $t_{\text{state}} \propto c_s$ or $t_{\text{adj}} \propto c_a$ are used as measures of the computational cost of solving state or solving adjoint and computing sensitivities, respectively. Using the measured wall time of the GT algorithm in Figure 17A, we compute the relative computational cost as: $r_c = t^{\text{GT}}_{\text{adj}}/t^{\text{GT}}_{\text{state}} = 14.81/149.0 \approx 0.10$. Solving the state equations is more expensive than solving the adjoint equations as shown in Figure 17B. We compare the derived relative cost of the CP algorithm in Equation (27) to the measured relative cost, respectively, as:

$$\frac{C^{\text{CP}}}{C^{\text{GT}}} \approx \frac{2 + r_c - 1/K}{1 + r_c} = 1.80, \qquad \frac{C^{\text{CP}}_{\text{comp}}}{C^{\text{GT}}_{\text{comp}}} = \frac{t^{\text{CP}}_{\text{solv}}}{t^{\text{GT}}_{\text{solv}}} = \frac{308.4}{163.8} = 1.88, \tag{63}$$

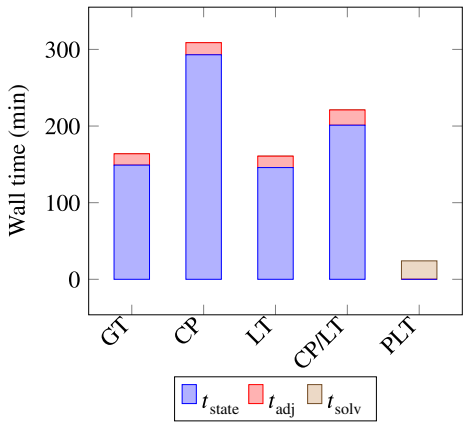
which shows a relatively low error of 8% between our approximation and the measure relative cost. Furthermore, t_{solv} is mainly increased by a growing t_{state} as shown in Figure 17B. Reducing the memory by using more intervals significantly increases the computational time required for the CP algorithm.

Secondly, we compare the computational cost of the LT algorithm to the GT and CP algorithms. Notably, the LT algorithm performed similarly to the GT algorithm. The LT algorithm is even slightly faster than the GT algorithm. This may be attributed to the fact that a different convergence path is followed by the LT algorithm as shown in Figure 16A with the objective convergence graph and in Figure 14B with the slightly different design layout. Different designs have different flow solutions which may take more or less time to solve.

Thirdly, the computational cost of the CP/LT algorithm is analyzed. We expected the CP/LT algorithm to have a cost between the GT and CP algorithms. Figure 18 depicts the corrected intervals during the CP/LT run. During the first 34 design iterations, all intervals except the last are corrected. Hereafter, the number of corrected intervals reduced and after 39 design iterations, no intervals were corrected. Even though for 39 of the 100 design iterations corrections are performed and $f_c = 0.38$, the computational cost remains lower than the cost of the CP algorithm, as the cost of computing the state

	GT	CP	LT	CP/LT	PLT
t_{solv} (min)	163.8	308.4	160.3	220.9	24.07
t_{state} (min)	149.0	293.0	145.8	201.1	—
t_{adj} (min)	14.81	15.37	14.54	19.77	—

(A) The measured wall times.



(B) A visualization of the wall times.

FIGURE 17 The wall times t_{solv}, t_{state} and t_{adj} for the designs in Figure 15. The PLT algorithm was ran on eight CPUs instead of one and consequently has a much lower t_{solv}. For the PLT algorithm some workers might still be working on the state solution while others are already working on the adjoint and we thus measure t_{solv} as the time from the start until the end of the sensitivity computation.

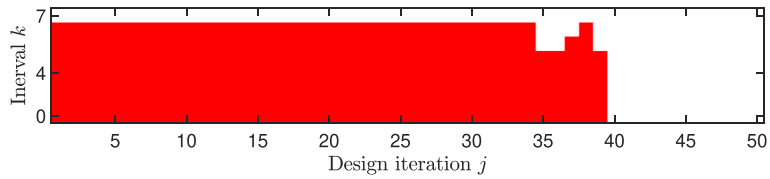


FIGURE 18 The intervals Θ_k, that require corrections for the CP/LT algorithm are flagged in red. Compared to the corrected intervals for the thermal problem in Figure 12, more corrections are performed. This is caused by a slower stabilization of the design and more significant changes in design over the first 30 iterations. After a stable topology is found, errors in adjoint reduce over design iterations 30–39. Since more corrections are performed in the 37th and 38th designs than in the 35th and 36th designs, adaptively correcting intervals helps in maintaining low sensitivity errors when design changes are large and disturb sensitivities.

solution is dominant and the cost of performing the adjoint corrections is relatively low ($r_c \approx 0.10$). We verify the relative computational cost of CP/LT to CP in Equation (29) to the measured cost, respectively, as:

$$\frac{C^{\text{CP/LT}}}{C^{\text{CP}}} \approx \frac{K-1}{K}(f_c+1)\frac{1+r_c}{2+r_c}+\frac{1}{K} = 0.78, \qquad \frac{C^{\text{CP/LT}}_{\text{comp}}}{C^{\text{CP}}_{\text{comp}}} = \frac{t^{CP/LT}_{solv}}{t^{\text{CP}}_{solv}} = \frac{220.9}{308.4} = 0.72, \tag{64}$$

which show that our measurements are in agreement with our approximations. As expected, although the CP/LT algorithm is more expensive than the LT algorithm, it outperforms the CP algorithm.

Finally, we compare the PLT algorithm to the other algorithms. While all other algorithms are run on one CPU, the PLT algorithm is run on eight CPUs. Furthermore, for the PLT algorithm some workers may still be working on the state solution while others are already working on the adjoint solution. Consequently, we do not measure t_{state} or t_{adj} separately. For the PLT algorithm we measure wall time t_{solv} starting when sending initial states and terminal adjoint to all intervals, and ending after retrieving terminal states, initial adjoints, and partial sensitivities $dF^*_k/d\boldsymbol{s}$, which are combined into total sensitivities $dF^*/d\boldsymbol{s}$. Each of the $K = 8$ intervals is attributed to one CPU. Since the PLT algorithm was allowed to use more computational power, the measured wall times are not a fair comparison of computational cost. Moreover, comparing the speedup with respect to the GT algorithm, we find it to be $t^{\text{GT}}_{\text{solv}}/t^{\text{PLT}}_{\text{solv}} \approx 6.8$. The speedup does not scale linearly with the computational power, which will be further examined in Section 5.2.4. However, as illustrated in Figure 17B, the increased computational power resulted in a large decrease in computational time.

5.2.4 | PLT speedup

In the previous section, we found that the speedup of the PLT algorithm does not scale linearly with the number of intervals and computational workers K. In this section, we will further examine the computational speedup and causes of slowdown for the PLT algorithm. There are two main causes for computational slowdown: communication overhead and load balancing. Communication overhead is defined as the idle time of computational workers waiting for data transfer. Load balancing issues are caused by an unequal distribution of tasks over computational workers, resulting in idle times on the workers with cheap computational tasks.

The speedup is examined for $K = 2, 4, 8, 16, 32, 48$ intervals. We investigate the designs and objectives to verify if the results can be used for a fair comparison of speedup. Two designs significantly differ in shape and objective value from the GT design. The $K = 2$ and $K = 48$ designs, as shown in Figure 19, display a more than 10% lower performance compared to the GT design. Moreover, the different designs have different flow solutions which lead to different computational costs. We measure the average computational time over five state solves in the optimal GT and PLT $K = 2$, $K = 48$ designs as $\bar{t}_{\text{state}} = 86.0$, $\bar{t}_{\text{state}} = 84.6$, and $\bar{t}_{\text{state}} = 92.5$ s, respectively. The $K = 48$ design requires significantly more time than the GT design. Moreover, the $K = 48$ design needs 20 additional iterations to converge due to global stability issues as we approach the predicted stability limit in Equation (62) of $K = 50$ intervals. Comparing the speedup between two different designs is inherently unfair and is a limitation of our methods to measure speedup. However, since the $K = 4, 8, 16, 32$ designs are similar in shape and objective to the GT design, a comparison is carried out based on these results.

First, we examine the slowdown due to communication overhead. We measure t_{solv} starting when sending initial states and terminal adjoints to the intervals and ending when the final sensitivities $dF^*/d\boldsymbol{s}$ are found, and $\hat{t}_{\text{solv}}$ the wall time

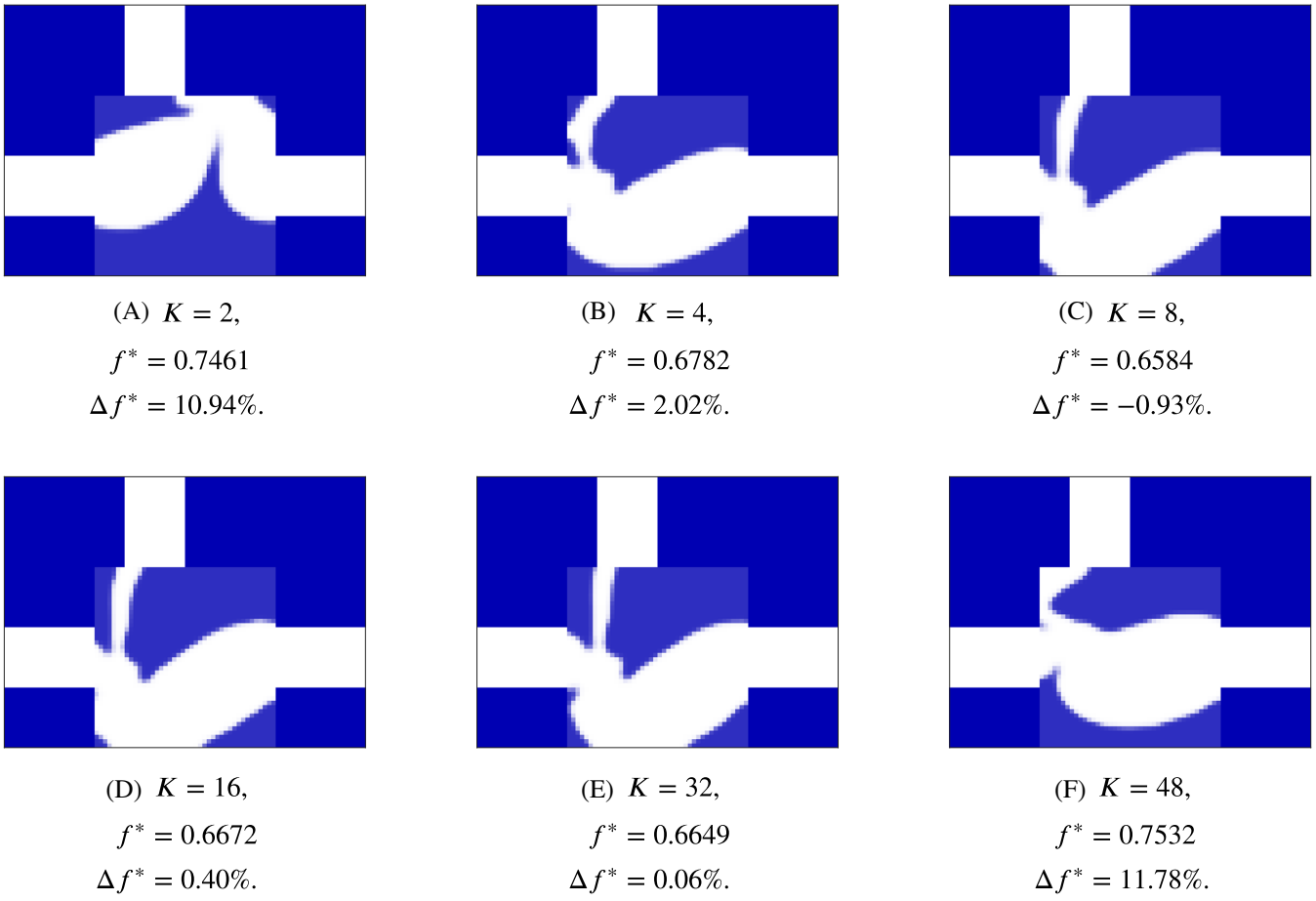


FIGURE 19 The designs, optimized objectives f^*, and relative changes in objective Δf^* computed using the PLT algorithm with varying number of intervals K. Two different designs are found for $K = 2$ and $K = 48$ compared to the designs for $K = 4, 8, 16, 32$ which are similar to the GT design in Figure 15A. Since different designs have different nonlinear flow solutions, computational times differ.

TABLE 9 The wall time t_{solv}, percentage of slowdown caused by overhead ($t_{\text{ov}}/t_{\text{solv}}$), and load balancing (LBS) for using an increasing number of CPUs.

K	1 (GT)	2	4	8	16	32	48
t_{solv} (min)	163.8	90.60	43.87	24.00	13.07	7.68	6.90
$t_{\text{ov}}/t_{\text{solv}}$	—	0.162%	0.269%	0.574%	1.65%	3.48%	4.52%
LBS	—	0.560%	1.95%	8.23%	11.5%	23.7%	29.2%

Notes: All CPUs are of the same type and are attributed one of the K intervals used in the PLT algorithm. Moreover, after measuring the communication overhead as in Equation (65), we find it to have a relatively small contribution to slowdown compared to the LBS.

on the slowest worker to solve the state and adjoint equations and combine them into partial sensitivities $dF_k^*/d\boldsymbol{s}$. On the slowest worker, we start the time measurement after initial states and terminal adjoint are received and end before sending terminal states, initial adjoints, and partial sensitivities. We note that t_{solv} and $\hat{t}_{\text{solv}}$ are summations of the wall times over all design iterations j. Subsequently, the overhead time is defined as the difference in wall times between the complete computation and the computation on the slowest worker:

$$t_{\text{ov}} = t_{\text{solv}} - \hat{t}_{\text{solv}}. \tag{65}$$

Table 9 provides the percentage of slowdown $t_{\text{ov}}/t_{\text{solv}}$ caused by communication overhead and shows that communication overhead is negligible for low number of intervals and increases with the number of intervals. For $K = 48$, 4.52% of the time is spent on communication. The increased communication overhead is caused by the relatively short interval lengths

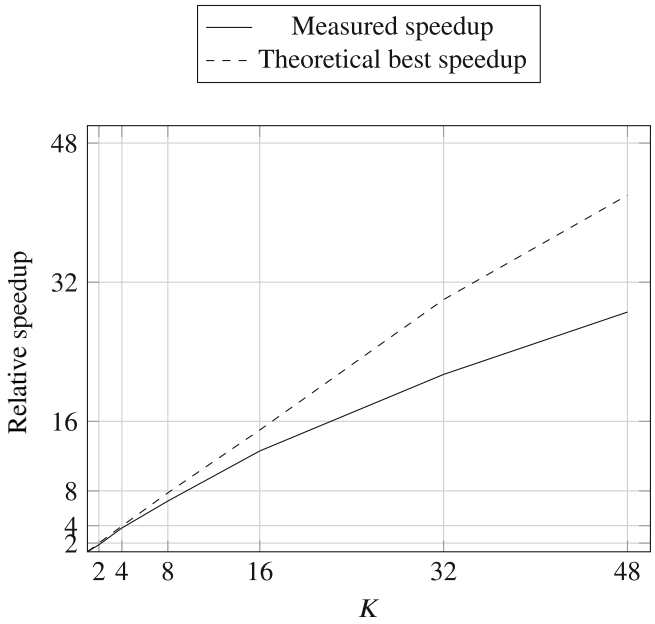


FIGURE 20 The speedup $t^{\mathrm{GT}}_{\mathrm{solv}}/t^{\mathrm{PLT}}_{\mathrm{solv}}$ of the PLT algorithm. The theoretical best speedup does not scale linearly with the number of intervals K due to the discrete interval lengths. Moreover, solving nonlinear equations finding the solution on some intervals requires more computational work as shown in Figure 21, which causes the slowdown found in Table 9. We note that the additional 20 design iterations were taken into account to compute the speedup for $K = 48$ by averaging the time for sensitivity computation and computing the speedup as $(120 \cdot t^{\mathrm{GT}}_{\mathrm{solv}})/(100 \cdot t^{\mathrm{PLT}}_{\mathrm{solv}})$.

of only four or five time steps for $K = 48$. The PLT algorithm suffers from similar problems as common parallel domain decomposition with small computational domains leading to large communication overhead.

Second, we investigate the slowdown due to load balancing. In Figure 20, we plot the relative speedup $t^{\mathrm{GT}}_{\mathrm{solv}}/t^{PLT}_{\mathrm{solv}}$. Additionally, we compute a theoretical best speedup which does not scale linearly due to the first load balancing issue. Dividing the $N = t_t/\Delta t = 210$ time steps computed using Table 7 by the $K = 48$ intervals, we find that the intervals either contain four or five time steps. The theoretical best speedup we can achieve is thus $210/5 = 42$ and not 48. Similar limits on the speedup are computed for $K = 4, 8, 16, 32$. The first load balancing issue is caused by discrete interval lengths.

In Figure 20 we find a lower performance than the theoretical best speedup, which cannot be fully explained by communication overhead. The additional decrease in speedup is caused by the nonlinear flow solution being more expensive on some intervals than on others. To measure the cost of solving the state equations on interval Θ_k in design iteration j, we measure the wall time for the state computation $t^{k,j}_{\mathrm{state}}$ locally on each CPU. Subsequently, we measure the average time for solving the state equations on an interval at a given design as:

$$\bar{t}^{j}_{\mathrm{state}} = \frac{\sum_{k=1}^{K} t^{k,j}_{\mathrm{state}}}{K}. \tag{66}$$

In the optimal scenario, each interval would have the same computational work and would spend the same time ($\bar{t}^{j}_{\mathrm{state}}$) on the state solution. Figure 21 shows a map of the relative computational cost of the intervals, compared to the average computational cost of the intervals $(t^{k,j}_{\mathrm{state}} - \bar{t}^{j}_{\mathrm{state}})/\bar{t}^{j}_{\mathrm{state}}$ for $K = 16$. The more expensive intervals take 20% longer than the cheaper ones, leading to a large decrease in speedup. We note that this measured increase may also result from the load balancing issues due to discrete intervals beside the issues due to expensive time steps. Subsequently, we measure load balancing slowdown (LBS) by comparing the optimal wall time $\bar{t}^{j}_{\mathrm{state}}$ with the wall time spend on solving the state solution on the slowest worker $\hat{t}^{j}_{\mathrm{state}}$, and average over design iterations j as:

$$\mathrm{LBS} = \frac{\sum_j (\hat{t}^{j}_{\mathrm{state}} - \bar{t}^{j}_{\mathrm{state}})}{\sum_j \bar{t}^{j}_{\mathrm{state}}}. \tag{67}$$

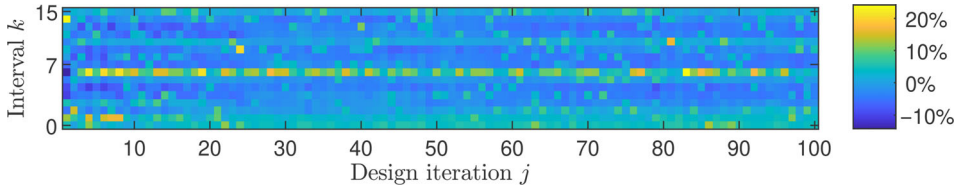


FIGURE 21 The relative cost of solving the state solution on an interval for $K = 16$. For each design iteration j we measure the time to solve the state equations on Θ_k as $t^{k,j}_{\text{state}}$ and compute the average $\bar{t}^{k}_{\text{state}}$ following Equation (66). Subsequently, we normalize the time required to solve the state equations on an interval $(t^{k,j}_{\text{state}} - \bar{t}^{j}_{\text{state}})/\bar{t}^{j}_{\text{state}}$. We find that certain intervals are more expensive to solve than others throughout the optimization procedure. Similar observations were made for different values of K.

In Table 9, the LBS can be found to be relatively large compared to the communication overhead. Load balancing issues are thus the main obstacle preventing speedup in the PLT algorithm for this example. When dealing with simulations that show less variation between intervals, this problem is expected to vanish.

6 | GUIDELINES FOR ALGORITHM SELECTION

Five algorithms for adjoint sensitivity computation are examined in this paper and their advantages and limitations are discussed. Using our results and experience, we formulate guidelines for the selection of an appropriate algorithm based on the characteristics of the transient optimization problem to be solved. A topic not discussed in the main body of this paper was implementation. All algorithms consist of sensitivity computations on intervals. The GT algorithm computes the complete sensitivity on the complete time interval at once. All other algorithms compute partial sensitivities on subintervals separately. To compute the sensitivity on any interval, an initial state, terminal adjoint, design, and (time dependent) boundary conditions are required. A piece of code that can compute partial sensitivities on an interval can thus be used for all algorithms. Even the adjoint correction in the CP/LT algorithm is a simplified version of the sensitivity computation on an interval. Following this approach implementing all the CP, LT, CP/LT and PLT algorithms should not take more time than implementing the GT algorithm. Implementation time is thus not included in the guidelines for algorithm selection.

As discussed in Section 1, only when boundary condition are complex and change drastically over time, or when nonlinear physics are considered should the algorithms in this work be used. Subsequently, algorithms should be compared on three properties discussed in this paper: (1) memory requirements, (2) computational cost, and (3) convergence behavior. A decision tree taking into account these properties is given in Figure 22 and is discussed in the remainder of this section.

When memory requirements are not an issue and the complete state equations may be stored, we recommend to choose between the GT and PLT algorithms and move to the left part of the decision tree in Figure 22. To select an algorithm, the computational cost and availability of a parallel architecture are considered. When the overall computational cost is low, resulting in short computational times, the GT algorithm is the safest choice as it does not suffer from global stability problems. If computational cost is high and a parallel computation architecture is available, we can consider the PLT algorithm. However, the PLT algorithm should only be used if weak global convergence, as discussed in Section 3.1.1, is satisfied. For this we found a weak global convergence requirement in Equation (43), the length of an interval $\Delta\theta = t_t/K$ is required to be longer than one tenth of the characteristic time $t_t/K > \tau/10$. If weak global convergence is not satisfied, we take the risk of ending up in an inferior local optimum with features of relatively short characteristic times, as illustrated in Sections 5.1.4 and 5.2.4. It is up to the user to approximate characteristic times and to weigh the speedup of the PLT algorithm against the possibility of inferior local optima. Moreover, if the user upgrades their computational system to a parallel system for speedup when setting up the optimization problem, available memory is also increased. Subsequently, the user should reevaluate the question of memory limitations, and the possibility of using the PLT algorithm, which has the authors' preference for its low computational time.

If memory limitations are constraining, the user should turn to the right part of the decision tree in Figure 22 and choose between the CP, LT, and CP/LT algorithms. We assume the smallest number of intervals K to satisfy the memory constraints are used. For the LT and CP/LT algorithms, we use the lowest K possible as this improves convergence behavior, for the CP algorithm the lowest K reduces computational cost. Selecting the appropriate algorithm depends on both

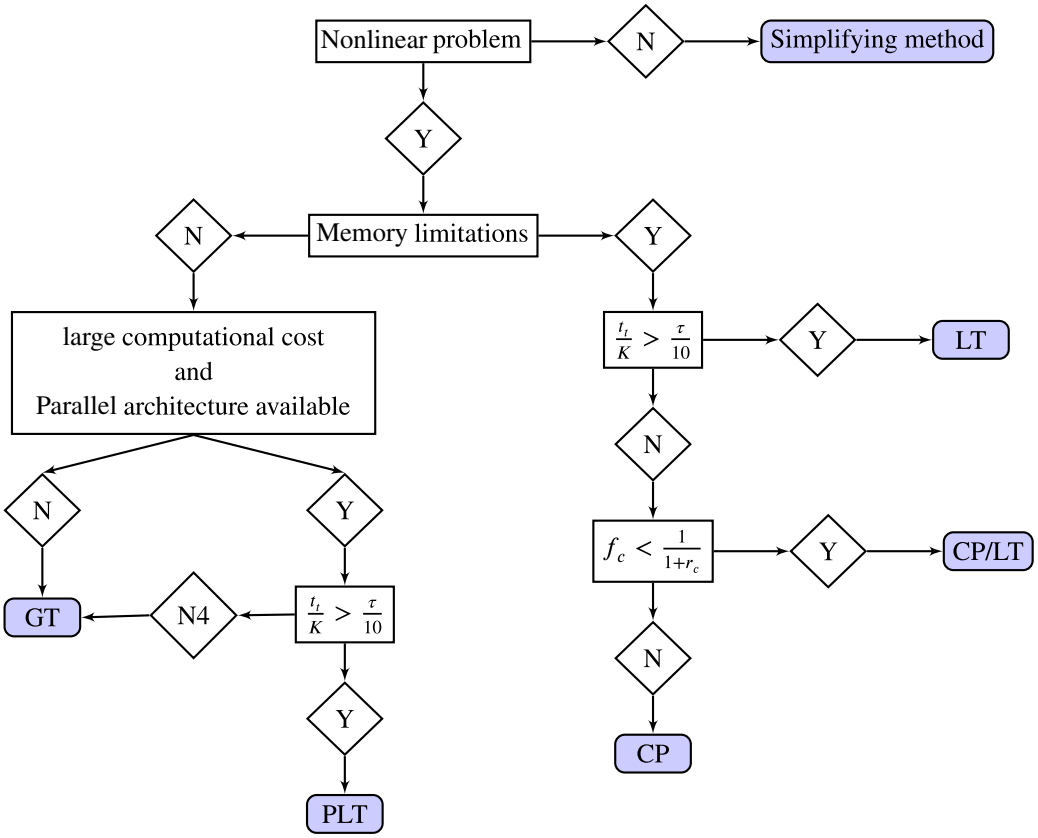


FIGURE 22 A decision tree for selecting the correct transient sensitivity computation algorithm. Decisions are based on memory requirements, computational cost and algorithmic stability. To analyze stability, characteristic time τ needs to be estimated. To compare computational cost of the CP/LT and CP algorithms, an estimation of the number of corrected intervals f_c, and relative computational cost r_c needs to be made.

computational cost and convergence behavior. As predicted in Section 4.2 and shown in Sections 5.1.5 and 5.2.3, the LT algorithm is generally the fastest followed by the CP/LT or CP algorithm. The exception is the convergence of the thermal problem using LT with $K = 2$ in Section 5.1.5. However, we included exactly these results as they illustrate this exception, but did not generally find this type of behavior for the LT algorithm. If weak global convergence is satisfied ($t_t/K > \tau/10$) we use the LT algorithm.

If memory requirements impose limitations and weak global convergence is not guaranteed, we need to choose between the CP/LT and CP algorithms. Selecting the appropriate algorithm depends on computational cost. In Equation 29 we show the relative computational cost of the CP/LT algorithm to the CP algorithm. The relative cost is dependent on the number of intervals K, the relative cost of the adjoint and sensitivity computation c_a to the state computation c_s: $r_c = c_a/c_s$, and an estimate for the percentage of corrected intervals f_c. To compare the computational cost of the CP/LT and CP algorithms, we thus need to compute at what percentage of corrected intervals the CP/LT algorithm becomes more expensive than the CP algorithm. Subsequently, the choice of algorithm is based on a prediction of f_c. We use the CP/LT algorithm when the ratio $C^{\mathrm{CP/LT}}/C^{\mathrm{CP}}$ from Equation (29) is lower than one, resulting in:

$$f_c < \frac{1}{1 + r_c}, \tag{68}$$

which allows us to decide between the CP and CP/LT algorithm in Figure 22. Relative cost r_c can be easily approximated by running the optimization procedure for only one iteration and measuring the time spent on adjoint and state equations as illustrated in Sections 5.1.5 and 5.2.3. However, for estimating f_c some insight into the problem is required. As shown in Figures 12 and 18, only during the first design iterations when design changes are large are corrections necessary. An experienced designer may thus estimate the number of design iterations associated with large design changes and the

number of iterations required to converge. Subsequently, an estimation of f_c can be made a priori, and an informed choice between the CP/LT and CP can be made. Furthermore, we note that for highly nonlinear systems where $r_c \to 0$, we find $f_c < 1$ and may thus always use the CP/LT algorithm.

7 | DISCUSSION AND CONCLUSION

In this work we thoroughly examined five algorithms for transient sensitivity computation in topology optimization. Two of these algorithms are new, and proposed for the first time in this paper. This has been motivated by the fact that in topology optimization of transient problems, memory- and time-efficient adjoint sensitivity analysis is crucial. The algorithms have been compared in terms of memory requirements, computational cost, and stability. Firstly, we examined the state-of-the-art GT, CP, and LT algorithms. The GT algorithm serves as the reference and suffers from severe memory limitations which motivates the development of new algorithms. To overcome these memory limitations, the most common approach is the CP algorithm. Although the CP algorithm reduces memory usage, it significantly increases the required computational time due to recomputation of the state solutions, as shown in Section 5.2.3. To avoid recomputation of the state solutions, the LT algorithm may be used. However, due to the approximations of adjoints and consequent stability and convergence issues described in Section 3.1.1 and illustrated in Section 5.1.4, using the LT algorithm may compromise the optimization outcome. To solve these stability issues, we introduced the hybrid CP/LT algorithm which does not show the convergence issues of the LT algorithm at the cost of an increased computational time. However, the hybrid algorithm showed a clear reduction in computational time compared to the CP algorithm.

We point out the importance of understanding the physics and characteristic times of the optimization problem for selecting an appropriate optimization approach. First of all, only when the characteristic timescale is long enough compared to the optimization time horizon or when sufficiently complex physics/load cases are examined do we benefit from the algorithms examined in this work. If the characteristic timescale is short compared to the time horizon of the optimization problem, ESL algorithms can be used, and when the physics/load cases are simple enough, model order reduction techniques can be considered. Secondly, comparing the characteristic time to the interval lengths associated to the LT algorithm, it can be determined whether stability and convergence issues arise and the hybrid CP/LT algorithm is required. When interval lengths are an order of magnitude smaller than the estimated characteristic time of the optimization problem, convergence issues can arise in the (P)LT algorithms.

To address the challenge of computational time in transient optimization, the novel PLT algorithm was proposed. In essence, the PLT algorithm is an extension of the LT algorithm where not only adjoints, but also states are approximated during optimization. Consequently, the PLT algorithm also suffers from similar stability and convergence problems as the LT algorithm. However, by inspecting the characteristic time of the optimization problem, stability and convergence issues may be identified a priori. This results in an upper limit on the number of intervals K. Although the PLT algorithm reduces computational time, it does not reduce memory requirements. However, when computational power is scaled up, the total available memory is also often increased.

For future research, we note the possibility to apply spatial domain decomposition techniques for further acceleration when speedup using the PLT algorithm is limited by stability and convergence constraints. A comparison between domain decomposition techniques and the PLT algorithm was not included in this work and is recommended. Similar to domain decomposition techniques, the PLT algorithm suffers from communication overhead when the decomposition becomes too fine. In addition, for problems with a varying cost per time step (e.g., nonlinear problems), the efficiency of the PLT algorithm is affected by load balancing issues. This presents an opportunity for the development of algorithms which address these load balancing issues by adaptively increasing the interval length of cheap intervals and decreasing the length of expensive intervals. Furthermore, another subject for further research is the implementation of a hierarchical approach to reduce the memory requirements of the PLT algorithm. In such an approach, the intervals used by the PLT algorithm can be further divided into subintervals which are used to locally apply the LT, CP, or CP/LT algorithm.

ACKNOWLEDGMENTS

We would like to acknowledge Svanberg[40] for the MMA optimizaiton code written in MATLAB used in this report.

CONFLICT OF INTEREST STATEMENT

The authors declare no potential financial or non-financial conflict of interests.

DATA AVAILABILITY STATEMENT

The code that supports the findings of this study are available from the corresponding author upon reasonable request.

ORCID

M. J. B. Theulings https://orcid.org/0000-0002-1545-844X
F. van Keulen https://orcid.org/0000-0003-2634-0110
M. Langelaar https://orcid.org/0000-0003-2106-2246

REFERENCES

1. Alexandersen J, Andreasen CS. A review of topology optimisation for fluid-based problems. *Fluids*. 2020;5(1):29.
2. Li Y, Saitou K, Kikuchi N. Topology optimization of thermally actuated compliant mechanisms considering time-transient effect. *Finite Elem Anal Des*. 2004;40(11):1317-1331.
3. Wu S, Zhang Y, Liu S. Topology optimization for minimizing the maximum temperature of transient heat conduction structure. *Struct Multidiscipl Optim*. 2019;60(1):69-82.
4. Nørgaard S, Sigmund O, Lazarov B. Topology optimization of unsteady flow problems using the lattice Boltzmann method. *J Comput Phys*. 2016;307:291-307.
5. Svanberg K. The method of moving asymptotes—a new method for structural optimization. *Int J Numer Methods Eng*. 1987;24(2):359-373.
6. Haftka RT. Techniques for thermal sensitivity analysis. *Int J Numer Methods Eng*. 1981;17(1):71-80.
7. Kang B, Choi W, Park G. Structural optimization under equivalent static loads transformed from dynamic loads based on displacement. *Comput Struct*. 2001;79:145-154.
8. Choi W, Park G. Structural optimization using equivalent static loads at all time intervals. *Comput Methods Appl Mesh Eng*. 2002;191:2077-2094.
9. Hooijkamp EC, Fv K. Topology optimization for linear thermo-mechanical transient problems: modal reduction and adjoint sensitivities. *Int J Numer Methods Eng*. 2018;113(8):1230-1257.
10. Qian X. On-the-fly dual reduction for time-dependent topology optimization. *J Comput Phys*. 2022;452:110917.
11. Li Q, Sigmund O, Jensen JS, Aage N. Reduced-order methods for dynamic problems in topology optimization: a comparative study. *Comput Methods Appl Mech Eng*. 2021;387:114149.
12. Zhao J, Wang C. Dynamic response topology optimization in the time domain using model reduction method. *Struct Multidiscipl Optim*. 2016;53(1):101-114.
13. Griewank A. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optim Methods Softw*. 1992;1(1):35-54.
14. Griewank A, Walther A. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans Math Softw*. 2000;26(1):19-45.
15. Grimm J, Pottier L, Rostaing-Schmidt N. Optimal time and minimum space-time for reversing a certain class of prorgams. *Comput Diff Techn Appl Tools*. 1996;161-172. https://inria.hal.science/inria-00073896/
16. Heuveline V, Walther A. Online checkpointing for parallel adjoint computation in PDEs: application to goal-oriented adaptivity and flow control. In Heuveline V, Walther A, (eds), *European Conference on Parallel Processing*. Springer; 2006:689-699.
17. Wang Q, Moin P, Iaccarino G. Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. *SIAM J Sci Comput*. 2009;31(4):2549-2567.
18. Yamaleev NK, Diskin B, Nielsen EJ. Local-in-time adjoint-based method for design optimization of unsteady flows. *J Comput Phys*. 2010;229(14):5394-5407.
19. Chen C, Yaji K, Yamada T, Izui K, Nishiwaki S. Local-in-time adjoint-based topology optimization of unsteady fluid flows using the lattice Boltzmann method. *Mech Eng J*. 2017;4(3):17-00120.
20. Yaji K, Ogino M, Chen C, Fujita K. Large-scale topology optimization incorporating local-in-time adjoint-based method for unsteady thermal-fluid problem. *Struct Multidiscipl Optim*. 2018;58(2):817-822.
21. Carraro T, Geiger M. Direct and indirect multiple shooting for parabolic optimal control problems. In Carraro T, Geiger M, (eds), *Multiple Shooting and Time Domain Decomposition Methods*. Springer; 2015:35-67.
22. Fang L, Vandewalle S, Meyers J. A parallel-in-time multiple shooting algorithm for large-scale PDE-constrained optimal control problems. *J Comput Phys*. 2022;452:110926.
23. Borrvall T, Petersson J. Large-scale topology optimizaiton in 3D using parallel computing. *Comput Methods Appl Mech Eng*. 2002;190:6201-6229.
24. Kristiansen H, Aage N. An open-source framework for large-scale transient topology optimization using PETSc. *Struct Multidisc Optim*. 2022;65(10):1-15.
25. Mahdavi A, Balaji R, Frecker M, Mockensturm E. Topology optimiziton of 2D continua for minimum compliance using parallel computing. *Struct Multidisc Optim*. 2006;32:121-132.
26. Aage N, Poulsen T, Gersborg-Hansen A, Sigmund O. Topology optimization of large scale stokes flow problems. *Struct Multidisc Optim*. 2008;35:175-180.
27. Gander MJ. 50 years of time parallel time integration. *Multiple Shooting and Time Domain Decomposition Methods*. Springer; 2015:69-113.

28. Versteeg H, Malalasekera W. *Computational Fluid Dynamics*. Pearson Education Limited; 1995.
29. Picioreanu C, Van Loosdrecht MC, Heijnen JJ. Effect of diffusive and convective substrate transport on biofilm structure formation: a two-dimensional modeling study. *Biotechnol Bioeng*. 2000;69(5):504-515.
30. Åström KJ, Murray RM. *Feedback Systems: an Introduction for Scientists and Engineers*. 2nd ed. Princeton University Press; 2020.
31. Lea DJ, Allen MR, Haine TW. Sensitivity analysis of the climate of a chaotic system. *Tellus A Dyn Meteorol Oceanogr*. 2000;52(5):523-532.
32. Wang Q, Hu R, Blonigan P. Least squares shadowing sensitivity analysis of chaotic limit cycle oscillations. *J Comput Phys*. 2014;267:210-224.
33. Bendsoe MP, Sigmund O. *Topology Optimization: Theory, Methods, and Applications*. Springer Science & Business Media; 2003.
34. Theulings MJB, Langelaar M, Keulen v F, Maas R. Towards improved porous models for solid/fluid topology optimization. *Struct Multidisc Optim*. 2023;66(6):133.
35. MATLAB. *Version 9.6.0 (R2019a)*. The MathWorks Inc.; 2019.
36. Gersborg-Hansen A, Bendsøe MP, Sigmund O. Topology optimization of heat conduction problems using the finite volume method. *Struct Multidisc Optim*. 2006;31(4):251-259.
37. Bruns TE, Tortorelli DA. Topology optimization of non-linear elastic structures and compliant mechanisms. *Comput Methods Appl Mech Eng*. 2001;190(26-27):3443-3459.
38. Wang F, Lazarov BS, Sigmund O. On projection methods, convergence and robust formulations in topology optimization. *Struct Multidisc Optim*. 2011;43:767-784.
39. Bendsøe MP, Sigmund O. Material interpolation schemes in topology optimization. *Arch Appl Mech*. 1999;69:635-654.
40. Svanberg K. *Some Modelling Aspects for the MATLAB Implementation of MMA*. KTH Royal Institute of Technology; 2004.

How to cite this article: Theulings MJB, Maas R, Noël L, van Keulen F, Langelaar M. Reducing time and memory requirements in topology optimization of transient problems. *Int J Numer Methods Eng*. 2024;125(14):e7461. doi: 10.1002/nme.7461