

Symbolic construction of the chemical Jacobian of quasi-steady state (QSS) chemistries for Exascale computing platforms

Hassanally, Malik; Wimer, Nicholas T.; Felden, Anne; Esclapez, Lucas; Ream, Julia; Henry de Frahan, Marc T.; Rood, Jon; Day, Marc

DOI

[10.1016/j.combustflame.2024.113740](https://doi.org/10.1016/j.combustflame.2024.113740)

Publication date

2024

Document Version

Final published version

Published in

Combustion and Flame

Citation (APA)

Hassanally, M., Wimer, N. T., Felden, A., Esclapez, L., Ream, J., Henry de Frahan, M. T., Rood, J., & Day, M. (2024). Symbolic construction of the chemical Jacobian of quasi-steady state (QSS) chemistries for Exascale computing platforms. *Combustion and Flame*, 270, Article 113740. <https://doi.org/10.1016/j.combustflame.2024.113740>

Important note

To cite this publication, please use the final published version (if applicable). Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights. We will remove access to the work immediately and investigate your claim.

Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

<https://www.openaccess.nl/en/you-share-we-take-care>

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.



Symbolic construction of the chemical Jacobian of quasi-steady state (QSS) chemistries for Exascale computing platforms

Malik Hassanaly^{a,*}, Nicholas T. Wimer^a, Anne Felden^b, Lucas Esclapez^a, Julia Ream^c, Marc T. Henry de Frahan^a, Jon Rood^a, Marc Day^a

^a Computational Science Center, National Renewable Energy Laboratory (NREL), Golden, CO 80401, United States of America

^b Process and Energy Department, Delft University of Technology, 2628 CB Delft, The Netherlands

^c Department of Mathematics, Florida State University, Tallahassee, FL 32306, United States of America

ARTICLE INFO

Keywords:

Quasi-steady state chemistry
Analytical Jacobian
High-performance computing

ABSTRACT

The Quasi-Steady State Approximation (QSSA) can be an effective tool for reducing the size and stiffness of chemical mechanisms for implementation in computational reacting flow solvers. However, for many applications, the resulting model still requires implicit methods for efficient time integration. In this paper, we outline an approach to formulating the QSSA reduction that is coupled with a strategy to generate C++ source code to evaluate the net species production rates, and the chemical Jacobian. The code-generation component employs a symbolic approach enabling a simple and effective strategy to analytically compute the chemical Jacobian. For computational tractability, the symbolic approach needs to be paired with common subexpression elimination which can negatively affect memory usage. Several solutions are outlined and successfully tested on a 3D multipulse ignition problem, thus allowing portable application across chemical model sizes and GPU capabilities. The implementation of the proposed method is available at <https://github.com/AMReX-Combustion/PelePhysics> under an open-source license.

Novelty and Significance

A symbolic method is proposed to write analytical chemical Jacobians. The benefit of the symbolic method is that it is easy to implement and flexible to any elementary reaction type. Its benefit is shown in the context of QSS-reduced chemistries: there, constructing an analytical chemical Jacobian is complex since one must include the effect of traditional elementary reactions and algebraic closure for the QSS species. To the authors' knowledge, there is no open-source package available to construct analytical Jacobians of QSS-reduced chemistries. We expect this work to facilitate the use of analytical Jacobians in arbitrarily complex chemical mechanisms. The proposed method was integrated into an open-source suite of reacting flow solvers <https://github.com/AMReX-Combustion/PelePhysics> to facilitate its dissemination.

1. Introduction

Exascale simulations of turbulent reacting flows have recently become possible [1–3] due to a sustained investment in the co-design of modern high performance GPU-based computing architectures and performance-portable simulation software [4–8]. As simulation capabilities grow, so does the complexity of the research questions that can be addressed computationally. It is now possible to consider detailed interactions between turbulence and chemistry in the context of multicomponent and alternative fuels [9,10], novel ignition strategies [11,12], and pollutant formation [13–16]. Enabling these detailed investigations, the chemical kinetic models employed have also grown in size and complexity, exhibiting an increasingly broad range of

time scales and ultimately becoming extremely “stiff” relative to the turbulent flow dynamics. As discussed in Ref. [17], operator-splitting has proven to be an effective framework for coupling stiff chemical kinetics to flow solvers, but even when using the most recent chemistry integration approaches of this type (e.g., [18–21]), reacting flow simulations employing a finite rate chemistry description can spend half or more of the computing resources required for the full simulation in the chemistry component of the solve alone [22]. To mitigate the cost of the chemistry integration, several approaches have been developed. From a computational standpoint, it was shown that the integration of the chemical system can be greatly accelerated through preconditioning approaches [23,24]. In addition, the matrices used to

* Corresponding author.

E-mail address: malik.hassanaly@nrel.gov (M. Hassanaly).

<https://doi.org/10.1016/j.combustflame.2024.113740>

Received 6 May 2024; Received in revised form 8 September 2024; Accepted 12 September 2024

Available online 20 September 2024

0010-2180/© 2024 The Combustion Institute. Published by Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

represent the chemical systems can be strategically sparsified thereby allowing to leverage efficient sparse-linear algebra methods [25]. In addition, the chemical mechanisms integrated can also be simplified [26]. Among the latter strategies, reducing the size of chemical mechanisms based on a-priori knowledge remains a key strategy to mitigate the computational burden of reacting flow simulations [27].

Many chemistry reduction techniques have been developed that identify and retain key kinetic information exhibited in an arbitrary set of idealized combustion scenarios (such as strain-free flame propagation speeds, extinction strain rates, etc.), that are asserted to be relevant to the model's intended application (e.g., see Table 1 in [28]). The first step usually results in a problem-specific "skeletal" mechanism with a smaller number of chemical species and reactions, and one that typically also exhibits reduced stiffness [29–31]. A common strategy for additional reduction of the size and stiffness of these models invokes the Quasi-Steady State Approximation (QSSA) [32,33], i.e. that the production and destruction of well-chosen chemical species are assumed to be everywhere in balance. As discussed in detail later in Section 2.1, the QSSA reduces the dimension of the chemical system; the molar concentrations of the removed species become algebraic constraints on the remaining system. Critically, the QSSA can result in complex non-linear algebraic relations between species for which either cumbersome numerical procedures must be designed, or simplified approximations should be formulated [34]. In the context of simulating turbulent combustion applications, QSSA-reduced chemistries also complicate the construction and evaluation of the chemical Jacobian, which is often required by time-implicit chemistry integration algorithms.

The evaluation of analytic chemical Jacobians has received significant recent attention [35–38], and efforts geared towards the way it is assembled have been recognized as beneficial from a computational standpoint [36,39]. However, fewer efforts have been dedicated to adapting analytic Jacobian construction methods to QSSA-reduced chemistries. Recently, Sharma [40] discussed the construction of analytical Jacobian of QSSA-reduced chemistry and how to approximate it, but did not explain how to encode the dependence of QSSA species on non-QSSA species, which is a significant hurdle we address here. To the authors' knowledge, there are also no currently available analytical chemical Jacobian construction tools for QSSA-reduced chemistry models.

Recent work in the design of exascale-ready codes [4,5] has also highlighted that software maintainability is crucial for long-term sustainability. Therefore, the software implementing the analytical Jacobian of QSS-reduced chemistry models should induce minimal additional complexity in the reacting flow solvers that use them. The novel contributions of this work are as follows:

1. A framework for the generation of the analytical chemical Jacobian of QSS-reduced mechanisms is developed. This capability is openly available in PelePhysics [6,41]. Jacobians of the chemical models are assembled via a symbolic method that significantly simplifies the logic involved in the resulting C++ implementation.
2. In general, evaluating a chemical Jacobian is computationally intensive and requires large amounts of memory. The problem is shown to be even worse when a symbolic method is used. Strategies to mitigate memory and computing time are proposed and shown to perform well and only mildly affect the overall computational cost associated with generating the evaluation routines.

The remainder of this paper is organized as follows. Section 2 discusses the QSSA in depth and describes how interdependencies and nonlinear coupling between QSS species are handled. Section 3 presents an approach to compute and validate the analytic evaluation of the Jacobian for QSSA-reduced chemistry models, and demonstrates increased stability and computational efficiency of implicit integration schemes over alternative approaches for stiff systems. In Section 4,

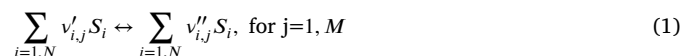
we address the key practical issue of efficient memory use when constructing the chemical Jacobian with a symbolic approach. Section 5 demonstrates the application and scalability of our combined generation and implementation approach on a test problem that models a pulsed-injection ignition. Conclusions are provided in Section 6.

2. Implementation of the QSS-reduced chemistries

In this section, a brief overview of the QSSA is provided. The difficulties caused by QSS species couplings in the construction of a tractable equation set are also discussed.

2.1. Rate laws and the quasi-steady state approximation

A set of N species interacting through M reactions will usually be expressed via a reaction mechanism of the form:



where S_i denotes any species i and $v'_{i,j}$ and $v''_{i,j}$ are the forward and backwards stoichiometric coefficients of species i in reaction j . The progress of reaction j (\dot{Q}_j) is the sum of the forward ($q_{f,j}$) and backward ($q_{b,j}$) contributions, which are a function of the species molar concentrations (C_i) as well as to the forward and reverse reaction rates ($k_{f/b}$):

$$\dot{Q}_j = q_{f,j} - q_{b,j} = k_{f,j} \prod_{i=1,N} C_i^{v'_{i,j}} - k_{b,j} \prod_{i=1,N} C_i^{v''_{i,j}}, \quad (2)$$

Thus, the evolution of species molar concentrations is the result of contributions from a subset of the M reactions, and is typically expressed via a series of ordinary differential equations (ODEs):

$$\frac{dC_i}{dt} = \sum_{j=1,M} v_{i,j} \dot{Q}_j = \dot{\omega}_i(\mathbf{C}, \mathbf{k}), \text{ for } i=1, N \quad (3)$$

where $v_{i,j} = v'_{i,j} - v''_{i,j}$, \mathbf{C} is the N vector of species molar concentrations, \mathbf{k} is a $2M$ vector of reaction rate functions and $\dot{\omega}_i$ is the net production rate of species i .

To reduce the computational burden and stiffness of reactive flow simulations, QSSA can be formulated on a set of N_{QSS} strategically chosen species. Good candidates are species with very short characteristic timescales and low molar concentration throughout the process of interest [34]. The QSS approximation for a species i is expressed as [42]:

$$\frac{dC_i}{dt} = 0, \quad (4)$$

The molar concentrations of QSS species can then be obtained via algebraic relations constructed by inverting this newly constructed system of N_{QSS} equations. Combining Eqs. (2) & (3) and substituting into Eq. (4) can lead to a set of nonlinear algebraic relations, which can involve a large number of species and reaction rates. In particular, a situation that typically arises is when two or more QSS species are on the same side of an elementary reaction: this situation is referred, hereafter, as a quadratic coupling. Quadratic couplings require complex solution strategies that can dominate the implementation and eventually reduce the interest of using QSSA in the first place [43], especially if the number of reactions is large. One linearization approach has been discussed in the past [34], but other linearization methods can be envisioned. For the sake of completeness, different linearization methods are described and compared in the following section. All linearization methods are compatible with a symbolic construction of the analytic Jacobian described in Section 3.

2.2. Nonlinear QSS couplings

Linearization approaches. One approach to avoid quadratic couplings would be to restrict the choice of QSS species to the ones that are not involved in quadratic couplings. A species is said to be involved in a quadratic coupling if it is a reactant in the elementary reaction where a quadratic coupling between QSS species concentrations is found. Here, this approach is referred to as the *species linearization* method. A simple but systematic method to identify the set of QSS species to turn into non-QSS species will be outlined in the following paragraphs. As will be illustrated, this approach often can end up being too restrictive. A second approach to eliminate quadratic couplings is to ignore chemical reactions involving intra-QSS collisions. This approach is rooted in the fact that since QSS species are present in lower molar concentrations than transported species, the probability of observing the reactions that induce quadratic coupling should also be very low. Lu and Law [34] proposed a linearization technique for irreversible reactions based on this observation. In the case when reversible reactions are considered, and following a similar approach as [34], one could ignore the complete reaction to conserve chemical balance, even if only the forward or the backward reaction induces the quadratic coupling. This approach is referred to as the *reaction linearization* method. Finally, a third approach, the *one-sided reaction linearization* method which can be seen as a softer version of the second method, removes only the reaction direction that induces the quadratic coupling. As discussed below, all three of these methods have been implemented in an open-source software distribution available at <https://github.com/AMReX-Combustion/PelePhysics>.

The example of n-dodecane. The three different linearization approaches discussed above are used to reduce a 53-species n-dodecane skeletal mechanism [44], assumed to contain 18 QSS species. This model is represented graphically in Fig. 1. The reaction mechanism is represented as a graph with 2 types of nodes: species (as textboxes) and elementary reactions (as circles). Representing elementary reactions with nodes allows to link more than 2 species to a given reaction. Non-QSS species are grayed out and QSS species that are involved in at least one quadratic coupling are highlighted in red. Quadratic couplings are observed in specific reactions that are also highlighted in red. Other reactions are grayed out. The species linearization method acts upon the red-highlighted QSS species, and both reaction linearization methods act upon the set of red-highlighted reactions.

In the first linearization approach (species linearization method), the objective is to identify a set of QSS species that can eliminate quadratic couplings, if turned into a set of non-QSS species. This set of QSS species must be as small as possible given that every new non-QSS species requires solving an additional transport equation in a reacting flow simulation. A naive but sufficiently fast procedure to identify this set of QSS species is outlined in Algo. 1.

In the case of the n-dodecane mechanism illustrated in Fig. 1, an initial set of candidate of QSS species to eliminate (\mathcal{A} in Algo. 1) contains 11 QSS species. Those species are the red-highlighted species in Fig. 1. If all 11 QSS species are turned into non-QSS species, quadratic couplings are all eliminated. However, not all 11 species need to be turned into non-QSS species to eliminate quadratic couplings. For instance, $\text{PXC}_7\text{H}_{15}$ and $\text{PXC}_5\text{H}_{11}$ and involved in a quadratic coupling through a single elementary reaction $\text{PXC}_7\text{H}_{15} + \text{PXC}_5\text{H}_{11} \rightleftharpoons \text{NC}_{12}\text{H}_{26}$. In this case, either $\text{PXC}_7\text{H}_{15}$ or $\text{PXC}_5\text{H}_{11}$ need to be turned into non-QSS species. Algorithm 1 shows a simple method to identify the minimal sets of QSS species that need to be turned into non-QSS species. Following this procedure, a set of 7 QSS species is identified as the smallest set of QSS species that should be turned into non-QSS species (\mathcal{B} in Algo. 1). Note that while this strategy is simple to implement, it can induce a significant computational overhead on the CFD solvers given that the original 35-species reduced mechanism has now become a 42-species mechanism.

Algorithm 1 Identification of QSS species to eliminate in the species linearization method.

```

1: Identify the set  $\mathcal{A}$ , the set of QSS species involved in at least one
   quadratic coupling, where  $n \equiv \text{card}(\mathcal{A})$ 
2: Initialize an empty list  $\mathcal{B}$ , that holds the candidate sets of QSS
   species that could eliminate quadratic couplings if turned into
   non-QSS species
3: for  $i = 1, n$  do
4:   Create an ensemble  $\mathcal{P}_i$  of subsets of  $i$  species from  $\mathcal{A}$  without
   repetition, i. e.
5:    $\mathcal{P}_i = \{\mathcal{A}_j\}$ , where  $\mathcal{A}_j \subset \mathcal{A}$ , and  $\text{card}(\mathcal{A}_j) = i, \forall j$ .
6:   for  $\mathcal{A}_j \in \mathcal{P}_i$  do
7:     Turn the set of QSS species  $\mathcal{A}_j$  into non-QSS species
8:     if No quadratic couplings then
9:       Append  $\mathcal{A}_j$  to  $\mathcal{B}$ .
10:    end if
11:   Turn back the set of non-QSS species  $\mathcal{A}_j$  into QSS species
12: end for
13: if  $\mathcal{B}$  non empty then
14:   Return  $\mathcal{B}$ 
15: end if
16: end for

```

Table 1

Ignition delay errors obtained with different QSS linearization methods of a reduced n-dodecane mechanism [44,45].

Linearization method	$\epsilon_{\text{ign,SK-QSS}}$
Species linearization	0.848%
Reactions linearization	13.3%
One-sided reaction linearization	8.97%

The other two linearization approaches (the reaction linearization method and the one-sided reaction linearization method) are more straightforward, and rely on eliminating the red-highlighted reactions shown in Fig. 1.

The mechanisms obtained with the three linearization methods are compared to the 53-species skeletal version of the n-dodecane mechanism [45] by computing ignition delays (t_{ign}) for 108 initial chemical states. The 108 initial conditions correspond to 9 initial temperatures evenly distributed in the set $T = [800-1600]$ K, 3 equivalence ratio from the set $\phi = \{0.5, 1, 2\}$, and 4 pressures from the set $P = \{1, 5, 10, 50\}$ atm. These conditions span the validity range of the QSSA for this mechanism [44]. The ignition delay t_{ign} was defined as the earliest time after which the initial temperature increased by at least 400 K. In the absence of ignition, t_{ign} was assigned the arbitrary value of 3 s. To eliminate any considerations regarding the analytic Jacobian construction for now, the results of this Section are obtained by using a Newton method with a Jacobian computed via finite differences [46]. Results are reported in terms of the ignition delay error (ϵ_{ign}), defined as:

$$\epsilon_{\text{ign},i-j} = \mathbb{E}_k \left(\frac{|t_{\text{ign},i,k} - t_{\text{ign},j,k}|}{t_{\text{ign},i,k}} \right), \quad (5)$$

where the error between results from method i and j are compared for the initial condition k , and \mathbb{E}_k denotes the averaging operation over all 108 initial conditions considered. In the present case, $i = \text{SK}$ since results obtained with the skeletal mechanism [45] are taken as a reference, and $j = \text{QSS}$ denotes the use of one of the linearized QSS mechanisms. Values for $\epsilon_{\text{ign,SK-QSS}}$ are shown in Table 1.

It can be seen that the species linearization strategy offers the highest accuracy at the expense of increasing the computational cost (by about 30% compared to the other two mechanisms). Additionally, the one-sided reaction linearization is 33% more accurate over the 108 initial conditions than the more stringent reaction linearization. Given these results, the one-sided reaction linearization option is the preferred

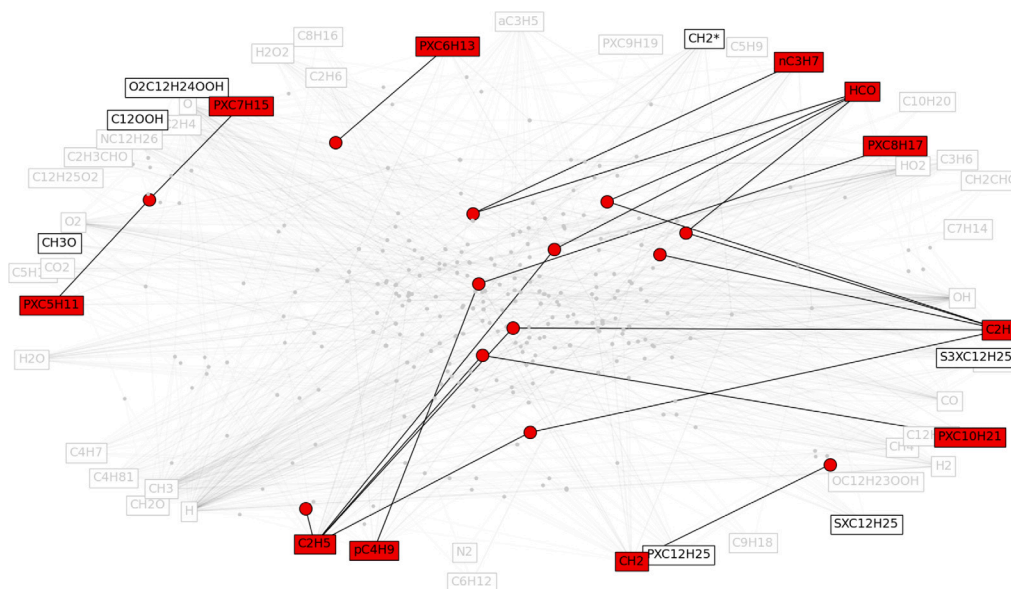


Fig. 1. Graph of the 35-species, QSSA reduced, n-dodecane mechanism [44]. Textboxes correspond to species, and circles correspond to elementary reactions. The 35-non-QSSA are grayed out, and 11 QSS species involved in quadratic couplings are highlighted in red. Red circles highlight the elementary reactions that induce the quadratic couplings. Links between reaction nodes and species nodes are grayed out except for inducing couplings (i.e., links between red-highlighted reaction nodes and red-highlighted species nodes). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

strategy for the rest of this manuscript. The analytical Jacobian construction method described in Section 3 could, however, easily adapt to any of these linearization approaches. All three methods are available in the companion repository.

2.3. Interdependencies between QSS species

After linearization, the system that results from applying Eqs. (3) and (4) must be inverted to formulate the algebraic closure equations for the QSS species molar concentrations. In the case of linear couplings between QSS species, the interdependencies can be represented by a block-diagonal matrix [34], allowing the use of standard solution approaches. The coupling between interdependent QSS species is a key complication that prevents one from simply aggregating the effect of each elementary reaction one by one into the chemical Jacobian. Instead, one needs to account for all the elementary reactions that appear in the algebraic relation function that maps non-QSS species concentration to QSS species concentration. The identification and organization of QSS interdependencies to facilitate this representation is summarized next.

The linear system for the N_{QSS} QSS species molar concentrations, stored in the vector C_{QSS} , can be written as follows:

$$\mathbf{A}C_{\text{QSS}} = \mathbf{b},$$

$$A_{m,n} = \frac{\sum_{j=1,M} \nu_{m,j} \left(\delta'_{n,j} q_{f,j} - \delta''_{n,j} q_{b,j} \right)}{C_n} \quad \text{for } m, n = 1, N_{\text{QSS}},$$

$$b_m = \sum_{j=1,M} \nu_{m,j} \left(\prod_{i=1,N_{\text{QSS}}} (\delta'_{i,j} - 1) q_{f,j} - \prod_{i=1,N_{\text{QSS}}} (\delta''_{i,j} - 1) q_{b,j} \right) \quad \text{for } m = 1, N_{\text{QSS}}. \quad (6)$$

where $\delta'_{i,j}$, resp. $\delta''_{i,j}$, equals 1 if (QSS) species i is on the left, resp. right, of reaction j . $A_{m,n}$ represents the contribution to the net production rate of QSS species m of those reactions that linearly couple QSS species n and m ; while b_m contains the remaining non-QSS contributions. The A matrix becomes more dense as QSS species interdependency increases, introducing the need for a non-trivial solution in order to solve for C_{QSS} .

Following the general procedure outlined by Lu and Law [47] the interdependencies between QSS species are expressed as a directed graph. Fig. 2(a) shows such a graph for a fictitious kinetic mechanism containing 8 QSS species (S_1 to S_8). The arrows express connectivity between species and point in the direction of dependence; for example, species S_2 is a product in at least one reaction that uses species S_1 as a reactant (S_2 needs S_1). Two types of interdependencies can be distinguished: connected components and strongly connected components (SCC). Vertices (the species) within a connected component are related only via unidirectional paths, such as S_6 and S_7 . Vertices within SCCs are related by bi-directional paths, which can be indirect; for every pair of vertices, S_x and S_y , within an SCC there exist paths both from S_x to S_y and from S_y to S_x [48]. Species S_1 to S_5 in Fig. 2(a) form a SCC group, labeled G_1 . Identifying SCCs enables partitioning QSS species into groups, as depicted in Fig. 2(b). This information can be used to reorder the species and simplify the expressions that define the entries of the block-diagonal matrix A [34]. Note also that connected components can be represented by an upper triangular structure, requiring only back substitutions to solve; while SCCs are represented by sub-blocks which require a more complex solution scheme that typically involves pivoting.

To sum-up, the order of algebraic resolution is determined as follows:

1. QSS species involved in SCCs are identified and partitioned into groups (Fig. 2(a))
2. The remaining unidirectional dependencies relating groups and individual QSS species are updated (Fig. 2(b))
3. Groups and individual QSS species are ordered based on the size and direction of dependencies

The procedure aforementioned requires identifying the set of SCCs and partitioning the set of QSS species appropriately. Several algorithms have been proposed to identify SCCs with various time complexities in terms of the number of graph vertices and edges [48,49]. The algorithm used herein [48] is comparable to that used in [34] in that both are recursive, depth-first search (DFS) algorithms that inherently topologically sort the resulting directed acyclic graph; the two differ in number of DFS traversals needed and storage structure of graph information. Our implementation requires only one DFS traversal of

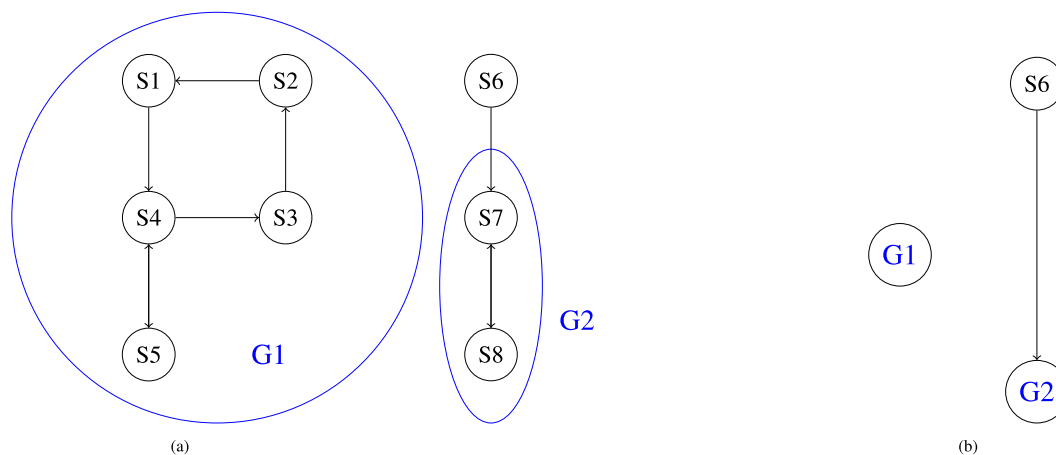


Fig. 2. (a) QSS dependencies through a directed graph: connected components are determined by acyclic paths (e.g., S6 to S7) while strongly connected components (SCC) are determined by cyclic paths (blue, solid groupings). (b) After SCCs are identified, they can be considered as single nodes within a simplified directed graph containing only acyclic paths. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the original graph, as opposed to the multiple traversals needed for both the original and transpose graphs as in [34]. Additionally, we use an adjacency list to represent graph information, as opposed to the adjacency matrix implementation in [34]; this eliminates storage of superfluous information regarding non-adjacent vertices. With the use of an adjacency list, the SCC identification has time complexity $\mathcal{O}(V + E)$ where V is the number of vertices and E is the number of edges in the directed graph. Further description of the algorithm is provided in Appendix B.

3. The analytical chemical Jacobian for QSS-reduced chemistries

The integration of stiff chemical kinetics in CFD solvers often relies on time-implicit methods to keep the overall timestep size constrained by the fluid dynamics of interest. The implicit time discretization results in a non-linear system generally tackled using a Newton approach. In this section, the shortcomings of using an iterative method to calculate the Newton direction (i.e. methods that do not rely on constructing the system Jacobian) are put forth when larger timesteps are required in CFD solvers. Challenges of constructing analytical Jacobians (AJ) for QSS-reduced chemistries are discussed, typical approximations are investigated and a symbolic approach for the construction of an exact AJ is presented and validated.

3.1. The need for a chemical Jacobian

At every timestep and in every computational cell of a reacting flow simulation, the instantaneous vector of production rates for the chemical state ϕ of the system should be evaluated in order to advance the solution. Typically, ϕ is an $(N + 1)$ -sized vector containing the transported species molar concentrations and an energy variable — here, temperature: $\phi = \{C_1, C_2, \dots, C_N, T\}$. Chemical mechanisms enabling the evaluation of these production rates are typically characterized by a wide range of timescales. This is all the more true as larger, more detailed, chemical mechanisms are being deployed within reacting flow simulations [36]. The integration of the resulting stiff system of ODE can require very small timesteps and when an explicit integration of the chemical kinetics over time is unpractical, such as in low-Mach solvers, an implicit strategy is often preferred.

Implicit time integration schemes usually employ a Newton method to solve the non-linear system resulting from the time discretization of the set of chemical ODEs, relying on forming and inverting the chemical Jacobian of ϕ [35,36,50]:

$$J_{i,j} = \frac{\partial \dot{\omega}_i}{\partial \phi_j}, \text{ for } i, j \in \{1, N + 1\}, \quad (7)$$

to advance the Newton step. Alternatively, matrix-free methods also exist, such as Krylov-based iterative methods, and these can be implemented without knowledge of the full chemical Jacobian [51]. Since the computation, storage and inversion of the *full* chemical Jacobian can be prohibitively expensive, these matrix-free and preconditioned Krylov techniques can be attractive. In what follows, we assess the need for a chemical Jacobian by comparing the robustness and efficiency of a Krylov iterative method without preconditioning, hereafter referred to as IM, with that of Newton strategy that uses a full Jacobian computed using finite differences (FD) [46]. Note that while an effective preconditioner could improve the performance of the IM solver [52], it would benefit most large chemical mechanism (>100 species [23,24]) and its construction would involve computing or approximating elements of J , thus rather reinforcing the argument for needing an approach to compute such elements. While an FD approach to computing J is robust and simple, the main focus of the paper is the construction of an analytical Jacobian that is considerably more efficient and flexible than its FD counterpart.

The comparison between the FD chemical Jacobian and the IM is conducted for the QSS-reduced, 35-species, n-dodecane chemical mechanism using the 0D calculations at constant volume; starting from the same 108 initial chemical states as in Section 2.2. All 108 0D cases are run for a total time interval of 0.1 s, which is divided into substeps over which the chemical state is advanced. The role of these substeps is to mimic timestepping in a fluid simulation.

First, a set of reference solutions is constructed using a finite difference chemical Jacobian for each one of the 108 initial states to approximate the true solution. The reference simulations use a substep of size 10^{-8} s and absolute tolerance of 10^{-15} . Second, the solutions predicted using the FD Jacobian and the Krylov-based IM are generated for absolute tolerances (η) varied between 10^{-8} and 10^{-14} , and substep size ranging from 3×10^{-8} s to 10^{-4} s. For each η and substep size, all 108 0D calculations are run, and the execution time is recorded, as well as the error in the temperature predictions compared to the reference solutions. The error reported is computed as:

$$\epsilon_T = \frac{\|T(t) - \overline{T_{ref}}(t)\|_2}{\sqrt{N_{\text{steps}}}}, \quad (8)$$

where T is the time-dependent temperature history obtained with a given number of substeps and tolerance level, $\overline{T_{ref}}$ is the temperature from the reference solution interpolated to the time instance where T is defined, and N_{steps} is the total number of substeps throughout the total time interval. The scaling factor $\sqrt{N_{\text{steps}}}$ ensures that the errors computed with different numbers of substeps are comparable.

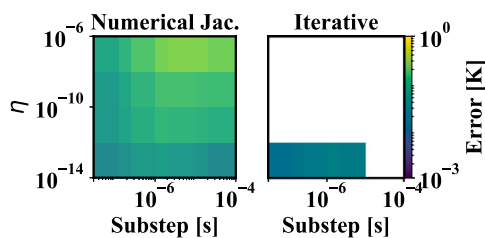


Fig. 3. Average temperature error of 0D constant volume calculations, starting from 108 initial conditions, done with a direct solve requiring a numerical Jacobian (left contour) and a Krylov-based iterative method (right contour). White spaces denote that at least one of the 108 calculations failed to complete in under 7200 s.

Convergence at large substep size. Hereafter, the error ϵ_T averaged over the 108 initial conditions is reported, where possible. Results are not reported when any of the 108 calculations took more than 7200 s to complete. This timeout procedure is implemented because, within each substep, the SUNDIALS ODE integrator employs its own variable size time-stepping strategy. Typically, when the ODE integrator uses more than 10^4 internal steps (threshold typically used in Pele runs [1]), the reacting flow simulation would stop and would need to be restarted with a smaller substep size (or timestep in the context of a reacting flow simulation). Here, this threshold is set to 10^9 internal steps to prevent such early stopping, and allow the integration method to take as many steps as needed to converge. In practice, none of the cases reached the 10^9 internal steps limit. However, with this approach, the time to solution can become unreasonably long (see also [17]), in which case we declare the simulation unsuccessful.

The results shown in Fig. 3 illustrate that, where all 108 calculations successfully complete, the averaged temperature error is similar between the FD Jacobian and the IM. Successful calculations can consistently be obtained with the IM only when the absolute tolerances are sufficiently small ($\leq 10^{-12}$). Moreover, some calculations failed to complete for substep sizes larger than 10^{-5} s. By contrast, all 108 OD runs successfully complete for all tolerances and substep sizes when using an FD Jacobian. These results emphasize that a chemical Jacobian would be typically needed for low-Mach number solvers [4,53,54] that use timestep sizes of the order $10^{-6} - 10^{-4}$ s. For compressible solvers, where timesteps are constrained by the acoustic timescales, the Krylov-based iterative method may however be an acceptable method if an explicit integration is not feasible.

Computational cost at large substep size. The wall time was recorded for each one of the 0D calculations and was averaged over the 108 initial conditions. The calculations were conducted on the Kestrel HPC cluster at NREL which uses Intel Xeon Sapphire Rapids. Hereafter, the ratio $W_{t_{IM}}/W_{t_{FDJ}}$ is reported, where $W_{t_{IM}}$ is the average wall time over the 108 initial conditions for the Krylov-based IM and $W_{t_{FDJ}}$ is the average wall time over the 108 initial conditions when using an FD Jacobian. The wall time ratio is computed for the smallest absolute tolerance level, ensuring successful completion of all computations. It can be observed from Fig. 4, that as the substep size (i.e. the simulation timestep size) increases, using a chemical Jacobian becomes computationally more efficient than using a GMRES IM. This result also suggests that a chemical Jacobian would be advantageous in low-Mach number simulations.

3.2. Construction of analytical chemical Jacobian

Finite differences evaluations of the chemical Jacobian can be a very important contribution to the total computational time in CFD simulations, especially if updates are frequently needed during the solve (as is often the case when relying on the BDF methods implemented in CVODE [46] for example). One FD evaluation of the chemical

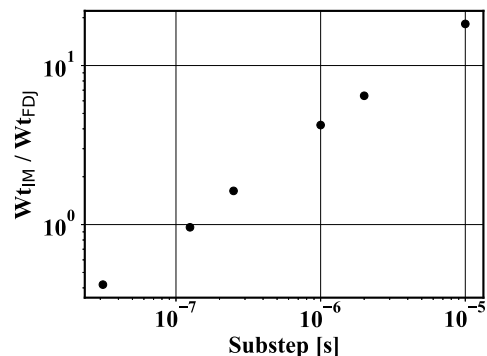


Fig. 4. Average wall time ratio of 0D constant volume calculations starting from 108 initial conditions between a Krylov-based iterative method and a numerical Jacobian for absolute tolerance of 10^{-14} .

Jacobian is typically more expensive than an analytical evaluation, because of the multiple reaction rate evaluations. Additionally, analytical evaluations can leverage the sparsity exhibited by most chemical Jacobians.

Although the computation of an AJ can be an involved process, due to the complexity of certain reaction types, several numerical methods have been successfully developed in the past [35,36]. Here, it is chosen to formulate the AJ in terms of molar concentrations (instead of mass fractions) in order to generate a sparser Jacobian [24,25]. Without QSSA, the AJ can be computed by accumulating the contributions of each reaction to entries in the AJ, one after the other (see, e.g., Eq.12 in [37]). In the presence of QSSA, a different approach is needed. One way to proceed is to differentiate between the set of N_{NQSS} non-QSS transported species, and the set of N_{QSS} QSS species which are algebraically related to the set of transported species. As a result, the production rates in Eq. (7) indirectly involve the contribution of QSS species, and the AJ now reads:

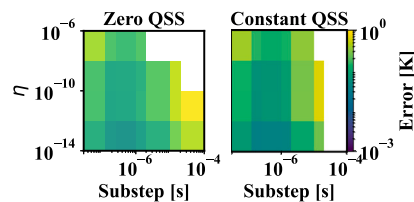
$$J_{i,j} = \sum_{k=1, N_{\text{NQSS}}} \frac{\partial \dot{\omega}_i}{\partial C_k} \frac{\partial C_k}{\partial C_j} = \sum_{k=1, N_{\text{NQSS}}} \left(\sum_{l=1, M_{\text{NQSS}}} \frac{\partial v_{i,l} \dot{Q}_l}{\partial C_k} + \sum_{l=1, M_{\text{QSS}}} \frac{\partial v_{i,l} \dot{Q}_l}{\partial C_k} \right) \frac{\partial C_k}{\partial C_j}, \quad (9)$$

A distinction is made between contributions from reactions that do not involve QSS species (the first term in parenthesis in Eq. (9)) and those who do (the second term in parenthesis in Eq. (9)). The non-QSS portion can be computed using classical AJ derivations reported in earlier work (e.g. [37]). The evaluation of the second term in the expression is more difficult because it entails resolving the dependencies of all QSS species (involved in the \dot{Q}_l) with respect to the non-QSS species. Note that a hybrid analytical-numerical strategy could be envisioned here, where one would numerically approximate the $\frac{\partial \dot{Q}_l}{\partial C_k}$ terms for each reaction l and non-QSS species k . However, the numerical approximation would involve additional production rate evaluations, which should be avoided for computational efficiency.

In the rest of this Section, the construction of AJ for QSS-reduced chemistries is described. First, common simplifications are investigated, before an exact version is constructed and tested. The validation of the approach is done on 0D integrations, as was done previously in Sections 2.2 and 3.1, by comparing results against that obtained with FD. Three different QSS-reduced mechanisms are used for this purpose, spanning the range of fuels typically considered in reacting flow simulations. The following nomenclature is adopted: X-Y-fuel, where X is the number of non-QSS species considered in the reduced mechanism, Y is the number of QSS species and fuel identifies the molecule for which the kinetic mechanism was designed. The first chemistry adopted is a reduced methane mechanism [55] referred to as 13-4-CH₄; the second

	Constant QSS AJ	Zero QSS AJ
13-4-CH ₄	8.3 %	9.9 %
35-18-N-C ₁₂ H ₂₆	22.7 %	25.1 %
55-33-N-C ₇ H ₁₆	65.7 %	72.1 %

(a) ϵ_J computed for three different QSS-reduced chemistries with simplified AJ for 100 randomly sampled chemical states.



(b) Mean temperature error profiles obtained with the 0D integration, done with either a Constant QSS AJ (left) or a Zero QSS AJ (right). White spaces denote that at least one of the 108 calculations failed to complete in under 7200s.

Fig. 5. Effect of simplified analytical Jacobian construction on QSS-reduced chemistries integration.

chemical mechanism is a reduced n-dodecane mechanism [44] referred to as 35-18-N-C₁₂H₂₆; the third chemical mechanism is a reduced n-heptane mechanism [56] referred to as 55-33-N-C₇H₁₆.

3.2.1. Simplified construction

The easiest simplifying strategy [40] has been suggested for QSS mechanisms, and consists of nullifying the second term of the right-hand side of Eq. (9), thus falling back on the classical formulation of non-QSS AJ. This approximation is termed the *constant QSS AJ*. Another approximation consists of nullifying the QSS species molar concentrations altogether in the *constant QSSA AJ* approximation by invoking the fact that QSS species molar concentrations are usually very low [34]. This second method is termed the *zero QSS AJ* approximation.

To evaluate these strategies, the resulting AJs are compared to an FD chemical Jacobian by computing 0D integrations of 100 initial chemical states. Note that these are different from the 0D integrations presented in Section 3.1; the only objective here is to interrogate the function that maps temperature and species molar concentrations to the chemical Jacobian. The molar concentrations of non-QSS species are uniformly sampled from the interval $[10^{-4}, 1]$ mol/m³ and initial temperatures are uniformly sampled from the interval [300, 1300] K. The finite difference approximation was done with perturbations of size 10^{-4} and using a central difference scheme. The error between the analytical approximation of the chemical Jacobian (J_A) and the FD one (J_{FD}) is computed using the Frobenius norm over all rows and columns:

$$\epsilon_J = \mathbb{E} \left(\frac{\|J_A - J_{FD}\|_F}{\|J_{FD}\|_F} \right), \quad (10)$$

\mathbb{E} denotes the average taken over the 100 random samples. Results are presented in Fig. 5(a). By considering the numerical chemical Jacobian as the ground truth, it can be observed that the constant QSS and zero QSS approximations lead to similar error levels. Errors observed are relatively low, considering how aggressive the approximation employed is. Note that the error increases with the number of QSS species, which suggests that both approximations may be appropriate for mechanisms with a small $N_{\text{QSS}}/N_{\text{QSS}}$ ratio.

Fig. 5(b) shows the results of the same experiment as reported in Fig. 4 (for the 35-18-N-C₁₂H₂₆ mechanism only). Even though both AJ errors are small (about 25% relative error), the ODE integration fails to converge in under 7200 s, especially at large substep size. For the Constant QSS AJ assumption, this is likely a consequence of the fact that the QSS concentrations cannot be assumed constant throughout the substep. For the Zero QSS AJ, the underlying system is less diagonally dominant as the substep increases and the Zero QSS AJ is too inaccurate to solve the linear system. Comparing Figs. 5(b) and 4 (left), it is also clear that the average error in the temperature predicted is higher than when using a finite difference Jacobian, especially for timesteps greater than 10^{-6} s. However, the errors quickly drop as the timestep and tolerance reduce. Overall simplified constructions of the chemical Jacobian are appropriate only if small timesteps and low tolerances are being used. In the following sections, it is proposed to not rely on the aforementioned approximations, but instead to obtain a Jacobian that can be used with a wider range of tolerances and timesteps.

Table 2

Size of the mechanism file generated when using a symbolic AJ computation with and without common subexpression elimination.

	Symbolic AJ with CSE (# of subexpressions)	Symbolic AJ
13-4-CH ₄	375 Kb (2327)	98 Mb
35-18-N-C ₁₂ H ₂₆	2.9 Mb (16 487)	(65 Gb+)
55-33-N-C ₇ H ₁₆	5.8 Mb (10 960)	1.5 Gb

3.2.2. Symbolic construction

For chemical mechanisms of moderate size (from 30 to 50 transported species) with more than 10 QSS species, which are of interest in reacting flow simulations, it was shown in Section 3.2.1 that neglecting the second right-hand-side term of Eq. (9) may adversely affect accuracy and stability. Including this term requires knowledge of the concentration of QSS species, which could in principle be obtained via a numerical inversion using a GPU-aware linear algebra library such as MAGMA [57]. However, the small size of the chemical system is too limited to exploit the high throughput of the GPU for such a subsystem. Grouping the linear systems of multiple cells into a single batched solve available in MAGMA could fully exploit the GPU, but this approach is incompatible with the function call stack of PelePhysics, which distributes individual cells across GPU threads, and uses a single GPU kernel launch for the entire Jacobian.

In this section, a method to construct the AJ of QSS-reduced mechanisms is described and demonstrated. A symbolic method is used whereby the QSS and non-QSS species molar concentrations are symbolically tracked. This can be simply achieved using any symbolic math package such as the sympy package [58]. When the AJ needs to be written, it can be obtained using symbolic differentiation tools. In practice, the differentiation steps can be expensive but need only to be done once as a pre-processing step, and takes less than 1 min for the 55-33-N-C₇H₁₆. An advantage of this approach is its flexibility with respect to the implementation of new reaction types. Any differentiable closure equation between transported and modeled species can be included in the chemical Jacobian. The symbolic approach therefore greatly simplifies the maintainability and the readability of the code, since the construction of the Jacobian is done based on the construction of reaction rates. While the manuscript describes the symbolic Jacobian construction for QSS-reduced chemistries, other types of algebraic reductions/simplifications could also benefit from the same approach [59].

The main drawback of the symbolic construction of the Jacobian is that a naive symbolic differentiation can be slow to compute and can result in verbose expressions. In the chemical Jacobian, several matrix entries may involve the contribution of the same set of reaction rates albeit scaled with a different factor. Without additional constraints, a symbolic differentiation would spell the same term many times, resulting in overly verbose expressions. Table 2 (second column) gives the size of the file mechanism.H generated by the code provided in the companion repository and that contains the function that assembles the AJ. It can be seen that, without further treatment, the symbolic

differentiation can generate several Gb of code. In the case of the 35-18-N-C₁₂H₂₆ mechanism [44], the code generation could not reach completion and the size indicated is a lower bound of the actual file size that should be expected. Note that the larger the function, the longer the compilation, and the larger the compiled object. Compilation typically failed on HPC clusters using mechanisms files that did not use further treatment, due to memory errors.

The function that assembles the AJ can be made significantly less verbose by first identifying common subexpressions in the Jacobian entries and precomputing them before assembly. In Appendix D, an example is provided to illustrate the practical benefit of common subexpression elimination strategies. Fortunately, common subexpression elimination (CSE) is available in modern symbolic math packages [58]. Despite the number of species, the CSE can be achieved in only 30 s on a single Apple M1 Pro chip for the 35-18-N-C₁₂H₂₆ mechanism. The first column of Table 2 shows that the code generated with CSE is also several orders of magnitude smaller, which allows compilation on HPC clusters. The number of common subexpressions does not necessarily increase with the mechanism size but with the complexity of the QSS species interdependencies. It can also be observed that the number of common subexpressions identified is a monotonic but sub-linear function of mechanism file size without common subexpression elimination.

Note that the symbolic procedure is only used for the first N rows and columns of the AJ. The contribution of the heat release rate can be simply obtained from the species production rates [36]. However the gradients of the production rates with respect to temperature, $\frac{\partial \dot{\omega}_i}{\partial T}$, need a specific and compute-intensive treatment [40]. The strategy adopted here is to approximate these terms via finite difference by computing the species' production rate twice at different temperatures. A finite difference strategy is useful here given that it avoids symbolically encoding the thermodynamics polynomials which can change for each species. In addition, differentiating with respect to temperature is less prone to numerical instability than differentiating with respect to species molar concentrations since the temperature is usually far from 0. The benefit of a numerical differentiation for gradients with respect to temperature was also recognized elsewhere [23,24].

3.2.3. Validation

The symbolic AJ is first validated using the 0D tests starting from the same 100 random initial conditions as the ones reported in Section 3.2.1. Fig. 6(a) shows the accuracy of the symbolic AJ evaluation in terms of the relative Frobenius error (Eq. (10)). The Frobenius error is several orders of magnitude lower than the one shown for the simplified AJ (Fig. 5(a)). A separate study in Appendix C shows that the error between the symbolic and the finite difference Jacobian can be attributed to the discretization errors in the construction of the finite difference Jacobian.

Next, the same 108 0D ignitions as in Section 3.2.1 are reproduced with the symbolic AJ. The stability domain (domain over which the ODE integrator converged in less than 7200 s) of the symbolic construction is the same as that of the finite differences one, and the errors incurred follow the same trend with respect to substep sizes and tolerances (Fig. 6(b)).

Finally, the accuracy of the symbolic AJ is evaluated by estimating ignition delay values for the same 108 initial conditions that were described and used in Section 2.2. The ignition delay error is reported in terms of the rescaled mean absolute error as defined in Eq. (5). Table 3 shows the ignition delay error between the skeletal mechanism and the reduced mechanism integrated with an FD Jacobian ($\epsilon_{\text{ign.SK-QSS}}$, first column), as well as the ignition delay error between the reduced mechanism integrated with an FD Jacobian and the symbolic AJ ($\epsilon_{\text{ign.QSS-QSSAJ}}$, second column). The QSSA affects the ignition delay by 8.97%. Meanwhile, the symbolic AJ exactly reproduces the ignition delay of that obtained with the FD Jacobian. Note that the ignition delay is accurate up to the size of the timestep used which

Table 3

Ignition delay errors between the skeletal and the QSS mechanisms integrated with an FD Jacobian (first column); and the QSS mechanisms integrated with an FD or a symbolic AJ (second column). The last column denotes the speedup when using a symbolic AJ against a FDJ on a CPU.

	$\epsilon_{\text{ign.SK-QSS}}$	$\epsilon_{\text{ign.QSS-QSSAJ}}$	$Wt_{\text{QSS}}/Wt_{\text{QSSAJ}}$
13-4-CH ₄	$4.02 \times 10^{-3}\%$	0%	1.35 ± 0.118
35-18-N-C ₁₂ H ₂₆	8.97%	0%	1.35 ± 0.0665
55-33-N-C ₇ H ₁₆	4.91%	0%	1.99 ± 0.195

Table 4

Fraction of non-zero entries in the AJ using an exact symbolic method, or approximated as in Section 3.2.1.

	Symbolic AJ	Constant QSS AJ	Zero QSS AJ
13-4-CH ₄	0.489	0.468	0.38
35-18-N-C ₁₂ H ₂₆	0.447	0.241	0.176
55-33-N-C ₇ H ₁₆	0.357	0.183	0.159

ranges from 10^{-10} s to 10^{-4} s. The 108 experiments were reproduced each 5 times on a single CPU, and timed. The computational speedup induced by the use of an AJ is denoted by $Wt_{\text{QSS}}/Wt_{\text{QSSAJ}}$ in Table 3. The error reported is the standard deviation of the speedup across the 108 realizations. It can be observed that the AJ strategy provides a significant speedup over an FD Jacobian as also noted elsewhere [60]. The speedup is also consistent across all the realizations considered. The speedup appears to increase with the number of species which dictates the number of production rate evaluations needed to compute the numerical Jacobian. However, the speedup observed is nearly the same between the 13-4-CH₄ and the 35-18-N-C₁₂H₂₆ mechanisms despite the fact that the latter mechanism contains three times as many non-QSS species. This observation points to the fact that the cost of evaluating the analytic chemical Jacobian can vary depending on its density, and the computational complexity of each term.

In terms of memory usage, which can be critical for memory-limited computing hardware, the use of an AJ for reduced mechanisms can induce significant memory pressure. Table 4 shows the fraction of non-zero entries in the first N rows and N columns of the AJ, as obtained over 100 randomly sampled chemical states. For the 35-18-N-C₁₂H₂₆ mechanism and the 55-33-N-C₇H₁₆ mechanism, the second term in the right-hand side of Eq. (9) increases the number of non-zero entries by a factor of about 2. This loss in sparsity is due to the coupling of each reaction with possibly many other reactions that can be related to the molar concentration of a single QSS species (see Section 2.3).

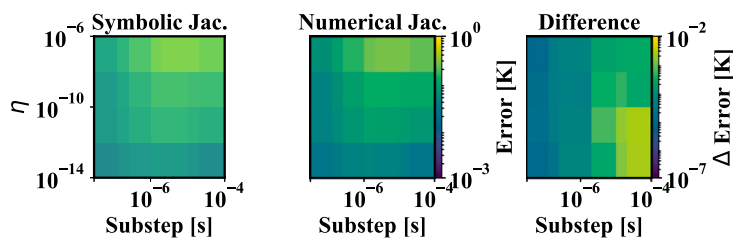
4. Performance improvements for the chemical Jacobian

Assembling a chemical Jacobian is computationally expensive in general, irrespective of whether the QSS approximation is used. In terms of computational intensity, each term of the AJ requires evaluating the derivative of several species' production rates. In terms of memory requirements, a matrix of size $(N + 1) \times (N + 1)$ needs to be stored in memory for every computational cell in the domain. The use of the QSSA can further result in a relatively dense Jacobian (Table 4) which increases the number of entries to compute. As mentioned in Section 3.2.2, the use of the symbolic method needs to be paired with CSE. However, the precomputed expressions are additional scalars that must be stored in memory. In the case of the 35-18-N-C₁₂H₂₆ mechanism, 16487 scalars need to be stored in memory which is nearly 13 times more than the number of entries in the chemical Jacobian matrix.

In this section, several strategies to mitigate memory use and reduce the computational of the AJ construction are outlined. Their effect is then quantified in Section 5.

	Symbolic AJ
13-4-CH ₄	6.5×10^{-8} %
35-18-N-C ₁₂ H ₂₆	1.35×10^{-6} %
55-33-N-C ₇ H ₁₆	4.33×10^{-4} %

(a) ϵ_J computed for three different QSS-reduced chemistries with symbolic AJ.



(b) Mean squared error of the temperature profiles obtained with the 108 0D ignitions, done with the symbolic analytic chemical Jacobian (left) and the Jacobian computed via finite differences (middle). Right: temperature error differences between the analytic and numerical Jacobian methods.

Fig. 6. Effect of symbolic analytical Jacobian construction on QSS-reduced chemistries integration.

4.1. Memory footprint reduction

In the case of a symbolic AJ that uses CSE, the memory use is primarily impacted by the number of common subexpressions that need to be precomputed. Reducing the number of common subexpressions can be done by simply replacing these expressions in the final Jacobian entry computation. However, if the common subexpressions are naively replaced, one will simply negate the benefits of CSE (Section 3.2.2). Instead, the common subexpressions must be strategically replaced depending on the amount of added computation that they will incur.

4.1.1. Replacement based on operation count

It is proposed to select the common subexpressions to replace based on the number of additional operations incurred if the common subexpression is replaced. The replacement procedure is shown in Algo. 2.

Algorithm 2 Algorithm of common subexpression replacement

```

1: User inputs a threshold of maximum operation count  $n_{op}$ 
2: for opThreshold = 1 :  $n_{op}$  do (Loop 1)
3:   for subexp  $\in$  allSubexpressions do (Loop 2)
4:     totalOpsubexp = (nUsesubexp - 1)  $\times$  nOpsubexp (number of
       operations added if subexp is eliminated)
5:     if totalOpsubexp < opThreshold then
6:       Replace subexp
7:     end if
8:   end for
9: end for

```

In Loop 2 of Algo. 2, for each common subexpression “subexp”, one first counts the number of times the common subexpression is used (nUse_{subexp}) and counts the number of operations (addition, multiplication, exponential, power, etc.) in the common subexpression nOp_{subexp}. By replacing the common subexpression in the rest of the AJ assembling function, one would trade the precomputation of one scalar with an added (nUse_{subexp} - 1) \times nOp_{subexp} operations. If the number of additional operations is less than the threshold opThreshold, the common subexpression is replaced.

The role of Loop 1 of Algo. 2, is to ensure that the number of common subexpressions eliminated grows as operation count threshold n_{op} inputted by the user increases. In our experience, this loop does not necessarily lead to a higher number of common subexpression eliminated but provides an intuitive control for memory footprint.

4.1.2. Recycling of common subexpression

The symbolic expressions constructed with the symbolic method can be divided into two categories: (1) common subexpressions, (2) AJ entries. The common subexpressions are computed sequentially and the role of some of the earlier CSEs is to simplify the computation of future CSEs. In this case, there is an opportunity to reuse precomputed

common subexpressions’ memory space once they are not useful for subsequent calculations. This method is termed *subexpression recycling* and is described in Algo. 3.

Algorithm 3 Algorithm of subexpression recycling

```

1: for subexp  $\in$  allSubexpressions do
2:   if subexp not in Jacobian terms then
3:     Find lastsubexp the last subexpression that uses subexp
4:     Reuse subexp and eliminate lastsubexp
5:   end if
6: end for

```

Algorithm 3 simply consists in looping through each common subexpression subexp and identifying the ones that are not directly used in the AJ entries. By finding the last common subexpression that uses subexp, one can reuse subexp instead of allocating an additional scalar.

4.2. Computational cost reduction on GPU

Assembling the chemical Jacobian is a computationally intensive process that requires several non-elementary operations, such as using power operations or exponential. Several best practices for writing GPU-ready code include transforming some non-elementary operations [61]. In particular, instead of using a pow operation for small integer powers (2 or 3), it can be more efficient to use explicit multiplication. Likewise, transforming powers of 10 with base-10 exponentiation was listed in Ref. [61] as possibly advantageous. In Section 5, the effect of these two optimizations on computing cost is explored.

5. Deployment on exascale machines

In previous sections, the validation of the symbolic approach for chemical Jacobian generation was done using 0D ignition. Here the focus is on exercising the symbolic chemical Jacobian constructed on a problem that can run on exascale-ready HPC clusters.

5.1. Configuration

The test-case simulated is a 3D multi-pulse fuel injection of pure N-C₁₂H₂₆ fuel into a premixed oxidizing air-fuel mixture with an equivalence ratio of $\phi = 0.5$ that consists of air and CH₄ at 900K and 60 bars. The multi-pulse scenario includes a first short pulse of the N-C₁₂H₂₆ into the prefilled chamber. As the jet develops, it entrains the preheated CH₄-air mixture and the first pulse ignites. After 0.5 ms, the first pulse injection stops. After a dwell time of an additional 0.5 ms, a second pulse of pure N-C₁₂H₂₆ fuel (same composition as the first pulse) enters the domain at 1 ms. The second pulse lasts 0.5 ms and interacts with the burning gases from the first pulse combustion process and ignites in turn. The simulation proceeds until there is significant

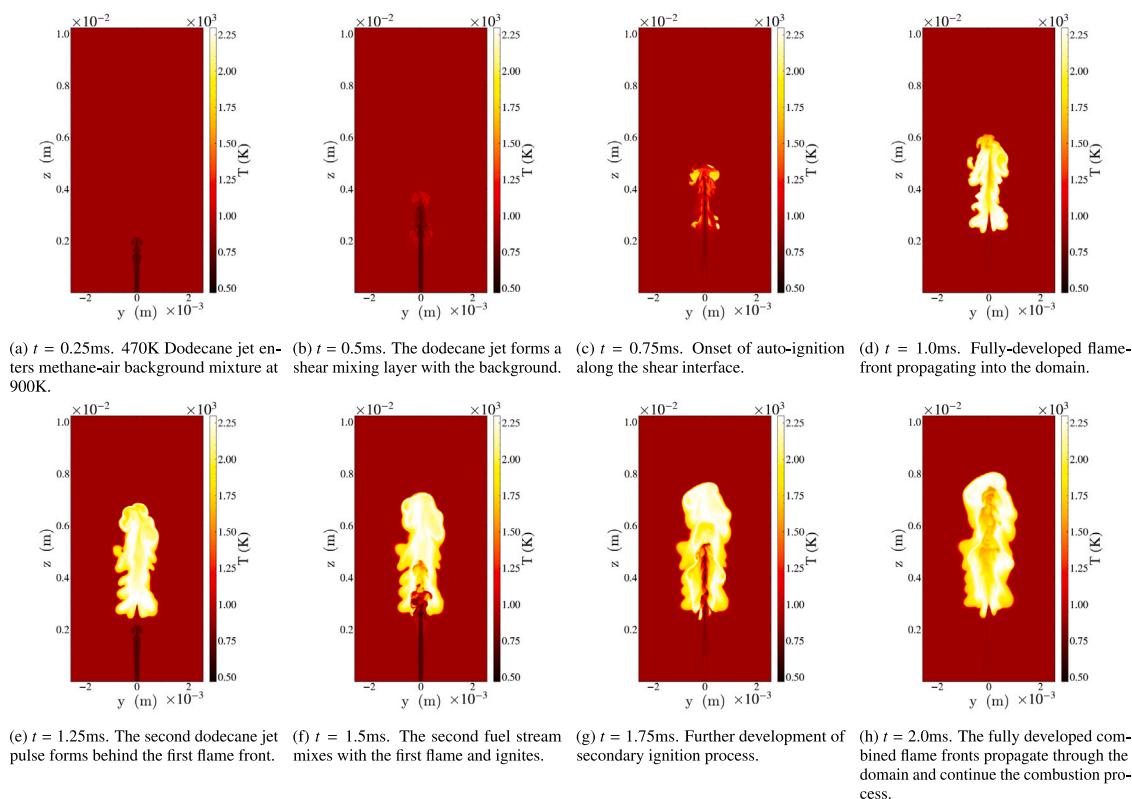


Fig. 7. PeleLMex multipulse n-dodecane fuel jet simulation using the analytic Jacobian QSS formulation at 0.25 ms increments during the simulation.

combustion of both jet pulses after about 2 ms. The entire ignition sequence is illustrated in Fig. 7. This test case was chosen since it is representative of many engineering combustion problems that include turbulent mixing of fuel and oxidizer on length scales that are typical of internal combustion engines.

The case was simulated on the Crusher exascale-ready HPC cluster located at Oak Ridge National Laboratory. Crusher is a testbed for the DOE first Exascale platform, Frontier, introduced in 2023. Each of the compute nodes on the Crusher machine consists of one 64-core AMD “Optimized 3rd Gen EPY” CPU and four AMD MI250X, each with 2 graphics compute dies that are equivalent to 8 standard GPUs per node. The case was simulated using the PeleLMex solver [4] which is an exascale capable adaptive mesh refinement (AMR) based low-Mach solver for reacting flow simulations. The simulations presented in this study used a total of 4 levels of AMR that refine based upon the vorticity and temperature gradient in order to capture the development of the shear layer of the incoming fuel stream as it mixes with the background oxidizer as well as adequately capture the start of the ignition process. Because the simulations are all performed using AMR, the total number of grid cells increases as the simulation progresses with the number of cells approximately 250 million during the onset of the ignition process. All of the cases in this study were simulated using between 8 and 30 Crusher nodes (64–240 GPUs) with fewer nodes required during the start of the simulation and the number of nodes increasing as the flow developed and during the ignition process. No subgrid-scale model is used in any of the simulations and the chemistry is modeled using a detailed $N-C_{12}H_{26}$ mechanism. In the following, either the 53-species skeletal $N-C_{12}H_{26}$ mechanism [45] or the QSS reduced 35-species $N-C_{12}H_{26}$ mechanisms [44] are used.

5.2. Results

Three cases are run to ensure that the symbolic Jacobian does not affect the simulation accuracy. All three cases rely on SUNDIALS

CVODE integrator, but use different linear solvers within the Newton method CVODE employs to solve the discretized nonlinear system of ODEs. Case 1 uses the skeletal mechanism [45] and its integration rely on an iterative Jacobian-free method (GMRES); Case 2 uses the QSS-reduced mechanism [44] combined with an iterative Jacobian-free method (GMRES); Case 3 uses the QSS-reduced mechanism [44] combined with a dense direct linear solve from the MAGMA library [62–64] which uses an analytic Jacobian constructed symbolically.

Fig. 8 shows a close-up view of the start of ignition for each of the three cases after 0.75 ms. Given the chaotic nature of the system investigated, any infinitesimal perturbations even due to rounding errors [65] will exponentially amplify over time [66,67]. Therefore, it is not possible to guarantee an exact agreement of instantaneous snapshots. The snapshots in Fig. 8 can only be used for qualitative assessment and show reasonable agreement between all three cases in terms of the location and sizes of the onset of the ignition kernels. A key observation in these images is that both of the QSS mechanisms’ run develop an ignition kernel at the leading edge of the penetrating jet, whereas the skeletal mechanism only develops two ignition kernels along the shear vortex ring and three ignition kernels at the lower portions of the shear jet layer. Overall, the use of the analytic Jacobian formulation leads to accuracy levels on par with Jacobian-free iterative methods.

Quantitatively, Fig. 9 shows the comparison of the domain average fields for: (a) temperature, (b) density, and (c) heat release for the three different mechanisms examined. The temperature and density averages are consistent among the different chemical mechanisms as these quantities are primarily dependent on the mixing between the fuel stream and background mixture. The integrated heat release tracks the ignition sequence. While all cases ignite similarly, Case 2 (QSS mechanism integrated with a Jacobian-free method) shows an initial spike in the heat release that is larger than the other cases. Case 3 (QSS mechanism with analytic Jacobian formulation) shows results that are closer to the original skeletal mechanism which suggests that the initial

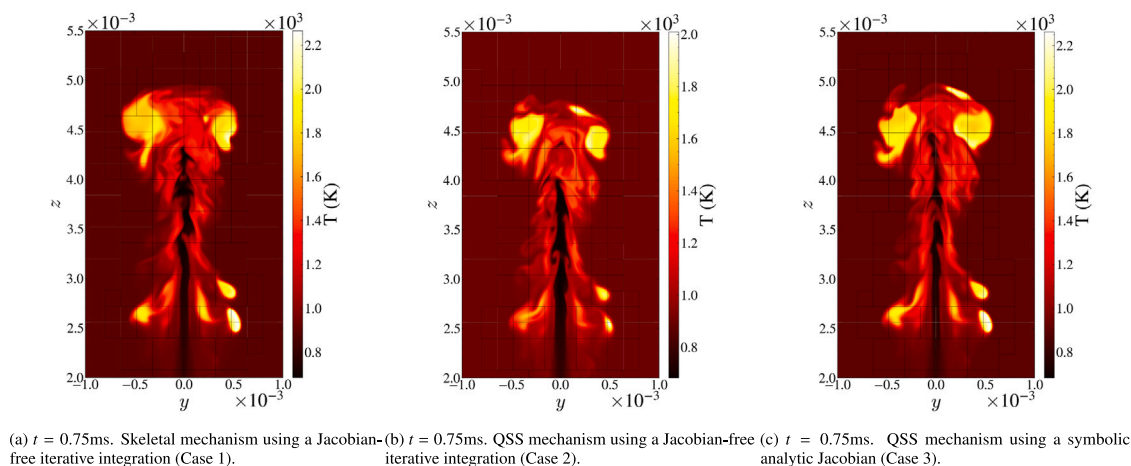


Fig. 8. Close-up comparison of the start of ignition for the three different solution methods.

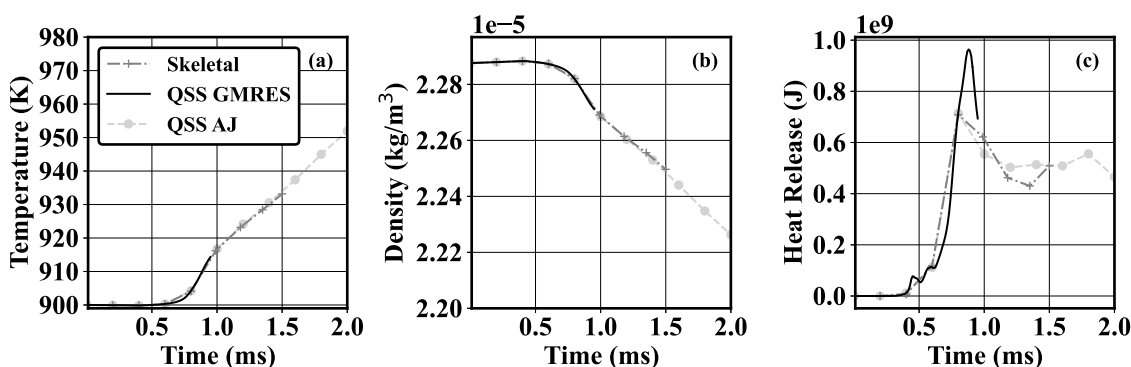


Fig. 9. Comparison of the domain averaged (a) temperature, (b) density, and (c) heat release for the skeletal mechanism using GMRES, the QSS mechanism using GMRES, and the QSS mechanism using the analytic Jacobian formulation.

heat release spike may be the results of a numerical inaccuracy during the chemistry integration.

For both mechanisms integrated with a Jacobian-free method (Case 1 and Case 2), the simulation was stopped before reaching 2 ms because of numerical instabilities that occurred during the ignition process. In practice, the cases would need to be restarted with a smaller timestep, or with a higher number of GMRES iterations. These instabilities were not observed when using the analytic Jacobian formulation and the solution was able to progress smoothly up to 5 ms without failure. This observation echos the 0D integration results shown in Fig. 4 which motivated the construction of the AJ.

In terms of computational time, significant differences were observed between all three cases. After the onset of ignition, Case 1 took approximately 162 s per timestep, Case 2 took 80 s per timestep, and Case 3 took 44 s per timestep. As could be expected, Case 2 was approximately twice as fast as Case 1 primarily because a smaller number of species needed to be transported. Case 3 was approximately 1.8 times faster than Case 2 while leading to higher robustness as discussed in the previous paragraph. Note that the robustness of Case 2 could be improved by increasing the number of iterations for the chemistry integration allowed per timestep (capped at 10000), at the cost of further widening the computational cost between Case 2 and Case 3. The computational speedup and robustness of Case 3 are particularly clear in the case simulated. However, they can be expected to be less clear when using a compressible solver where the fluid timestep size is small enough to make the chemical Jacobian diagonally dominant.

5.3. Performance optimization

In this section, the effect of the performance optimization methods described in Section 4 is evaluated using the multipulse ignition

case described in Section 5. In total, 24 different formulations of the chemical Jacobian function were tested by varying the operation count threshold on common subexpression replacement (n_{op} in Algo. 2) in the set $\{0, 10, 20\}$, by either recycling or not recycling subexpressions (Algo. 3), by either replacing small integer power with multiplications or not, by either replacing powers of 10 with base-10 exponentials or not.

For each of the 24 mechanisms, Case 3 (see Section 5.2) was simulated for 10 timesteps on 30 nodes (240 GPUs) starting from $t = 0.75$ ms. The criteria used to evaluate each of the 24 cases are (1) the total computational time needed by the chemistry integration module (which dominates the computational cost) and (2) the number of common subexpressions precomputed. The effect of each optimization method is obtained by computing the relative compute time and the relative memory use between cases that are the same, except for the one optimization parameter being investigated.

5.3.1. Effect of memory footprint optimization

The effect of Algo. 2 is shown in Fig. 10 in terms of memory use (left) and computing time (right). It can be observed that using $n_{op} = 20$ can reduce the number of precomputed common subexpressions by a factor of 5 to 10 which brings the number of precomputed subexpressions close to the number of entries in the Jacobian matrix. While small values of n_{op} drastically reduce the number of precomputed subexpressions, the number of precomputed expressions decreased sub-linearly as n_{op} further increases. In terms of computing time, n_{op} has almost no effect on chemistry integration computational intensity for the time range considered. This result suggests that the highest value of n_{op} investigated can be used when deploying QSS-reduced chemistries on exascale-ready HPC platforms.

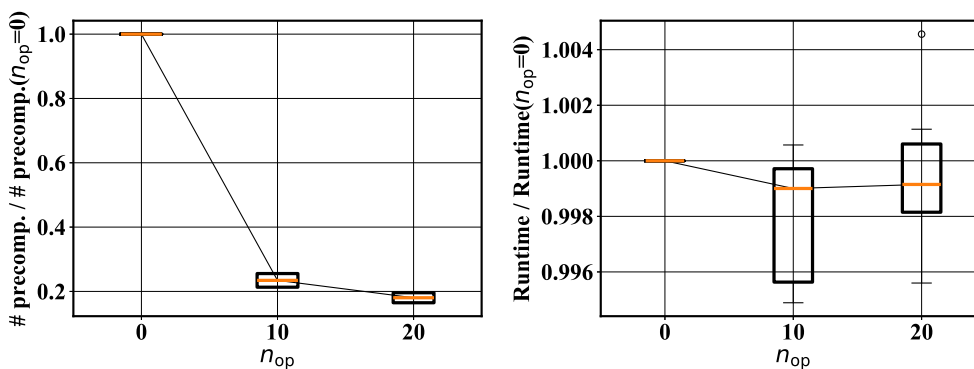


Fig. 10. Effect of n_{op} in Algo. 2 on the number of precomputed subexpressions (left) and the computing time of the chemistry integration (right). The orange line denotes the median, and the box plot denotes the 95% confidence interval.

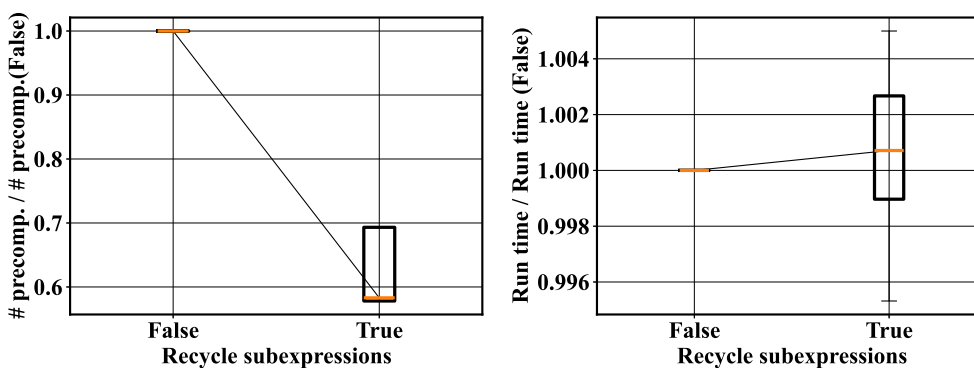


Fig. 11. Effect of Algo. 3 on the number of precomputed subexpressions (left) and the computing time of the chemistry integration (right). The orange line denotes the median, and the box plot denotes the 95% confidence interval.

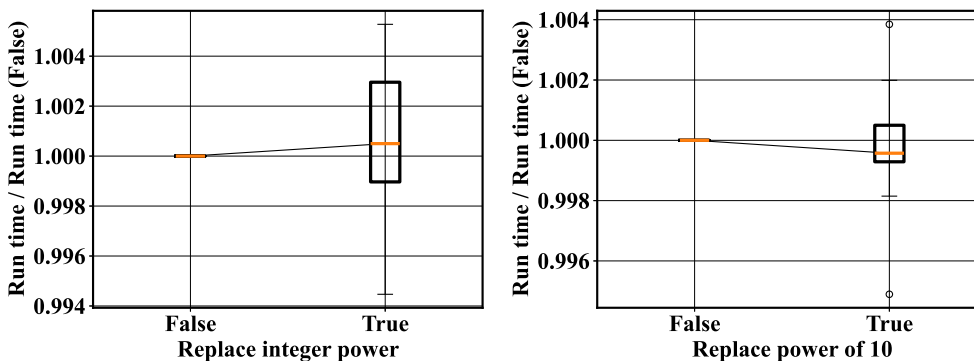


Fig. 12. Effect of replacing small integer powers with multiplications (left) and powers of 10 with base-10 exponential (right) on the computing time of the chemistry integration (right). The orange line denotes the median, and the box plot denotes the 95% confidence interval.

Fig. 11 suggests that Algo. 3 also has a significant effect on memory use since it tends to reduce the number of precomputed expressions almost by 45%. Similar to Algo. 2, it has almost no effect on the computational time. This result suggests that subexpression recycling should be used in general on exascale-ready HPC platforms.

5.3.2. Effect of computing optimization

The effects of mathematical operations optimizations for GPU described in Section 4.2 are shown in Fig. 12. Neither the small integer power replacement nor the power of 10 replacement has a significant effect on the computing time. This result may be due to modern compiler optimization that already performs operation optimization [61]. Overall, this result suggests that the mathematical operation optimizations listed in Section 4.2 are not necessary on exascale-ready HPC platforms.

5.4. Scaling

Finally, to ensure that the Jacobian optimizations do not negatively impact the scalability of the solver, strong scaling of the analytic Jacobian evaluation was performed for 8 random mechanisms out of the 24 mechanisms described in Section 5.3. None of the mechanisms show significant deviation from the ideal scaling (not shown here) which can be expected because the modifications in the Jacobian formulation are local to each computing unit. The scaling results are shown for two QSS mechanisms: one with all memory optimization turned on, resulting in 1691 precomputed terms; and one with all the memory optimization turned off, resulting in 14884 precomputed terms. Note that the latter number of precomputed terms departs from the original value of 16487: in all the 24 mechanisms tested, precomputed expressions that did not include any operation were replaced. Fig. 13

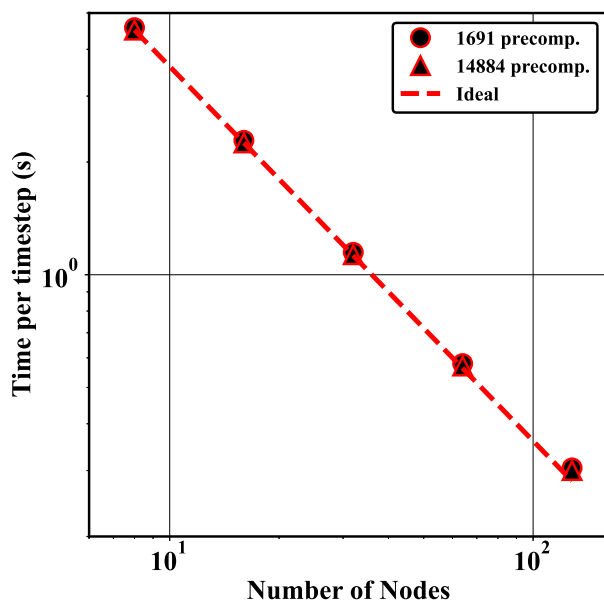


Fig. 13. Strong scaling of the AJ routines for a QSS mechanism without memory optimization (\blacktriangle) and with memory optimization (\bullet) shown against ideal scaling ($- -$).

shows the results of the strong scaling study from 4–128 nodes (32–1024 GPUs) on the Frontier exascale-ready cluster (owing to its larger size than Crusher). The scaling was conducted by simulating Case 3 for 10 timesteps, starting at $t = 2$ ms. The results show that despite the differences in the formulation of the Jacobian, no adverse scaling issue was introduced. The consequence of the above results is that the analytic Jacobian can be written in a variety of ways that each preserve the computational runtime and scalability of the solver while still providing memory optimizations.

6. Conclusions

A comprehensive analysis of chemistry integration of QSS-reduced chemistries using an analytic Jacobian was performed. Several QSS coupling linearization methods were discussed and compared and a 0D analysis of different ODE integration methods suggested that the construction of a Jacobian was needed, especially for low-Mach number solvers. The construction of an analytic Jacobian can provide significant speedup compared to a finite difference Jacobian and a Jacobian-free method. However, constructing an analytic Jacobian of QSS-reduced chemistries can be fairly involved. In this work, a symbolic method was used which allowed to generate an analytic Jacobian if algebraic closure for QSS species can be obtained. Without further treatment, a symbolic Jacobian was found to be overly verbose. Using common subexpression eliminations addressed the problem but increased the memory requirements given that precomputed quantities needed to be stored in memory. Several optimization methods were devised that could reduce the number of precomputed terms by an order of magnitude without incurring additional computational costs. The method was exercised in a 3D multipulse ignition problem that was run on an Exascale-ready platform. It was verified that the symbolic Jacobian provided significant speedup compared to a Jacobian-free method and that the procedure did not affect scalability.

CRediT authorship contribution statement

Malik Hassanaly: Writing – original draft, Software, Methodology, Investigation. **Nicholas T. Wimer:** Writing – review & editing, Visualization, Software, Methodology, Investigation. **Anne Felden:** Writing – review & editing, Software, Methodology. **Lucas Esclapez:** Writing –

review & editing, Software. **Julia Ream:** Writing – review & editing, Software, Methodology. **Marc T. Henry de Frahan:** Writing – review & editing, Software. **Jon Rood:** Writing – review & editing, Software. **Marc Day:** Writing – review & editing, Software, Funding acquisition, Conceptualization.

Declaration of competing interest

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

Acknowledgments

This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding was provided by U.S. Department of Energy Office of Science and National Nuclear Security Administration. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. A portion of the research was performed using computational resources sponsored by the Department of Energy's Office of Energy Efficiency and Renewable Energy and located at the National Renewable Energy Laboratory. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

Appendix A. Automated code generation

Automated generation of the mechanism files to be included in application codes is a critical step because of the large variety of possible mechanisms and the importance of being able to switch mechanism types rapidly to explore different cases. We have written a Python-based code generator, called Chemistry Evaluation for Pele Through Recasting (CEPTR), which is included in PelePhysics [6,41]. CEPTR is used to generate C++ code for the evaluation of chemical reaction mechanisms. It is particularly formulated to generate efficient code for heterogeneous computing architectures, e.g., graphics processing units (GPUs). The input file for CEPTR is a Cantera mechanism input file, which can easily be obtained from a CHEMKIN mechanism file. This input file is parsed for finite rate chemistry evaluation and transport model specifications. These specifications are used to automatically formulate C++ expressions that are written to C++ header and source code files. The rate expressions are parsed to eliminate redundant computation by precomputing expressions, e.g., the arguments of the exponential functions in the Arrhenius rate equations. Efforts were made to reduce thread private arrays, in favor of constant global arrays to reduce register pressure, with a bias towards the GPU programming model.

Appendix B. An overview of Tarjan's algorithm for SCC QSS species identification

At the start, all vertices (QSS species) are considered unvisited. Choose any vertex to initiate the recursion; this will be known as the parent. Add the parent to a potential SCC stack and assign it a low-link value, which should match the discovery order upon initialization (note: position in the stack implicitly stores the discovery order via the

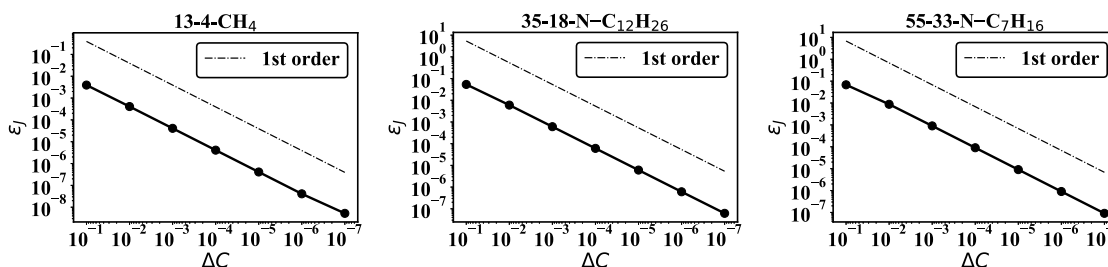


Fig. C.14. Convergence of the finite difference Jacobian towards the symbolic Jacobian as a function of the discretization size for the molar concentrations. Left: 13-4-CH₄. Middle: 35-18-N-C₁₂H₂₆. Right: 55-33-N-C₇H₁₆.

use of an ordered dictionary). Loop over the adjacent vertices of the parent; these are considered the parent's children. If a child has not been visited yet (i.e., the child does not have a low-link value), recurse; the child becomes the next parent, enabling the depth-first search to continue.

Throughout the recursion, low-link values are updated in two scenarios: (1) the current child has been visited before and exists on the stack (note: if a child has been visited before and is not on the stack, it is part of a previously discovered SCC and can thus be ignored). The parent's low-link value is updated with the child's discovery order if this value is lower. (2) the end of a recursion is reached (i.e., all children of a given parent have been visited and updates have commenced as described in (1); the parent then reverts to being a child for its parent). The parent's low-link value is updated with the child's low-link value if this value is lower. If the low-link value and discovery order of the parent are still equal to each other after the end-of-recursion update, that vertex is the root of an SCC. Vertices are then popped from the stack until the root vertex is removed, generating the fully sequestered SCC. Vertices that were visited but are part of a different SCC remain on the stack until the recursion they are a part of is completed and the next root is found.

Once all SCCs are found, edge information is updated in the adjacency list to reflect SCC dependencies, as demonstrated in Fig. 2(b), and to determine the order of resolution. A conclusive order of execution for back substitution will always be possible at this point, as no cyclical paths will occur outside of the already determined SCCs.

With only unidirectional edges remaining, back substitution is implemented for the final dependencies while an in-house Gaussian pivoting routine is used to manage the algebraic relationships within individual SCCs.

Appendix C. Convergence between the finite difference and the symbolic Jacobian

In Section 3.2.2, the accuracy of the symbolic Jacobian is assessed by comparing it to a finite difference Jacobian computed with a single discretization size for the species concentrations which allows a direct comparison with the zero QSS and the constant QSS approaches. Here, the error between the numerical Jacobian and the symbolic Jacobian is studied for different discretization sizes in the molar concentration space.

In Fig. C.14, the relative error between the finite difference Jacobian and the semi-analytical Jacobian (Eq. (10)) is computed and averaged over 1000 random samples throughout the composition space (molar concentration of species uniformly sampled from [0.1–1.1] mol/m³ and temperature uniformly sampled from $T = [300 - 1300]$ K. The FD Jacobian is computed with a one-sided, first-order method, computed with discretization sizes in the molar concentration space $\Delta C \in [10^{-1}, 10^{-7}]$. It can be seen that the FDJ converges towards the symbolic AJ generated, for all three mechanisms considered. The error also converges with a first-order rate which suggests that the error between the symbolic and the finite difference Jacobian is explained by the discretization size.

Appendix D. Practical benefit of common subexpression elimination

In Section 3.2.2, the common subexpressions identified are recurrent mathematical expressions that are found to appear several times in several entries of the chemical Jacobian. By precomputing such expressions, one can (a) abbreviate the file size and (b) reduce the total number of arithmetic operations. Abbreviating the file size is necessary to ensure that the mechanism generated can be compiled.

As an illustrative example, consider the case of the 13-4-CH₄ mechanisms without memory optimization. One common expression found is `const amrex::Real x54 = sc[12] * kf_qss[28];`. Here, `x54` is the name of the common subexpression, `sc[12]` is one of the non-QSS species concentrations, and `kf_qss[28]` is one of the rates constants. This expression is reused three times when constructing the analytical Jacobian.

In terms of file size abbreviation: creating the common subexpression requires 44 characters and using the expression later requires 5 characters for each common subexpression occurrence, for a total of 59 characters. In contrast, without the common subexpression, one would need 21 characters for every common subexpression occurrence, hence 63 characters.

In terms of the number of arithmetic operations: creating the aforementioned common subexpression requires one multiplication. Not creating the expression requires performing one multiplication three times. Creating a common subexpression allows to reduce the total number of arithmetic operations, at the expense of one variable stored in memory.

Obviously, the larger the common subexpression (in terms of characters and number of arithmetic operations), and the more often it is used, the larger the gains. The question of which common subexpression to keep is tackled in Section 5.3 where it is shown that it is possible to reduce the number of precomputations without modifying the computational cost.

References

- [1] N.T. Wimer, L. Esclapez, N. Brunhart-Lupo, M.T. Henry de Frahan, M. Rahimi, M. Hassanaly, J. Rood, S. Yellapantula, H. Sitaraman, B. Perry, et al., Visualizations of a methane/diesel RCCI engine using PeleC and PeleLMEx, *Phys. Rev. A* 8 (11) (2023) 110511.
- [2] F. Alexander, A. Almgren, J. Bell, A. Bhattacharjee, J. Chen, P. Colella, D. Daniel, J. DeSlippe, L. Diachin, E. Draeger, et al., Exascale applications: skin in the game, *Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci.* 378 (2166) (2020) 20190056.
- [3] N. Malaya, B. Messer, J. Glenski, A. Georgiadou, J. Lietz, K. Gottiparthi, M. Day, J. Chen, J. Rood, L. Esclapez, J. White III, G.R. Jansen, N. Curtis, S. Nichols, J. Kurzak, N. Chalmers, C. Freitag, P. Bauman, A. Fanfarillo, R.D. Budiardja, T. Papatheodore, N. Frontiere, D. McDougall, M. Norman, S. Sreepathi, P. Roth, D. Bykov, N. Wolfe, P. Mullowney, M. Eisenbach, M.T. Henry de Frahan, W. Joubert, Experiences readying applications for exascale, in: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '23*, Association for Computing Machinery, New York, NY, USA, 2023, pp. 1–13.

- [4] L. Esclapez, M. Day, J. Bell, A. Felden, C. Gilet, R. Grout, M. Henry de Frahan, E. Motheau, A. Nonaka, L. Owen, B. Perry, J. Rood, N. Wimer, W. Zhang, PeLeMeX: An AMR low mach number reactive flow simulation code without level sub-cycling, *J. Open Source Softw.* 8 (90) (2023) 5450.
- [5] M.T. Henry de Frahan, J.S. Rood, M.S. Day, H. Sitaraman, S. Yellapantula, B.A. Perry, R.W. Grout, A. Almgren, W. Zhang, J.B. Bell, J.H. Chen, PeleC: An adaptive mesh refinement solver for compressible reacting flows, *Int. J. High Perform. Comput. Appl.* 37 (115–131) (2022) 109434202211211.
- [6] M.T. Henry de Frahan, L. Esclapez, J. Rood, N.T. Wimer, P. Mullowney, B.A. Perry, L. Owen, H. Sitaraman, S. Yellapantula, M. Hassanaly, M.J. Rahimi, M.J. Martin, O.A. Doronina, S.N. A., M. Rieth, W. Ge, R. Sankaran, A.S. Almgren, W. Zhang, J.B. Bell, R. Grout, M.S. Day, J.H. Chen, The pele simulation suite for reacting flows at exascale, in: *Proceedings of the 2024 SIAM Conference on Parallel Processing for Scientific Computing*, 2024, pp. 13–25.
- [7] S. Treichler, M. Bauer, A. Bhagatwala, G. Borghesi, R. Sankaran, H. Kolla, P.S. McCormick, E. Slaughter, W. Lee, A. Aiken, et al., S3D-legion: An exascale software for direct numerical simulation of turbulent combustion with complex multicomponent chemistry, in: *Exascale Scientific Applications*, Chapman and Hall/CRC, 2017, pp. 257–278.
- [8] D. Mira, E.J. Pérez-Sánchez, R. Borrell, G. Houzeaux, HPC-enabling technologies for high-fidelity combustion simulations, *Proc. Combust. Inst.* 39 (4) (2023) 5091–5125.
- [9] F. Pignatelli, M. Passad, A. Åkerblom, T. Nilsson, E. Nilsson, C. Fureby, Predictions of spray combustion using conventional category a fuels and exploratory category C fuels, in: *AIAA Scitech 2023 Forum*, American Institute of Aeronautics and Astronautics, 2023, p. 1486.
- [10] A. Felden, L. Esclapez, E. Riber, B. Cuenot, H. Wang, Including real fuel chemistry in LES of turbulent spray combustion, *Combust. Flame* 193 (2018) 397–416.
- [11] W.T. Chung, N. Ly, M. Ihme, LES of HCCI combustion of iso-octane/air in a flat-piston rapid compression machine, *Proc. Combust. Inst.* 39 (4) (2023) 5309–5317.
- [12] Y. Tang, M. Hassanaly, V. Raman, B.A. Sforzo, J. Seitzman, Probabilistic modeling of forced ignition of alternative jet fuels, *Proc. Combust. Inst.* 38 (2) (2021) 2589–2596.
- [13] M. Grader, P. Gerlinger, Influence of operating conditions on flow field dynamics and soot formation in an aero-engine model combustor, *Combust. Flame* (2023) 112712.
- [14] T. Jaravel, E. Riber, B. Cuenot, G. Bulat, Large eddy simulation of an industrial gas turbine combustor using reduced chemistry with accurate pollutant prediction, *Proc. Combust. Inst.* 36 (3) (2017) 3817–3825.
- [15] O. Mathieu, E.L. Petersen, Experimental and modeling study on the high-temperature oxidation of ammonia and related NO_x chemistry, *Combust. Flame* 162 (3) (2015) 554–570.
- [16] A. Attili, F. Bisetti, M.E. Mueller, H. Pitsch, Formation, growth, and transport of soot in a three-dimensional turbulent non-premixed jet flame, *Combust. Flame* 161 (7) (2014) 1849–1865.
- [17] C.J. Balos, M. Day, L. Esclapez, A.M. Felden, D.J. Gardner, M. Hassanaly, D.R. Reynolds, J. Rood, J.M. Sexton, H. Sitaraman, N.T. Wimer, C.S. Woodward, SUNDIALS time integrators for exascale applications with many independent ODE systems, 2024, arXiv preprint arXiv:2405.01713.
- [18] Y. Shi, W.H. Green, H.-W. Wong, O.O. Oluwole, Accelerating multi-dimensional combustion simulations using GPU and hybrid explicit/implicit ODE integration, *Combust. Flame* 159 (7) (2012) 2388–2397.
- [19] J. Kodavasal, K. Harms, P. Srivastava, S. Som, S. Quan, K. Richards, M. García, Development of a stiffness-based chemistry load balancing scheme, and optimization of input/output and communication, to enable massively parallel high-fidelity internal combustion engine simulations, *J. Energy Resour.-ASME* 138 (5) (2016).
- [20] H. Wu, P.C. Ma, T. Jaravel, M. Ihme, Pareto-efficient combustion modeling for improved CO-emission prediction in LES of a piloted turbulent dimethyl ether jet flame, *Proc. Combust. Inst.* 37 (2) (2019) 2267–2276.
- [21] R. Mao, M. Lin, Y. Zhang, T. Zhang, Z.-Q.J. Xu, Z.X. Chen, DeepFlame: A deep learning empowered open-source platform for reacting flow simulations, *Comput. Phys. Comm.* 291 (2023) 108842.
- [22] T. Lu, C.K. Law, Toward accommodating realistic fuel chemistry in large-scale computations, *Prog. Energy Combust.* 35 (2) (2009) 192–215.
- [23] M.J. McNenly, R.A. Whitesides, D.L. Flowers, Faster solvers for large kinetic mechanisms using adaptive preconditioners, *Proc. Combust. Inst.* 35 (1) (2015) 581–587.
- [24] A.S. Walker, R.L. Speth, K.E. Niemeyer, Generalized preconditioning for accelerating simulations with large kinetic models, *Proc. Combust. Inst.* 39 (4) (2023) 5395–5403.
- [25] N.J. Curtis, K.E. Niemeyer, C.-J. Sung, Using SIMD and SIMT vectorization to evaluate sparse chemical kinetic Jacobian matrices and thermochemical source terms, *Combust. Flame* 198 (2018) 186–204.
- [26] T. Turányi, A.S. Tomlin, *Analysis of Kinetic Reaction Mechanisms*, Vol. 20, Springer, 2014.
- [27] T. Lu, C.K. Law, Linear time reduction of large kinetic mechanisms with directed relation graph: n-heptane and iso-octane, *Combust. Flame* 144 (1–2) (2006) 24–36.
- [28] A. Felden, P. Pepiot, L. Esclapez, E. Riber, B. Cuenot, Including analytically reduced chemistry (ARC) in CFD applications, *Acta Astronaut.* 158 (2019) 444–459.
- [29] T. Lu, C.K. Law, Strategies for mechanism reduction for large hydrocarbons: n-heptane, *Combust. Flame* 154 (1–2) (2008) 153–163.
- [30] P. Pepiot, H. Pitsch, An efficient error-propagation-based reduction method for large chemical kinetic mechanisms, *Combust. Flame* 154 (1–2) (2008) 67–81.
- [31] A.S. Tomlin, M.J. Pilling, T. Turányi, J.H. Merkin, J. Brindley, Mechanism reduction for the oscillatory oxidation of hydrogen: Sensitivity and quasi-steady-state analyses, *Combust. Flame* 91 (2) (1992) 107–130.
- [32] M. Bodenstein, Eine theorie der photochemischen reaktionsgeschwindigkeiten, *Z. Phys. Chem.* 85 (1) (1913) 329–397.
- [33] S.J. Fraser, The steady state and equilibrium approximations: A geometrical picture, *J. Chem. Phys.* 88 (8) (1988) 4732–4738.
- [34] T. Lu, C.K. Law, Systematic approach to obtain analytic solutions of quasi steady state species in reduced mechanisms, *J. Phys. Chem. A* 110 (49) (2006) 13202–13208.
- [35] C. Safta, H.N. Najm, O. Knio, TChem-A Software Toolkit for the Analysis of Complex Kinetic Models, Technical Report, Sandia National Lab.(SNL-CA), Livermore, CA (United States), 2011.
- [36] K.E. Niemeyer, N.J. Curtis, C.-J. Sung, pyJac: Analytical Jacobian generator for chemical kinetics, *Comput. Phys. Comm.* 215 (2017) 188–203.
- [37] F. Perini, E. Galligani, R.D. Reitz, An analytical Jacobian approach to sparse reaction kinetics for computationally efficient combustion modeling with large reaction mechanisms, *Energy Fuel* 26 (8) (2012) 4804–4822.
- [38] F. Bisetti, Integration of large chemical kinetic mechanisms via exponential methods with Krylov approximations to Jacobian matrix functions, *Combust. Theory Model.* 16 (3) (2012) 387–418.
- [39] T. Dijkmans, C.M. Schietekat, K.M. Van Geem, G.B. Marin, GPU based simulation of reactive mixtures with detailed chemistry in combination with tabulation and an analytical Jacobian, *Comput. Chem. Eng.* 71 (2014) 521–531.
- [40] P. Sharma, Acceleration Techniques for Efficient and Accurate Particle PDF Simulations of Large-Scale Turbulent Combustion (Ph.D. thesis), Cornell University, 2022.
- [41] H. Sitaraman, S. Yellapantula, R. Grout, PelePhysics [SWR-17-36], 2017, <http://dx.doi.org/10.11578/dc.20171025.1989>, [Computer Software].
- [42] T. Turányi, A.S. Tomlin, M.J. Pilling, On the error of the quasi-steady-state approximation, *J. Phys. Chem.* 97 (1) (1993) 163–172.
- [43] C.K. Law, C.J. Sung, H. Wang, T. Lu, Development of comprehensive detailed and reduced reaction mechanisms for combustion modeling, *AIAA J.* 41 (9) (2003) 1629–1646.
- [44] G. Borghesi, A. Krisman, T. Lu, J.H. Chen, Direct numerical simulation of a temporally evolving air/n-dodecane jet at low-temperature diesel-relevant conditions, *Combust. Flame* 195 (2018) 183–202.
- [45] T. Yao, Y. Pei, B.-J. Zhong, S. Som, T. Lu, K.H. Luo, A compact skeletal mechanism for n-dodecane with optimized semi-global low-temperature chemistry for diesel engine simulations, *Fuel* 191 (2017) 339–349.
- [46] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, C.S. Woodward, SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers, *ACM Trans. Math. Software* 31 (3) (2005) 363–396.
- [47] T. Lu, C.K. Law, A directed relation graph method for mechanism reduction, *Proc. Combust. Inst.* 30 (1) (2005) 1333–1341.
- [48] R. Tarjan, Depth-first search and linear graph algorithms, *SIAM J. Comput.* 1 (2) (1972) 146–160.
- [49] M. Sharir, A strong-connectivity algorithm and its applications in data flow analysis, *Comput. Math. Appl.* 7 (1) (1981) 67–72.
- [50] M. Gadalla, Implementation of analytical Jacobian and chemical explosive mode analysis (CEMA) in OpenFOAM, 2022, arXiv preprint arXiv:2205.07416.
- [51] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, 2003.
- [52] D.A. Knoll, D.E. Keyes, Jacobian-free Newton-Krylov methods: a survey of approaches and applications, *J. Comput. Phys.* 193 (2) (2004) 357–397.
- [53] M. Hassanaly, H. Koo, C.F. Lietz, S.T. Chong, V. Raman, A minimally-dissipative low-mach number solver for complex reacting flows in OpenFOAM, *Comput. & Fluids* 162 (2018) 11–25.
- [54] O. Desjardins, G. Blanquart, G. Balarac, H. Pitsch, High order conservative finite difference scheme for variable density low mach number turbulent flows, *J. Comput. Phys.* 227 (15) (2008) 7125–7159.
- [55] R. Sankaran, E.R. Hawkes, J.H. Chen, T. Lu, C.K. Law, Structure of a spatially developing turbulent lean methane-air bunsen flame, *Proc. Combust. Inst.* 31 (1) (2007) 1291–1298.
- [56] C.S. Yoo, T. Lu, J.H. Chen, C.K. Law, Direct numerical simulations of ignition of a lean n-heptane/air mixture with temperature inhomogeneities at constant volume: Parametric study, *Combust. Flame* 158 (9) (2011) 1727–1741.
- [57] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, Accelerating numerical dense linear algebra calculations with GPUs, *Numer. Comput. GPUs* (2014) 1–26.
- [58] A. Meurer, C.P. Smith, M. Paprocki, O. Čertík, S.B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J.K. Moore, S. Singh, T. Rathnayake, S. Vig, B.E. Granger, R.P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M.J. Curry, A.R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kula, R. Cimrman, A. Scopatz, SymPy: symbolic computing in python, *PeerJ Comput. Sci.* 3 (2017) e103.

- [59] M. Rein, The partial-equilibrium approximation in reacting flows, *Phys. Fluids A* 4 (5) (1992) 873–886.
- [60] M. Rouhi Youssefi, Development of Analytic Tools for Computational Flame Diagnostics (Ph.D. thesis), University of Connecticut, 2011.
- [61] D. Guide, *Cuda C Best Practices Guide*, NVIDIA, 2013, July.
- [62] S. Tomov, R. Nath, H. Ltaief, J. Dongarra, Dense linear algebra solvers for multicore with GPU accelerators, in: 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, IPDPSW, IEEE, 2010, pp. 1–8.
- [63] S. Tomov, J. Dongarra, M. Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems, *Parallel Comput.* 36 (5–6) (2010) 232–240.
- [64] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, Accelerating numerical dense linear algebra calculations with GPUs, *Numer. Comput. GPUs* (2014) 3–28.
- [65] J.-M. Senoner, M. García, S. Mendez, G. Staffelbach, O. Vermorel, T. Poinso, Growth of rounding errors and repetitivity of large eddy simulations, *AIAA J.* 46 (7) (2008) 1773–1781.
- [66] M. Hassanaly, V. Raman, Ensemble-LES analysis of perturbation response of turbulent partially-premixed flames, *Proc. Combust. Inst.* 37 (2) (2019) 2249–2257.
- [67] M. Hassanaly, V. Raman, Lyapunov spectrum of forced homogeneous isotropic turbulent flows, *Phys. Rev. A* 4 (11) (2019) 114608.