ARL-TR-9908 ● Apr 2024

Computational Advances in Modeling Opposed-Flow Diffusion Flames with Detailed Chemical Kinetics

by Christopher P Stone

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-9908 ● APR 2024

Computational Advances in Modeling Opposed-Flow Diffusion Flames with Detailed Chemical Kinetics

Christopher P Stone
DEVCOM Army Research Laboratory

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
April 2024	Technical Report	1/10/2022	1/9/2023

4. TITLE AND SUBTITLE
Computational Advances in Modeling Opposed-Flow Diffusion Flames with Detailed Chemical Kinetics

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Christopher P Stone

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
DEVCOM Army Research Laboratory ATTN: FCDD-RLA-WC Aberdeen Proving Ground, MD 21005	ARL-TR-9908

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID: Christopher P Stone, 0000-0002-9621-5334

14. ABSTRACT
The opposed-flow diffusion flame model, OPPDIF, has been refactored to improve its computational performance when using chemical kinetics mechanisms with hundreds or thousands of chemical species. A novel, block-tridiagonal matrix LU factorization method was formulated and used within the flame model's nonlinear (Newton–Raphson) solver to directly solve the Jacobian-vector linear system. The new method has a theoretical run-time that is a factor of more than four faster than the original factorization method, based on the LAPACK banded matrix format, and requires only half the storage. This was achieved by maintaining a block-tridiagonal matrix structure even while performing (partial) row pivoting for numerical stability. Additionally, new methods for assembling the system's Jacobian matrix and approximating its condition number were implemented and benchmarked using multicore (multithreaded) parallelism. In benchmark tests performed with 48 cores with two Intel Xeon Skylake CPUs and a large chemical kinetics mechanism with more than 1,300 species, the algorithm optimizations reduced the run-time by a factor of 5.7 over a previous parallel implementation. With a reduced, 600-species version of the mechanism, the run-time was reduced by a factor of 3.8. In addition to the net performance improvement, the parallel scalability of the OPPDIF model was improved by more than 100% on the large mechanism, allowing more efficient use of high-performance computing resources.

15. SUBJECT TERMS
Weapons Sciences, diffusion flame modeling, OPPDIF, block-tridiagonal matrix factorization, high-performance computing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	37

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Christopher P Stone	(678) 938-7292

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures iv

List of Tables iv

Acknowledgments v

1. Introduction 1

2. Model Formulation 3

3. Numerical Method 7

4. Algorithm Enhancements 9

4.1 Parallelization 10

4.2 Jacobian Factorization 11

4.3 Condition Number Estimation 14

5. Results and Discussion 15

6. Summary and Conclusions 21

7. References 23

Appendix. Complexity Analysis 26

List of Symbols, Abbreviations, and Acronyms 30

List of Figures

Fig. 1 Matrix sparsity with five mesh points. (a) Exact block-tridiagonal matrix without infill. (b) Factored banded matrix with padding (yellow) and pivot in-fill (orange). (c) Exact block-tridiagonal matrix with superblock pivot (yellow). 11

Fig. 2 Total run-time and function breakdown for OPPDIF simulation with the (a) full and (b) reduced (S600) EODT mechanisms using the baseline parallel algorithm and the four optimizations. Optimizations are applied cumulatively. 17

Fig. 3 Parallel speedup for OPPDIF simulation with the (a) full, (b) medium (S813), and (c) small (S600) mechanisms with the four optimizations 19

List of Tables

Table 1 Leading cost estimates for factorization and linear system solve for the banded and block-tridiagonal matrix methods 13

Table 2 EODT mechanism sizes and mesh resolution used in benchmarks 15

Table A-1 Cost estimates for LAPACK matrix operations used by band and block matrix factorization and linear solve methods 27

Acknowledgments

The author is eternally grateful for the many contributions from colleagues during this research effort. Dr Michael J McQuaid provided essential technical guidance and feedback on this report and shared the initial OPPDIF simulation results with a reduced mechanism from which the larger simulations were based. Dr Chiung-Chu Chen provided the latest EODT (*6-ethenyl-2,8,12,16-octadecetetraene*) chemical kinetics mechanism and the species Simplified Molecular Input Line Entry System (SMILES) strings the author used to estimate their transport properties.

Computing resources for this work were provided by the DOD High-Performance Computing Modernization Program. Systems used include Narwhal, Koehr, and Gaffney (US Navy Defense Shared Resource Center [DSRC]); Warhawk (US Air Force Research Laboratory DSRC); and Jean (US Army Combat Capabilities Development Command [DEVCOM] Army Research Laboratory [ARL] DSRC).

Funding for this study was primarily provided by ARL mission programs. Partial funding was provided by the Office of Naval Research project titled, "Fundamental Analysis of Pressurized Combustion and Decomposition of Hydroxyl-Terminated Polybutadiene."

1. Introduction

Opposed-flow diffusion flames (OPDFs) provide a useful laboratory setup for studying the combustion of separate impinging fuel and oxidizer streams or of solid fuels (e.g., strand burners) burning against an incoming oxidizer jet. Under well-defined and idealized conditions (i.e., steady-state, laminar flow, and constant pressure), the OPDF configuration can be accurately simulated using a 1-D numerical model. The computational cost savings from the 1-D approximation, compared to 2-D or 3-D simulations, affords the use of detailed, finite-rate chemical kinetics mechanisms. By comparing with experimental measurements, an OPDF model can be used to validate these mechanisms. Once validated, the models and the kinetics mechanisms can then be used to provide insight into the flame structure inaccessible to experimental methods. Furthermore, these models can be used to predict regression rates of solid fuel and propellant formulations.[1,2]

As detailed later, the numerical model of the OPDF, when discretized using a finite-different method, requires the solution of a highly nonlinear system of differential equations with $N_g M_b$ unknowns where N_g is the number of mesh points used to discretize the spatial domain and $M_b = (4 + n_s)$ is the number of unknowns at each point with n_s gas-phase chemical species.

Two commonly used OPDF models, OPPDIF[3] and Cantera,[4] solve the nonlinear equations iteratively using a damped, modified Newton–Raphson method. This approach requires the frequent assembly and factorization of the system's Jacobian matrix (or an approximation thereof), a block-tridiagonal matrix with N_g block rows, and square matrix blocks of size ${M_b}^2$. Both software packages use direct matrix factorization (e.g., lower–upper factorization with partial row pivoting [LUP]) to solve the block-tridiagonal linear system. Assuming $n_s \gg 1$ and $N_g \gg 1$, the computational cost* of the matrix factorization scales as $\mathcal{O}\left(N_g {n_s}^3\right)$ and the storage as $\mathcal{O}\left(N_g {n_s}^2\right)$. Since the factorized Jacobian matrix can be reused many times in the modified Newton–Raphson method, the overall run-time of the OPDF model scales between ${n_s}^2$ and ${n_s}^3$ since other more frequent operations (e.g., linear system solves) have leading costs of $\mathcal{O}\left(N_g {n_s}^2\right)$.

While the computational cost of evaluating the reaction rates and mixture-averaged transport properties (e.g., species diffusion coefficients) is high, the Jacobian assembly and factorization dominates for large chemical mechanisms due to the cubic scaling. Typical values of N_g are $\mathcal{O}(100 - 1{,}000)$ for detailed chemical

*Computational cost is approximated as the leading number of floating-point operations.

mechanisms for combustion simulations. The mesh resolution requirement can be influenced by the chemistry (e.g., faster reactions or diffusion); however, the overall mesh size does not vary greatly across mechanisms owing to the resolving efficiency of the adaptive mesh refinement capabilities provided by OPPDIF and Cantera.

Unlike mesh sizes, mechanism sizes vary widely across different combustion modeling applications with n_s spanning $\mathcal{O}(10 - 10{,}000)$. The number of reactions (n_r) is typically 5–20 times larger than n_s. For example, the GRI-Mech (v3.0) mechanism,[5] often used to model light hydrocarbon (e.g., methane) combustion, has 53 species and 325 reactions while the US Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory (ARL) "HTPB" mechanism, which models the decomposition of $C_{20}H_{32}$, is 25 times larger with over 1,300 species and nearly 6,000 reactions. The ARL mechanism is designed for modeling the gas-phase combustion of the hydrocarbon *6-ethenyl-2,8,12,16-octadecetetraene* (EODT), which is postulated as being representative of the nascent products (NPs) emanating from the pyrolysis of R45M-type *hydroxyl-terminated polybutadiene* (HTPB), a common propellant ingredient in rocket motors. See Chen and McQuaid,[2,6] and references therein for details on the motivation and evolution of the ARL EODT mechanism.

OPDF modeling using the full EODT mechanism, and similarly large mechanisms related to propellant pyrolysis and combustion modeling, are intractable using the standard versions of OPPDIF or Cantera packages due to the cubic time and quadratic storage costs. For example, the run-time, typically $\mathcal{O}(0.1 - 1)$ min with GRI-Mech on a modern (ca., 2023) desktop computer, would increase to $\mathcal{O}(10)$ days with the full EODT mechanisms and would require $\mathcal{O}(100)$ GB.

OPDFs, along with burner-stabilized premixed flames, are used to model the gas-phase combustion in models coupling condensed- and gas-phase combustion. These models, such as that of Miller and Anderson[7] and Geipel et al.,[8] are used to predict propellant linear burning rates by iteratively adjusting the fuel boundary condition to converge energy and mass conservation across the condensed- and gas-phase interface. This means that the flame models are solved repeatedly, often requiring $\mathcal{O}(10)$ successive flame solutions, to converge the condensed- and gas-phase interface conditions. Similarly, the trial mechanism method (TMM),[1] a statistical method used to reduce the size of chemical kinetics mechanisms for specific applications, repeatedly solves a set of predefined flame simulations using a sequence of smaller mechanisms to identify important reactions and species. As a statistical method, TMM may require $\mathcal{O}(10 - 1{,}000)n_r$ simulations though the size of the mechanism (and the computational cost) declines as the mechanism is reduced. The high cost of the flame models is exacerbated when applied within

these iterative methods and is a driving factor in the current effort to optimize the computational performance of the models and reduce their cost.

To reduce the high run-time and memory requirements of the OPDF model, an alternative direct matrix factorization method, specialized for block-tridiagonal matrices, has been implemented within the OPPDIF model. The factorization method has lower theoretical run-time and storage cost compared to the original banded matrix format used in OPPDIF and Cantera. This method does not alter the nonlinear algorithm and can be used directly within the existing flame models (and similar applications); however, it retains the same cubic cost scaling of the band-matrix method. Iterative linear solvers avoid direct factorization and, if well-conditioned, may reduce the time complexity to quadratic or even linear. For example, Lapointe et al.[9] demonstrated that (steady) freely propagating premixed flames can be solved using a Jacobian-free Newton–Krylov (JFNK) method in nearly linear time (i.e., $\mathcal{O}(n_s)$). While this, and similar Krylov-space methods, avoid direct matrix factorization costs, convergence is strongly dependent on preconditioning, which can be application-specific and, as such, is not as robust as direct methods.

In addition to optimizing direct matrix factorizations, more efficient methods for approximating the Jacobian matrix and for estimating its condition number were implemented to reduce the overall cost of the OPDF model.

This report is organized as follows:

- OPPDIF's mathematical model is detailed;
- the nonlinear solver method and related algorithms are summarized;
- the new matrix factorization and related optimizations are detailed; and
- finally, performance benchmarks are provided to demonstrate the impact of these advances on the performance and efficiency of the OPPDIF model.

2. Model Formulation

For this study, the author employed a modified version of the OPPDIF combustion model.[3] It used the pre-commercial CHEMKIN library (version 3.0)[10] for thermodynamic and chemical kinetics computations and the associated TRANLIB library for molecular transport properties[11] (e.g., conductivity, viscosity, and species diffusivity).

The OPPDIF model simulates a steady and laminar stagnation flame formed by the impingement of two opposing, circular gaseous streams. Under an axisymmetric assumption, the mass conservation in cylindrical coordinates is

$$\frac{\partial \rho u}{\partial z} + \frac{1}{r}\frac{\partial \rho v r}{\partial r} = 0\,. \tag{1}$$

Here, ρ is the gas density and $[u, v]$ are the axial (z) and radial (r) velocity components. As detailed in Lutz et al.,[3] the 2-D model can be reformulated as a 1-D model with the introduction of the similarity variables, $G(z) = -\rho v/r$ and $F(z) = \rho u/2$, that are functions only of z. With these, the continuity equation reduces to

$$\mathrm{G}(z) = \frac{dF(z)}{dz}\,. \tag{2}$$

The momentum, energy, species conservation equations also become functions of z only. Assuming constant axial pressure (i.e., ignoring gas compressibility), these are

$$\begin{gathered} H - \frac{d}{dz}\left(\frac{FG}{\rho}\right) + \frac{3G^2}{\rho} + \frac{d}{dz}\left[\mu\frac{d}{dz}\left(\frac{G}{\rho}\right)\right] = 0 \\ \rho u\frac{dT}{dz} - \frac{1}{c_p}\frac{d}{dz}\left(\lambda\frac{dT}{dz}\right) + \frac{\rho}{c_p}\sum_k c_{p_k} Y_k V_k \frac{dT}{dz} + \frac{1}{\mathrm{c_p}}\sum_k h_k \dot{\omega}_k = 0 \\ \rho u \frac{dY_k}{dz} - \frac{d}{dz}(\rho Y_k V_k) - \dot{\omega}_k W_k = 0, \end{gathered} \tag{3}$$

where T and Y_k are the temperature and species mass fractions; μ and λ are the mixture-averaged gas viscosity and conductivity; c_p is the mixture-averaged heat capacity at constant pressure; c_{p_k}, h_k, $\dot{\omega}_k$, V_k, and W_k are the heat capacity, molar enthalpy, net molar reaction rate, diffusion velocity, and molar mass of the k^{th} species, respectively. Lastly, $H = \frac{1}{r}\frac{\partial \mathrm{p}}{\partial \mathrm{r}}$ is a constant (eigenvalue) satisfying the radial momentum. To maintain a block-tridiagonal matrix structure, this constraint is solved as $dH/dz = 0$.

The system of equations is closed by specifying a model for 1) the gas equation of state; 2) the thermochemical reaction rates; 3) the diffusion velocity (e.g., mixture-averaged, or multicomponent species transport) and other transport properties; and 4) the velocity, chemical composition, and temperature at the fuel and oxidizer boundaries. The closure models are summarized in the following.

We assume a thermally perfect mixture of the gaseous chemical species governed by the ideal gas law. All thermodynamic properties and chemical reaction rates are

computed by the CHEMKIN library. Note that we have added support for pressure-dependent reactions via tabulated log-pressure interpolation (PLOG) and Tsang and Herron[12] models to our in-house version of the CHEMKIN library.

The diffusion velocity is modeled as mixture-averaged, such that,

$$V_k = -\frac{1}{X_k} D_k^m \frac{dX_k}{dz} - \frac{D_k^T}{\rho Y_k} \frac{1}{T} \frac{dT}{dz}$$

$$D_k^m = \frac{1 - Y_k}{\sum_{j \neq k}^{K} X_j / \mathcal{D}_{jk}}, \tag{4}$$

where X_k are the mole fractions; and D_k^m, D_k^T and $\mathcal{D}_{jk}$ are the mixture-averaged diffusion, thermal diffusion, and multicomponent diffusion coefficients of the species. The mixture-averaged and multicomponent species diffusion, thermal conductivity, and viscosity coefficients are computed by the TRANLIB library given the local thermodynamic state.

The use of mixture-averaged transport is justified due to the lack of reliable information on the transport properties for many of the chemical species in our chemical mechanism. For example, the EODT mechanism currently involves 1,373 species, of which only 115 have published data for the molecular properties required by the TRANLIB library when generating polynomial fits of the transport properties from kinetic theory.[11] That is, we lack data for the Lennard–Jones well depth (ε) and collision diameter (σ); the dipole moment (μ); the polarizability (α); and the rotational relaxation collision number (z_{rot}). We approximate the remaining species using the Joback[13] group additivity method. We employ the Reaction Mechanism Generation[14–16] software to approximate the necessary molecular properties via Joback's method given the molecular structure of the chemical species (e.g., a canonical Simplified Molecular Input Line Entry System [SMILES] string).[17] Since we lack rigorous data for so many species in the EODT mechanism and rely upon approximations, multicomponent diffusion properties would be a wasteful expense with little improved model fidelity.

The final closure is the specification of the composition, temperature, and velocity at the fuel and oxidizer boundaries. The oxidizer composition, temperature and velocity are all presumed to be known. The boundary values are also assumed known when modeling a gaseous fuel jet. However, the fuel boundary conditions are not explicitly known when modeling a pyrolyzing solid propellant. In that scenario, we follow the framework of Miller and Anderson.[7,18] Their method, known as CYCLOPS, has been used to successfully predict the regression rates of burning propellants.

In CYCLOPS, we assume that condensed-phase chemical reactions and diffusion can be neglected and that only the energy and mass flux across the condensed- and gas-phase interface must be modeled. Under these assumptions, we must 1) specify the gaseous products emanating from the condensed-phase pyrolysis (i.e., we specify the fuel gas-phase composition explicitly) and 2) compute the gas-phase fuel temperature (T_f) and velocity (V_f) such that energy and mass are conserved across the interface. The energy flux across the interface can be expressed as

$$\left[\lambda_{\mathrm{g}} \frac{\mathrm{dT}}{\mathrm{dz}}\right]^{+0} = \dot{m}'' \left[\sum_{\mathrm{k}}^{\mathrm{NP}} (\mathrm{y_k})^{-0} (h_k)^{+0} - (h_c)^{-\infty}\right]. \tag{5}$$

On the left-hand side, the gas-phase conductivity (λ_g) and temperature gradient are evaluated at the interface boundary using only data from the gas phase (i.e., denoted by the +0 superscript). The right-hand side of Eq. 5 represents the net enthalpy carried into the gas phase with the mass flux rate ($\dot{m}''$). The net enthalpy is the difference between the bulk enthalpy of the condensed phase (h_c) and the gas phase enthalpy at the interface of the NP set (i.e., the set of chemical species assumed to emit from the pyrolysis of the condensed phase). The mass fractions $(y_k)^{-0}$ of the NP species are assumed to be known and specified as a boundary condition and their enthalpies $(h_k)^{+0}$ are determined by the OPDF model solution. The h_c is assumed to be a known property of the condensed phase and is generally some function of the heat of formation (ΔH_f), bulk temperature, and specific heat capacity.

To conserve mass, $\dot{m}''$ must be constant across the interface. The energy flux, Eq. 5, assumes that all heat conducted back into the pyrolysis layer is balanced entirely by the advection of chemical energy from the condensed phase. The drop in enthalpy across the interface represents all processes required to convert the solid propellant to the NP gases (e.g., endothermic phase chance).

The predicted fuel regression rate ($\dot{r}$) (i.e., the linear burning rate) and V_f are determined from the condensed- and gas-phase mass flux definitions:

$$\dot{m}'' = \rho_c \dot{r} = \rho_g V_f, \tag{6}$$

where ρ_c and ρ_g are the condensed- and gas-phase mass densities, respectively, with ρ_c being a constant or some function of the bulk temperature in the condensed phase and ρ_g being evaluated at the gas-phase side of the interface at the at the fuel boundary temperature (T_f) and composition.

T_f can be specified, if known, or can be similarly determined by introducing a second interface model coupling $\dot{m}''$ with T_f and solving simultaneously with Eq. 5. That is, we specify some relation

$$\dot{m}'' = \dot{m}''(T_f) \,. \tag{7}$$

Different empirical or physics-based models can be used in Eq. 7. For example, the mass flux can be modeled with the (empirical) pyrolysis law of Chiaverini et al.,[19] or as the evaporation of the fuel expressed via the Hertz–Langmuir–Knudsen relation (i.e., a physics-based model) as was demonstrated by Chen and McQuaid.[20]

These boundary flux equations constitute a nonlinear system of equations (i.e., two equations and two unknowns) that is solved iteratively. At each iteration, OPPDIF is solved with a given V_f and T_f using a previous solution as the initial guess for the gas-phase solution. In the present benchmark study, T_f is fixed at 750 K and only V_f is solved (i.e., only Eq. 5 is solved). Note that this does not alter the cost of solving an individual OPDF problem with a given set of fuel boundary conditions.

3. Numerical Method

The 1-D conservation equations for T, F, G, H, and Y_k are solved using a finite-difference method on a mesh with variable spacing. The convective terms are approximated with upwind differences (first-order) and the diffusion terms with central (second-order) differences.

At steady-state, the previous equations constitute a nonlinear, algebraic two-point boundary value problem that must be solved numerically. That is, OPPDIF seeks to find the root of the residual function

$$\mathcal{R}(g) = 0, \tag{8}$$

where g is the global state vector and $g_i = \left[T_i, F_i, G_i, H_i, Y_{i,k=1,n_s}\right]^T$ is the state vector at the i^{th} mesh point. Here, g is a block vector with N_g rows (i.e., mesh points) and M_b columns per row (i.e., unknowns per mesh point) leading to $N_g M_b$ unknowns overall.

OPPDIF employs the TWOPNT nonlinear boundary value solver[21] to solve the multivariate boundary value problem iteratively using a damped, modified Newton–Raphson method. When the steady-state system does not converge or converges too slowly, TWOPNT will advance the equations in pseudo-time (generally an easier nonlinear problem to solve) to drive the system toward a well of convergence.

In addition to solving the nonlinear boundary value problem, TWOPNT also refines the finite-difference mesh to resolve solution gradients and curvatures (i.e., second-derivatives) to within specified tolerances (i.e., `CURV` and `GRAD`). Values of 0.05 and 0.25 for `CURV` and `GRAD` are used, respectively, throughout this study resulting in meshes of between 300 and 400 points.

Absolute (`ATOL`) and relative (`RTOL`) tolerances of 10^{-9} and 10^{-6}, respectively, are used to terminate the TWOPNT nonlinear solver. Note that TWOPNT does not directly control the error in Eq. 8, as measured by some norm of $\mathcal{R}$, but considers the solution converged when the length of the Newton correction step (s) is less than the prescribed tolerances as measured in the ∞-norm. That is, the solution is deemed converged when the following is satisfied for all unknows at all mesh points:

$$|s| \leq \max(ATOL, RTOL * |g|) \, . \tag{9}$$

The step correction at each nonlinear iteration is found by solving the Newton–Raphson equation:

$$\mathcal{J}^m s^k = -\mathcal{R}(g^k) \tag{10}$$

where $\mathcal{J}$ is the Jacobian (or an approximation) of Eq. 8 (i.e., $\mathcal{J} = \partial\mathcal{R}/\partial g$).

As noted, TWOPNT uses a *modified* Newton method meaning that $\mathcal{J}$ does not have to be evaluated at the most recent solution iterate (i.e., $k \geq m$). This approach is a common optimization as the assembly (and factorization) of the Jacobian is costly. TWOPNT will update the Jacobian after a fixed number of iterations (`NJAC`) or if the step correction is rejected at some iterate.

TWOPNT accepts a new solution iterate based on the following criteria:

$$|s^{k+1}| < |s^k| \tag{11}$$

where the trial step (s^{k+1}) is found by solving Eq. 10 with $g^{k+1} = g^k + \lambda s^k$. Here, λ is the damping factor used to keep the solution within bounds (with $\lambda = 1$ being a full Newton step). TWOPNT does not search for an optimal λ (e.g., a *line search* method) but simply halves λ recursively for up to five iterations (i.e., $\lambda^l = 2^{-l}$, $l = 0{,}4$). If Eq. 11 is never satisfied and the Jacobian is out of date (i.e., $k > m$), TWOPNT updates $\mathcal{J}$ using g^k and searches for a new, acceptable damping coefficient. If that also fails, TWOPNT will resort to a fixed number of pseudo-time steps to drive the solution closer to a convergence well.

While it is possible to analytically evaluate many (or all) of the terms in $\mathcal{J}$ exactly or approximately,[9] OPPDIF approximates $\mathcal{J}$ entirely using a first-order finite

difference method. Each column of $\mathcal{J}$ is approximated by perturbing a single element of the state vector, such that,

$$\mathcal{J}_j = \frac{\mathcal{R}(g + \epsilon\hat{e}) - \mathcal{R}(g)}{\epsilon};\ \hat{e}_i = \begin{cases} 1, & i = j \\ 0, & else \end{cases} \tag{12}$$

where ϵ is a small but significant value relative the j^{th} state variable and $\hat{e}$ is a vector with only 1 in row j.

Since a three-point, central finite-difference stencil is used to approximate the spatial derivatives, the residual function at some point i (i.e., $\mathcal{R}_i$) is only dependent on itself and its neighboring points at $i-1$ and $i+1$. This results in a block-tridiagonal matrix with N_g block rows (i.e., one per mesh point) and square $M_b \times M_b$ blocks. Note that the boundary points are included in the residual function to allow the enforcement of the radial pressure gradient eigenvalue and other gradient conditions at the boundaries. For example, species can diffuse into the boundary points causing the steady-state composition at the boundaries to differ from the specified boundary values.

As noted earlier, the cost of the OPPDIF flame model is often dominated by the Jacobian assembly and factorization when factorized and solved directly. The Jacobian is reused over several nonlinear iterations to reduce the cost. Two additional optimizations (or approximations) are used by OPPDIF to reduce the assembly and factorization costs of the Jacobian:

1) Since the residual function at points i $\pm$ 2 and further apart are decoupled, $N_g/3$ columns of $\mathcal{J}$ are evaluated concurrently by perturbing every third block row of g. This greatly reduces the number of residual function evaluations needed to assemble the Jacobian matrix from $N_g M_b$ to only $3M_b$ and reduces the cost scaling of the approximation from quadratic to linear with respect to the mesh size.

2) OPPDIF does not reevaluate the transport coefficients when approximating $\mathcal{J}$ at the perturbed state in Eq. 12. Instead, the transport coefficients are evaluated only at unperturbed solution state (g^k) and frozen. The impact on the convergence is expected to be minor; however, this has not been investigated in this study.

4. Algorithm Enhancements

Improvements to the OPPDIF and underlying CHEMKIN libraries have been implemented to improve both the computational performance and the numerical efficiency. Note that while the improvements mentioned here have been

implemented in an in-house version of the legacy CHEMKIN library and OPPDIF, the improved linear algebra and Jacobian approximation methods can be applied to similar flame models in the Cantera[4] software package as they use the same matrix storage approach and finite-difference Jacobian approximation described here.

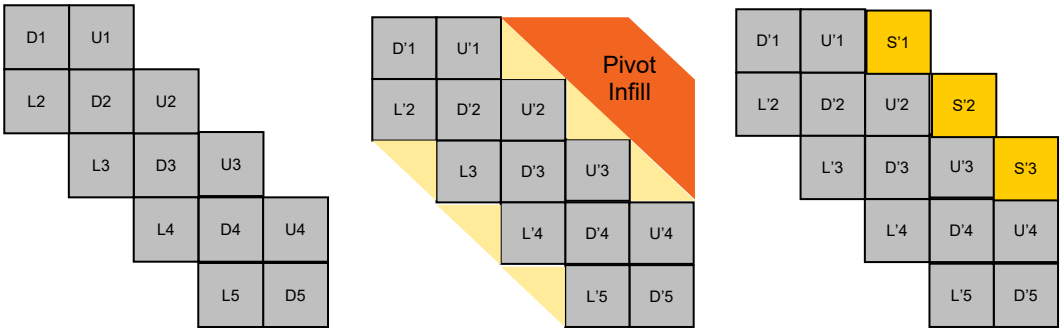
4.1 Parallelization

OPPDIF was previously modified[21] to support multicore parallel processing with OpenMP[22] to reduce the computational cost. Briefly summarizing the methods in Stone,[23] the residual function and the Jacobian assembly were refactored such that parallelism can be applied across independent mesh points and matrix columns, respectively. For the residual function, the convective and diffusive fluxes and thermochemical reaction rates and states (e.g., density) are evaluated across the N_g mesh points concurrently and in parallel. With the mesh tolerances and adaptivity discussed earlier, converged solutions commonly have 300–600 mesh points, which is sufficient to saturate most modern high-performance computing (HPC) systems with $O(10-100)$ cores. For the Jacobian assembly, the $3M_b$ independent columns are evaluated currently. Since M_b is typically $O(100-1{,}000)$ for chemical mechanisms of interest, ample parallelism exists in this operation as well. Note, when assembling the Jacobian in parallel, parallelism within the residual function itself is disabled. That is, nested parallelism was not exploited in Stone[23] and this approach is retained here. See Stone[23] for further details on these parallelization strategies and the necessary modifications to the CHEMKIN and TRANLIB libraries to support this.

An additional optimization has been implemented for the Jacobian assembly operation. A direct application of the finite difference approximation in Eq. 12 results in redundant computation of the nonspatial (i.e., inhomogeneous) terms in the residual function. These terms (e.g., $\dot{\omega}_k$ in the species and energy equations) only contribute to the diagonal block on each block row. To address this, the residual function was split into spatial and nonspatial contributions. The Jacobian is then assembled in two steps: First, M_b nonspatial residual functions are evaluated initializing the diagonal block matrices of the Jacobian. Then, the spatial residual terms are evaluated on the $3M_b$ matrix columns updating the diagonal blocks and setting the off-diagonal blocks in the Jacobian matrix. In both steps, multicore parallelism is used. If the chemical reaction rates dominate the residual function cost, an easily justifiable assumption when using mixture-averaged transport properties, splitting the Jacobian assembly in this manner should reduce the Jacobian assembly cost by nearly a factor of three.

4.2 Jacobian Factorization

As noted, the Jacobian matrix is block tridiagonal with dense blocks. Figure 1a shows an example of a block-tridiagonal matrix with equal sized blocks. The original OPPDIF implementation stored the sparse Jacobian matrix using a banded matrix format with upper and lower bandwidths equal to $2M_b - 1$. This format is the most appropriate format supported by the LAPACK[24] linear algebra library, which was used to directly factor and solve the resulting linear systems (i.e., Eq. 10) during a nonlinear iteration. However, storing a block-tridiagonal system with a band matrix format results in a 33% storage overhead. That is, the block-tridiagonal matrix in Fig. 1a requires $3M_b^2N_g$ storage but the band-matrix equivalent (shown in Fig. 1b) requires $4M_b^2N_g$. On each block row (with slight differences at the first and last two block rows), zero-filled upper and lower triangular matrices must be added to make the underlying system banded.



(a) Block-tridiagonal matrix (no infill)

(b) Banded matrix with padding (yellow) and in-fill from pivoting (orange)

(c) Block-tridiagonal matrix with pivot in-fill (yellow)

Fig. 1 Matrix sparsity with five mesh points. (a) Exact block-tridiagonal matrix without infill. (b) Factored banded matrix with padding (yellow) and pivot in-fill (orange). (c) Exact block-tridiagonal matrix with superblock pivot (yellow).

The storage overhead increases substantially when row pivoting is included in the factorization to increase the numerical stability, a necessity in this model. This makes the effective storage requirement for the banded matrix format $6M_b^2N_g$ for $M_b \gg 1$ as opposed to only $4M_b^2N_g$ to store the unfactored Jacobian. The extra overhead accounts for pivot rows coming from the zero-filled triangular matrices (i.e., the padded regions of the banded matrix). Note that the storage requirements for direct matrix factorization (banded or dense) with LAPACK make no regard for zeros in the system as is done by sparse matrix methods such as SuperLU.[25]

A new block-tridiagonal LUP factorization and linear system solver algorithm were formulated and implemented to reduce the storage requirement and improve

performance. This method is based on Intel Corporation's *Factoring General Block Tridiagonal Matrices*,[26] but extends that method such that a *strictly* block-tridiagonal structure is maintained, even with partial row pivoting, reducing the storage requirement to only $3M_b^2N_g$, half of that needed with the band matrix format. For clarity and comparison, the full algorithm from Intel's *Factoring General Block Tridiagonal Matrices*[26] is presented first.

In *Factoring General Block Tridiagonal Matrices*,[26] a composite matrix, D_i', is formed and factored sequentially and recursively for $\mathrm{i} = 1 \rightarrow N_g - 2$. The 2×3 rectangular block matrix is defined as

$$D_i' = \begin{bmatrix} D_i & U_i & 0 \\ L_{i+1} & D_{i+1} & U_{i+1} \end{bmatrix} \tag{13}$$

where U_i, L_i, and D_i are the upper, lower, and diagonal blocks on the i^{th} block row. Note that the last two block rows (i.e., $\mathrm{i} = (\mathrm{N_g} - 1) \rightarrow N_g$) are factorized separately as a single square system. The resulting factorization leads a super-diagonal block matrix at $(i, j+2)$ (i.e., to the right of U_i, replacing the null block) and accounts for pivot rows coming from the L_{i+1} lower block matrix. This method ensures that the resulting LUP factorization matches that generated by the equivalent dense (or banded) algorithm and as such ensures equivalent numerical stability. This factorization results in a storage of $4M_b^2N_g$, a 33% improvement over the band format with equivalent stability.

As an optimization, the pivot row search is constrained to only rows on the same block row (i.e., only rows in D_i). That is, potential pivot rows from L_{i+1} in Eq. 13 are not considered. This allows D_i to be factored directly, eliminating the need for the super-diagonal block. The resulting factorization remains strictly block tridiagonal and reduces the storage to $3M_b^2N_g$.

Restricting the pivot search can reduce the numerical stability of the factorization compared to Eq. 13. Buttari et al.[27] used a similar restricted pivot approach in their *tiled*, parallel LUP factorization of dense matrices and noted that the stability is reduced when small tiles are used. (Note, a tile in Buttari et al.[27] is equivalent to a full block in our method.) While no formal stability can be proven for this method, we note that the largest Jacobian terms below the diagonal, which would be selected as the pivots, tend to be associated with reaction rates and those terms are only within D_i. As such, while this method does produce different factorizations than with a complete pivot row search, no change in the convergence rate or the matrix condition number has been observed. It is also important to recall that the Jacobian is only used to drive the system toward convergence and is not part of the solution itself.

The estimated leading (serial) cost of the three Jacobian factorization methods is shown in Table 1. Only the highest power is retained to show scalability trends. See the Appendix for further details on the derivation of the cost estimates.

Table 1 Leading cost estimates for factorization and linear system solve for the banded and block-tridiagonal matrix methods

Method	Factorization	Linear system solves
Band matrix	$16N_gM_b^3$	$12N_gM_b^2$
Full block-tridiagonal matrix	$19/3N_gM_b^3$	$8N_gM_b^2$
Strictly block-tridiagonal matrix	$11/3N_gM_b^3$	$6N_gM_b^2$

The full block-tridiagonal LUP factorization following Eq. 13 has a leading time cost approximately 2.5 times faster than the original band matrix with in-fill and equivalent row pivoting. The strictly block-tridiagonal method reduces the leading cost constant by an additional 42%, or nearly 4.4 times faster than using a band matrix format during the Jacobian factorization.

The block-tridiagonal approach is also theoretically more efficient during the forward- and backward-substitution phases of the solution of the linear systems due to the lack of in-fill and the expanded LUP matrix system. The banded matrix linear system solver cost is approximately 50% more expensive than the full block-tridiagonal LUP solver and twice as expensive as the strictly block-tridiagonal variant. While solving the linear systems (with a previously factorized matrix) is significantly faster than the matrix factorization and scales quadratically, it still constitutes a substantial portion of the overall run-time since it is executed at least twice per nonlinear iteration (like the residual function itself) when searching for the acceptable damping coefficient via Eq. 11.

In practice, both block-tridiagonal methods are expected to outperform their theoretical advantage over their band matrix counterparts since they can exploit optimized dense matrix factorization and multiplication kernels provided by vendor math libraries (e.g., Intel's Math Kernel Library [MKL][28]). Further, band matrix methods are generally limited to level-1 Basic Linear Algebra Subprograms (BLAS) operations (i.e., vector–vector operations) while the dense matrix operations in LAPACK can exploit multithreaded dense matrix–vector and matrix–matrix operations (i.e., matrix–vector and matrix–matrix) that tend to be more cache efficient.

As noted, Jacobian-free (matrix-free) methods exist (e.g., inexact Newton–Krylov) and have been used for similar 1-D, steady flame models.[9] This avoids the explicit assembly and factorization of the Jacobian matrix and instead solves Eq. 10

iteratively with a Krylov method and approximates the Jacobian-vector product using a form like Eq. 12. Convergence of these methods depends strongly upon preconditioning, which itself may require the assembly of a Jacobian-like matrix. We have investigated using the KINSOL nonlinear solver library[29] within the OPPDIF framework. KINSOL provides both modified Newton as well as inexact Newton–Krylov methods. However, convergence challenges were encountered due to the treatment of constraint conditions. Solution constraints are necessary to prevent nonphysical values such as negative species concentrations. It was not possible to incorporate OPPDIF's constraint conditions within the limited constraint framework provided by KINSOL to attain efficient solutions. Furthermore, simple preconditioners (i.e., not application-specific) were not sufficient for the resulting linear system to converge. Both shortcomings require further research to resolve.

4.3 Condition Number Estimation

TWOPNT reports the condition number of the Jacobian whenever it is formed if a nonzero value is provided. The reported value is used only for diagnostic purposes and does not impact the solution accuracy. However, generating the estimate can be costly. OPPDIF estimates the condition number (or its reciprocal) using the LAPACK function `DGBCON`. According to Intel's LAPACK documentation,[28] the iterative method requires "usually 4 or 5 and never more than 11" solutions of the linear system in addition to evaluating the 1-norm of the original and factorized matrices. Since this is only used for diagnostics, an option has been added to OPPDIF to forego the estimation and not report the diagnostic. Hager's method[30] has also been implemented for when the estimate is desired. The new estimates are sufficiently close to those from DGBCON (e.g., within a factor of 10) for diagnostic purposes. This method reduces the number of linear solves generally needed to estimate the condition number reducing the computational cost. As the linear solver algorithms require n^2 floating-point operations, this savings can be significant as the Jacobian matrix size increases.

5. Results and Discussion

This section presents a set of benchmarks to demonstrate the performance improvements and scalability of the four proposed parallelization and factorization methods. Enumerated, these are as follows:

1) Hager's method for the condition number estimation.
2) Using a block-tridiagonal matrix storage format and factorization method.
3) Restricting row pivoting to rows within the same diagonal block.
4) And, splitting the residual function into spatial and nonspatial terms to avoid redundant computations when approximating the Jacobian with finite differences.

These four optimizations are applied to the same model and input conditions for repeatability. For this work, an opposed-flow diffusion flame stabilized above an HTPB fuel grain is modeled following the experimental setup of McDonald et al.[31] A single oxidizer flow rate (5.5 L/min) and a single composition (100% O_2) were selected from that study.

We used the full EODT as well as two smaller reduced versions to determine the performance impact and scalability of the optimization methods. The number of species and reactions for each mechanism is listed in Table 2. With the full mechanism, 343 mesh points were required to satisfy the mesh gradient and curvature requirements. Slightly fewer points where required for the reduced mechanisms. Adaptivity was disabled for this benchmark so that all tests used the same mesh size.

Table 2 EODT mechanism sizes and mesh resolution used in benchmarks

Mechanism	Species	Reactions	Points
Full	1,373	5,790	343
S813	813	2,800	323
S600	600	1,283	312

To mimic the behavior of a CYCLOPS calculation, the benchmark was restarted from a previously converged solution with a small perturbation applied to the fuel boundary velocity. OPPDIF was run for 100 pseudo-time steps and then the steady Newton solver was run to convergence. A new (unsteady) Jacobian is evaluated ever 10 pseudo-time steps and each step required only one iteration. Two nonlinear iterations were required to solve the steady solver to the tolerances. In total, the Jacobian was assembled and factorized 11 times; the linear system was solved 202 times; and the residual function was evaluated 314 times. Note that the number of

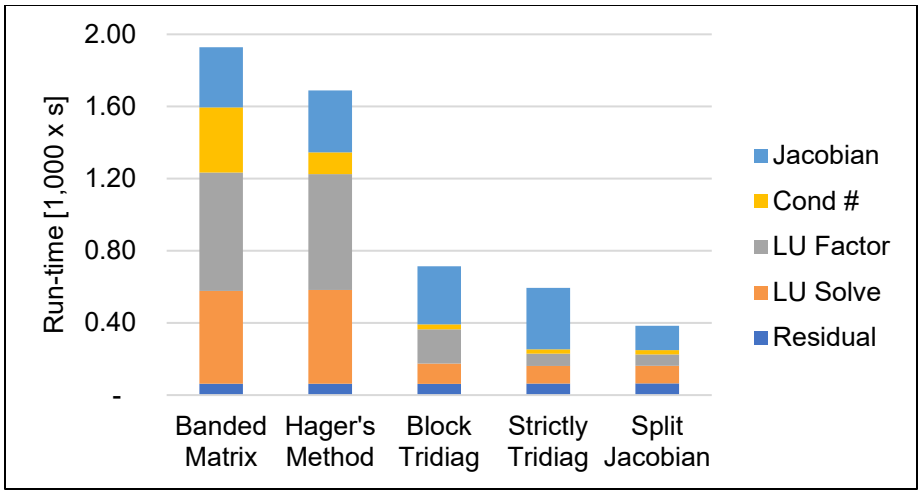
residual function evaluations does not include those needed to approximate the Jacobian matrix.

The benchmark was conducted on a single compute node on the Koehr system[32] at the US Navy Defense Shared Resource Center (DSRC). Each node has two Intel Xeon Skylake 8168 (platinum) processors with 24 cores and 170 GB of memory. Parallel benchmarks used up to 48 cores with OpenMP thread parallelism with one thread per code.

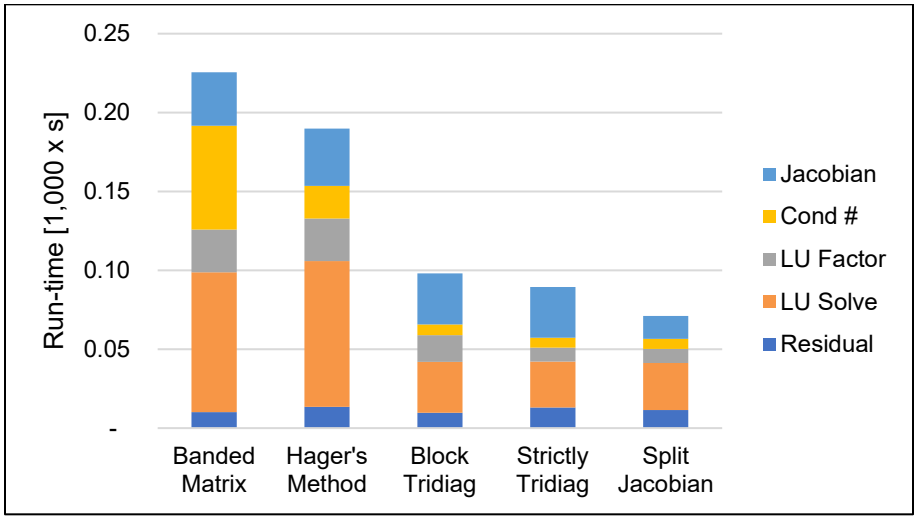
The OPPDIF code is compiled using the Intel Parallel Studio (XE) 2019 (version 19.0.4.243) compiler suite with default optimization (i.e., `-O2`). All LAPACK (and BLAS) matrix and vector operation function calls link with the external MKL implementations. The MKL release matched the compiler suite version.

Figure 2 shows the net run-time for the full and reduced (S600) versions of the EODT mechanism with the baseline, banded-matrix-based formulation (i.e., from Stone[22]) and the four optimizations, cumulatively applied, with 24 cores on the same CPU.[†] The major function run-times are also shown to highlight the individual contributions of each method. The very high cost of the Jacobian operations is evident; combined, the assembly, factorization, condition number, and linear solver operations account for over 95% of the run-time with the full and smaller S600 mechanisms. The condition number estimation is seen to take between 19% and 29% of the run-time with the proportional cost increasing as the mechanism size decreases. The linear solver also has the same quadratic cost scaling and follows the same proportional cost trend (i.e., 27%–39% from largest to smallest).

[†]Faster run-times can be attained using all 48 cores with both processors, but the run-to-run variability was higher than when isolating to within a single processor due to nonlocal memory accesses. A single processor was used to highlight the function cost improvements.



(a) Full



(b) S600

Fig. 2 Total run-time and function breakdown for OPPDIF simulation with the (a) full and (b) reduced (S600) EODT mechanisms using the baseline parallel algorithm and the four optimizations. Optimizations are applied cumulatively.

The new condition number estimation method is approximately 3 times faster than the baseline LAPACK version (i.e., between 3.0 and 3.3 times faster across the mechanism sizes). The new estimation method requires only 1–2 linear solver operations, several times fewer than the baseline version. As the linear solver operations are the dominant cost of this operation, this translates into significant run-time savings. The net savings for the full mechanism was 12.4% and 15.9% for the S600 mechanism.

The new block-tridiagonal matrix storage and factorization provided a much larger cost savings, the largest of all the optimizations. For the full mechanism, the cost was reduced by more than 50% over the baseline solver. The savings are seen both

in the factorization (3.4×) and linear solver phases (4.6×) compared to the baseline format and implementations. When limiting pivot rows to only the diagonal blocks (i.e., strictly block tridiagonal), the overall run-time is reduced by an additional 6.2%. The extra savings derive mostly from the factorization (2.8×) phase, but the linear solver is also improved by 15%.

With the smaller S600 mechanism, the overall savings with the strictly block-tridiagonal method is 44% over the baseline banded matrix. The factorization phase has a proportionally smaller cost than the linear solver phase with the small matrix, which may be explained by the cubic versus the quadratic cost scaling (i.e., the factorization savings with smaller matrix size scale cubically, too).

As anticipated, the realized gains for the block matrix factorization methods significantly exceeded their theoretical advantage shown in Table 1 due to improved parallelism over the band matrix methods. Analyzing the measured serial run-time cost (i.e., 1 thread) with the S600 mechanism showed that the strictly block-tridiagonal method was 82% faster than the full block-tridiagonal method and the full block-tridiagonal method was 67% faster than the banded matrix method. This is less than the theoretical scaling advantage. This discrepancy could be due to additional data movement overhead (i.e., copy in/out) required to temporarily form the intermediate matrix D_i' in Eq. 13.

The final optimization involves avoiding redundant computation when approximating the Jacobian with finite difference by splitting the assembly into two steps as detailed previously. This optimization reduces the run-time between 2.5 and 2.3 times for the largest and smallest mechanisms. As noted, this method is constrained to improving the Jacobian assembly cost by up to 3 times in the limit of overwhelmingly costly reaction rate terms compared to all other residual terms. Overall, this final optimization improved the net run-time by 11% and 8% for the large and small mechanism, respectively.

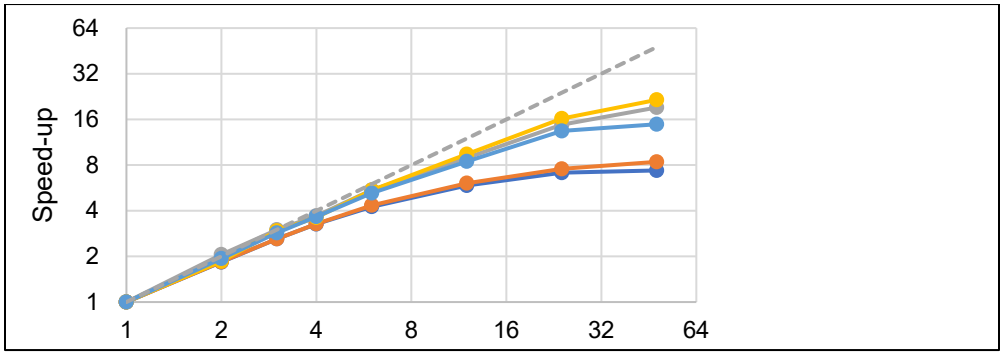
When combined, the four optimizations improve the run-time by 5.0 and 3.1 times on the large and small mechanism, respectively, using a single CPU with 24 cores. While not shown in Fig. 2, the net improvement increases to 3.8, 4.1, and 5.8 times when using the full compute node (i.e., both CPUs) with the small, medium, and large mechanisms, respectively. The higher impact on the larger mechanisms is due to the larger proportional cost of matrix factorization and linear system solver operations but also to improved parallel scalability of these methods.

The overall parallel run-time speedup (S_p) is shown in Fig. 3. S_p is defined as

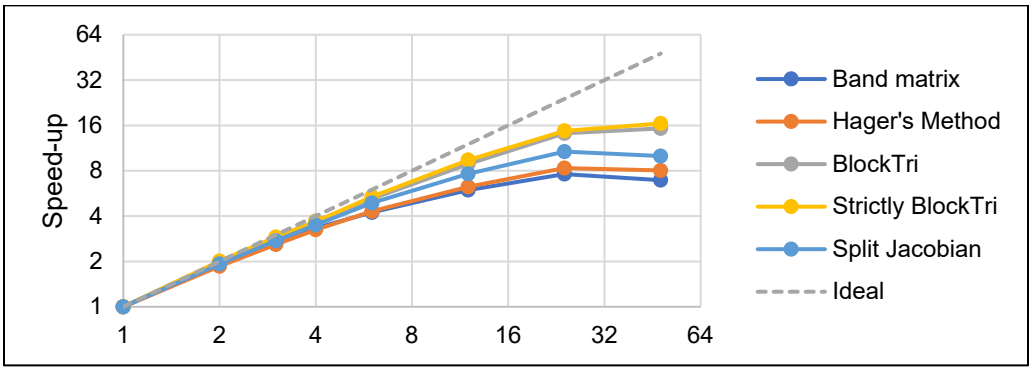
$$S_p = \frac{T_1}{T_p} \tag{14}$$

where T_1 is the run-time when using 1 core and T_p is the run-time with p cores. The ideal speedup (S_{ideal}) is shown for comparison where $S_{ideal} = p$. For all three mechanisms, the original banded-matrix method results in the lowest scalability achieving less than 16% efficiency where efficiency (E_p) is defined as

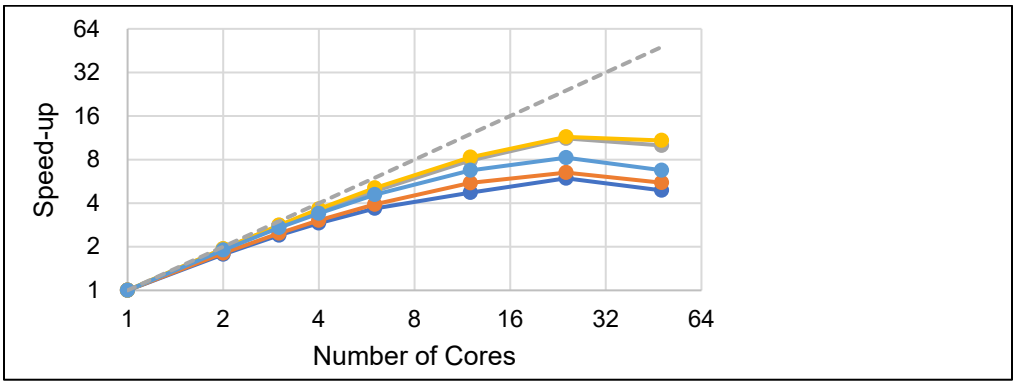
$$E_p = \frac{S_p}{p} \tag{15}$$



(a) Full



(b) S813



(c) S600

Fig. 3 Parallel speedup for OPPDIF simulation with the (a) full, (b) medium (S813), and (c) small (S600) mechanisms with the four optimizations

Ideal speedup (as defined) may be unattainable on modern CPUs due to dynamic frequency scaling (i.e., throttling) in which core frequencies are increased or decreased to manage temperatures while maximizing single-core performance. In this scenario, the core frequency when using only one core may be higher than when using many (or all) cores leading to a drop in *realized* parallel efficiency though this decrease may not be due entirely to implementation inefficiencies (i.e., Amdahl's law effects). This technology is available on the present system; however, the extent of the impact is difficult to discern.

As a test, the speedup of the Jacobian assembly with the full EODT mechanism was analyzed. This operation should exhibit the best scalability in the current algorithm due to its high computational cost, large loop operation count (i.e., well balanced across threads), and minimal thread synchronization overhead. $S_p > 45$ (i.e., $E_p > 94\%$) was measured with all 48 cores using the original assembly method.[‡] With only one CPU (i.e., 24 cores), the efficiency exceeded 96%. From this, it appears that frequency throttling does not have a significant impact on the present system and loss of scalability should be attributed to the application's (and the implementation's) parallel overhead.

Returning to Fig. 3, S_p using the original banded-matrix method (with and without the new condition number method) shows a significant drop compared to using the new block-tridiagonal method. The culprit for this poor scalability is the linear solver operation, which achieved no parallelism (i.e., $S_p \leq 1$) for all p. While accounting for only 4% of the net run-time in the serial case with the baseline implementation, it limits S_p to less than 25 times, per Amdahl's law. With zero scalability, its proportional cost increases to over 25% of the overall run-time. With the strictly block-tridiagonal method, the linear solver is 5% of the run-time serially but achieves up to 2.5-times speedup, which, while small, reduces the negative impact. The factorization scalability of the banded matrix method was less than 7 times on the full compute node and increased to over 12 times with the strictly block-tridiagonal method. From both the linear solver and factorization costs it is evident that the block-tridiagonal method provided the scalability improvements observed in Fig. 3 for the full mechanism.

As a final remark on the scalability, it must be noted that the split Jacobian method reduces the realized speedup, especially when using both CPUs. As noted earlier, this method improves the overall performance significantly; however, it reduces the

[‡]Note, the efficiency of the Jacobian assembly dropped to 87% when using the splitting approach due to an increase in thread synchronization (necessary to avoid race conditions between the two phases) and an overall reduction in computational cost. However, this method, as shown, was significantly faster overall.

cost of the most scalable operation (i.e., the Jacobian assembly)—thus, accentuating the remaining, and less scalable operations, particularly the linear solver operations, which become the dominant cost accounting for over 50% of the run-time for all three mechanisms. For the two small mechanisms, this, coupled with the very negative scalability of the linear solver method across multiple CPUs (i.e., spanning memory regions) leads to negative overall scalability and limits the OPPDIF application to a single CPU for efficient execution.

6. Summary and Conclusions

A new matrix factorization method was introduced for block-tridiagonal matrix systems. This was applied to the opposed-flow diffusion flame model, OPPDIF, using a finite-rate chemical kinetics mechanism to model the decomposition and combustion of EODT in pure oxygen at atmospheric pressure and temperature. A variant of the block-tridiagonal matrix method was introduced that restricted the pivot row search such that the resulting factorized matrix retained a block-tridiagonal structure. That is, it does not incur in-fill due to row pivoting. This variant reduced the overall memory storage requirement by 50% compared to the original method using a banded matrix format and was possible (and stable) since the dominant terms (attributed to reaction rates) tend to cluster in the diagonal block matrices.

When combined with a new estimation for the Jacobian-matrix condition number using Hager's method and a faster Jacobian assembly method that eliminated redundant computation, the net performance of the flame solver was improved (i.e., the run-time reduced) by a factor of 5 for the largest (full) mechanism with 1,373 species and 3.15 with the smallest mechanism with only 600 species.

The parallel scalability was also improved by over a factor of two with the largest mechanism leading to more effective use of HPC resources in addition to an overall reduction in run-time. It was shown that a key factor in improving the scalability was achieving positive scaling with the linear system solver using the block tridiagonal and strictly block-tridiagonal variant.

While the new factorization methods were demonstrated in the OPPDIF flame model, block-tridiagonal matrices occur frequently in combustion models (e.g., burner stabilized premixed flames) and more broadly in computational models of 1-D, multivariable physical systems. The strictly block-tridiagonal method could be directly implemented in similar 1-D combustion models in the Cantera library.[4] Cantera does not support multithreaded parallelism as used here; however, the performance could be improved by accelerating the matrix factorization and linear solver operations alone as those are external to Cantera itself (e.g., a library function

call as done here). The factorization and linear solver functional costs accounted for 29% of the baseline serial run-time using the full EODT mechanism in OPPDIF. Assuming the major operations have comparable costs, replacing only the Jacobian factorization and linear system solver operations with the parallel to strictly block-tridiagonal implementation would reduce the overall run-time by 40% using 24 cores. Additionally, this would reduce the memory usage by 50% allowing larger models to be simulated.

7. References

1. McQuaid MJ. An opposed-flow diffusion flame simulation-based implementation of the trial mechanism method for chemical kinetics mechanism reduction: application to the San Diego mechanism for HTPB-air combustion modeling. Army Research Laboratory (US); 2020 Aug. Report No.: ARL-TR-9032.

2. Chen C-C, McQuaid M. A skeletal finite-rate chemical kinetics mechanism for modeling HTPB-air combustion in a gun-launched solid-fuel ramjet combustor. Army Research Laboratory (US); 2020. Report No.: ARL-TR-8891.

3. Lutz AE, Kee RJ, Grcar JF, Rupley FM. OPPDIF: a Fortran program for computing opposed-flow diffusion flames. Sandia National Laboratories (US); 1997 May. Report No.: SAND96-8243.

4. Goodwin D, Moffat H, Schoegle I, Speth R, Weber B. Cantera: an object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes. Ver. 3.0. Cantera; 2023 [accessed 2024 Apr 9]. https://www.cantera.org

5. Smith G, Golden D, Frenklach M, Moriarty NEB, Goldenberg M, Bowman C, Hanson R, Song S, Gardiner W, Lissaianski V, Qin Z. Ver. 3.0. GRI-Mech; c n.d. [accessed 2023 Mar 28]. http://www.me.berkeley.edu/gri_mech/

6. Chen C-C, McQuaid M. Thermochemical and kinetics modeling pertaining to AP-HTPB composite propellant combustion. In: 39th JANNAF Propellant and Explosives Development and Characterization Meeting; 2015; Salt Lake City, UT.

7. Miller M, Anderson W. Burning-rate predictor for multi-ingredient propellants: nitrate-ester propellants. J Propul Power. 2004;20(3):440–454.

8. Geipel C, Bojko B, Pfutzner C, Fisher B, Johnson R. Regression of solid polymer fuel strands in opposed-flow combustion with gaseous oxidizer. P Combust Inst. 2023;39:3389–3399.

9. Lapointe S, Whitesides R, McNenly M. Sparse, iterative simulation methods for one-dimensional laminar flames. Combust Flame. 2019;204:23–32.

10. Kee J, Rupley F, Meeks E, Miller JA. CHEMKIN-III: a Fortran chemical kinetics package for the analysis of gas-phase chemical and plasma kinetics. Sandia National Laboratories (US): 1996 May. Report No.: SAND96-8216.

11. Kee RJ, Dixon-Lewis G, Warnatz J, Coltrin ME, Miller JA, Moffat HK. A Fortran computer code package for the evaluation of gas-phase, multicomponent transport properties. Sandia National Laboratories (US); 1998 Mar. Report No.: Supercedes SAND86-8246B.

12. Tsang W, Herron J. Chemical kinetic data base for propellant combustion I. Reactions involving NO, NO_2, HNO, HNO_2, HCN and N_2O. J Phys Chem Ref Data. 1991;20(4):609–663.

13. Joback K, Reid R. Estimation of pure-component properties from group-contributions. Chem Eng Commun. 1987;57:233–243.

14. Gao C, Allen J, Green W, West R. Reaction mechanism generator: automatic construction of chemical kinetic mechanisms. Comput Phys Commun. 2016;203:212–225.

15. Liu ML, Grinberg Dana A, Johnson M, Goldman M, Jocher A, Payne A, Grambow C, Han K, Yee N, Mazeau E, Blondal K, West R, Goldsmith C, Green W. Reaction mechanism generator v3.0: advances in automatic mechanism generation. J Chem Inf Model. 2021;61(6):2686–2696.

16. Johnson M, Dong X, Grinberg Dana A, Chung Y, Frina D, Gillis R, Liu M, Yee N, Blondal K, Mazeau E, Grambow C, Payne A, Spiekermann K, Pang H-W, Goldsmith C, West R, Green W. RMG database for chemical property prediction. J Chem Inform Model. 2022;62:4906–4915.

17. Simplified Molecular Input Line Entry System (SMILES). SMILES tutorial. Environmental Protection Agency (US); c1987–1989 [accessed 2023 Apr 10]. https://archive.epa.gov/med/med_archive_03/web/html/smiles.html.

18. Miller M, Anderson W. Energetic-material combustion modeling with elementary gas-phase reactions: a practical approach. Prog Astronaut Aero. 2000;185:501–531.

19. Chiaverini M, Hartin G, Lu Y-C, Kuo K, Peretz A. Pyrolysis behavior of hybrid-rocket solid fuels under rapid heating conditions. J Propul Power. 1999 Nov;15(6).

20. Chen C-C, McQuaid MJ. A detailed, finite-rate chemical kinetic mechanism for modeling the thermal decomposition and combustion of gaseous nitroglycerin. DEVCOM Army Research Laboratory (US); 2022 July. Accession No.: AD1179961.

21. Grcar J. The TWOPNT program for boundary value problems. Sandia National Laboratories (US); 1996. Report No: SAND91-8231.

22. OpenMP. Ver. 5.1. OpenMP; c 1997–2020 [2024 Mar 25]. https://www.openmp.org/.

23. Stone C. Summary of user productivity enhancement, technology transfer, and training (PETTT) refactoring assistance for chemical kinetics analysis tools at CCDC Army Research Laboratory. Army Research Laboratory (US); 2020 Jan. Report No.: ARL-CR-0843. Accession No.: AD1090612.

24. Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D. LAPACK users' guide, 3rd ed. Society for Industrial and Applied Mathematics; 1999.

25. Li X, Demmel JW, Gilbert JR, Grigori L, Sao P, Shao M, Yamazaki I. SuperLU users'guide. Department of Energy (US); 1999 Sep.

26. Intel Corporation. Factoring general block tridiagonal matrices. Intel Corporation; 2021. Intel oneAPI Math Kernel Library Cookbook.

27. Buttari A, Langou J, Kurzak J, Dongarra J. A class of parallel tiled linear algebra algorithms for multicore architectures. Parallel Comput. 2009;35(1):38–53.

28. Intel Corporation. Developer Reference for Intel oneAPI Math Kernel Library for C. [accessed 2024 Mar 22]. https://www.intel.com/content/www/us/en/docs/onemkl/developer-reference-c/2023-0/gecon.html.

29. Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS. SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. ACM T Math Software. 2005;31(3):363–396.

30. Higham N. FORTRAN codes for estimating the 1-norm of a real or complex matrix, with applications to condition estimation. ACM T Math Software. 1988;14(4):381–396.

31. McDonald A, Baier M, Son S, Mcquaid M, Chen C-C, Veals J, Stone C, Goldnestein C. Laser absorption measurements of temperature and CO profiles in opposed-flow diffusion flames of solid propellants. In: 13th US National Combustion Meeting; 2023; College Station, TX.

32. DOD High Performance Computing Modernization Program. Department of Defense (US); c2022 [accessed 2023 Apr 10]. https://centers.hpc.mil/systems/unclassified.html#Koehr

Appendix. Complexity Analysis

The computational cost of the band matrix and the two block-tridiagonal matrix factorization methods are approximated here. The cost is approximated as the number of floating-point operations with all operations counted equally; that is, addition, subtraction, multiplication, and division operations are given the same cost factor of 1.

All three factorization and linear solver implementations rely on LAPACK subroutines internally to perform dense matrix operations. Table A-1 lists the cost estimates for these core functions based on reference implementations of LAPACK kernels from Netlib.[§]

Table A-1 Cost estimates for LAPACK matrix operations used by band and block matrix factorization and linear solve methods

Function	Description	Cost
$DGETRF(M,N)$	Lower–upper factorize dense matrix with M rows and N columns with $M \leq N$	$[M^2N - 1/3M^3 - MN + 1/3M] + 1/2[M^2 - M]$
$DGEMM(M,N,K)$	Multiple two dense matrices of sizes $M\ x\ K$ and $K\ x\ N$	$N[M + 2KM]$
$DTRSM(M,N)$	Solve upper or lower triangular system of a dense matrix with M rows and N columns	$1/2MN[M + 1]$
$DTBSV(M,K)$	Solve upper or lower triangular system of a band matrix with M rows and bandwidth K	$2M[K - 1]$
$DGER(M,N)$	Perform rank-1 update $A = A + \alpha xy^T$ on matrix with M rows and N columns and vectors with length $M\ x\ 1$ and $N\ x\ 1$	$N[1 + 2M]$
$SCAL(M)$	Scale a vector of length M	M

Band matrix factorization uses the LAPACK function DGBTRF to factor a matrix with M rows, N columns, and lower and upper bandwidths KL and KU, respectively. For OPPDIF, the Jacobian matrix is square with $M = N = N_g M_b$ and $KL = KU = K = 2M_b - 1$. Since $N_g \gg 1$, we can ignore complicated analysis near the top and bottom of the matrix. That is, we shall not account for nonuniform costs for rows $j <$ K and $j > M -$ K (where j is the row index) and instead treat all rows uniformly. With this simplification, the factorization cost *per row* is

$$SCAL(K) + DGER(K, 2K) = K + 2K[1 + 2K] = 3K + 4K^2\ . \qquad \text{(A-1)}$$

Expanding with the Eq. A-1 definitions leads to the following cost estimate for band matrix factorization:

[§]Netlib LAPACK. [accessed 2024 Mar 25]. https://netlib.org/lapack/

$$N_g[16M_b^3 - 10M_b^2 + M_b] \approx 16N_g M_b^3 . \quad \text{(A-2)}$$

The band matrix linear system solve cost is approximately $12N_g M_b^2$ of which $2/3$ of the cost is in the backward substitution phase.

We use the same simplification for the block matrix methods as for the (scalar) band matrix and ignore the special costs on the first and last few block rows. Thus, the full block-tridiagonal matrix factorization cost per *block* row, with block size K, is

$$\begin{aligned} &DGETRF(2K,K) + DTRSM(K,2K) + DGEMM(K,2K,K) \\ &\quad = 19/3K^3 + 3K^2 - 1/2K . \end{aligned} \quad \text{(A-3)}$$

Expanding, this leads to the overall leading cost of

$$N_g[19/3M_b^3 + 3M_b^2 - 1/2M_b] \approx 19/3N_g M_b^3 . \quad \text{(A-4)}$$

Similarly, the strictly block-tridiagonal factorization method has a cost per block row of

$$\begin{aligned} &\mathrm{DGETRF}(K,K) + 2\ \mathrm{DTRSM}(K,K) + \mathrm{DGEMM}(K,K,K) \\ &\quad = 11/3K^3 + 3/2K^2 - 1/6\mathrm{K} . \end{aligned} \quad \text{(A-5)}$$

Expanding, this leads to the overall leading cost of

$$N_g[11/3M_b^3 + 3/2M_b^2 - 1/6M_b] \approx 11/3N_g M_b^3 . \quad \text{(A-6)}$$

The linear system solve for the three methods is found in similar fashion. For a band matrix with M rows, and bandwidth K, the cost of the DGBTRS function is approximately:

$$\begin{aligned} &[M]DGER(K,1) + DTBSV(M,2K) = M[1+2K] + 2M[2K-1] \\ &\quad = 6MK - M . \end{aligned} \quad \text{(A-7)}$$

Expanding, with $M = N_g M_b$ and $K = 2M_b - 1$ leads to the overall leading cost of

$$6N_g\mathrm{M_b}[2\mathrm{M_b} - 1] - \mathrm{N_g M_b} = 12N_g M_b^2 - 7N_g M_b \approx 12N_g M_b^2 . \quad \text{(A-8)}$$

The full block-tridiagonal linear system solve cost per block row is approximately

$$\begin{aligned} &[2]DTRSM(K,1) + [3]DGEMM(K,1,K) = 2[K^2 + K] + 3[K + 2K^2] \\ &\quad = 8K^2 + 5K . \end{aligned} \quad \text{(A-9)}$$

Expanding, with N_g block rows and $K = M_b$ leads to the overall leading cost of

$$N_g\left[8{M_b}^2 + 5M_b\right] = 8N_g M_b^2 - 5N_g M_b \approx 8N_g M_b^2 . \quad \text{(A-10)}$$

Finally, the strictly block-tridiagonal linear system solve cost per block row is approximately

$$[2]DTRSM(K,1) + [2]DGEMM(K,1,K) = 2[K^2 + K] + 2[K + 2K^2] \\ = 6K^2 + 4K\ , \tag{A-11}$$

which leads to an overall leading cost of

$$N_g\left[6{M_b}^2 + 4M_b\right] = 6N_g M_b^2 - 4N_g M_b \approx 6N_g M_b^2\ . \tag{A-12}$$

List of Symbols, Abbreviations, and Acronyms

1-/2-/3-D	one-/two-/three-dimensional
ARL	Army Research Laboratory
BLAS	Basic Linear Algebra Subprograms
CPU	computer processing unit
DEVCOM	US Army Combat Capabilities Development Command
DOD	Department of Defense
DSRC	Defense Shared Resource Center
EODT	*6-ethenyl-2,8,12,16-octadecetetraene*
GB	gigabyte
HPC	high-performance computing
HTPB	*hydroxyl-terminated polybutadiene*
JFNK	Jacobian-free Newton–Krylov (method)
LUP	lower–upper matrix factorization with row pivoting
MKL	Math Kernel Library
NP	nascent product
OPDF	opposed-flow diffusion flame
SMILES	Simplified Molecular Input Line Entry System
TMM	trial mechanism method