COS597E Project Final Report: NGA_LeWI
Dynamic Load Balancing in a Massively Parallel Reacting Flow Solver

Michael D. Walker *
Princeton University

Abstract

Chemically-reacting flows, like combustion, are highly nonlinear. Reaction kinetics follow an exponential Arrhenius law. Such a flow is simulated by applying the system of conservative partial differential equations over a domain, often with a finite-difference discretization and parallelized across many processing units. Using the operator-splitting technique commonly applied in reactive flow solvers, chemistry can be treated as an independent stiff ordinary differential equation (ODE) system within each computational cell, but large variation in the convergence rates of the ODE integrator may occur. Calculation of the chemical source terms in the system of equations, therefore, can lead to severe load imbalance across a distributed computing architecture. The independent nature of the chemistry ODE system leads to a problem that can be parallelized and balanced during the flow solution without the introduction of significant error. This work implements Lend When Idle (LeWI), a program that monitors and dynamically balances the load distribution across cores, on an existing massively parallel reacting flow solver, NGA, providing significant speed-up in clock time. Additionally, a reference mapping scheme is developed which applies reference variables, like mixture fraction Z_i and activation temperature T_i, to predict areas of low chemical reaction activity (and thus low computational load). Rather than evaluating the full chemical Jacobian in these regions, a reference cell solution is simply copied, thereby generating further speed-up. Performance is evaluated by comparing to simulations of hydrogen-air and methane-air jet flames, whose unbalanced results were published previously [13, 14]. A modest improvement is shown, especially for early simulation timesteps, where computational load is most difficult to predict.

[1] netID: mw6136
Email: michael.walker@princeton.edu
Computational Turbulent Reacting Flow Laboratory (CTRFL)
Department of Mechanical and Aerospace Engineering
Princeton University, Princeton, NJ 08544
Submitted: December 11, 2024 for fulfillment of
COS597E: Programming Languages, Distributed Systems

1 Introduction and Motivation

Modern Computational Fluid Dynamics (CFD) software relies on distributed-memory parallel computer architectures for large-scale computations. Using resources efficiently while at the same time maintaining high accuracy of the numerical solution is a major challenge. *Computational load imbalance* is a well-known performance issue in multiprocessor reacting flow (e.g., combustion) simulations utilizing directly integrated chemical kinetics. As the need for more detailed combustion models has become prominent, chemical kinetics models have grown in size and complexity, resulting in higher computational cost and often exceeding the computational cost of fluid dynamics by a factor of 100 [15].

For each chemical reaction j with species $\mathcal{A}_k$ (such as oxidizer and fuel) and stoichiometric coefficients ν_k:

$$\nu'_{k,j}\mathcal{A}_k \leftrightarrow \nu''_{k,j}\mathcal{A}_k \quad ,$$

where f and b scripts indicate the forward and backward modes of an equilibrium reaction. The governing reaction rate $\dot{\omega}_j$ is defined by

$$\dot{\omega}_j = k_{f,j}\prod^{k}\left(\frac{\rho Y_k}{W_k}\right)^{\nu'_{k,j}} - k_{b,j}\prod^{k}\left(\frac{\rho Y_k}{W_k}\right)^{\nu''_{k,j}} \quad .$$

W_k is the molecular weight of species k, Y_k is the species mass fraction, and ρ the gas density. The reaction rate coefficient is dependent on temperature according to the modified Arrhenius form

$$k_{f,j} = A_j T^{n_j}\exp\left(-\frac{E_{a,j}}{\overline{R}T}\right)$$

where A_j is an experimentally derived pre-exponential factor, $\overline{R}$ the ideal gas constant, n_j is the temperature exponent corresponding to the reaction order, and E_a is the activation energy. $E_a/\overline{R}$ is considered the activation temperature, the temperature at which the exponential function becomes dominant. Forward rate coefficients k_f are tabulated and parameterized in the modified Arrhenius form. Backward rate coefficients

are determined from equilibrium constant $k_{b,j} = k_{f,j}/K_{p,j}$. The chemical source term is a forcing function in the conservation equations for species k, for each reaction j

$$\dot{m}_k = \sum_j (\nu^f_{k,j} - \nu^b_{k,j}) W_k \dot{\omega}_j \quad .$$

The reaction rate is evaluated at the temperature T_b of the reaction zone $\omega^o_b = \omega_b(T^o_b)$.

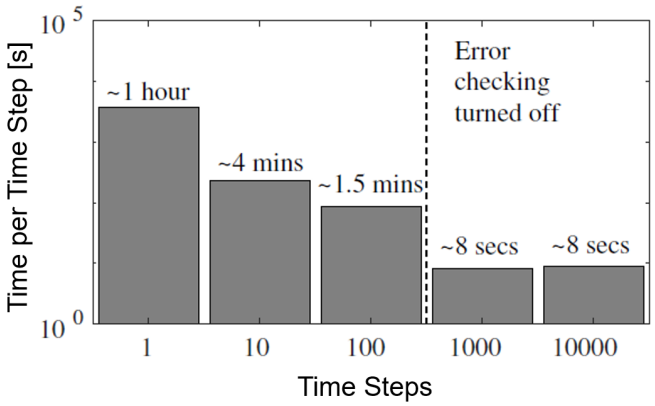

Figure 1: Typical evaluation time (wall clock time) for reacting flow Large Eddy Simulations. The semi-log plot illustrates an exponential decay in time per timestep, but with severe cost in initial timesteps due to chemical Jacobian construction, evaluation, and inversion.

Typically, the computational cost of the chemical source term evaluation dominates the performance metrics and creates an uneven computational load distribution in parallel applications. The difficulties occur due to the intrinsic and non-linear nature of the ODE system. Figure 1 shows typical evaluation time for a laboratory scale jet flame [13]. The computational cost of solving the associated stiff ODEs scales quadratically with the number of species. Furthermore, due to the vast scale separation between the fastest and slowest chemical reaction time scales, the system of ODEs is practically always numerically stiff, requiring the use of implicit time integrators with low timestep values. **This project implements a *dynamic load balancing* scheme (specifically Lend When Idle) on the low Mach number flow solver, `NGA` [7, 17].** Additionally this project develops a *reference mapping* paradigm to sort group cells with similar thermochemical composition, thereby evaluating their chemical source terms only when sufficiently dissimilar. The structure of the paper is as follows:

1. The low Mach flow solver `NGA`, its associated dependencies, and its current scheme for approximating and distributing load is introduced. Additional background is provided on numerical approaches for solution of multi-dimensional reacting flows, especially chemical source term evaluation.
2. The Lend When Idle (LeWI) scheme, an existing runtime-based, task-neutral dynamic load balancing code that distribute resources through the `MPI` protocol, is introduced and implemented with `NGA`.
3. A *reference mapping scheme* based on combustion theory and multi-zone reduction models is developed and implemented.
4. The balanced `NGA` code is evaluated using two canonical turbulent jet flames. An overview of load balancing in state-of-the-art reacting flow solvers is given and directions for further future development is discussed.

2 Background: Existing Numerical Scheme

2.1 Conservation Equations and Fluid Solver

Advanced schemes solve the coupled Navier-Stokes system using a second order MUSCL-Hancock scheme (Monotonic Upstream-Centered Scheme for Conservation Laws). This scheme was designed for solving any system of non-linear hyperbolic conservation laws. The method is employed for discretizing the governing equations, starting with a control volume Φ and flux F:

$$F(\Phi) \rightarrow \underbrace{\boxed{\Phi}}_{\Delta x} \rightarrow F(\Phi + \Delta\Phi)$$

The conservation equations for three-dimensional, multi-component, reactive compressible flows can be written in index notation as

$$\frac{\partial \Phi}{\partial t} + \frac{\partial F(\Phi)}{\partial x_j} + N\frac{G(\Phi)}{x_j} = \frac{\partial F_\nu(\Phi)}{\partial x_j} + S(\Phi)$$

F_ν is the diffusion flux, S is the reaction term, and G is the geometric shape factor [6]. The vectors Φ, $F(\Phi)$, $G(\Phi)$, $F_\nu(\Phi)$, and $S(\Phi)$ are defined as

$$\Phi = \begin{bmatrix} \rho Y_1 \\ \rho Y_2 \\ \vdots \\ \rho Y_k \\ \rho u \\ E \end{bmatrix} \quad \mathbf{F}(\Phi) = \begin{bmatrix} \rho u Y_1 \\ \rho u Y_2 \\ \vdots \\ \rho u Y_k \\ \rho u^2 + P \\ (E+P)u \end{bmatrix} \quad \mathbf{G}(\Phi) = \begin{bmatrix} \rho u Y_1 \\ \rho u Y_2 \\ \vdots \\ \rho u Y_k \\ \rho u^2 \\ (E+P)u \end{bmatrix}$$

$$\mathbf{F}_\nu(\Phi) = \left[\text{viscous terms}\right] \quad \mathbf{S}(\Phi) = \left[\dot{m}_1, \dot{m}_2, \ldots, \dot{m}_k, 0, 0\right]$$

where N is the geometry factor ($N = 0$, rectangular coordinate; $N = 1$, cylindrical coordinate; $N = 2$ spherical coordinate). u is the flow velocity and $x(= r)$ is the spatial coordinate. ρ represents the fluid density, Y_k is the mass fraction of species k, E is the total energy, and P is the pressure of the fluid. Energy in the low Mach number limit is $E = \rho e + \cancel{\rho u^2/2} = \rho e$. Along with the ideal gas equation of state, these equations represent a closed system.

2.2 Operator Splitting Technique

Commonly, engineering flow solvers assume an operator-splitting strategy in the reacting flow implementation, enabling the decoupling of the chemistry and fluid dynamics in the solution during the calculation of the chemical source terms. Therefore, the changes in thermochemical composition Φ can be described as a stiff system of ordinary differential equations (ODEs) $\partial\Phi/\partial t = F(\Phi,t)$. A linear or nonlinear system can be defined in matrix form as $\Phi \equiv \phi_{i,j}$ and $\mathbf{f}(\Phi) \equiv f_{i,j}(\Phi)$ for $i = 1,\ldots,N$, $j = 1,\ldots,N$. Parallelization is achieved with geometrical domain decomposition. The Strang splitting scheme is achieved with the spatially discretized governing equations:

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \underbrace{\mathbf{T}(\Phi)}_{\text{transport}} + \underbrace{\mathbf{S}(\Phi)}_{\text{chemistry}}$$

The two steps are denoted by operator $\mathbf{T}^{(t)}$ (for transport) and operator $\mathbf{S}^{(t)}$ (chemical source), respectively. Based on the above splitting, the solution can be evolved from its initial value Φ^n at time t^n, by one timestep of size Δt, to a value Φ^{n+1} at time $t^{n+1} = t^n + \Delta t$,

$$\Phi^{n+1} = \mathbf{S}^{\Delta t}\,\mathbf{T}^{\Delta t}(\Phi^n)$$

The above procedure for solving the inhomogeneous system is only first-order accuracy in time. A scheme with second-order accuracy in time called Strang splitting is

$$\Phi^{n+1} = \mathbf{S}^{\Delta t/2}\,\mathbf{T}^{\Delta t}\,\mathbf{S}^{\Delta t/2}(\Phi^n)$$

where $\mathbf{S}$ and $\mathbf{T}$ are at least second-order accurate solution operators in time. The time derivative is approximated by finite-difference involving the current and future values. The gradient and divergence operators are approximated with linear functions involving multiple neighboring cells. Transport is relatively linear and has a highly structured Jacobian, with weak coupling between species. Chemistry is highly nonlinear and typically solved implicitly. Chemistry terms are local, so each mesh volume can be solved independently (i.e., in parallel). The chemistry and transport substeps are then:

$$\begin{aligned}
&\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \mathbf{S}(\Phi^{(1)}), && \Phi^{(1)}(x,0) = \Phi(x,t_n) && \text{on } [t_n, t_n + \Delta t/2]\\
&\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \mathbf{T}(\Phi^{(2)}), && \Phi^{(2)}(x,0) = \Phi^{(1)}(x,\Delta t/2) && \text{on } [t_n, t_n + \Delta t]\\
&\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \mathbf{S}(\Phi^{(3)}), && \Phi^{(3)}(x,0) = \Phi^{(2)}(x,t_n) && \text{on } [t_n + \Delta t/2, t_n + \Delta t]
\end{aligned}$$

Considering the time integration of a stiff reacting system, each implicit integration step involves evaluation of the source terms by Jacobian evaluation and factorization in a Newton-Raphson scheme. A system of ODEs in chemistry is defined based on multiple chemical species N_k (i.e., types of molecules) evolving through coupled reactions $d\Phi/dt = \mathbf{S}\Phi$, where $\Phi = [\phi_1, \phi_2, \ldots \phi_N] \in \mathbb{R}^N$ and $\mathbf{S} \in \mathbb{R}^{N\times N}$. Any time integration scheme requires the eigenvalues of $\mathbf{S}$ to lie inside the stability region of the method. The system is stiff if for the set of the eigenvalues $[\lambda_1, \lambda_2, \ldots \lambda_N]$ of matrix $\mathbf{S}$: $|\lambda_{\max}|/|\lambda_{\min}| \gg 1$. A stiff problem requires a very small timestep for stability with respect to $|\lambda_{\max}|$ in conjunction with a very long time integration to see any significant change in magnitude associated to $|\lambda_{\min}|$.

The system $\mathbf{S}(\Phi) = 0$ is solved first by constructing the Jacobian in 2-D $\mathbb{J} \equiv \partial S_{i,j}/\partial\phi_{i,j}$. Then the Newton-Raphson scheme computes the components of the matrix. Each iteration is defined by the linear system $\mathbb{J}(\Phi^{n+1} - \Phi^n) = -\mathbf{S}(\Phi^n)$ and can then be solved through direct matrix inversion sequentially,

$$\Phi^{n+1} = \Phi^n - [\mathbb{J}^{-1} \cdot \mathbf{S}(\Phi^n)] \quad .$$

Here the Jacobian is an $N^2 \times N^2$ matrix and thus **Jacobian evaluation and factorization is the most expensive step in combustion simulations**. To solve the conservative system, the stiff source term $\mathbf{S}$ is treated by the fractional-step procedure. In the first fractional step, the non-reactive flow is solved using the MUSCL-Hancock scheme from Toro [23].

$$\left.\begin{aligned}
&\text{PDE}: \frac{\partial\Phi}{\partial t} + \frac{\partial F(\Phi)}{\partial x_j} + N\frac{G(\Phi)}{x_j} = \frac{\partial F_{\mathrm{v}}(\Phi)}{\partial x_j}\\
&\text{IC}: \Phi(x,t^n) = \Phi^n
\end{aligned}\right\} \rightarrow \overline{\Phi}^{n+1}$$

MUSCL-Hancock scheme To numerically solve the system of equations, a second-order MUSCL-Hancock scheme (Monotonic Upstream-Centered Scheme for Conservation Laws) is used. The steps of this solver include:

1. *Data Reconstruction*: Cell-centered values are extrapolated into slopes between grid cells so that the fluid is represented by a piecewise-linear plane in each of the cells. The *boundary extrapolated values* ($\Phi_i^L, \Phi_i^R, \Phi_j^L, \Phi_j^R$) are then calculated.

2. *Evolution*: Each of the four boundary extrapolated values move forward by a half timestep.

3. *Riemann Initial Value Problem*: A HLLC (Harten-Lax-van Leer-Contact) Riemann solver was implemented to calculate the fluxes in the i and j direction at the cell faces [23]. It determines the states of nearby cells and then calculates the variable fluxes across those cells using conservation laws.

4. *Conservative Update*: The conserved variables are updated. The MUSCL-Hancock scheme is repeated for each consecutive timestep.

2.3 Chemical Source Term evaluation

With this technique, chemistry can be treated as an independent stiff ODE system within each computational cell. The

chemistry is solved in the second fractional step for a homogeneous system

$$\left.\begin{array}{l} \text{ODE}: \dfrac{d\Phi}{dt} = \mathbf{S}(\Phi) \\ \text{IC}: \overline{\Phi}^{n+1} \end{array}\right\} \rightarrow \Phi^{n+1}$$

For the **S** operator, the mass fraction of species are updated by using the CVODE solver [12]. Note that the density ρ, momentum ρu, and total energy E remain constant during updating the mass fraction of all the species, after which the temperature T is solved according to the definition of total energy:

$$F(T) = R_0 T/\overline{M} - u^2/2 - h + E/\rho = 0$$

where the equation of state $\rho = \sum \rho_i = P\overline{M}/R_0 T$ is utilized to eliminate the pressure term. The above equation can be solved numerically by using the Newton iteration method and the updated temperature at each iteration step is

$$T^{\text{new}} = T^{\text{old}} - F(T^{\text{old}})/(R_0/\overline{M} - c_P)$$

2.4 Load Balancing and NGA

NGA [7, 17] is a 3-D finite-difference energy-conserving low-Mach number flow solver, using derivative operators of arbitrarily high order. It is capable of solving the variable-density Navier-Stokes equations on structured meshes using a fractional-step method. Spatial discretization errors are reduced using local $(n-1)$-th order Lagrange polynomial interpolation to achieve n-order accurate viscous terms. The code utilizes detailed temperature- and composition-dependent transport and thermodynamic properties, detailed chemical kinetics, and an ideal gas equation of state. NGA was written for massively parallel large-eddy simulation (LES) and direct numerical simulation (DNS) of both premixed and non-premixed reacting turbulent flows. It is parallelized using a hybrid MPI/OpenMP approach and is highly scalable. NGA uses an operator splitting technique to decouple the conservation equations and chemical source term calculations.

NGA then uses SUNDIALS [12] (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) CVODE (unsteady solver) for solving stiff systems of initial value ODEs for detailed chemistry (chemical Jacobian construction, inversion, and Newton iteration). Load balancing is not handled internally by the SUNDIALS integrators but rather is the responsibility of the application developer. As a result of the highly non-linear characteristics of chemical kinetics, a large variation in the convergence rates of the ODE integrator may occur, leading to a high load imbalance across multiprocessor configurations. However, the independent nature of chemistry ODE systems leads to a problem that can be parallelized easily (embarrassingly parallel) during the flow solution. The presented LeWI model takes advantage of this feature and balances the chemistry load across available resources.

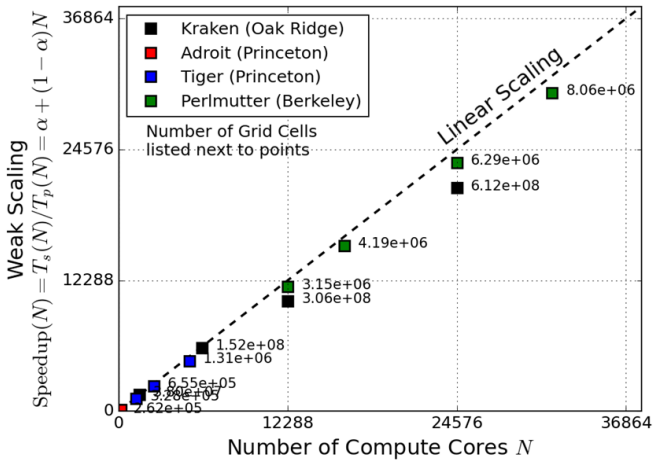

Figure 2: NGA exhibits excellent scale-up characteristics on various high performance architectures. Scaling studies were conducted by increasing the grid resolution and number of processors while maintaining ~ 256 cells per core. The largest run so far has consisted of over 1.6 billion cells, and almost 50,000 compute cores.

Because the ODEs are uncoupled across the domain, there is considerable freedom in staging their integration. In multi-threaded CPU-based implementations, the total work to integrate all the cells in the domain can be distributed arbitrarily across the available threads with no race or synchronization concerns. NGA currently uses a rudimentary load balancing scheme by approximating the work for each cell ahead of time by tracking the work required to integrate the previous timestep. Thus, it is in the earliest timesteps that more robust load balancing scheme could provide considerable speed-up.

3 Implementation

3.1 Systems Component: Dynamic Load Balancing

Lend When Idle [8–10] (LeWI) is a novel *work stealing* [5] algorithm that distributes resources equally among MPI processes in a node while they are doing computation, and reassigns resources of MPI processes while they are blocked in communication calls. One of its main properties is that the load balancing is task neutral, done dynamically at runtime without analyzing nor modifying the application previously. This dynamism allows LeWI to react to different sources of imbalance: algorithm, data, hardware architecture, and resource availability.

Figure 3 shows an example of how the algorithm works. The application is running in a node with 4 CPUs. It starts two MPI processes in the same node and each MPI process spawns

2 OpenMP threads. MPI process 1 gets into the blocking call and will lend its two OpenMP threads to the MPI process 2. The second MPI process will use the newly acquired CPUs as fast as the programming model allows it. When the MPI process 1 gets out of the blocking call it retrieves its CPUs from the MPI process 2 and the execution continues with an equal CPU partition until another blocking call is met.

The LeWI scheme has previously been implemented with both OpenMP and SMPSuperscalar protocols at the inner layer of parallelism [8]. Because NGA is parallelized in a hybrid MPI/OpenMP approach, the following description will only consider OpenMP. OpenMP is limited by the fork-join model and can only change the number of threads outside a parallel region (e.g., "DO" loop). This means that when an MPI process lends its CPUs, the MPI process that wants to use them is not able to do so until reaching a new parallel region. This limitation makes the performance of the algorithm highly dependent on the number of parallel regions that the application presents between MPI blocking calls.

The LeWI module is used in hybrid MPI/OpenMP applications to dynamically and transparently change the process number of threads and their CPU affinity. LeWI mode must be enabled before using the environment variable DLB_ARGS with the value -lewi, and optionally -lewi-keep-one-cpu. Then, place calls to DLB_Borrow() before parallel regions with a high computational load, or at least those near MPI blocking calls. LeWI cannot manage the CPU pinning of each thread and so each MPI rank is run without exclusive CPU binding:

```
$ mpicc -fopenmp opt.c -o opt -I"$DLB_PREFIX/include"\
        -L"$DLB_PREFIX/lib" -ldlb_mpi -Wl,\
        -rpath,"$DLB_PREFIX/lib"
$ export DLB_ARGS="--lewi"
$ mpirun -n 2 --bind-to none ./opt
```

where opt is the NGA makefile. By intercepting MPI calls, LeWI is able to detect when the process reaches a state where the worker threads are idle, then it may temporarily yield the process assigned CPUs to another process that would benefit from them.

In using the OpenMP protocol, both actions (lend and retrieve) are restricted in execution one when entering a new parallel region. Thus while retrieving CPUs the application can run with more threads than CPUs available. If there are more than two MPI processes running in the same node the algorithm must decide to which MPI process (of the ones still running) lends the idle CPUs. Each time a idle CPU is waiting to be assigned the algorithm calculates a load factor for each MPI process. This load factor is obtained by dividing the computational load of the MPI process by the number of threads it has assigned. The idle CPU will be assigned to the MPI process with the higher load factor. For iterative applications, the computational load of each MPI process will correspond to the time accumulated since the beginning of the application.

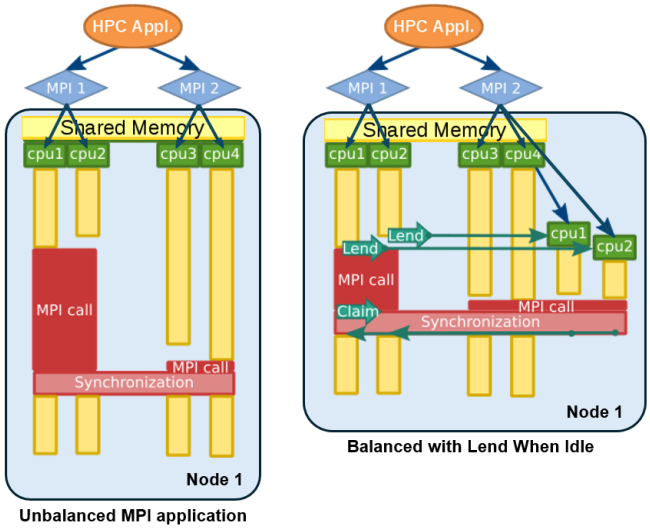

Figure 3: Diagram of MPI thread execution with and without LeWI protocol. Figure adapted from [9].

In the context of reacting flow simulation, the reaction source term for an individual cell can be computed independently from all other cells, through operator splitting as described in previous sections. However, due to the non-linear nature of chemistry, the floating point operations required to evaluate source terms can vary significantly from cell to cell. The computational geometry is decomposed using available domain decomposition algorithms. However, dividing the domain geometrically into blocks of close to equal cell count may lead to chemical imbalance due to the different convergence rates of chemistry ODE problems within processes. Hence, it is beneficial to send some of the chemical problems from the overloaded processors (senders) to the less occupied processors (receivers) for more uniform load balancing. Once the chemistry calculation has been completed, the roles interchange once the updated chemical solutions are sent back to where they physically belong, as source terms in the conservation equations.

3.2 Programming Languages Component: Reference Mapping scheme

The timestep value and hence the total number of floating-point operations during the integration depends on the initial thermochemical composition. Parallelization is achieved with geometrical domain decomposition, leading to explicit chemistry load imbalance due to spatially and temporally varying values. A simple reference mapping feature allows a further reduction in computational cost. This approach groups cells sharing similar thermochemical composition values together and solving the chemistry only once for this group [22]. Such a mapping approach is intended to be used for regions

with low reactivity, (e.g., where no fuel is present or in low-temperature regions far upstream and downstream of the flame, where few chemical reactions are occurring), similar to multi-zone reduction models [20]. **The reference mapping acts as a filter for load balancing, where the reaction rates of cells satisfying a user-given criteria are copied from a reference cell solution.** At a given time instance, a reference cell is picked and the chemistry source term of that cell is solved and copied to other reference cells. The criteria used for identifying the reference cells is: $Z_i < Z_{\text{tol}}$, $|T_i - T_{\text{ref}}| < T_{\text{tol}}$, where Z_i and T_i are mixture fraction (the mass fraction of fuel in a fuel/oxidizer stream, a conserved scalar with no source term) and temperature of i-th cell, respectively, T_{ref} denotes the temperature of the chosen reference cell and Z_{tol} and Z_{tol} are the user-defined tolerance values, specified in the `input` and `config` files.

The reference mapping implemented is applied to each process separately. As long as conservative tolerances are used, the introduced error is rather small and does not affect the global characteristics of the reactive simulation. Figure 4 illustrates the partitioning of cells, where the gray cells indicate regions of low reactivity and are not evaluated, but rather mapped to a similarly low reactivity reference cell. The error is likely negligible compared to typical uncertainties found in various combustion models. The reference mapping model improves the balancing performance significantly by further reducing the load of the more idle processes and increasing their potential to receive more load from the busier processes.

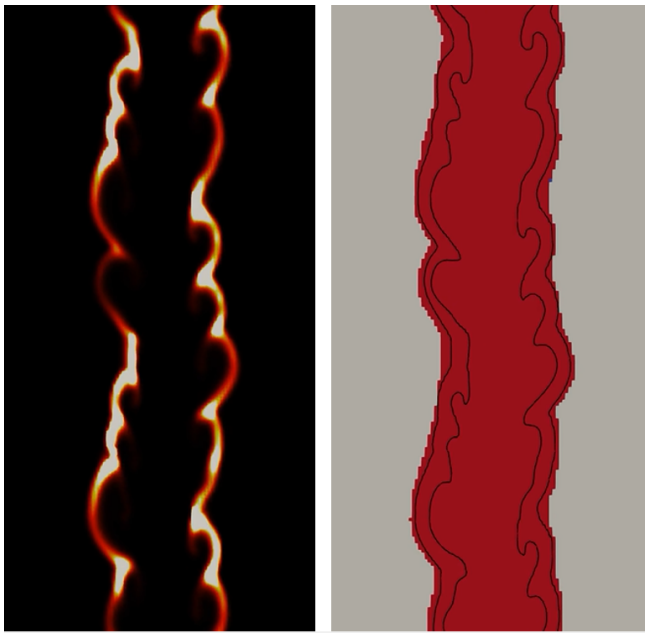

Figure 4: Visualization of the reference mapping implemented in Sandia Flame D. (**left**) Colored by heat release rate, (**right**) **red** cells are those with full chemical Jacobian evaluation, **gray** cells are those where a non-reacting reference cell is copied. $Z_{\text{tol}} = 10^{-2}$ and $T_{\text{tol}} = 1$ [K].

4 Evaluation

4.1 Benchmarks

Runtime performance is evaluated by comparing to simulations of hydrogen-air and methane-air jet flames. These configurations were initially experiments from the turbulent non-premixed flames (TNF) workshop https://tnfworkshop.org/ and are representative of laboratory-scale problems in the combustion science field. These experiments were replicated using Large Eddy Simulation in `NGA` without load balancing and published by Lacey et al. [13, 14]. The experimental data agrees strongly with the simulation, validating the `NGA` modeling paradigm within acceptable accuracy for these cases. The two jet flames are shown qualitatively in Figure 5.

Configuration I: Hydrogen Jet Flame The first configuration is a turbulent non-premixed hydrogen jet flame with a Reynolds number of 10,000 [1–3]. Only pure hydrogen, undiluted by helium, is considered in this work. The flame was simulated in cylindrical coordinates using a stretched grid with $256 \times 144 \times 64$ grid points along the axial, radial, and circumferential directions, respectively. A uniform inlet velocity profile was imposed for the coflow, and the central jet inflow velocity profile was generated from a separate simulation of fully-developed turbulent pipe flow. The chemical mechanism for hydrogen combustion [16] consists of 9 species and 21 reactions. For this flame, statistics were computed using a constant Smagorinsky constant of 0.12 and a constant turbulent Schmidt number of 0.7. However, timings are shown with the dynamic models for comparison with the other flame configuration.

Configuration II: Sandia Flame D The second configuration is Sandia Flame D, a turbulent piloted partially premixed methane/air jet flame with a Reynolds number of 22,400 [1, 4]. The flame was simulated in cylindrical coordinates using a stretched grid with $256 \times 144 \times 64$ grid points along the axial, radial, and circumferential directions, respectively. Uniform inlet velocity profiles were imposed for the coflow and pilot, and the central jet inflow velocity profile was generated from a separate simulation of fully-developed turbulent pipe flow. The mechanism for methane combustion, `GRI-3.0` [21], consists of 35 species with the nitrogen chemistry removed.

4.2 Lend When Idle Results

To achieve statistical convergence, the simulations were run for 30 characteristic "flame time scales" in accordance with

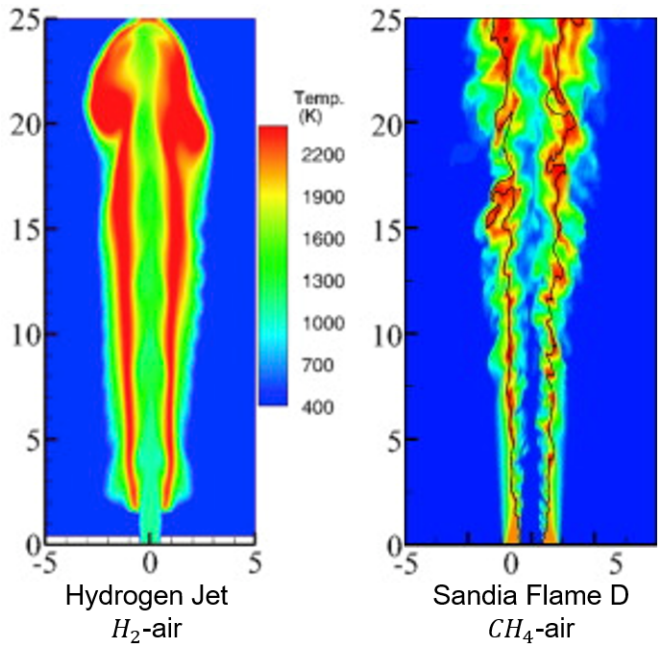

Figure 5: Hydrogen Jet and Sandia Flame D benchmark test cases. Flames are axially symmetric. Spatial numbering represents normalization by the jet diameter: radial (r/D) and axial (x/D). Colored by temperature.

the pseudo-Central Limit Theorem. The chemical timescale τ_C is the laminar flame thickness divided by the laminar burning velocity. For hydrogen, $\tau_C = 0.204$ [msec] and for methane $\tau_C = 1.18$ [msec]. Thus the simulations were run to completion $30\Delta t \cdot \tau_C$. Figures 6 and 7 show the wall clock time per simulation timestep for the Hydrogen Jet and Sandia Flame D, respectively, in a semi-log bar graph. Here, the comparison of "time per timestep" for the balanced and unbalanced simulations serves as an indicator for load balancing. Each simulation timestep will be shorter with computational resources saturated (improved parallel efficiency), all else constant.

It can be seen that, as discussed in previous sections, the first timesteps are extremely costly, taking nearly an hour, due to the highly non-linear initialization of flow and chemical systems. The unbalanced `NGA` solver then estimates the load for each cell using what was required to integrate the previous timestep. For these test cases, the simulation quickly plateaus to only a few seconds per timestep by the 3rd or 4th timestep. It can be seen that Lend When Idle provides a modest improvement between 10% and 30% for the earliest timesteps. In later timesteps, the speed is comparable and slightly slower than the unbalanced version. This is likely due to the additional I/O bottleneck overhead associated with `MPI` calls that may be unnecessary for already fast calculations. A future implementation feature may be to "turn off" Lend When Idle after certain speed plateau criteria are met.

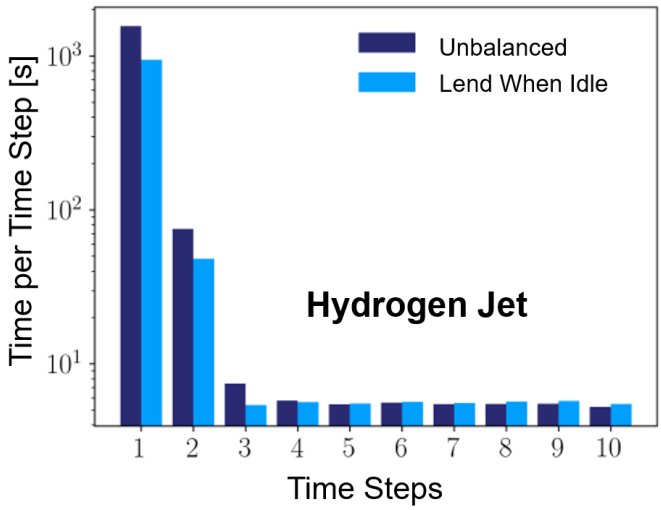

Figure 6: Wall clock time per timestep in the Hydrogen Jet simulation for both the balanced and unbalanced cases. For equal problem size, time per timestep is an indicator of parallel efficiency.

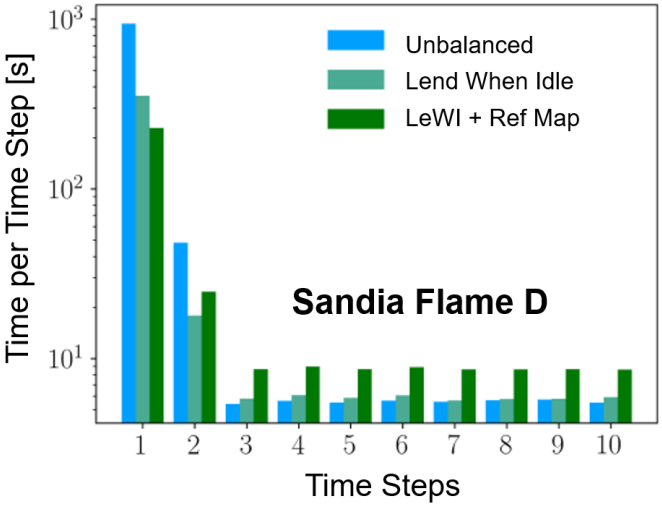

Figure 7: Wall clock time per timestep in the Sandia Flame D simulation for both the balanced and unbalanced cases, as well as balanced with reference mapping. For equal problem size, time per timestep is an indicator of parallel efficiency.

Nonetheless, the speedup in the initial timesteps significantly reduces the cumulative simulation time.

4.3 Reference Mapping Results

The reference mapping model should improve the balancing performance modestly by further reducing the load of

the more idle processes and increasing their potential to receive more load from the busier processes. Implemented only for Sandia Flame D in Figure 7, the scheme provides a further improvement beyond load balancing alone for the initial timesteps. However, past the speed plateau, the scheme slows down iterative timesteps by a factor of 2. This is likely because the scheme is implemented as a looping function through all cells in the domain, which as the flame reaches steady-state, becomes significant in the overall evaluation. This feature too may be worth "turning off" beyond the plateau.

Lend When Idle is purely a manipulation of Message Passing Interface protocol to improve parallel efficiency and makes no assumptions or corrections to the flow physics and chemistry. However, reference mapping seeks to mitigate load imbalance using a custom decomposition with a prior knowledge on the spatial activity of chemical kinetics, which will introduce errors. As long as properly strict tolerances are used (chemical reactions are highly non-linear based in an Arrhenius law), the introduced error should be rather small and not affect the global characteristics of the reactive simulation.

Figure 8 quantifies the error introduced by the reference mapping model. The figure shows the L_2 norm of the error of the volume integral of the heat release rate $\int_V \dot{\omega}\, dV$ over all time instances relative to the results obtained with the standard chemistry model. The results are obtained from the Sandia Flame D case with mixture fraction tolerances $Z_{\text{tol}} = 10^{-6}$-10^{-2} and $T_{\text{tol}} = 1$ [K] fixed. The norm of error is shown for simulations of increasing total number of cells N. When the load balancing algorithm is used without mapping, identical results to the standard model are obtained for all process counts as expected. When the reference mapping is used, overall errors are small, approaching machine precision and scale as $\mathcal{O}(\log(N))$. Spikes in error corresponding to non-powers of 2 are a result of the number of processors per node on the computer hardware. Initially a large deviation around 10% is observed for $Z_{\text{tol}} = 10^{-2}$. With tighter tolerances, this error drops to below 2%.

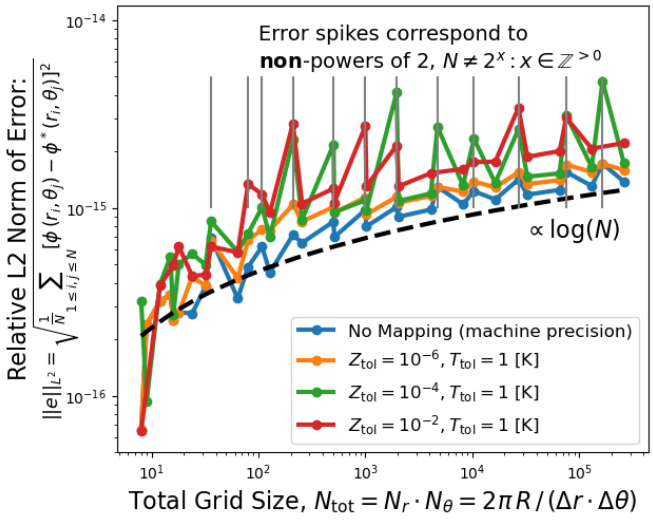

Figure 8: L_2 norm of the error of the volume integral of the heat release rate $\int_V \dot{\omega}\, dV$ over all time instances relative to the results obtained with the standard chemistry model for increasing problem (total grid) size.

5 Related Work: Load Balancing in other reacting flow solvers

Most advanced reacting flow solvers like `Pele` [11, 19] and `OpenFoam` [18, 22] are built on the `AMReX` structure [24]. `AMReX` is a publicly available software framework designed for building massively parallel block-structured adaptive mesh refinement (AMR) applications. `AMReX` provides a very general approach for decomposition of the computational domain into individual logically rectangular grids, and how to distribute those grids to `MPI` ranks. The load balancing process combines grid creation (and re-creation when regridding) and distribution of grids to `MPI` ranks.

For single-level calculations, `AMReX` provides the flexibility to have different size grids, more than one grid per `MPI` rank, and different strategies for distributing the grids to `MPI` ranks. For multi-level calculations, the same principles for load balancing apply as in single-level calculations, but there is additional complexity in how to tag cells for refinement. The process of load balancing is typically independent of the process of grid creation; the inputs to load balancing are a given set of grids with a set of weights assigned to each grid.

Single-level load balancing algorithms are sequentially applied to each AMR level independently, and the resulting distributions are mapped onto the ranks taking into account the weights already assigned to them (assign heaviest set of grids to the least loaded rank). Note that the load of each process is measured by how much memory has already been allocated, not how much memory will be allocated. Distribution options supported by `AMReX` include the following (the default is SFC):

- **Knapsack**: the default weight of a grid in the knapsack algorithm is the number of grid cells, but `AMReX` supports the option to pass an array of weights–one per grid–or alternatively to pass in a MultiFab of weights per cell which is used to compute the weight per grid.
- **SFC**: enumerate grids with a space-filling Z-morton curve, then partition the resulting ordering across ranks in a way that balances the load.
- **Round-robin**: sort grids and assign them to ranks in round-robin fashion–specifically FAB i is owned by CPU $i \in N$ where N is the total number of `MPI` ranks.

6 Future Work

Apart from the optimization of the ODE solver algorithms themselves, several modeling strategies have been proposed to reduce the high computational requirements associated with reacting flows. In particular certain strategies are formulated in a "processor-based" approach, being still prone to yield computational load imbalance across available resources.

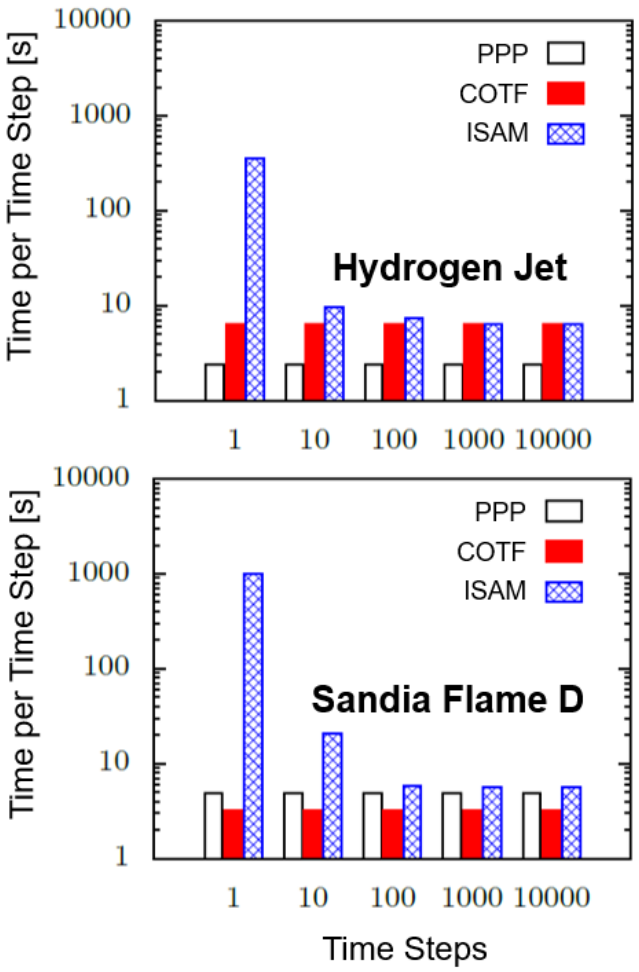

Figure 9: Baseline performance of In-Situ Adaptive Manifold (ISAM) modeling and Convolution-on-the-Fly (COTF) compared to solutions without any optimizations (pre-computed, pre-convoluted, and pre-tabulated, PPP) for the Hydrogen Jet and Sandia Flame D. Reproduced from [14].

Schemes like In-Situ Adapative Manifold (ISAM) modeling and Convolution-on-the-Fly (COTF) developed by Lacey et al. [14] are based on binary-tree search. Thus are costly in the early timesteps as the tree of thermochemical state solutions is built. The cost sharply declines over the first ten timesteps and rapidly reaches a plateau for both test cases, shown in Figure 9. Thus it may be the case that a load balancing scheme, as implemented in this work, could provide a beneficial speed improvement in the early timesteps.

7 Conclusion

Computational load imbalance across large-scale parallel computing architectures can significantly slow simulations of chemically-reacting fluid flows, like combustion. The exponential nature of reaction kinetics makes evaluation of chemical sources terms highly sensitive to local conditions, making prediction and balancing of computational load difficult. This project implemented a dynamic load balancing scheme (Lend When Idle) on a massively parallel low Mach number flow solver `NGA`. Additionally this project developed a reference mapping paradigm to sort group cells with similar thermochemical composition, thereby evaluating their chemical source terms only when sufficiently dissimilar. A modest speed-up is demonstrated for Large Eddy Simulations of two turbulent jet flames, particularly for early simulation timesteps, where computational load is most difficult to predict. These jet flames are characteristic of laboratory-scale problems. It is believed this load balancing scheme will pair well with several processor-based modeling strategies to reduce the high computational requirements associated with reacting flows.

8 Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award Number DE-SC0024386.

The work presented was substantially performed using the Princeton Research Computing resources at Princeton University which is consortium of groups led by the Princeton Institute for Computational Science and Engineering (PICSciE) and Office of Information Technology's Research Computing.

This research used resources of the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory (LBL), a Department of Energy Office of Science User Facility using NERSC award DDR-ERCAP0026889 (project m1266).

References

[1] Barlow, R. S. Sandia H2/He flame data. https://tnfworkshop.org/data-archives/simplejet/, 2003. Release 2.0.

[2] Barlow, R. S., and Carter, C. D. Raman/Rayleigh/LIF measurements of nitric oxide formation in turbulent hydrogen jet flames. *Combustion and Flame 97*, 3 (1994), 261–280.

[3] Barlow, R. S., and Carter, C. D. Relationships among nitric oxide, temperature, and mixture fraction in hydrogen jet flames. *Combustion and Flame 104*, 3 (1996), 288–299.

[4] BARLOW, R. S., AND FRANK, J. H. Effects of turbulence on species mass fractions in methane/air jet flames. *Symposium (International) on Combustion 27*, 1 (1998), 1087–1095. Twenty-Seventh Sysposium (International) on Combustion Volume One.

[5] BLUMOFE, R. D., AND LEISERSON, C. E. Scheduling multithreaded computations by work stealing. *J. ACM 46*, 5 (Sep 1999), 720–748.

[6] CHEN, Z. *Studies on the Initiation, Propagation, and Extinction of Premixed Flames*. Phd thesis, Princeton University, Princeton, NJ, Jan 2009. Available at http://www2.coe.pku.edu.cn/tpic/2011812212957550.pdf.

[7] DESJARDINS, O., BLANQUART, G., BALARAC, G., AND PITSCH, H. High order conservative finite difference scheme for variable density low mach number turbulent flows. *Journal of Computational Physics 227*, 15 (2008), 7125–7159.

[8] GARCIA, M., CORBALAN, J., BADIA, R., AND LABARTA, J. A dynamic load balancing approach with SMPSuperscalar and MPI. In *Facing the Multicore - Challenge II*, R. Keller, D. Kramer, and J.-P. Weiss, Eds., vol. 7174 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 10–23.

[9] GARCIA, M., CORBALAN, J., AND LABARTA, J. LeWI: A runtime balancing algorithm for nested parallelism. In *International Conference on Parallel Processing, 2009. ICPP '09.* (Sep 2009), pp. 526–533.

[10] GARCIA, M., LABARTA, J., AND CORBALAN, J. Hints to improve automatic load balancing with lewi for hybrid applications. *Journal of Parallel and Distributed Computing 74*, 9 (2014), 2781–2794.

[11] HENRY DE FRAHAN, M. T., ROOD, J. S., DAY, M. S., HARISWARAN, S., YELLAPANTULA, S., PERRY, B. A., GROUT, R. W., ALMGREN, A., ZHANG, W., BELL, J. B., AND CHEN, J. H. PeleC: An adaptive mesh refinement solver for compressible reacting flows. *The International Journal of High Performance Computing Applications 37*, 2 (2023), 115–131.

[12] HINDMARSH, A. C., BROWN, P. N., GRANT, K. E., LEE, S. L., SERBAN, R., SHUMAKER, D. E., AND WOODWARD, C. S. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw. 31*, 3 (2005), 363–396.

[13] LACEY, C. E. *Computationally Efficient Data-Enhanced Manifold Modeling of Multi-Modal Turbulent Combustion*. PhD thesis, Princeton University, 2023.

[14] LACEY, C. E., NOVOSELOV, A. G., AND MUELLER, M. E. In-Situ Adaptive Manifolds: Enabling computationally efficient simulations of complex turbulent reacting flows. *Proceedings of the Combustion Institute 38*, 2 (2021), 2673–2680.

[15] LAW, C. K. *Combustion Physics*. Cambridge University Press, 2006.

[16] LI, J., ZHAO, Z., KAZAKOV, A., AND DRYER, F. L. An updated comprehensive kinetic model of hydrogen combustion. *International Journal of Chemical Kinetics 36*, 10 (2004), 566–575.

[17] MACART, J. F., AND MUELLER, M. E. Semi-implicit iterative methods for low mach number turbulent reacting flows: Operator splitting versus approximate factorization. *Journal of Computational Physics 326* (2016), 569–595.

[18] MOREV, I., TEKGÜL, B., GADALLA, M., SHAHANAGHI, A., KANNAN, J., KARIMKASHI, S., KAARIO, O., AND VUORINEN, V. Fast reactive flow simulations using analytical Jacobian and dynamic load balancing in OpenFOAM. *Physics of Fluids 34*, 2 (Feb 2022), 021801.

[19] OWEN, L. D., GE, W., RIETH, M., ARIENTI, M., ESCLAPEZ, L., SORIANO, B. S., MUELLER, M. E., DAY, M., SANKARAN, R., AND CHEN, J. H. PeleMP: The Multiphysics Solver for the Combustion Pele Adaptive Mesh Refinement Code Suite. *Journal of Fluids Engineering 146*, 4 (Feb 2024), 041103.

[20] RAJU, M., WANG, M., DAI, M., PIGGOTT, W., AND FLOWERS, D. Acceleration of detailed chemical kinetics using multi-zone modeling for CFD in internal combustion engine simulations. In *SAE 2012 World Congress & Exhibition* (Apr 2012), SAE International.

[21] SMITH, G. P., GOLDEN, D. M., FRENKLACH, M., MORIARTY, N. W., EITENEER, B., GOLDENBERG, M., BOWMAN, C. T., HANSON, R. K., SONG, S., GARDINER, JR., W. C., LISSIANSKI, V. V., AND QIN, Z. Gri–mech 3.0. http://www.me.berkeley.edu/gri_mech/, 2018.

[22] TEKGÜL, B., PELTONEN, P., KAHILA, H., KAARIO, O., AND VUORINEN, V. DLBFoam: An open-source dynamic load balancing model for fast reacting flow simulations in OpenFOAM. *Computer Physics Communications 267* (2021), 108073.

[23] TORO, E. F. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[24] ZHANG, W., MYERS, A., GOTT, K., ALMGREN, A., AND BELL, J. AMReX: Block-structured adaptive mesh refinement for multiphysics applications. *The International Journal of High Performance Computing Applications 35*, 6 (2021), 508–526.

Notes

The core code libraries for the results presented are available at the following repository, including instructions for running and compiling.

Github: https://github.com/mw6136/NGA_LeWI