Michael D. Walker (mw6136)
COS597E: Programming Languages, Distributed Systems
October 25, 2024

Course Project Literature Survey

1. Introduction

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. **This project proposes to implement a *dynamic load balancing* scheme (specifically Lend When Idle) on my research group's (lab: CTRFL) low Mach number flow solver, `NGA`** [1, 2]. Additionally this project proposes to develop a *reference mapping* paradigm to sort group cells with similar thermochemical composition, thereby evaluating their chemical source terms only when sufficiently dissimilar.

This is literature review covers the related work within the area of the proposed project: **Dynamic Load Balancing in a Massively Parallel Reacting Flow Solver**. This is accomplished by surveying related literature in four areas:

1) An introduction to the low Mach flow solver `NGA`, its associated dependencies, and its current scheme for approximating and distributing load. Additional background is provided on numerical approaches for solution of multi-dimensional reacting flows, especially chemical source term evaluation.
2) Overview of selected existing runtime-based, task-neutral dynamic load balancing schemes and implementations. Specifically *work stealing* algorithms that distribute resources through the `MPI` protocol, like Lend When Idle (LeWI) scheme, are presented.
3) Explanation of *reference mapping schemes* and multi-zone reduction models. Overview of previous implementations in literature.
4) Overview of state-of-the-art reacting flow solvers and how load balancing is accomplished in these libraries.

2. `NGA` and existing Numerical Scheme

`NGA` [1, 2] 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 nonpremixed 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.

A linear or non-linear system can be defined in matrix form as $\Phi \equiv \varphi_{i,j}$ 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 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:

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

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 (i.e., types of molecules) evolving through coupled reactions $\frac{d\Phi}{dt} = \mathbf{S}\Phi$, where $\Phi = [\varphi_1, \varphi_2, ...\varphi_N] \in \mathbb{R}^N$ 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 $[\lambda_1, \lambda_2, ...\lambda_N]$ 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 $\mathbb{J} \equiv \partial S_{i,j} / \partial \varphi_{i,j}$. 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**.

With this technique, chemistry can be treated as an independent stiff ODE system within each computational cell. `NGA` then uses `SUNDIALS` [3] (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).

Typically, the computational cost of the chemical source term evaluation dominates the performance metrics and creates an un-even computational load distribution in parallel applications. The difficulties occur due to the intrinsic and non-linear nature of the ODE system. Cost of solving the associated stiff ODEs scales quadratically with the number of species [4, 5]. 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.

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.

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. The work for each cell can be approximated ahead of time by tracking the work required to integrate the previous timestep. Thus, it is in the earliest timesteps that a Lend When Idle scheme could provide considerable speedup. (Note that unlike the examples [6, 7, 8] presented in the `SUNDIALS` paper, `NGA` does not have adaptive mesh refinement, so it should be a simpler implementation.)

3. Systems Component: Dynamic Load Balancing schemes

Lend When Idle [9, 10, 11] (LeWI) is a novel *work stealing* [12] algorithm that distributes resources equally among `MPI` processes in a node while they are doing computation, and re-assigns 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 at runtime without analyzing nor modifying the application previously.

The LeWI scheme has previously been implemented with both `OpenMP` and `SMPSuperscalar` protocols at the inner layer of parallelism [11]. `OpenMP` 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 (i.e., if there is just one parallel loop between `MPI` blocking calls we cannot change the number of threads, therefore, the application cannot be balanced). Conversely, `SMPSuperscalar` is a shared memory programming model that allows for the change of the number of threads at any time.

4. Programming Languages Component: Reference Mapping schemes

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 will allow 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 [4]. 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 [13]. **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) 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.

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. 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.

5. Other reacting flow solvers

`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 speedup. Several test cases will be used to evaluate performance. First, to verify proper code execution a fully-developed turbulent pipe flow configuration (presented in Appendix C) will be simulated. This case has previously been used to conduct weak scaling studies of `NGA` with increasing grid resolution. More detailed performance benchmark test cases would be simulated using Large Eddy Simulation data of a hydrogen jet flame [14, 15, 16] and Sandia Flame D [17]. Such simulations were published[18] using `NGA` without load balancing (note the massive time per timestep for the early timesteps of Figures 4 and 5 therein).

Most advanced reacting flow solvers like `Pele` [7, 8] and `OpenFoam` [4, 5] are built on the `AMReX` structure [6]. `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.

Fluid Solver Advanced schemes solve the coupled Navier-Stokes system using a second order finite volume 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 system of equations is shown below. 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. Along with the ideal gas equation of states, these equations represent a closed system. The finite volume method is employed for discretizing the governing equations, starting with a control volume Φ and flux F:

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

The conservation equations for one-dimensional, multi-component, reactive compressible flows can be written as

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

F_ν is the diffusion flux, S is the reaction term, and G is the geometric shape factor [19]. The vectors Φ, $F(\Phi)$, $G(\Phi)$, $F_\nu(\Phi)$, and $S(\Phi)$ are defined as

$$\Phi = \begin{bmatrix} \rho Y_1 \\ \rho Y_2 \\ \vdots \\ \rho Y_k \\ \rho u \\ E \end{bmatrix} \quad \mathbf{F}(\Phi) = \begin{bmatrix} \rho u Y_1 \\ \rho u Y_2 \\ \vdots \\ \rho u Y_k \\ \rho u^2 + P \\ (E+P)u \end{bmatrix} \quad \mathbf{G}(\Phi) = \begin{bmatrix} \rho u Y_1 \\ \rho u Y_2 \\ \vdots \\ \rho u Y_k \\ \rho u^2 \\ (E+P)u \end{bmatrix} \quad \mathbf{F}_\nu(\Phi) = \big[\text{viscous terms}\big] \quad \mathbf{S}(\Phi) = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_k \\ 0 \\ 0 \end{bmatrix}$$

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. Energy is the low Mach number limit is $E = \rho\,\mathrm{e} + \cancel{\rho u^2/2} = \rho\,\mathrm{e}$.

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 [20].

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

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

1) Data Reconstruction
2) Evolution
3) Riemann Solution
4) Conservative Update

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 2-D cells. The equations describing cell slopes in the i and j direction are:

$$\Delta_i = \begin{cases} \mathbf{max}[0, \mathbf{min}(\beta\Delta_{i-1/2}, \Delta_{i+1/2}), \mathbf{min}(\Delta_{i-1/2}, \beta\Delta_{i+1/2})], & \Delta_{i+1/2} > 0 \\ \mathbf{min}[0, \mathbf{max}(\beta\Delta_{i-1/2}, \Delta_{i+1/2}), \mathbf{max}(\Delta_{i-1/2}, \beta\Delta_{i+1/2})], & \Delta_{i+1/2} < 0 \end{cases}$$

$$\Delta_j = \begin{cases} \mathbf{max}[0, \mathbf{min}(\beta\Delta_{j-1/2}, \Delta_{j+1/2}), \mathbf{min}(\Delta_{j-1/2}, \beta\Delta_{j+1/2})], & \Delta_{j+1/2} > 0 \\ \mathbf{min}[0, \mathbf{max}(\beta\Delta_{j-1/2}, \Delta_{j+1/2}), \mathbf{max}(\Delta_{j-1/2}, \beta\Delta_{j+1/2})], & \Delta_{j+1/2} < 0 \end{cases}$$

where $\Delta_{i\pm1/2}$, $\Delta_{j\pm1/2}$ are the slopes in i and j at the cell faces, using a value of $\beta = 1$. The *boundary extrapolated values* (Φ_i^L, Φ_i^R, Φ_j^L, Φ_j^R) are then calculated.

$$\Phi_i^L = \Phi_{i,j}^n - \frac{1}{2}\Delta_i, \;\; \Phi_i^R = \Phi_{i,j}^n + \frac{1}{2}\Delta_i$$

$$\Phi_j^L = \Phi_{i,j}^n - \frac{1}{2}\Delta_j, \;\; \Phi_j^R = \Phi_{i,j}^n + \frac{1}{2}\Delta_j$$

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

$$\bar{\Phi}_{i,j}^{L,R} = \Phi_{i,j}^{L,R} + \frac{1}{2}\frac{\Delta t}{\Delta x}\left(\begin{bmatrix} \rho u_j \\ \rho u_i u_j \\ \rho u_j^2 + P \\ u_j(E+P) \end{bmatrix}_i^L - \begin{bmatrix} \rho u_j \\ \rho u_i u_j \\ \rho u_j^2 + P \\ u_j(E+P) \end{bmatrix}_i^R + \begin{bmatrix} \rho u_i \\ \rho u_i^2 + P \\ \rho u_i u_j \\ u_i(E+P) \end{bmatrix}_j^L - \begin{bmatrix} \rho u_i \\ \rho u_i^2 + P \\ \rho u_i u_j \\ u_i(E+P) \end{bmatrix}_j^R\right)$$

The Riemann problem can now be solved using the following equations for the left and right states:

$$\Phi_{i,riemann}^L = \bar{\Phi}_i^R, \;\; \Phi_{i,riemann}^R = \bar{\Phi}_{i+1}^L$$

$$\Phi_{j,riemann}^L = \bar{\Phi}_j^R, \;\; \Phi_{j,riemann}^R = \bar{\Phi}_{j+1}^L$$

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 [20]. It determines the states of nearby cells and then calculates the variable fluxes across those cells using conservation laws.

Conservative Update: The conserved variables are then updated through the equation below. This is the final step of the MUSCL-Hancock scheme, and is repeated for each consecutive time step.

$$(1) \qquad \Phi_{i,j}^{n+1} = \Phi_{i,j}^n + \frac{\Delta t}{\Delta x}\left(\begin{bmatrix} \rho u_j \\ \rho u_i u_j \\ \rho u_j^2 + P \\ u_j(E+P) \end{bmatrix}_{i-\frac{1}{2}} - \begin{bmatrix} \rho u_j \\ \rho u_i u_j \\ \rho u_j^2 + P \\ u_j(E+P) \end{bmatrix}_{i+\frac{1}{2}}\right) + \frac{\Delta t}{\Delta y}\left(\begin{bmatrix} \rho u_i \\ \rho u_i^2 + P \\ \rho u_i u_j \\ u_i(E+P) \end{bmatrix}_{j-\frac{1}{2}} - \begin{bmatrix} \rho u_i \\ \rho u_i^2 + P \\ \rho u_i u_j \\ u_i(E+P) \end{bmatrix}_{i+\frac{1}{2}}\right)$$

Chemical Source Term Evaluation The chemistry is solved in the second fractional step for a homogeneous system

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

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 time step 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 exceedingly simple but only has first-order accuracy in time, when S and C are at least first-order accurate solution operators. 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. For the $\mathbf{S}$ operator, the mass fraction of species are updated by using the `CVODE` solver. 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

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

Entropy Generation While the inlet-to-outlet mass flow ratio is one way to assess the quality of the solution, it is not always reliable. The flow field can be incorrect despite a ratio near unity. Entropy generation in the system can be computed as a better measure of accuracy. The total entropy change across the system, that is, between the inlet and outlet, can be estimated using the mean static temperature and pressure values from the inlet and summing the entropy across each j line [21].

$$\Delta s = \sum_{j_{outlet}} c_p \cdot \ln\left(\frac{T_{j,outlet}}{\overline{T}_{static,inlet}}\right) - R \cdot \ln\left(\frac{P_{j,outlet}}{\overline{P}_{static,inlet}}\right)$$

Entropy generation can also be evaluated between each grid point. This is under the assumption that entropy is being advected along the j lines, like streamlines.

$$\Delta s_{i,j} = c_p \cdot \ln\left(\frac{T_{i,j}}{T_{i-1,j}}\right) - R \cdot \ln\left(\frac{P_{i,j}}{P_{i-1,j}}\right)$$

Grid Generation and Load Balancing 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 and how to create the union of grids at levels greater than 0 where that union most likely does not cover the computational domain. 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.

References

[1] O. Desjardins, G. Blanquart, G. Balarac, and H. Pitsch. High order conservative finite difference scheme for variable density low mach number turbulent flows. *Journal of Computational Physics*, 227(15):7125–7159, 2008.

[2] J. F. MacArt and M. E. Mueller. Semi-implicit iterative methods for low mach number turbulent reacting flows: Operator splitting versus approximate factorization. *Journal of Computational Physics*, 326:569–595, 2016.

[3] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, 2005.

[4] B. Tekgül, P. Peltonen, H. Kahila, O. Kaario, and V. Vuorinen. DLBFoam: An open-source dynamic load balancing model for fast reacting flow simulations in OpenFOAM. *Computer Physics Communications*, 267:108073, 2021.

[5] I. Morev, B. Tekgül, M. Gadalla, A. Shahanaghi, J. Kannan, S. Karimkashi, O. Kaario, and V. Vuorinen. Fast reactive flow simulations using analytical Jacobian and dynamic load balancing in OpenFOAM. *Physics of Fluids*, 34(2):021801, Feb 2022.

[6] W. Zhang, A. Myers, K. Gott, A. Almgren, and J. Bell. AMReX: Block-structured adaptive mesh refinement for multiphysics applications. *The International Journal of High Performance Computing Applications*, 35(6):508–526, 2021.

[7] M. T. Henry de Frahan, J. S. Rood, M. S. Day, S. Hariswaran, S. Yellapantula, B. A. Perry, R. W. Grout, A. Almgren, W. Zhang, J. B. Bell, and J. H. Chen. PeleC: An adaptive mesh refinement solver for compressible reacting flows. *The International Journal of High Performance Computing Applications*, 37(2):115–131, 2023.

[8] L. D. Owen, W. Ge, M. Rieth, M. Arienti, L. Esclapez, B. S. Soriano, M. E. Mueller, M. Day, R. Sankaran, and J. H. Chen. PeleMP: The Multiphysics Solver for the Combustion Pele Adaptive Mesh Refinement Code Suite. *Journal of Fluids Engineering*, 146(4):041103, Feb 2024.

[9] M. Garcia, J. Corbalan, and J. Labarta. LeWI: A runtime balancing algorithm for nested parallelism. In *Parallel Processing, 2009. ICPP '09. International Conference on*, pages 526–533, Sep 2009.

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

[11] M. Garcia, J. Corbalan, R.M. Badia, and J. Labarta. A dynamic load balancing approach with SMPSuperscalar and MPI. In R. Keller, D. Kramer, and J.-P. Weiss, editors, *Facing the Multicore - Challenge II*, volume 7174 of *Lecture Notes in Computer Science*, pages 10–23. Springer Berlin Heidelberg, 2012.

[12] R. D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, Sep 1999.

[13] M. Raju, M. Wang, M. Dai, W. Piggott, and D. Flowers. Acceleration of detailed chemical kinetics using multi-zone modeling for CFD in internal combustion engine simulations. In *SAE 2012 World Congress & Exhibition*. SAE International, Apr 2012.

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

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

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

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

[18] C. E. Lacey, A. G. Novoselov, and M. E. Mueller. In-Situ Adaptive Manifolds: Enabling computationally efficient simulations of complex turbulent reacting flows. *Proceedings of the Combustion Institute*, 38(2):2673–2680, 2021.

[19] Z. Chen. *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.

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

[21] P. G. Tucker. *Advanced Computational Fluid and Aerodynamics*. Cambridge Aerospace Series. Cambridge University Press, 2016.