Computer Physics Communications 267 (2021) 108073

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

DLBFoam: An open-source dynamic load balancing model for fast reacting flow simulations in OpenFOAM ☆,☆☆

Bulut Tekgül [a,*], Petteri Peltonen [a], Heikki Kahila [b], Ossi Kaario [a], Ville Vuorinen [a]

[a] Department of Mechanical Engineering, Aalto University School of Engineering, Otakaari 4, 02150 Espoo, Finland
[b] Wärtsilä Finland Oy, 65101 Vaasa, Finland

ARTICLE INFO

Article history:
Received 17 November 2020
Received in revised form 25 May 2021
Accepted 8 June 2021
Available online 18 June 2021

Keywords:
Reacting flow
Combustion
Load balancing
OpenFOAM
Chemical kinetics

ABSTRACT

Computational load imbalance is a well-known performance issue in multiprocessor reacting flow simulations utilizing directly integrated chemical kinetics. We introduce an open-source dynamic load balancing model named `DLBFoam` to address this issue within OpenFOAM, an open-source C++ library for Computational Fluid Dynamics (CFD). Due to the commonly applied operator splitting practice in reactive flow solvers, chemistry can be treated as an independent stiff ordinary differential equation (ODE) system within each computational cell. 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 (called an *embarrassingly parallel* problem in the literature) during the flow solution. The presented model takes advantage of this feature and balances the chemistry load across available resources. Additionally, a reference mapping model is utilized to further speed-up the simulations. When `DLBFoam` it utilized with both these features enabled, a speed-up by a factor of 10 is reported for reactive flow benchmark cases. To the best of our knowledge, this model is the first open-source implementation of chemistry load balancing in the literature.

Program summary

Program Title: DLBFoam
CPC Library link to program files: https://doi.org/10.17632/bb9zjfzcmm.1
Developer's repository link: https://github.com/blttkgl/DLBFoam
Licensing provisions: GPLv3
Programming language: C++
Nature of problem: Solution of chemical kinetics in parallel reacting flow solvers raises a computational imbalance across multiprocessor architectures. DLBFoam balances the load distribution evenly, providing significant speed-up in reacting CFD applications.
Solution method: The dynamic load balancing is implemented by distributing the point-wise chemistry problems from most loaded processes to less loaded ones using MPI communication protocol.
Additional comments including restrictions and unusual features: The present model is designed to work with the standard chemistry model class available in OpenFOAM (versions 7 and 8). For the time being, the model does not support derived combustion models such as "TDAC" and covers gas-phase reaction kinetics only. In addition, the boundary surface chemistry problems are neglected by the model.

1. Introduction

The development of efficient Computational Fluid Dynamics (CFD) simulation tools for reacting flows is a crucial step in the research and development of less polluting combustion concepts [1]. 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 review of this paper was arranged by Prof. Hazel Andrew.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author.
E-mail address: bulut.tekgul@aalto.fi (B. Tekgül).

https://doi.org/10.1016/j.cpc.2021.108073

the computational cost of fluid dynamics by a factor of 100 [2]. Even though most CFD software utilize distributed-memory parallel architectures to their full extent, incorporating an efficient solution for reacting flows tends to lead to a high computational load imbalance during parallel execution [3].

Commonly, engineering CFD codes assume an operator-splitting strategy in the reacting flow solver implementation, enabling the decoupling of the chemistry and fluid dynamics in the reacting flow solution during the calculation of the chemical source terms in the governing equations [4,5]. Therefore, the changes in thermochemical composition Φ (temperature, pressure, and species concentrations) due to chemical reactions per computational element (e.g. finite-volume cell) are computed by solving a cell-wise independent homogeneous reaction system. The reaction system describes the rate of change of thermochemical composition as a stiff system of ordinary differential equations (ODEs), i.e. $\partial\Phi/\partial t = f(\Phi, t)$ [4].

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. As the computational cost of solving the associated stiff ODEs scales quadratically with the number of species [6], detailed reaction mechanisms easily become impractical. 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 time step values [7]. The time step value and hence the total number of floating-point operations during the integration depends on the initial thermochemical composition [8]. Furthermore, in most CFD codes parallelization is achieved with geometrical domain decomposition, leading to explicit chemistry load imbalance due to spatially and temporally varying Φ values. A visual illustration of the chemistry load imbalance is presented in Fig. 1.

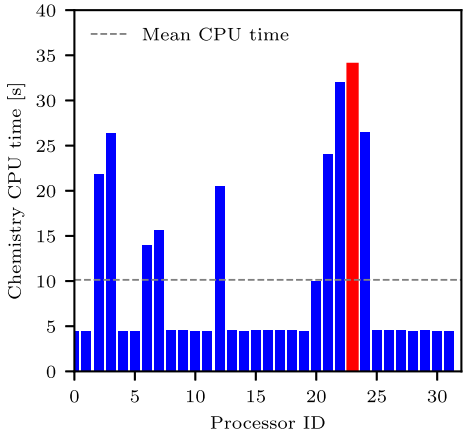

Fig. 1. An illustration of the computational imbalance in a reactive CFD simulation. The cost of the chemistry solution across different processes within a given CFD time step varies, creating a bottleneck at the process with highest computational load (marked with red). Data, plotting scripts and figure are available under the CC-BY license [9]. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Apart from the optimization of the ODE solver algorithms and implementations [6,10,11], several modeling strategies have been proposed to reduce the high computational requirements associated with reacting flows. To name a few successful approaches: in-situ adaptive tabulation method [12], dynamic and adaptive reduction methods for chemical kinetics [13] and dynamic stiffness removal methods [14] each provide performance gains to a certain extent. However, these strategies are often formulated in a processor-based approach, being still prone to yield computational load imbalance across available resources [15].

In the context of the present study, dynamic load balancing of chemistry has been previously covered in literature. One method to mitigate the load imbalance is to use a custom decomposition with a prior knowledge on the spatial activity of chemical kinetics [16]. However, in large complex geometries, custom decomposition becomes impractical [16,17]. In terms of dynamic run-time load-balancing algorithms, Antonelli et al. [18] developed a Message-Passing Interface (MPI) based parallel solver which utilizes a cell distribution based load balancing algorithm. Both Kodavasal et al. [19] and Shi et al. [20] considered stiffness detection approaches as balancing criteria for their balancing algorithms. Recently, Muela et al. [3] presented a dynamic load balancing method which also utilized a stiff cell detector to choose the optimal ODE integration method. In practical benchmark cases, the aforementioned methods reported speed-up factors in the range three to five.

In contrast to previous studies where implementations are either not publicly available or based on commercial CFD codes, in the present study we introduce a robust open-source load balancing algorithm for parallel reacting flow simulations. To the best of our knowledge, this model is the first open-source implementation of chemistry load balancing in literature. In addition to the load balancing model, a zonal reference mapping model is also introduced to further improve the available balancing performance. The implementation is carried out in OpenFOAM, an open-source C++ library targeted for CFD applications [21]. It is worth noting that an earlier version of this now publicly shared model has already been utilized in multiple published combustion studies by the authors [22–24].

The paper is outlined as follows: The implemented dynamic load balancing algorithm is presented in Section 2. Section 3 provides benchmarks highlighting the theoretical performance gains of this model, while Section 4 reports performance metrics in practical reacting flow configurations. Section 5 concludes the study with a brief discussion on the results along with the potential for further improvements.

2. Implementation details

2.1. Reacting flow solver

A schematic of the different stages of the reacting flow solver used in the paper is presented in Fig. 2. Solution for mass, momentum, species, and energy conservation laws is achieved in an iterative manner. The `reactingFoam` and `sprayFoam` reactive solvers of OpenFOAM [21] with compressible PIMPLE algorithm are used. Within each time step, a Poisson equation for pressure is solved and the velocities obtained from the solution of the momentum equation are corrected to ensure that the mass is conserved.

Here we consider the solution of chemistry source terms (i.e. net production/consumption rates of species concentrations) as a separate step, as highlighted in Fig. 2. In the present study, the chemistry source terms are evaluated by direct integration of chemical kinetics with no additional models. The source term evaluation is performed in a separate module which is interfaced at the solver level (chemistry model). The implementation of the load balancing algorithm is carried out in this module as a separate compilation unit. The details of the algorithm are discussed in the following subsection.

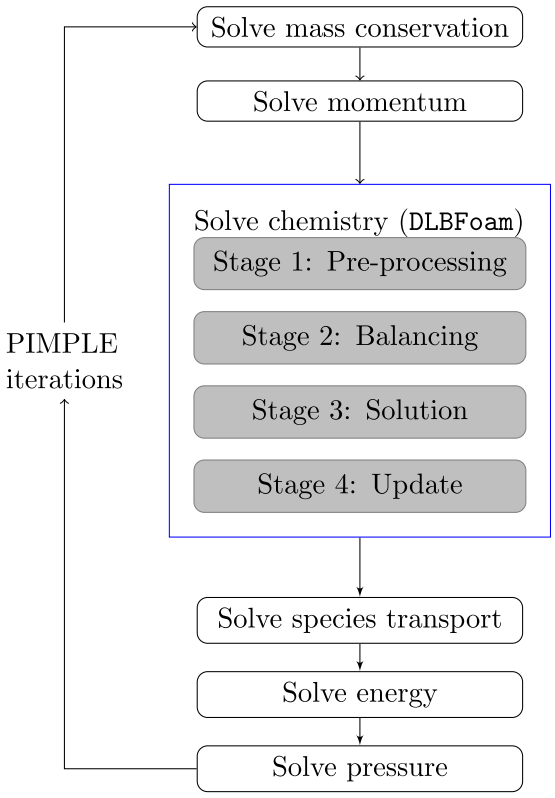

Fig. 2. A schematic showing the main steps of the reactive solver within a single CFD time step. The `DLBFoam` part is highlighted in the figure.

2.2. Dynamic load balancing

As mentioned before, the reaction source term for an individual CFD cell can be computed independently from all other cells. 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. In most CFD simulations, the computational geometry is decomposed using some of the available domain decomposition algorithms. However, dividing the domain geometrically into chunks of close to equal cell count may lead to chemical imbalance due to the different convergence rates of chemistry ODE problems within processes, as illustrated in Fig. 1. 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. Obviously, once the chemistry calculation has been completed, the roles interchange once the updated chemical solutions are sent back to where they physically belong i.e. as source terms in the species and enthalpy equations.

The dynamic load balancing algorithm implemented in the present paper consists of four stages: 1) preprocessing stage, 2) balancing stage 3) solution stage, and 4) update stage. In the preprocessing stage, the thermochemical variables and other variables needed for the ODE solution for each cell n are packed into a single data structure, here denoted as a `problem`, and an array of these structures is formed. Fig. 3 describes the variables within a `problem` data structure needed to describe a single chemistry ODE problem. A `problem` consists of N_{sp}+6 double precision floating point variables and one integer variable. Here, N_{sp} is the number of species in the utilized chemical kinetics mechanism. The amount of data to be transferred between a sender-receiver pair depends on various case setup parameters such as the size of the utilized chemical mechanism, number of cells per process and the level of imbalance between a sender-receiver pair.

problem
`Species concentration (double array, length` N_{sp}`)`
`Temperature (double)`
`Pressure (double)`
`Density (double)`
`Cell ID (int)`
`Chemistry Timestep (double)`
`CFD Timestep (double)`
`Cell CPU Time (double)`

Fig. 3. Data structure of a problem that contains the necessary information to solve the chemistry ODE system of the cell n. In C++, the `problem` is implemented as a `struct` data type. N_{sp} denotes the number of species in the utilized chemical kinetics mechanism.

Algorithm 1 Pseudo-code representation of the load balancing algorithm stage two over N_p processes w.r.t. a nth cell CPU times t^n_{cpu} at a time $t = n + 1$.

1: $l_k \leftarrow \sum_{n=1}^{N_c} t^n_{\text{cpu}}$ // load for rank k at given time
2: $L \leftarrow$ allGather(l_k) // List of loads $L = \left[l_0, l_1, ..., l_{N_p-1}\right]$
3: $\overline{L} \leftarrow \sum_{i=0}^{N_p} -1 l_k / N_p$ // average load over all processes
4: $L, R \leftarrow$ sort(L) // Sort L, and get a new rank indexing R
5: $i \leftarrow 0$
6: $j \leftarrow N_p - 1$
7: **while** $i \neq j$ **do**
8: $\quad \Delta L_{send} \leftarrow \min(\overline{L} - L_i, L_j - \overline{L})$ // Define load to be transferred
9: $\quad r_r \leftarrow R_i$ // Rank ID receiving data
10: $\quad r_s \leftarrow R_j$ // Rank ID sending data
11: $\quad$ find N_s, s.t. $\Delta L_{send} \approx \sum_{n=1}^{N_s} t^n_{\text{cpu}}$ // Corresponding N_s problems
12: $\quad$ `operations.insert(`r_s, r_r, N_s`)` // List of balancing operations
13: $\quad L_i \leftarrow L_i + \Delta L_{send}$ // Increase by the send value
14: $\quad L_j \leftarrow L_j - \Delta L_{send}$ // Decrease by the send value
15: $\quad$ **if** $|L_i - \overline{L}| \approx 0$ **then**
16: $\quad\quad i \leftarrow i + 1$ // Move to next receiver
17: $\quad$ **else**
18: $\quad\quad j \leftarrow j - 1$ // Move to next sender
19: $\quad$ **end if**
20: **end while**
21: `process(operations)` // Call MPI functions
22: // based on the operations list.

Next, stage 2 is discussed together with a pseudocode illustration presented in Algorithm 1. Stage 2 is initiated by defining a processor-based load value l_k for all available N_p processes (line 1). Here, we define the load l_k as the sum of total CPU time spent on solving N_c problems on process k. To compute the global balancing statistics, a sorted global list of loads $L = \left[l_0, l_1, ..., l_{N_p-1}\right]$ is formed and broadcasted to all available processes (line 2). Next, the algorithm aims at setting the global arithmetic mean load ($\overline{L}$) to all processes by finding sender-receiver rank pairs (r_s, r_r) sharing a load ΔL_{send} (line 8). Then, the CPU time based load value is converted to N_s chemistry problems to be transferred between the sender-receiver pair (line 11). It is important to note that the smallest amount of load that can be sent between processes is controlled by explicitly removing very small load balancing communications and not executing them. This is done by ensuring that for a given sender-receiver pair, $\Delta L_{send} \geq 0.01 \times \overline{L}$ is satisfied. Finally, r_s, r_r and N_s are inserted into a list denoted as `operations` (line 12), which is utilized to execute multiple non-blocking MPI calls to communicate the problems across processes (line 21). Note that in the case of a sender with multiple receivers or a receiver with multiple senders, the communication protocol is organized so that all the data are sent/received simultaneously.

In stage 3, the distributed chemistry problems are solved by their new processes. We note that the input data (a problem) is tagged with a cell id and the tag is copied to the output data (a solution) to ensure the correct placement of the reaction rate in the CFD domain in the original process. In stage 4, the solutions are communicated back to the original processes based on the same (r_s, r_r, N_s) information from stage 2 and the chemical source term values of the cells are updated. It should be noted that the stage 4 also includes an explicit MPI barrier to ensure that the processes update the reaction rates only after all the problems are communicated back to their original processes.

2.3. Reference mapping

In addition to the load balancing, we have implemented a simple reference mapping feature which allows us to further reduce the computational cost. The aim is to group cells sharing similar thermochemical composition (Φ) values together and solve the chemistry only once for this group. Such a mapping approach is intended to be used for regions with low reactivity, e.g. where no fuel is present. Reference mapping model is very similar to multi-zone reduction models found in commercial CFD software [25]. However, by design our approach is far more conservative since it is mostly intended for mapping non-reacting mixtures including no fuel to one another. In our implementation, the reference mapping acts as a filter in stage 1 of the load balancing algorithm, 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:

$$Z_i < Z_{tol},$$

$$|T_i - T_{ref}| < T_{tol},$$

where Z_i and T_i are mixture fraction and temperature of ith cell, respectively, T_{ref} denotes the temperature of the chosen reference cell and Z_{tol} and T_{tol} denote the user-given tolerance values. In the reacting flow community, the mixture fraction stands for a conserved scalar describing the mixing state of a fuel and oxidizer uniquely. Although there are different definitions of Z in the literature, in this paper the Bilger's definition [26] is used. The reference solution is computed from the first reference cell found, and this solution is then mapped to the subsequent reference cells.

The reference mapping implemented in the present study is applied to each process separately. Depending on the value set of Z_{tol} and T_{tol}, as well as whether the temperature criterion is applied or not, the mapped solution and the actual solution may be slightly different from one another. As long as proper tolerances are used, the introduced error is rather small and does not affect the global characteristics of the reactive simulation. According to the authors' experience, the error is negligible compared to typical uncertainties found in various combustion models. Details on the validation of the reference mapping against the standard model as well as a discussion on the level of error introduced are presented in Appendix A. We emphasize that 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. However, due to its process-based formulation it often provides infinitesimal speedup if it is utilized without the load balancing. It is also worth noting that reference mapping may introduce some approximation to the chemistry solution especially at the decision boundary between the mapped and unmapped regions. Therefore, it should be utilized with caution like any other zonal reduction model.

3. Unit benchmarks

In this section, the performance of the load balancer model is demonstrated by using a simple test environment created in OpenFOAM. Inspired by the performance analysis presented by Muela et al. [3], we present a similar analysis to show the efficiency as well as the scaling performance of our load balancer model for: i) different number of processes and ii) different number of cells per process.

To carry out the performance analysis, we have developed a benchmarking suite in OpenFOAM, where a given set of chemistry problems are solved in parallel by the ODE solver and the total execution times are measured. Two different *pseudo-problems* are predefined: a *heavy problem*, which is a stiff ODE problem and takes longer to compute, and a *light problem*, which is a less stiff problem with a shorter solution time. Here, a benchmark environment is created by defining the number of chemistry problems (N_c), available resources (N_p processes), and a predefined θ value, describing the ratio of *heavy problems* to the number of problems in each process.

Throughout the benchmark analysis, the ratio of all *heavy problems* to the total number of problems across all processes is fixed as 0.2. Different configurations are then created to distribute the total heavy load either evenly (balanced load configuration) or unevenly (unbalanced load configuration). The following benchmark configurations are investigated:

- C_1: "very unbalanced", θ = 1 for 20% of N_p
- C_2: "slightly unbalanced", θ = 0.8 for 25% of N_p
- C_3: "slightly balanced", θ = 0.4 for 50% of N_p
- C_4: "very balanced", θ = 0.2 for 100% of N_p.

The benchmark statistics are obtained by running 10 separate samples for each configuration to minimize the variance due to any hardware or parallel communication issues. Each case is run first with the unbalanced (standard) model, then with the balanced model, and the speed-up ratio χ_{su} for each condition is calculated as:

$$\chi_{su} = \frac{\tau_{unb}}{\tau_{bal}}, \tag{1}$$

where τ_{unb} and τ_{bal} are the mean execution times over all samples of unbalanced and balanced cases, respectively. In addition, using the ratio of a *heavy* problem to a *light* problem (ξ), the maximum theoretical speed-up that can be attainable for cases C_1-C_4 are calculated as:

$$\frac{\theta \times \xi + (1-\theta)}{\frac{\theta \times \xi + (1/x - 1)}{1/x}}, \tag{2}$$

where x is the percentage of processes which θ condition applies for each condition (for instance, it is 20% (0.2) for C_1).

All simulations in this paper are carried out by the Mahti supercomputer, provided by the CSC - Finnish IT Center for Science. Mahti contains 1404 nodes with two 64 core AMD EPYC (Rome) processors, each node running at 2.6 GHz, as well as high-speed 200 Gpbs infiniband HDR interconnection network between the nodes, all running on a RHEL 7.8 Linux operating system [27].

Fig. 4 shows the benchmark results for configurations C_1-C_4 with varying N_p and N_c values. Fig. 4a highlights the steady performance over a large range of process counts with a fixed N_c = 200. It can be seen that the balancing algorithm scales up to N_p = 1280 with no apparent performance issue. In addition, the χ_{su} value increases with increasing imbalance ($C_4 \rightarrow C_1$), since the potential balancing gain to be obtained is higher for cases with higher imbalance.

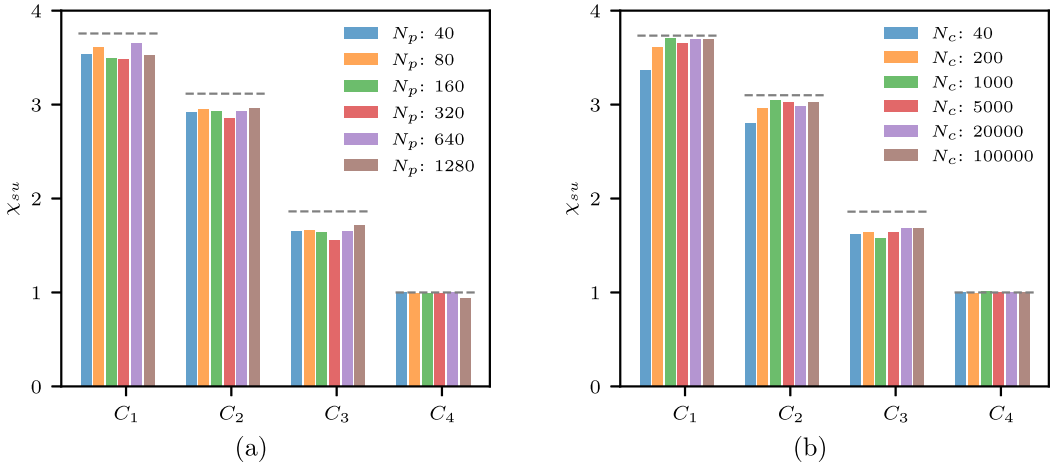

Fig. 4. Speed-up ratio of a) different process counts N_p with number of problems N_c = 200 b) different N_c with N_p = 80, from imbalanced (C_1) to balanced (C_4) configurations. The dashed lines show the maximum theoretical speed-up that can be obtained for each configuration. Data, plotting scripts and figure are available under the CC-BY license [9].

Fig. 4b shows the influence of problem count N_c to balancing performance for a fixed N_p = 80. It can be clearly seen from the high imbalance condition (C_1) that the χ_{su} first increases with increasing N_c from 40 up to 1000, then it stays in the same level for higher N_c values. The reason is two-fold: i) for lower N_c values the global mean load value ($\overline{L}$) that the balancer tries to reach is smaller, therefore the cost of communication is comparable to the cost of gain obtained from the balancing, and ii) for lower N_c values the number of problems describing the extra load is very sensitive, e.g. the balancer may overshoot/undershoot the target load condition with a greater margin by sending one cell more/less with lower N_c values. Once the N_c value is large enough so that the communication overhead is insignificant and the number of problems that are being transferred between processes are large enough to create the balancing statistics accurately, the χ_{su} is stabilized.

One common characteristic observed with both configurations presented in Fig. 4 is the change in balancing performance with respect to the level of imbalance in the benchmark. While the increase in χ_{su} with increased imbalance is already discussed, it is also important to note that $\chi_{su} \approx 1$ for the very balanced case (C_4). The χ_{su} values close to 1 indicate that the operations that create the load balancing statistics and determine the inter-processor communications are not resulting in significant overhead compared to ODE solution. Since the case is already perfectly balanced, the balancer does not communicate any data between the processes. A more detailed analysis on the computational overhead associated with the dynamic load balancing algorithm is presented in Appendix B.

In summary, this benchmarking analysis shows that the balancing algorithm described in Section 2.2 shows very good performance and scalability for a range of process counts and the number of chemistry problems per process. The selected values for these two parameters are typical for reacting CFD simulations, ranging from small simulations that can be run on personal computers to very large simulations that require high-performance computing clusters.

4. Results

Following the performance benchmark of the load balancing model in the previous section, here we demonstrate the model performance in actual reactive CFD simulations. First, a 2D reactive shear layer case with a uniform mesh is simulated to show the effects of load balancing, reference mapping model, number of processes, and decomposition methods. Furthermore, a 3D reactive spray LES simulation is performed to indicate a potential speed-up that can be obtained in large-scale reactive simulations. We do not present physics-based case validation, but aim to show the performance gain available with a load balancing algorithm.

4.1. Reactive shear layer

A schematic describing the 2D reactive shear layer configuration is presented in Fig. 5. A square domain (8 mm × 8 mm) with cyclic boundary conditions is discretized by a structured mesh with 400 × 400 resolution. A hyperbolic tangent function is utilized to generate a smooth shear layer of $L/10$ width in the domain. The momentum shear layer is characterized by a relative velocity difference of ΔU_x = 40 m/s. The shear layer instability is initiated by introducing a sinusoidal perturbation to the vertical velocity component U_y.

The fuel, n-dodecane (n-$C_{12}H_{26}$), is placed in the middle while air (0.77 N_2 + 0.23 O_2) is set elsewhere. The fuel and air temperatures are set to T_{fuel} = 400 K and T_{amb} = 900 K, respectively. Constant pressure of p = 60 bar is initialized in the domain. The chemical kinetics is modeled by the skeletal mechanism developed by Yao et al. [28], including 54 species and 269 reactions. The relative and absolute tolerances of the ODE solver tolerances are set to $1e^{-5}$ and $1e^{-8}$, respectively. The reference cell would be chosen at an arbitrary point outside the shear layer with Z_{tol} = $1e^{-4}$ and T_{tol} = 1 K. All cases are simulated over 1,000 CFD time steps with a maximum Courant number of 0.5.

The reactivity (and the computational load) of the case is higher within the shear layer between fuel and oxidizer across X direction. Therefore, this case can be decomposed using different approaches to have high or low load imbalance. Different decomposition approaches tested here are illustrated in Fig. 6. While a simple decomposition in X direction (left) would ensure minimal imbalance since the decomposition is orthogonal to the shear layer, the same decomposition in Y direction (middle) would result in higher imbalance since now only the processes assigned to/around the shear layer will have high load, creating a bottleneck. Finally, the Scotch decomposition (right) is also investigated since in many CFD applications automated and optimized decomposition methods are used to ensure uniform cell counts and minimize the process boundaries. In addition, since the majority of the cells in the configuration are initially pure oxidizer (and identical), the reference mapping method can be used to significantly reduce the computational load in the processes assigned to the oxidizer part,

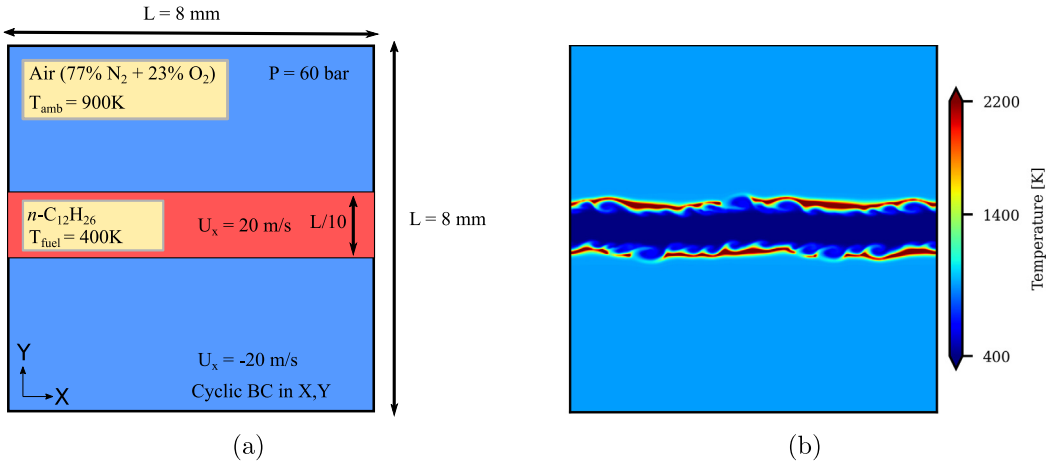

Fig. 5. a) Schematic of the test case and b) an instantaneous representation of the temperature field during the simulation.

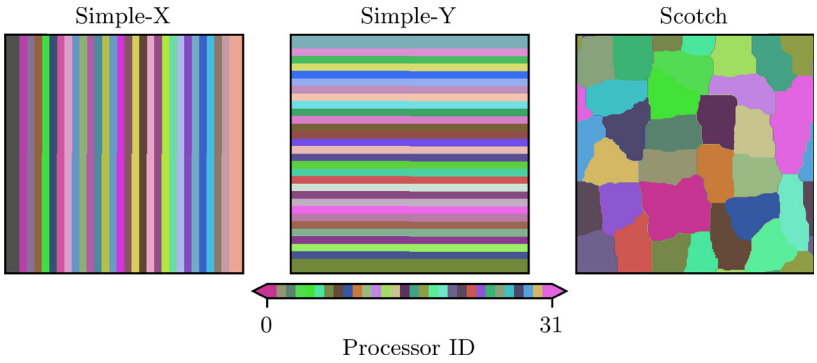

Fig. 6. Decomposition of the test case to 32 processes using 1) Simple decomposition in X direction (left), 2) Simple decomposition in Y direction (middle) and 3) Scotch decomposition (right). While Simple-X attempts to reduce load imbalance, Simple-Y promotes it. Scotch decomposition by design attempts to minimize the number of process boundaries. Data, plotting scripts and figure are available under the CC-BY license [9].

Table 1
The difference decomposition methods, number of processors, and balancing models investigated in reactive shear layer simulations.

Decomposition	Simple-X, Simple-Y, Scotch
Number of processors	32, 64, 128, 256
Model	Standard, Load Balanced, Load Balanced + Reference Mapping

increasing the balancing performance. Therefore, the reactive shear layer is a good test case to investigate: i) the effect of load imbalance on χ_{su} and ii) the additional performance that can be gained from using the reference mapping method together with the load balancer. The details of the parameters investigated in this case (decomposition, number of processors and balancing model) are given in Table 1.

The results obtained from the 3 different decomposition methods for N_p = 32 are presented in Fig. 7. It can be seen that the total execution times are significantly reduced when the load balancing is utilized. While the domain decomposition plays an important role on the performance of the standard case with no balancing, for load balanced models the execution time is not dependent on the decomposition strategy. For instance, while the Simple-Y decomposition has the highest execution time when using the standard model due to load imbalance caused by the decomposition, the execution times of load balanced models are similar to other decomposition methods. Finally, it is important to mention that while the load balancer alone already provides a substantial performance increase, utilizing the reference mapping model further increases the performance of the balancer by reducing the mean

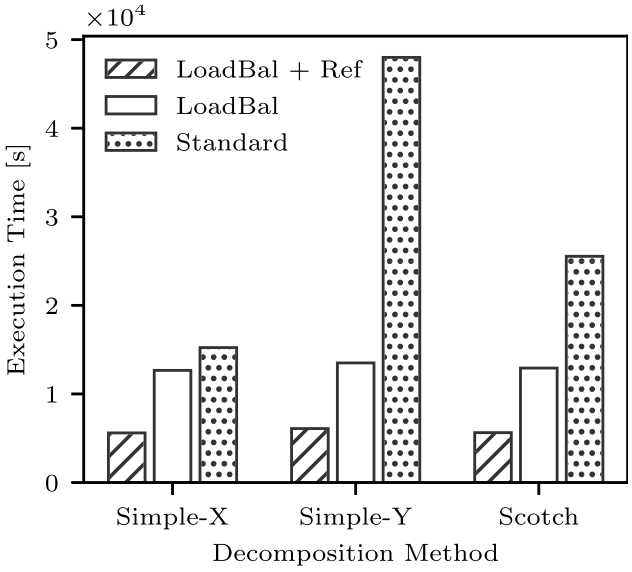

Fig. 7. A bar chart showing the execution times for Simple-X, Simple-Y and Scotch decomposition methods for N_p = 32. Increase in speed-up with increased imbalance can be observed particularly from Simple-Y decomposition. Data, plotting scripts and figure are available under the CC-BY license [9].

load of the simulation and creating further imbalance to increase the balancing performance.

To further demonstrate the effect of load balancer on reactive simulations, the load, i.e. CPU time for solving the chemistry (τ^k_{cpu})

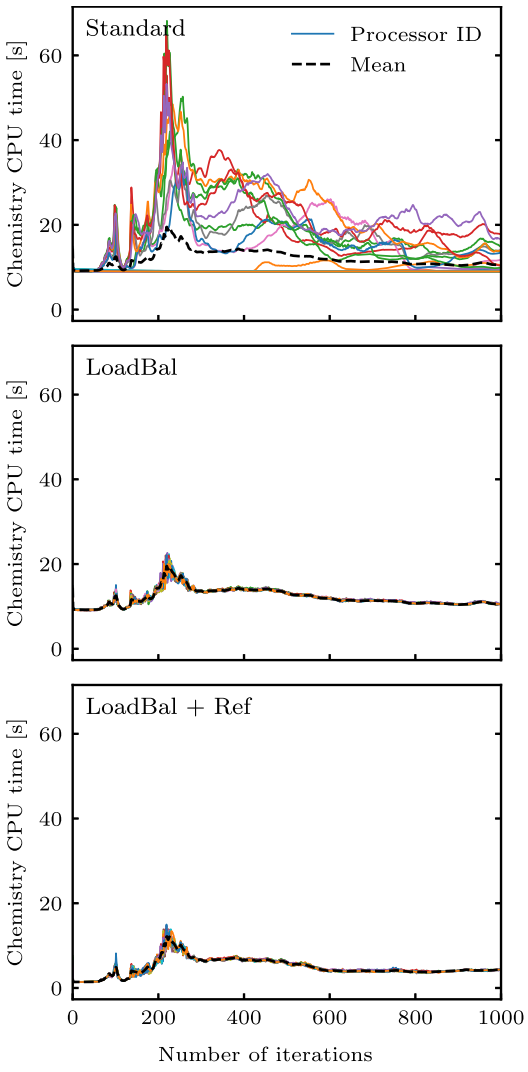

Fig. 8. Chemistry solution CPU time for each process and the corresponding arithmetic mean in each configuration for the reactive shear layer problem. It is noted that the deviation from the mean value is higher for the standard model, which causes computational performance issues. Data, plotting scripts and figure are available under the CC-BY license [9].

on each process along with its arithmetic mean across processes is presented in Fig. 8 for the case with N_p = 32 and Scotch decomposition. It is important to note that the profiles given here only show the time spent on solving the chemistry and do not include the time spent on balancing, which is not significant compared to τ^k_{cpu} as shown in Section 3. It can be seen that for the non-balanced case (bottom), the deviation of τ^k_{cpu} with respect to its mean value is very high. In this scenario, the most computationally loaded process becomes the bottleneck, while the other processes are waiting for that process to finish the chemistry computation. For load balanced cases (middle and bottom), there is a very small deviation in τ^k_{cpu} between processes with respect to the arithmetic mean, which eliminates the bottleneck caused by the unbalanced chemistry load and reduces the simulation time.

Finally, the χ_{su} for all cases described in Table 1 with respect to the standard model are reported in Fig. 9. For the load balanced model, the χ_{su} is small for Simple-X decomposition due to already balanced load distribution. However, for the other 2 decomposition methods a χ_{su} around 2 to 5 can be observed. When the reference mapping is also utilized, we observe that χ_{su} increases even further. While for Scotch decomposition $\chi_{su} \approx 4$ to 6 is achieved, for Simple-Y decomposition we can observe $\chi_{su} \approx 8$ to 12.

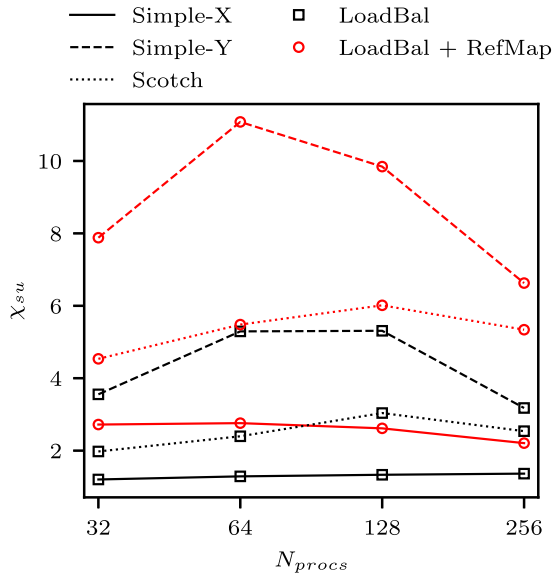

Fig. 9. Speedup factor for all the cases described in Table 1. We note that the highest speedup is achieved with Simple-Y decomposition with reference mapping utilized, due to imbalance created by decomposition and balancing gain obtained by reference mapping, respectively. Data, plotting scripts and figure are available under the CC-BY license [9].

4.2. Three dimensional reacting diesel spray

To further demonstrate the performance of the implemented model, a larger and more challenging case is simulated in this subsection. A 3D reactive spray LES with liquid fuel injection, featuring the Engine Combustion Network Spray A condition [29] is utilized. The details of this simulation configuration can be found in our earlier work [22,24]. A summary of the case setup is given in Table 2. A cylindrical constant volume domain with dimensions 108x108 mm and a 1 mm base mesh along with a uniform mesh refinement region of 125 µm enclosing the spray is used. A total of ≈4.5 M cells are decomposed into 256 processes with Scotch decomposition, resulting with ≈17,000 cells per process. A constant CFD time step of $2e^{-7}$ s is utilized. The same chemical mechanism used in the reactive shear layer case was also used here. The reference cell would be chosen as a non-reactive ambient cell with Z_{tol} = $1e^{-4}$ and T_{tol} = 1 K. Due to the poor performance of the standard model and our computational limitations, all simulations are started from a time instance prior to ignition, and simulated for 500 timesteps instead of the full simulation. A schematic demonstrating a portion of the computational domain is presented in Fig. 10.

Table 2
Spray case setup details. See [29] for further details.

	ECN Spray A
Injected fuel	n-$C_{12}H_{26}$
Nominal nozzle diameter, D	90 µm
Injection pressure	150 MPa
Temperature	900 K
O_2 molar fraction	0.15

Fig. 11 shows the total execution times for the 3 investigated cases: standard, load balancing and load balancing with reference mapping. The computational cost of different solver operations is

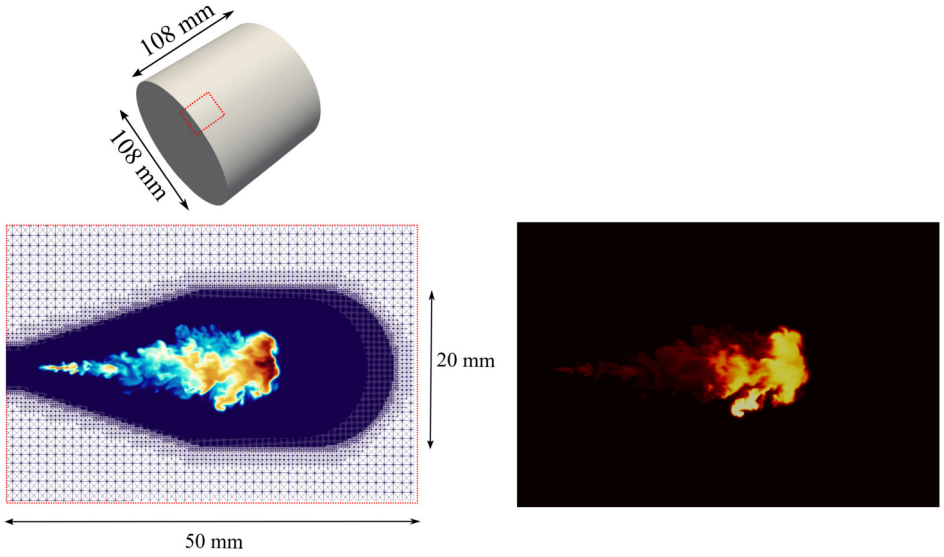

Fig. 10. Schematic of the test case and the computational grid used in spray simulations based on our earlier work [22–24] (left) and an instantaneous representation of the temperature field during the simulation (right). In left figure, the mixture fraction (Z) field is used for visualization, which gives information about the fuel mass fraction and spray vapor.

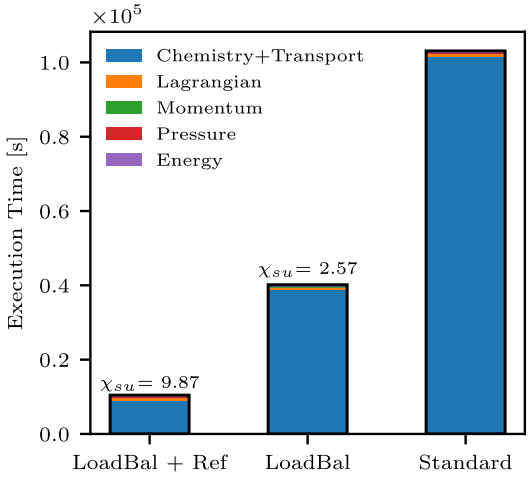

Fig. 11. Speedup for different models. A speed-up by a factor of 2.57 and 9.87 obtained with load balancing and load balancing + reference mapping, respectively. Data, plotting scripts and figure are available under the CC-BY license [9].

also timed and presented in the figure. It can be observed that the computational cost of flow equations (momentum, pressure, energy, and lagrangian) takes up a very small fraction of the total execution time. In contrast, the chemistry and species transport equations take up about 90 to 98 % of the total execution time. A speedup χ_{su} = 2.57 is obtained compared to the standard model when load balancing is utilized. When the reference mapping is also utilized, the χ_{su} increases to 9.87. The gain obtained by using the reference mapping model is higher in this case compared to the reactive shear layer case presented in Section 4.1 due to the transient nature of the spray injection. The chemistry is solved within the spray region which takes up a small percentage of the total cells at the early stages of the simulation. Utilizing the reference mapping provides further speedup due to the large number of idle processes outside of the spray region.

5. Conclusions and future work

A dynamic load balancing and a reference mapping model named `DLBFoam` to speed up the parallel reactive CFD simulations in OpenFOAM is presented and investigated. Dynamic load balancing utilizes MPI routines to send chemistry problems from processes with higher computational load to ones with less load, to avoid computational bottlenecks in parallel simulations. In addition, the reference mapping acts as a filter, where the chemical source terms of the cells satisfying a given criteria are copied from a reference solution, instead of direct integration.

A thorough performance analysis of the model has been presented, showing the balancing efficiency to scale up to a high number of processes and number of problems per process. Furthermore, the model has been utilized on two reactive cases: a 2D reactive shear layer and a 3D diesel spray combustion case. Significant speed-up up to a factor of 10 to 12 has been achieved and demonstrated when load balancing and reference mapping models are used together. It is worth noting that the reported speed-up results are for the particular cases investigated, as well as the ODE solver tolerances chosen and the chemical mechanism used. Changing ODE convergence tolerances and using larger/smaller chemical mechanisms may result in less or more favorable results in terms of load balancing performance.

In addition to the improvements we have introduced in this study, there are still certain aspects that can be modified to further increase the performance of the load balancing model and extend its compatibility. Two potential improvements are following:

- *Improving performance and accuracy of the ODE solvers*: Our experience on state-of-the-art ODE solution algorithms including analytical Jacobian [6] and problem-type tailored linear algebra [11,22] is noted to significantly decrease ODE integrator iterations, enhance its robustness and even further increase the load-balancing efficiency.
- *Extension to in-situ tabulation and reduction models*: Presently, the implementation is not compatible with in-situ adaptive reduction and tabulation model TDAC available in OpenFOAM. Extending the load balancer to these models will both increase the load balancing performance and mitigate the performance issues still present in TDAC due to its processor-based formulation.

Including these features to our open-source library will be investigated in the near future.

CRediT authorship contribution statement

Bulut Tekgül: Conceptualization, Methodology, Software, Formal analysis, Writing – original draft, Visualization. **Petteri Peltonen:** Conceptualization, Methodology, Software, Formal analysis, Writing – original draft, Visualization. **Heikki Kahila:** Conceptualization, Methodology, Software, Writing – original draft. **Ossi Kaario:** Supervision, Writing – review & editing. **Ville Vuorinen:** Supervision, Funding acquisition, Writing – review & editing.

Declaration of competing interest

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

Acknowledgements

The present study has been financially supported by the Academy of Finland (grant number 318024). The computational resources for this study were provided by CSC - Finnish IT Center for Science. The first author has been financially supported by the Merenkulun Säätiö.

Appendix A. Reference mapping model validation

The purpose of this appendix is to quantify the error introduced by the reference mapping model. The error introduced by using the reference mapping model is illustrated in Fig. A.1. The figure shows the mean absolute percentage error of the volume integral of the heat release rate over all time instances relative to the results obtained with the standard chemistry model in OpenFOAM. The results are obtained from the reactive shear layer case in Section 4.1 with different process counts. Z_{tol} = 1e^{-2}, 1e^{-4}, 1e^{-6} and T_{tol} = 1 K values are used as model tolerance. 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, initially a large deviation around 10% is observed for Z_{tol} = 1e^{-2}. With tighter tolerances, we observe this error to drop below 2% and converge between Z_{tol} = 1e^{-4} to 1e^{-6}. This observation is consistent with our earlier studies, in which a $Z_{tol} = 1e^{-4}$ value is successfully used to

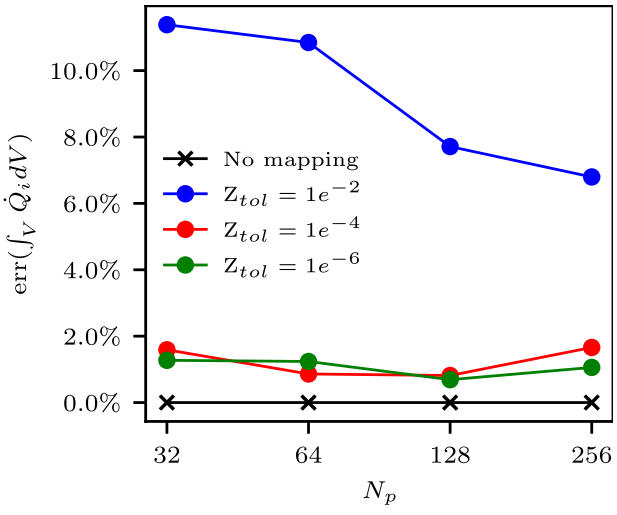

Fig. A.1. The absolute percentage error of volume integrated heat release rate averaged over all time steps for different process counts. Data, plotting scripts and figure are available under the CC-BY license [9].

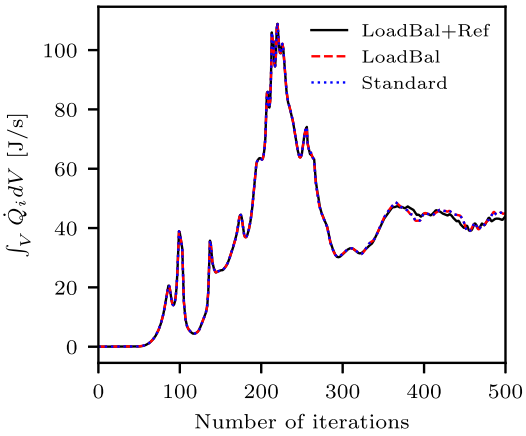

Fig. A.2. Volume integrated heat release rate for N_p = 64, using load balanced + reference mapped (with Z_{tol} = 1e^{-4}), load balanced and standard models. Data, plotting scripts and figure are available under the CC-BY license [9].

model various spray combustion phenomenon [22–24]. The deviation from the standard model depends on the process count due to the process-local nature of the reference mapping. A video animation demonstrating how the reference mapping operates during the simulation is provided in the supplementary materials of the paper.

The evolution of the volume integrated heat release rate is given in Fig. A.2. As seen, the reference mapping shows a very good match with the non-mapped models for the presented simulation data. It is important to note that the error due to the reference mapping may accumulate over time depending on the model tolerance values used. Although the performance gain when using the reference mapping is significant compared to the introduced error, for applications that require very high accuracy, a careful model tolerance analysis is recommended.

Appendix B. Computational overhead

This appendix quantifies the computational overhead introduced by the load balancing algorithm. As described in Section 2.2, the dynamic load balancing algorithm consists of 4 different stages: 1) preprocessing stage, 2) balancing stage, 3) solution stage, and 4) update stage. Among those, stages 1,2 and 4 can be considered in computational overhead context. Although we have noted the overhead to be insignificant compared to the solution time, here we quantify the overhead for the three dimensional reacting diesel spray benchmark case we utilized at the end of the paper.

Fig. B.1 shows the CPU time spent on solving the chemistry and the percentage of computational overhead with respect to the chemistry CPU time for 100 CFD iterations. Following the case setup described in Section 4.2, the domain is decomposed into 256 processors. It should be noted that the idling time associated with the explicit barrier at the end of the stage 4 is not counted towards computational overhead. It can be seen that the computational overhead associated with load balancing is less than 1% of the total chemistry solution CPU time. While the overhead may increase by increasing the cells per process or the size of the utilized chemical mechanism, we note that it is insignificant for cases and mechanisms of relevant size to reactive CFD applications. We note that Fig. B.1 shows outlier processors, i.e., processors with slightly higher load than the mean value after balancing. Based on our experience, this behavior is due to 1) transient nature of the process-based load value due to local chemistry stiffness, and 2) possible hardware and parallel communication issues that are briefly mentioned in Section 3. These aspects exist in any reactive

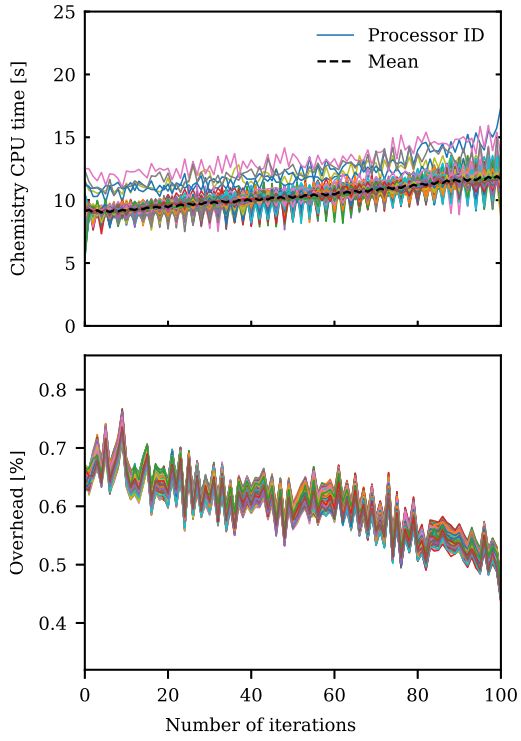

Fig. B.1. Chemistry solution CPU time and corresponding computational overhead percentage related to dynamic load balancing for each process. The solid colored lines represent the CPU time of each processor, while the black dashed line represents their arithmetic mean. The computational overhead accounts for less than 1% of the chemistry solution time. Data, plotting scripts and figure are available under the CC-BY license [9].

CFD simulation running on any hardware architecture, and are not caused by the balancing algorithm.

Appendix C. Supplementary material

All the analysis in this paper is performed using the `DLBFoam`. `DLBFoam` is open-source and publicly available at https://github.com/blttkgl/DLBFoam. The repository contains instructions for compilation and tutorials. The simulation data, plotting scripts, and the figures featured in this paper are available as supplementary material under the CC-BY license [9].

References

[1] T. Poinsot, D. Veynante, Theoretical and Numerical Combustion, Edwards, 2001.
[2] N. Peters, Turbulent Combustion, No. 11, Cambridge University Press, Cambridge, 2000.
[3] J. Muela, R. Borrell, J. Ventosa-Molina, L. Jofre, O. Lehmkuhl, C.D. Pérez-Segarra, Comput. Fluids 190 (2019) 308–321, https://doi.org/10.1016/j.compfluid.2019.06.018.
[4] C.K. Law, Combustion Physics (Google eBook), Cambridge University Press, Cambridge, 2006.
[5] R.L. Speth, W.H. Green, S. MacNamara, G. Strang, SIAM J. Numer. Anal. 51 (6) (2013) 3084–3105, https://doi.org/10.1137/120878641.
[6] K.E. Niemeyer, N.J. Curtis, C.J. Sung, Comput. Phys. Commun. 215 (2017) 188–203, https://doi.org/10.1016/j.cpc.2017.02.004.
[7] E. Hairer, G. Wanner, Solving Ordinary Differential Equations. II, Springer-Verlag, 1996.
[8] C.P. Stone, A.T. Alferman, K.E. Niemeyer, Comput. Phys. Commun. 226 (2018) 18–29, https://doi.org/10.1016/j.cpc.2018.01.015.
[9] B. Tekgül, P. Peltonen, H. Kahila, O. Kaario, V. Vuorinen, Supplementary material for "DLBFoam: an open-source dynamic load balancing model for fast reacting flow simulations in OpenFOAM", https://doi.org/10.6084/m9.figshare.14143931, Mar 2021.
[10] N. Curtis, Accelerating Reactive-Flow Simulations via Vectorized Chemical Kinetic Evaluation, Doctoral Dissertations, Jan 2019.
[11] A. Imren, D.C. Haworth, Combust. Flame 174 (2016) 1–15, https://doi.org/10.1016/j.combustflame.2016.09.018.
[12] S. Pope, Combust. Theory Model. 1 (1) (1997) 41–63, https://doi.org/10.1080/713665229.
[13] Z. Ren, Y. Liu, T. Lu, L. Lu, O.O. Oluwole, G.M. Goldin, Combust. Flame 161 (1) (2014) 127–137, https://doi.org/10.1016/J.COMBUSTFLAME.2013.08.018.
[14] T. Lu, C.K. Law, C.S. Yoo, J.H. Chen, Combust. Flame 156 (8) (2009) 1542–1551, https://doi.org/10.1016/j.combustflame.2009.02.013.
[15] F. Contino, H. Jeanmart, T. Lucchini, G. D'Errico, Proc. Combust. Inst. 33 (2) (2011) 3057–3064, https://doi.org/10.1016/j.proci.2010.08.002.
[16] H. Turkeri, X. Zhao, S.B. Pope, M. Muradoglu, Combust. Flame 199 (2019) 24–45, https://doi.org/10.1016/j.combustflame.2018.10.018.
[17] A. Cuoci, A. Frassoldati, T. Faravelli, E. Ranzi, Energy Fuels 27 (12) (2013) 7730–7753, https://doi.org/10.1021/ef4016334.
[18] L. Antonelli, P. D'Ambra, in: Proceedings - 19th International Euromicro Conference on Parallel, Distributed, and Network-Based Processing, PDP 2011, 2011, pp. 133–140.
[19] J. Kodavasal, K. Harms, P. Srivastava, S. Som, S. Quan, K. Richards, M. García, J. Energy Resour. Technol. 138 (5) (2016) 052203, https://doi.org/10.1115/1.4032623.
[20] Y. Shi, W.H. Green, H.W. Wong, O.O. Oluwole, Combust. Flame 159 (7) (2012) 2388–2397, https://doi.org/10.1016/j.combustflame.2012.02.016.
[21] H.G. Weller, G. Tabor, H. Jasak, C. Fureby, Comput. Phys. 12 (6) (1998) 620, https://doi.org/10.1063/1.168744.
[22] H. Kahila, A. Wehrfritz, O. Kaario, V. Vuorinen, Combust. Flame 199 (2019) 131–151, https://doi.org/10.1016/j.combustflame.2018.10.014.
[23] H. Kahila, O. Kaario, Z. Ahmad, M. Ghaderi Masouleh, B. Tekgül, M. Larmi, V. Vuorinen, Combust. Flame 206 (2019) 506–521, https://doi.org/10.1016/J.COMBUSTFLAME.2019.05.025.
[24] B. Tekgul, H. Kahila, O. Kaario, V. Vuorinen, Combust. Flame 215 (2020) 51–65, https://doi.org/10.1016/j.combustflame.2020.01.017.
[25] M. Raju, M. Wang, M. Dai, W. Piggott, D. Flowers, in: SAE 2012 World Congress & Exhibition, SAE International, 2012.
[26] R. Bilger, S. Stårner, R. Kee, Combust. Flame 80 (2) (1990) 135–149, https://doi.org/10.1016/0010-2180(90)90122-8.
[27] Mahti supercomputer technical specifications https://research.csc.fi/techspecs. (Accessed 10 September 2020).
[28] T. Yao, Y. Pei, B.J. Zhong, S. Som, T. Lu, K.H. Luo, Fuel 191 (2017) 339–349, https://doi.org/10.1016/j.fuel.2016.11.083.
[29] Engine combustion network, combustion research facility, Sandia National Laboratories, Livermore, CA, https://ecn.sandia.gov.