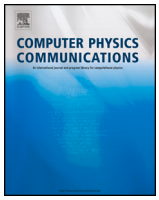
Computer Physics Communications 305 (2024) 109322

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc



Computational Physics

A chemistry load balancing model for OpenFOAM ☆

Jan Wilhelm Gärtner *, Ali Shamooni, Thorsten Zirwes, Andreas Kronenburg

Institute for Combustion Technology, University of Stuttgart, Pfaffenwaldring 31, 70569 Stuttgart, Germany

ARTICLE INFO

Dataset link: https://github.com/ITV-Stuttgart/loadBalancedChemistryModel

Keywords:
OpenFOAM
Load balancing
Combustion
TDAC
Finite-rate chemistry

ABSTRACT

Efficient simulation tools are crucial for studying complex systems such as reacting flows where computational costs of computing chemical reaction rates can vastly exceed the costs for the integration of the convective and diffusive transport terms. Load imbalance in parallel computing poses a significant challenge for massively parallel reacting flow simulations. In response, a novel load balancing library has been developed to enhance OpenFOAM's solver performance in parallel environments. This library seamlessly integrates with OpenFOAM, offering ease of use and applicability to any OpenFOAM reacting solver incorporating finite-rate chemistry. In addition, it supports the standard and the dynamic adaptive chemistry model (TDAC) of OpenFOAM.

The newly developed load-balanced standard and TDAC models address significant load imbalances by exchanging information between processes via MPI calls and tracking ODE solution times on a cell level. The TDAC model introduces dual tables on each core and enables immediate addition of computed solutions, enhancing computational efficiency. Validation on various test cases, including simulations on the HLRS Hawk supercomputer with up to 8000 cores, confirms identical results compared to the original unbalanced models, with notable speed-up factors of up to 6 for the standard and 5 for the TDAC model. Despite non-linear scaling at lower cell count per processor, load-balanced models consistently outperform unbalanced counterparts, making them the preferred choice for reacting flow simulations in OpenFOAM.

1. Introduction

Efficient Computational Fluid Dynamics (CFD) simulation tools are essential for studying complex systems such as reacting flows in application of engineering interest. High-fidelity numerical simulations of reactive systems usually involve detailed chemical kinetics with hundreds of chemical species and thousands of elementary reactions, and numerical integration of reaction rates is computationally very demanding. This results in the computational cost of chemistry-related computations often being magnitudes larger than that of solving the transport equations. While modern CFD codes utilize parallel computation quite efficiently, the commonly used approach of domain decomposition leads to a high load imbalance in reacting solvers [28]. This is mostly due to the typically rather confined locations of a flame where reaction occurs. This leads to large inhomogeneities in terms of requirements on the chemistry solver, as outside of the flame equilibrium conditions or non-flammable mixtures prevail and computing times will be negligible.

One effective strategy to alleviate load imbalances involves custom domain decomposition, particularly when the spatial dynamics of chemical kinetics are ascertainable [28]. However, this information is rarely available or, in the cases of an advancing flame front, a pulsating flame or an ignition study, not applicable. Consequently, load balancing techniques leveraging Message Passing Interface (MPI) to efficiently redistribute cell level computations from heavily burdened cores to those with lighter computational loads have been proposed [30,28,23,17]. Although these approaches demonstrate notable acceleration, the available open-source implementations either lack versatility due to tight integration with the bespoke solvers or remain untested in massively parallel simulations spanning thousands of cores.

Addressing these challenges, we present a novel load balancing library designed to improve the performance of OpenFOAM chemistry models within parallel computing environments. This library offers a robust solution that can seamlessly integrate with the default structure of OpenFOAM, providing efficient means to mitigate computational load imbalances. Key advantages of this library are its ease of use and general applicability for any OpenFOAM solver which requires chemical kinetics integration. Moreover, it extends support not only to the standard chem-

☆ The review of this paper was arranged by Prof. Andrew Hazel.

* Corresponding author.
E-mail address: jan-wilhelm.gaertner@itv.uni-stuttgart.de (J.W. Gärtner).

https://doi.org/10.1016/j.cpc.2024.109322
Received 9 April 2024; Received in revised form 3 July 2024; Accepted 15 July 2024
Available online 18 July 2024

istry model of OpenFOAM but also to advanced methodologies such as *in-situ* adaptive tabulation and dynamic adaptive chemistry (TDAC) [7].

To validate the efficacy of the novel load-balanced standard and TDAC chemistry models, we conduct simulations across three distinct test cases encompassing a spectrum of scenarios ranging from small-scale cases with extensive reaction mechanisms on a single node to large-scale simulations featuring small and large reaction mechanisms. In addition, the strong scaling behavior of the two models is investigated on the super-computer Hawk at the HLRS in Stuttgart using up to 64 nodes and 8192 cores.

The remainder of this paper is organized as follows: Section 2 provides a comprehensive exposition of the load-balanced standard and TDAC algorithms within OpenFOAM and their implementation details. Subsequently, Sec. 3 presents the validation and scaling analyses of the chemistry models across the three test cases.

2. Load balancing algorithm

Reactive simulations with finite-rate chemistry integration often suffer from high load imbalance due to varying computational demands across computational cells, driven by factors such as temperature, pressure, and species composition. Unlike solving the transport and conservation equations, which exhibit uniform computational loads per cell and scale linearly with the cell count, solving ordinary differential equations (ODEs) for reactive simulations presents disparities. To achieve load balancing for reactive simulations, the computational time required for each cell must be known. Once this information is available, cells can be distributed among the MPI processes to ensure equal computational loads. While various methods exist for load balancing, such as mesh decomposition adjustments and cell redistribution, these can significantly impact solving the primary partial differential transport equations (PDEs) and exacerbate computational overhead. Thus, we adopt a lightweight approach: selectively transmitting essential cell information for ODE integration to other processes and receiving computed results.

OpenFOAM provides two distinct methods to solve the chemistry reaction rates. The first model is called the standard model, solving the ODEs sequentially for each cell. In addition, OpenFOAM provides a second model which uses a combined tabulation and reduction algorithm with *in-situ* adaptive tabulation and dynamic adaptive chemistry, the so-called TDAC model [7–10]. Note that OpenFOAM uses the syntax of processors to refer to the MPI processes of a simulation. However, in this paper we use the term “process” to refer to a single running entity of a program, corresponding to a single MPI rank.

2.1. Standard chemistry model

In the standard chemistry model of OpenFOAM, ODEs are solved sequentially in each cell. The adaptive ODE solvers tend to use sub-steps for the solution procedure, and the number of sub-steps is dependent on the local stiffness of the ODE system. This causes a high computational load in regions with high reactivity, such as flame fronts. These regions, however, typically comprise a low fraction of the total simulation domain. Therefore, a high load imbalance is generated at each time step/iteration over the course of the simulation. Here, we propose a load-balanced algorithm comprising the following steps:

1. Determine the load imbalance of the processes and calculate the percentage of CPU time to be sent to *one* or *multiple* processes to achieve an equal load, see Fig. 1.
2. Determine the cell lists based on the individual cell CPU time to send to or receive from other processes.
3. Send and receive the cell information required to solve the ODEs
4. Solve the ODEs for all local and received cells.
5. Send the computed results back to the original processes.

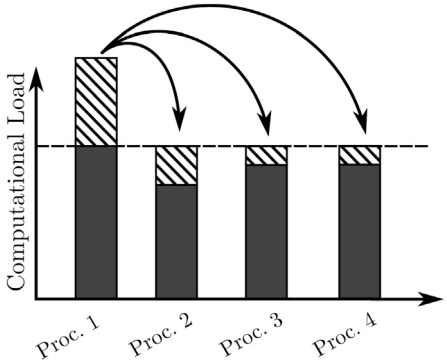


Fig. 1. Sketch of load balancing to solve chemical reactions for different processes and their overheads. Here, the load of the first process is distributed to the other three to achieve an equal load for all four processes.

6. Update the total computational load on each process as the sum of all computational cells (including the ones sent to other processes).

This approach, which was proposed in Shamooni et al. [26] is straightforward, and similar to the one employed in the published DLBFoam solver [28].

One critical challenge of this approach lies in the inter-processor MPI communication. In the load-balanced standard chemistry model, MPI communication occurs in three instances, see Fig. 2. The first occurrence is when the load imbalance is calculated and all processes need to know about the computational load on all others, necessitating an all-to-all communication. The two other MPI communications are required for sending/receiving the cell list for computation and the subsequent reversed “send and receive” once they have been computed. In OpenFOAM, the required exchange of the computational load of all processes can be achieved by built-in *gather* and *scatter* functions. For the exchange of the cell lists between individual processes, the cells to send are streamed to a char array, which is then sent and received using MPI commands. OpenFOAM provides a native solution called `PstreamBuffers` for this task. However, despite the idea of sending the buffers to distinct processes, the `PstreamBuffers` class requires an all-to-all communication to exchange the buffer sizes prior to sending and receiving. This step is required as the `PstreamBuffers` class does not know which processor sends and receives data prior to this step. Although the size exchange process involves merely a few hundred bytes, the all-to-all communication emerges as a bottleneck, particularly once multiple nodes are used. For instance, in scenarios involving approximately 4000 cores, the exchange of buffer sizes alone consumes over 30% of computational time, dwarfing the actual buffer exchange, which accounts for less than 1%.[1]

In the case of the load-balanced chemistry model the connection of sending and receiving processes is known from the computational load calculation. Therefore, we propose a novel point-to-point buffer class which uses this information to circumvent the all-to-all MPI calls and only exchanges the buffers and their sizes with the set send and receive processor IDs.

The importance of this novel process-to-process communication approach is shown in Fig. 3, which depicts the mean CPU time required on all processes to solve the ODEs for varying intervals of updating the computational load balance for a case of premixed combustion with a moving flame front run on 32 nodes (4096 cores). If the computational load is updated in each iteration, the mean and maximum CPU time required to solve the ODEs are much higher than updating the computational load every 50th step. However, for the other extreme, when the computational load is only set once and never updated, the load

[1] MPI performance and load balance measured with ScoreP, see here.

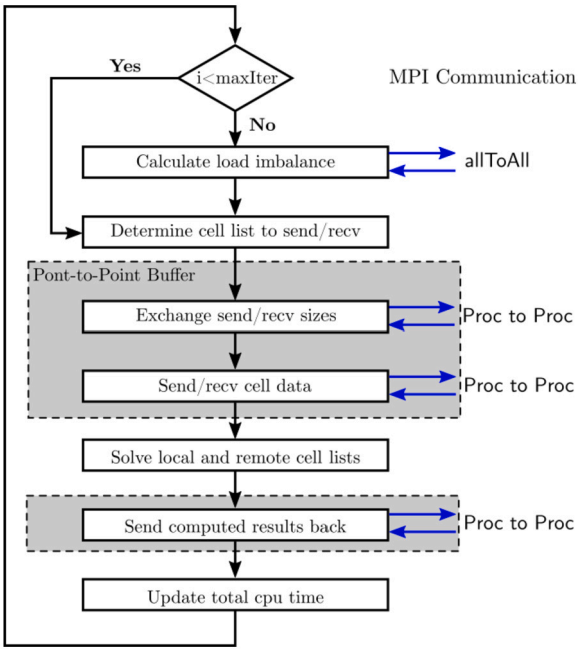


Fig. 2. Flow chart of the load balanced chemistry model for the standard chemistry model of OpenFOAM, using the novel point-to-point buffer.

imbalance increases over time as the flame front moves from one processor to the next, requiring an update of the load balancing. At the time of writing, the proposed library allows the user to set the update frequency, which needs to be adjusted to the given case. However, future work shall include an automatic update frequency detection based on the current load imbalance.

2.2. TDAC chemistry model

The *in-situ* adaptive tabulation (ISAT) [25] is a storage-retrieval algorithm which stores the ODE solutions and the data needed to retrieve them in memory and reuses them during the simulation. When combined with on-the-fly kinetics reduction algorithms such as dynamic adaptive chemistry (DAC) [21], the computational cost of ODE integrations in reactive solvers can be significantly reduced [7–10]. Since the previously computed results can be retrieved from the table and do not need to be recomputed, the computational load for the chemistry solution is significantly reduced. However, it can also introduce significant load imbalance, as solutions within the reaction zone typically need to be recomputed whereas results outside the reaction zone can be retrieved from the table. A distribution of the computational load can be split up into retrieving results from the table, solving the ODE, and storing in the table. This is displayed in Fig. 4 for the jet in hot co-flow DLR burner test case of the exaFOAM benchmark set with 11 million cells run on 64 cores.

Attempts have been made by designing parallel ISAT strategies to circumvent the above described issue [22,15,16]. These include random and uniform distribution of ODE solution tasks among all the processes (URAN) [22], the URAN distribution but among user-defined process groups/partitions (P-URAN) [15,16], and preferential distribution (PREF) where tasks are preferentially distributed among processes that have already seen/not seen the data during the previous/current step [22]. Apart from the fact that these strategies are not available in OpenFOAM, it has been already shown in a relatively large SandiaD flame simulation with particle methods using a 16 species chemical mechanism on 1024 cores that URAN and PREF are not efficient enough, mainly due to the required high communication times [15]. Further, while the P-URAN strategy attempts to reduce the communication times by restricting the communications among process groups, an imbalance is generated among these partitions. Moreover, the partitioning requires prior knowledge of the load distribution, e.g., where the flame is located, which is not applicable in scenarios where the load has an unsteady nature e.g. in flame propagation simulations where the flame front enters/exits process boundaries during the simulation period. Here, we propose an alternative distribution strategy to avoid costly all-to-all communications which is suitable for both stationary and transient load scenarios by adapting the algorithm explained in the previous section to the TDAC environment.

First, the load-balanced TDAC model checks if the solution for the local cells can be retrieved from the table. Cells for which the solution could not be found in the table are then flagged for computation. The computational load imbalance is computed, and the amount of CPU time to be sent to other processes is set. Following this, the cells are added to the list of cells to send until the CPU time determined in the previous step is reached. However, in contrast to the load-balanced standard model, not only the computational time to solve the ODE is considered, but also the time required to add the new solution to the table. Considering the time to add the solution to the table is important. Depending on the reaction mechanism and table settings, this can be of the order of the time to solve the ODE itself. Now, the cell data is sent to the processes for computation using the novel point-to-point buffer presented in the previous section. The ODEs for the local and remote cells are then solved on each MPI rank. Hereby, the solution of the ODE is added to the table directly after computing a cell. This has the advantage that subsequent cells can potentially use the previously computed solution. However, the load-balanced algorithm uses two tables, one for the local cells and one for cells received from other processes. The reasoning is that the local cells are closer to the solution space, allowing for a more efficient table, whereas the remote cells may be located far away in the solution space. Once all cells are computed, the solution of the remote cells is sent back to the original process. As a final step, the local tables have to be updated with the remotely calculated solution, and the CPU time required to solve and add the cells to the table is updated. This process is also visualized in the flow chart of Fig. 5.

The importance of storing the received cell solution in a table during computation is shown in Fig. 6 depicting the speed-up for the new load-balanced model as presented above for the standard counterflow flame case of the OpenFOAM tutorial. For comparison, the orange line provides the speed-up for cases where the table is not updated by the solution received from the remotely computed chemistry integration. The results show clearly how the performance of the model deteriorates if the received cells are only computed but the solution is not stored in the table of the computing process. For more than 5000 cells per core, the load-balanced TDAC model, without adding the received cell solution to it's own table, performs worse than the unbalanced TDAC model.

2.3. Compilation and usage of the load-balanced chemistry models

A key advantage of the new library is that it is not tied to any specific solver of OpenFOAM so that it can be dynamically selected for any OpenFOAM solver that uses a chemistry model. To use the load-balanced chemistry model, the user has to compile the library with the OpenFOAM compilation framework, called wmake. Once the library has been compiled, it can be selected in the chemistryProperties file by setting the method of the chemistry type to load-balanced. In addition, the user has the option to set the iteration count to update the load balance. Listing 1 gives a typical dictionary entry. The library, unit-tests and instructions for compilation are publicly available at https://github.com/ITV-Stuttgart/loadBalancedChemistryModel. Further, a static version of the code is available at the data repository of the University of Stuttgart (DaRUS) at https://doi.org/10.18419/darus-4121.

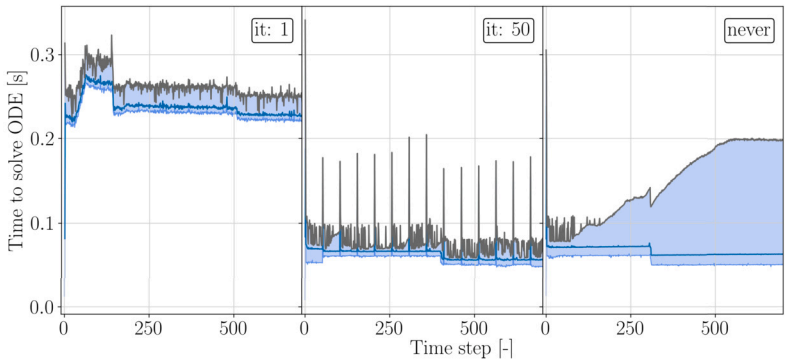


Fig. 3. Mean computational load on all processes for each time step for a case involving a propagating flame front (Case B in Sec. 3) with varying computational load update frequencies. The minimum and maximum time required to solve the reaction sub-step on all processes are marked by the shaded area. The maximum time of all processes is the determining factor for overall computational speed of the solver and is marked by a gray line.

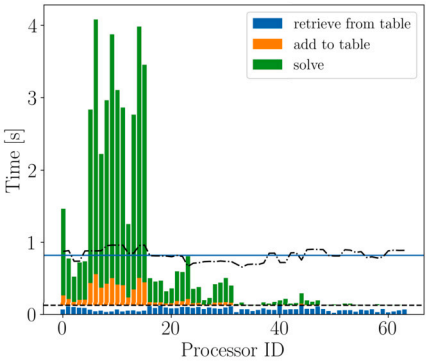


Fig. 4. Load distribution for TDAC computation of the JHC DLR burner case, run on 64 cores. The bars denote the unbalanced time required to retrieve, solve, and store the ODE solution for each process. The black dashed line marks the MPI wait-all required to compute the computational load, see Fig. 5. The black point-dashed line is the total load-balanced time for each process, and the blue solid line marks the mean value of the total time of all processes. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

```
chemistryType
{
    solver            ode;
    method      LoadBalancedStandardChemistryModel;
    //or
    method      LoadBalancedTDAC;
}

LoadBalancedTDACCoeffs
{
    updateIter 10;  // update the load balancing
    // distribution every 10 iterations
}
```

Listing 1: Example chemistryProperties dictionary.

3. Test cases

The new load-balanced standard and TDAC models are applied to three test cases. All test cases involve detailed (but of variable size) finte-rate chemical kinetics which is the main bottleneck in the solution algorithm as described in the previous section. The first test case is the 2D counterflow flame test case from the OpenFOAM tutorial and is examined in Sec. 3.1. This test case features a small mesh and a large chemistry mechanism, where the time needed for chemistry integration accounts for a large portion of the total computational load of each solution step. To simulate large-scale reactive flows, the developed load-balanced algorithms have to scale up to several thousand cores to allow the use of HPC resources. To examine the behavior of the

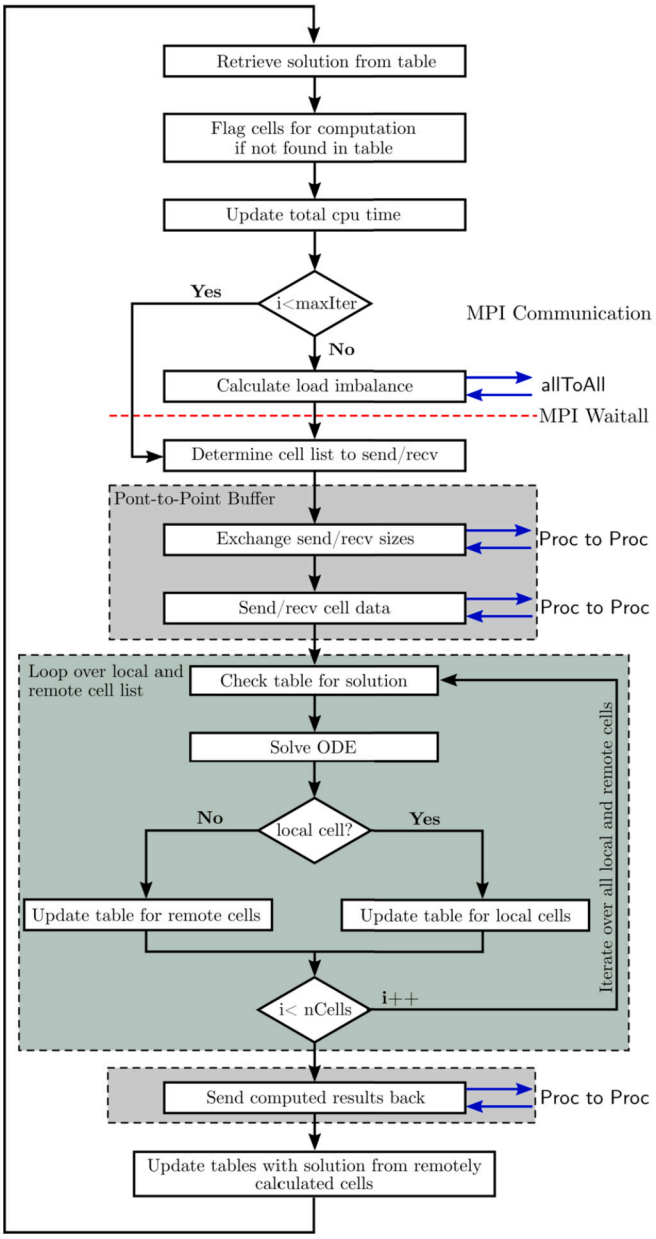


Fig. 5. Flow chart of the standard load-balanced chemistry model and the TDAC load-balanced chemistry model.

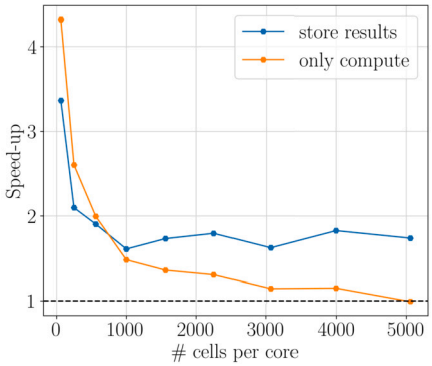


Fig. 6. Comparison of the speed-up of the load-balanced TDAC model for the counterflow flame case of the OpenFOAM tutorial decomposed for 64 cores. The blue line denotes the new load-balanced model and the orange line the load-balanced model with the modification of not adding the received cells to the table. The black dashed line marks a no speed-up compared to the unbalanced TDAC model.

solver under HPC relevant conditions, two further test cases are investigated on the supercomputer Hawk at the HLRS in Stuttgart. The second case examined in Sec. 3.2 is a Large Eddy Simulation (LES) of an H_2-air flame acceleration in an obstructed channel [11]. The test case accounts for a large mesh, small chemical mechanism. The unsteady characteristic of the evolving flame, which causes the load to constantly move among the individual process domains, makes the case a challenging test for any load balancing algorithm. The final test case which will be investigated in Sec. 3.3 is a 3D LES of the DLR jet-in-hot-coflow (JHC) [4,3,12] configuration listed among Grand Challenges in the exaFoam project [1]. The test case constitutes a large mesh and a large chemical mechanism, which is typical for engineering relevant applications. In the following these three cases are investigated and the scaling behavior and speed-up for the load-balanced standard and TDAC models are reported. The scaling behavior always compares the scaling of the chemistry solution only; thus, it does not consider the parts that are unaffected by the load-balancing algorithm, such as solving the transport equations for momentum and energy. Hence, settings such as the ODE solver and its tolerances will not directly affect the load balancing, except for slightly shifting the balance of compute vs. communication times. However, they affect the solver's overall scaling. If the ODE tolerances are set very low, e.g., to 1E-15, the computational effort to solve the chemistry increases; hence, the overall share of the chemistry solution on the total run-time of the solver increases as well. In contrast, if lower accuracies in the ODE integration are acceptable and the tolerances are set very high, e.g., 1E-3, the computational effort decreases, which will lead to a reduction of the chemistry share of the overall runtime of the solver. As only the load of the chemistry can be balanced, the maximum performance increase of a solver is limited by the share of the chemistry load and thus case dependent. Therefore, we kept the official ODE settings for each case. For the counterflow case, the settings are kept the same as those provided by the official OpenFOAM tutorial. The ODE tolerances of the "flame propagation and acceleration" case are the highest tolerances that can be used with the Rosenbrock34 ODE solver, which gives an acceptable agreement with the reference case [11]. The settings for the "DLR JHC burner" case are the ones set in the exaFoam benchmark set [1].

3.1. 2D counterflow flame

3.1.1. The case description

The first test case investigated is the 2D methane-air counterflow flame from OpenFOAM tutorials. It employs the GRI chemical mechanism comprising 53 species and 325 reactions [2]. The computational load predominantly arises from solving the ODEs for the chemical source terms, accounting for over 90% of the computational time. The case

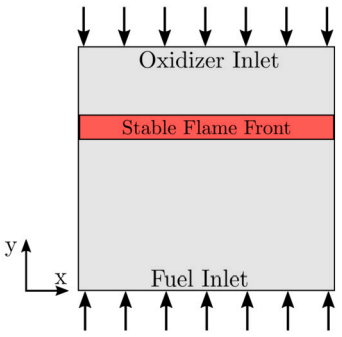


Fig. 7. Sketch of the 2D counterflow flame domain setup.

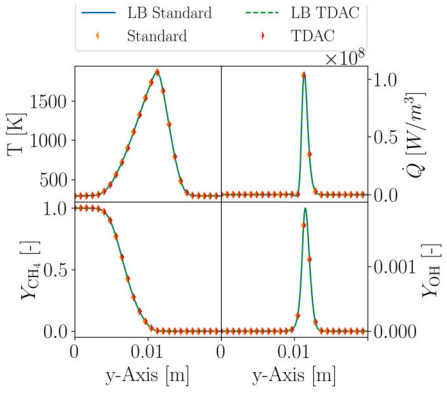


Fig. 8. Line plot along the y-axis of the counterflow flame. To distinguish between the four data sets, the reference solution of the unbalanced methods uses half-filled diamond markers, whereas the load-balanced methods use lines without a marker. Further, note that only every 8th data point is marked with a marker to allow visibility of the other data sets.

setup consists of a 2D slice with opposing oxidizer and fuel inlets with a flow velocity of 0.1 m/s, creating a stable flame front in the center of the domain, see Fig. 7. The simplicity of the 2D setup with no turbulence allows to scale up the original base mesh of 4000 cells to investigate the effect of the number of cells per core on the solution, similar to a weak-scaling test. The standard reactingFoam solver of OpenFOAM is then used with an adaptation to report the total time required to solve the ODEs for each process.

In this section, the load-balanced standard and TDAC chemistry models are tested on a single node equipped with two AMD Epyc 7543 processors, each with 32 cores. All investigated cases are decomposed into 64 domains using the scotch mesh decomposition algorithm. Except for the chemistry model, all other settings are kept as given by the tutorial.

3.1.2. Model validation

To validate the novel load-balanced standard and TDAC models, the temperature, fuel (CH_4) mass fraction, heat release, and OH mass fraction fields are sampled along the y-axis of the domain and compared to the standard OpenFOAM models without load balancing. The results are depicted in Fig. 8 and show no deviation between the different models, proving that the load-balanced models return the correct chemical reaction rates. In addition to this test, the library is equipped with a unit-test framework that checks the calculation of the reaction rate for a single time step and compares it to the standard OpenFOAM models. The advantage of the unit tests is that they can be executed within seconds, allowing frequent execution during the code development.

3.1.3. Standard chemistry model

The performance of the load-balanced standard chemistry model is compared with the default unbalanced standard chemistry model in this

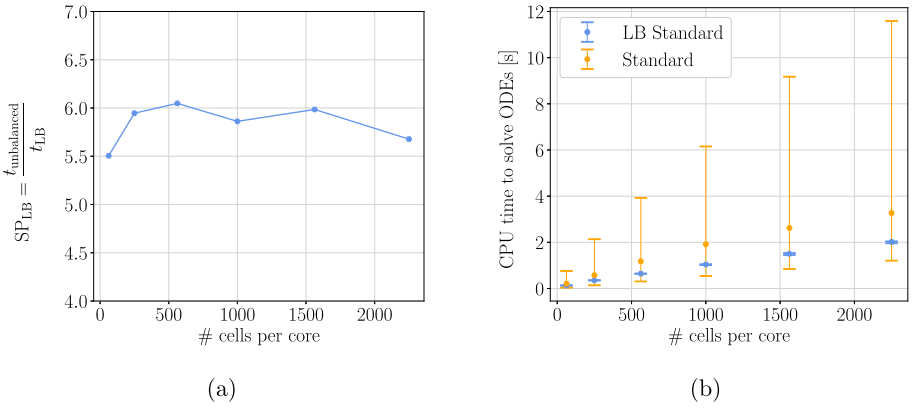


Fig. 9. 2D counterflow case using a single node with 64-cores: (a) speed-up of the load-balanced standard chemistry model. (b) The required CPU time to solve the ODEs in the reaction sub-step per time step for the unbalanced and load-balanced standard chemistry models. The error bars denote the minimum and maximum time required to solve the ODEs over all 64 processes.

sub-section. Note that in the standard chemistry model no tabulation/reduction is performed. An additional comparison to the DLBFoam [28] can be found in Appendix A. Fig. 9a shows the speed-up gained by enabling the load balancing algorithm compared to the unbalanced model versus the number of cells per core. The speed-up by enabling the load balancing algorithm is defined by

$$SP_{LB} = \frac{t_{unbalanced}}{t_{LB}}, \tag{1}$$

where SP_{LB} is the speed-up by enabling the load balancing algorithm, $t_{unbalanced}$ is the average CPU time per time step to solve the chemistry reaction mechanism of the unbalanced model and t_{LB} is the one of the load-balanced model. It can be seen that over the entire investigated range, a speed-up factor above 5.5 is achieved. Note that here and for the following test cases, only the scaling of the chemistry solution sub-step is analyzed. We chose this approach to exclude the effect of solving the PDEs and all auxiliary functions that are highly dependent on the selected solver and the specifics of the setup and flow configuration.

To reiterate, starting from the base mesh of 4000 cells, the mesh is increased in each step by multiplying the number of cells in the x and y axis by a scaling factor. This factor starts from 1.0, retaining the base mesh resolution, and is then increased by a counter of 1.0 in each step. The number of cores is, however, kept constant throughout.

The load imbalance is visualized in Fig. 9b, showing the mean, minimum, and maximum time of all processes to solve the ODEs. For the unbalanced standard model, the large spread of the computational load is apparent. In contrast, the load-balanced model (denoted by "LB Standard" in the figure) shows a very narrow band displaying a load balance of nearly 90%. Here, it shall be stressed that determining the computational load on each core cannot be easily achieved by tracking the computational time required for each process. While this approach is used in several publications to estimate the load balance [17,28], it neglects the effect that the load-balanced algorithms use MPI communications which mandates in most cases a final MPI_Waitall, particularly if native OpenFOAM MPI wrappers are used. The often hidden MPI wait then forces all processes to report the same time, even though the load balance is not equal. Therefore, more advanced tools are required to investigate this behavior. One possibility is the use of the open-source software Score-P [19]. With this library the source code can be instrumented to distinguish between MPI wait and code execution. This tool reports a load-balancing of 87% for the case of 1000 cells per core and an overall parallel performance of 86% in the current study.

3.1.4. TDAC chemistry model

The test case with a stable flame front is ideally suited for the use of the TDAC model. Due to the nearly stationary flame, the majority of the ODE solutions can be stored and retrieved from the tables. Hence, the total computational time required to solve the chemical reaction is significantly reduced.

The speed-up of the load-balanced TDAC model is plotted in Fig. 10a. It can be seen that a speed-up factor of about 1.6 can be obtained for this case as well. However, compared to the load-balanced standard chemistry model, the overall speed-up is lower. This is not surprising as the solution can now be retrieved from the table for the majority of the cells. Hence, a minor fraction of the cells requires direct chemistry integration, which significantly decreases the load required to be balanced. This can be observed by comparing the mean, minimum, and maximum time on all processes to solve the ODEs shown in Fig. 10b with the corresponding results in Fig. 9b. While this is an unfavorable condition for the load-balancing algorithm, it is shown that the load-balanced TDAC still achieves a computational gain.

As mentioned before, it can be seen in Fig. 10b that the mean time over all processes of the load-balanced model is higher than for the standard TDAC model. However, for the performance of the chemistry model, only the maximum time of all processes is important, as the subsequent solution of the convective and diffusive terms in the PDEs can only start once all processes have finished. Further, a larger spread of the minimum and maximum times of all processes is observed for the load-balanced model compared to the load-balanced standard chemistry model presented in Fig. 9b. This larger spread is caused by the challenge of determining the computational load required to solve the ODEs. Whereas for the standard model, the CPU times of the previous time step are available for all cells, the TDAC model has to use the CPU time required to solve the ODE from the last computation of that cell, which does not need to be the previous time step. In fact, as the result of the previous time step is likely to be found in the table, the CPU time to solve the ODE stored in the cell is likely to be different than the one required for the current composition. Hence, the estimation of the expected computational load on each process is not as precise as for the standard chemistry model, leading to the observed spread between minimum and maximum times in Fig. 10b.

The difference between the load-balanced standard and TDAC model is also shown in Fig. 11a, which shows the time spent solving the ODEs and waiting for MPI communication for a case with 13 500 cells per core, decomposed in 24 domains. The MPI wait and communication times are obtained with a ScoreP trace of the simulation over five time steps.

The load-balanced TDAC model has, in contrast to the standard chemistry model, a small fixed part required for retrieving the solution and adding it to the table, which cannot be load-balanced. Further, a significant increase in the MPI wait and communication times is observed for the TDAC model. This increase in MPI wait times can be attributed to two factors: First, the aforementioned difficulty in predicting the computational load of a cell, leading to a load imbalance and MPI wait times. Second, the TDAC model has to transfer four times as many bytes over

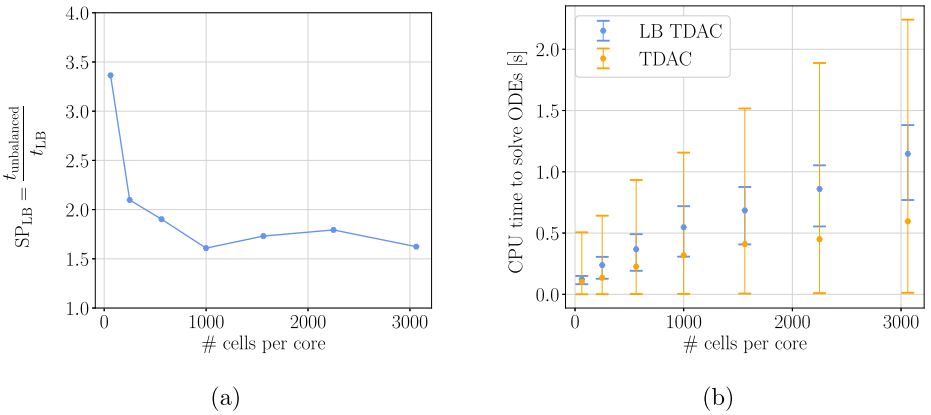


Fig. 10. 2D Counterflow Case using a single node with 64-cores: (a) Speed-up of the load-balanced TDAC chemistry model. (b) Required CPU time to solve the reaction per time step for the TDAC chemistry model. The error bars denote the minimum and maximum time required to solve the ODEs over all 64 processes.

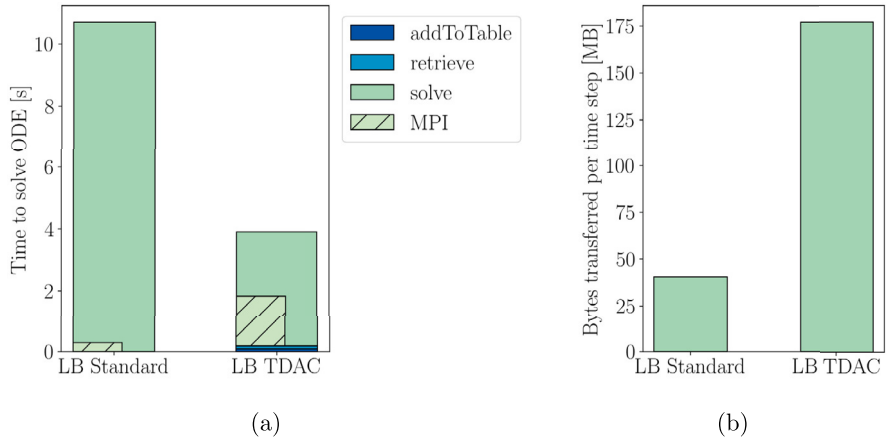


Fig. 11. (a) CPU time per time step to solve the ODEs with the MPI wait times and times to add and retrieve a solution from the table. (b) Bytes transferred via MPI in each time step to solve the ODEs. The presented data is obtained by tracing with ScoreP and analyzing the data with Vampir [18].

MPI than the standard chemistry model, see Fig. 11b. The reason for the increased MPI traffic of the TDAC model is that only cells located within the flame front will need to be solved, whereas the solution for cells outside the flame front can be retrieved from the tables. Therefore, a few processes send most of their cells for computation to the other processes. This gives an explanation for the observed drop in speed-up of the load-balanced TDAC model (cf. Fig. 10a) compared to the load-balanced standard model (cf. Fig. 9a).

3.2. Flame propagation and acceleration

3.2.1. The case description

In this section, the performance of the load balancing algorithm is analyzed with the simulation of hydrogen-air flame propagation and acceleration in an obstructed channel [11,13,14]. The schematic of the simulation domain is shown in Fig. 12. The 2D channel has a length of 48.8 cm and a width of $d = 4$ cm. Starting with $d/2$ from the closed end, the channel is obstructed by rectangular objects of width $d/16$ every 4 cm, blocking half of the channel width. The mesh consists of cubic cells with a cell size of 25 µm, resulting in a total of 15 million cells. In contrast to the counterflow flame case, this test uses a small hydrogen-air reaction mechanism with 9 species and 27 reactions [20]. The domain is initially filled with a premixed stoichiometric H_2-air mixture. A cylindrical ignition patch consisting of hot combustion products with a radius of 5 mm is placed at the top closed end. A symmetric plane boundary condition is applied at the symmetry plane, while at the walls zero-gradient and no-slip boundary conditions are applied for scalars and velocity, respectively. At the open end of the channel zero-gradient boundary conditions are used for all transported variables except for the pressure which uses a wave transmissive outlet boundary condition. An eddy viscosity sub-grid scale model using a modeled balance equation for turbulent kinetic energy [29] is employed for turbulence modeling, while no sub-grid combustion model is applied. Further details can be found in [11]. The Rosenbrock34 ODE solver [27] with absolute and relative tolerances of $1.0 \cdot 10^{-12}$ and $1.0 \cdot 10^{-2}$ is selected to solve the ODEs. The solver is a modified version of the rhoReactingFoam solver of OpenFOAM. The modifications include solving a modified species and sensible enthalpy transport equations to account for correction diffusion velocities in mass and heat transport equations [24]. Further, different Lewis numbers are applied for the species involved to account for the differential diffusion effects in hydrogen-air combustion. The list of applied Lewis numbers is given in Table 1. An snapshot of the results at 1.75 ms showing the heat release contours and vorticity structures is illustrated in Fig. 13.

All simulations are run on the supercomputer Hawk, starting from 8 nodes, each with 128 cores, up to 64 nodes corresponding to a total of 8192 cores. The scotch decomposition method is chosen for all cases. Further, due to the small time step of 5×10^{-9} s the computational load is recalculated every 50 iterations, significantly reducing the required CPU time for the all-to-all communication.

3.2.2. Speed-up and strong scaling behavior

The computational times required to solve the chemistry reaction mechanism are shown in Fig. 14a displaying the results for the standard chemistry model and Fig. 14b showing the corresponding values for the TDAC model. In both cases the number of nodes (cores) varies between

Table 1
Lewis numbers of different species employed in the simulation.

Species	H_2	H_2O	H	O	H_2O_2	OH	HO_2	O_2	N_2
Lewis number	0.35	0.99	0.22	0.79	1.23	0.81	1.22	1.21	1.01

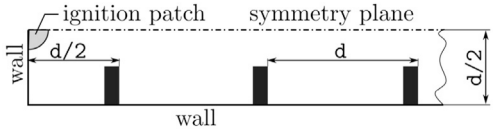


Fig. 12. Sketch of the advancing flame case setup.

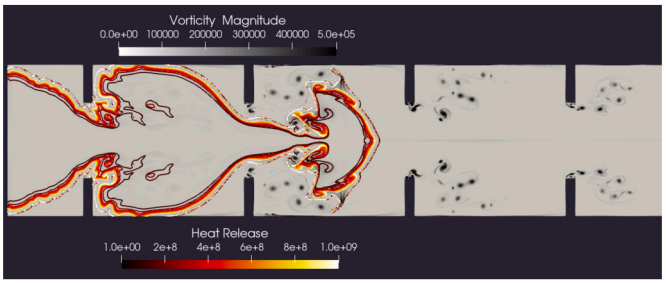


Fig. 13. Flame propagation at 1.75 ms showing the heat release [J/s] contours and vorticity magnitude [1/s] as a gray scale background image for the first part of the computational domain.

8 (1024) and 64 (8192). The values are calculated by considering the maximum wall clock time for solving the chemistry reaction among all processes at each time step, averaged over the simulation time. The ratio of the computational times for unbalanced and load-balanced models, i.e., the speed-up gained by enabling the load-balancing algorithm compared to the unbalanced models, are shown in Fig. 14c and Fig. 14d as well.

In comparison to the counterflow case, the chemistry load reduces significantly due to the relatively small reaction mechanism. This can be inferred by comparing the CPU times in Fig. 9b (the case ran with ≈3000 cells per core) with the approximately equivalent case in Fig. 14a (the case ran with 32 nodes i.e., 3600 cells per core). Hence, the overall gain by enabling the load balancing is reduced. However, it is interesting to see in Fig. 14c that the load-balanced model can still halve the required solution times even when using 64 computational nodes, corresponding to 8192 cores. The plot in Fig. 14c shows a drop of the speed-up from 3 in the case of 8 nodes (14 600 cells per core) to about 2 in the case of 8 nodes (1800 cells per core) which is due to the prevalence of the required MPI communications required for load balancing with respect to the total available load to be balanced. Similar behavior is observed when TDAC is enabled in Fig. 14b and Fig. 14d. The speed-up of the load-balanced TDAC model compared to the unbalanced TDAC is at first in the range determined in the counterflow flame case, see Fig. 14d. However, as the total load to be balanced decreases and the MPI load increases with the increasing number of nodes, the speed-up reduces to a factor of about 1.2 for 64 nodes (8192 cores).

The strong scaling behavior of the load-balanced and default standard chemistry model is plotted in Fig. 15a. The speed-up in the strong scaling test is defined by

$$SP = \frac{t}{t_{ref}}, \tag{2}$$

where SP is the *Strong scaling* speed-up, t is the average CPU time per time step and t_{ref} is the one of the reference case, which is the simulation with 8 nodes corresponding to 1024 cores. Here we want to emphasize, that the strong scaling speed-up shown here does not relate unbalanced with balanced implementations, but show scaling of the unbalanced and balanced implementations to themselves. The better scaling performance for the unbalanced case explains the reduction in speed-up depicted in Fig. 14c and Fig. 14d for larger numbers of nodes. For the unbalanced chemistry model, near perfect scaling is observed up to 64 nodes. This is the expected behavior as the chemistry load is local, i.e., solution of ODEs does not require any MPI communication. Hence, the standard chemistry model scales directly with the number of cells. The load-balanced chemistry model starts to deviate from the ideal scaling for more than 16 nodes or less than 7300 cells per core. This observed behavior is in alignment with typical mesh decomposition suggestions for OpenFOAM, which determine a lower cell limit of around 10 000 cells per core, at which point MPI communications start to outweigh the performance increase. Still, some scaling is seen up to 64 nodes and 1800 cells per core. Despite the observed non-linear strong scaling of the load-balanced model, it shows a speed-up factor of at least 2 compared to the unbalanced model, see Fig. 14c.

For the TDAC model, the strong scaling behavior of the load-balanced TDAC model is comparable to the load-balanced standard model, see Fig. 15b. Further, as the computational load of the TDAC model is not directly linear with the cell count, as some cells are tabulated whereas others require solving the ODEs, the strong scaling behavior of the standard TDAC model deviates as well from the ideal line, see the orange TDAC line in Fig. 15b. Again a cell limit of about 7000 cells per core can be identified for the linear strong scaling of the load-balanced model which also halves the required computational time compared to the unbalanced model (see Fig. 14d).

3.3. DLR JHC burner

3.3.1. The case description

The last test case investigated is the jet in hot co-flow DLR burner which is part of the exaFoam benchmark set [1]. The test case setup provides three different meshes with 1, 4 and 11 million cells, i.e. small, medium and large cases. Each setup consists of a 90 degree wedge of the simulation domain with symmetry boundary conditions.

In the following, the large case is selected for the scaling investigation, whereas the small case is chosen to validate the correct behavior of the load-balanced model. The small case is chosen for the validation as running a simulation from 0 to 5 ms requires a considerable amount of CPU hours. Therefore, the result of the 1 million cells mesh is then mapped to the 11 million cells case and ran for 0.05 ms to generate a good starting solution for the scaling investigations. The large case is then decomposed starting from 1 node (128 cores) up to 24 nodes (4800 cores) using the scotch decomposition method. All scaling results presented were run on the supercomputer Hawk at the HLRS in Stuttgart.

This test case uses a large reaction mechanism with 36 species and 218 reactions, which is the GRI mechanism without the NO_x sub-mechanism [2]. For the sake of brevity, only the results with TDAC are discussed. Based on the results in the previous section, the load balancing of the chemistry load with tabulation is challenging due to the dynamic nature of the tabulation method. Therefore, TDAC and load-balanced TDAC models are investigated. For the chemistry model, the TDAC and load-balanced TDAC models are investigated, with the reduction of the chemical mechanism switched off for the large case.

A similar turbulence model as with the previous case is employed [29] while a partially stirred reactor (PaSR) model [6] is applied for turbulent combustion modeling. The Rosenbrock34 ODE solver [27] with absolute and relative tolerances of $1.0 \cdot 10^{-12}$ and $1.0 \cdot 10^{-7}$, respectively, is applied. Further details can be found in [1]. In analogy to the counterflow flame, the adapted reactingFoam solver is utilized.

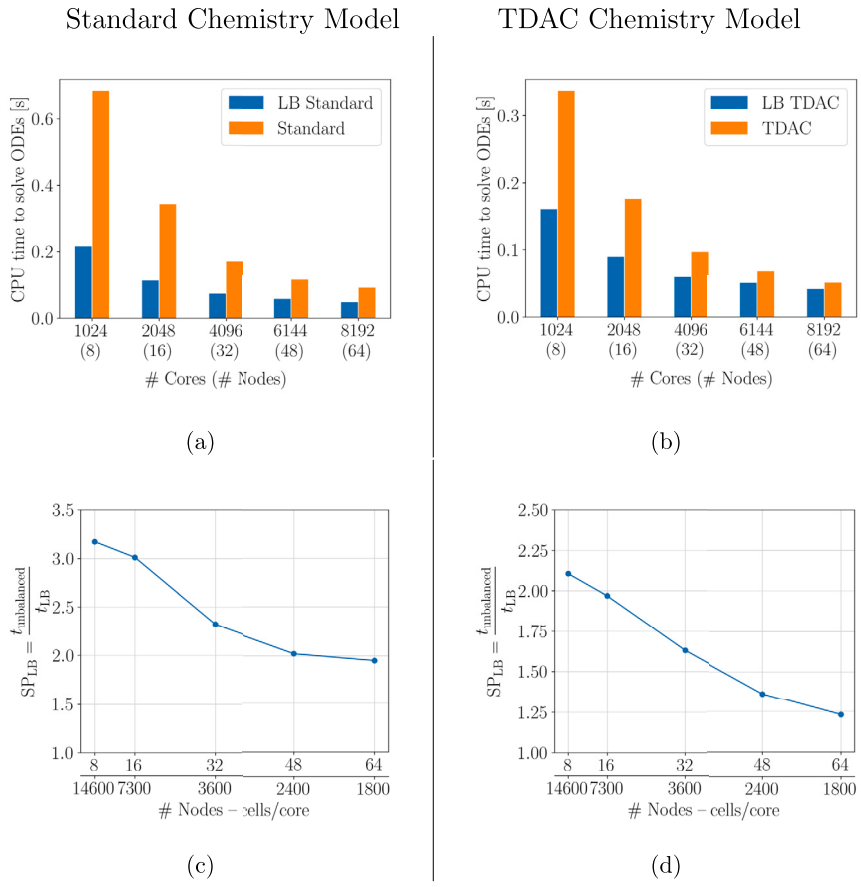


Fig. 14. The flame propagation case: (a) & (b) Bar plots of the maximum time on all processes required time (per time step) to solve the ODEs averaged over the total simulation time. (c) & (d) Speed-up of the load-balanced standard and TDAC chemistry models compared to the unbalanced models of OpenFOAM.

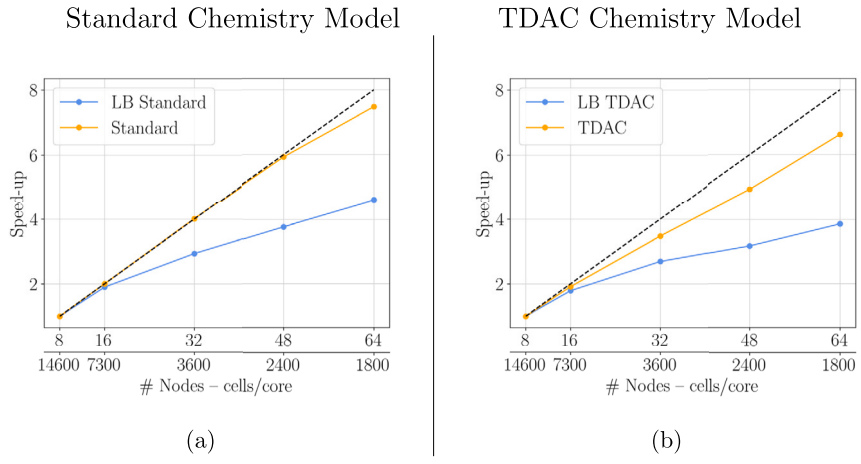


Fig. 15. The flame propagation case: strong scaling of the load-balanced and unbalanced standard and TDAC chemistry model, considering only the time spent on solving the reaction rate. The top x-axis description shows the number of nodes with each 128 cores and the bottom x-axis description denotes the number of cells per core.

3.3.2. *Model validation*

Four sampling positions are available for model validation, see Fig. 16, and are used in the following to serve for validation of the load-balanced algorithm. The resulting temperature and OH distribution at these four sampling lines is plotted in Fig. 17, exhibiting a perfect agreement between the load-balanced and unbalanced TDAC models. Here, both tabulation and dynamic adaptive reduction of the TDAC have been utilized, and the results with and without load balancing are compared with the experimental data. It can be seen that the load-balanced TDAC model returns the same results as the unbalanced model and that the load-balanced TDAC model supports the on-the-fly mechanism reduction option as well.

3.3.3. *Speed-up and strong scaling behavior*

After the validation of the new load-balanced model, we analyze in this sub-section the performance of the model for large-scale parallel simulations, selecting the case with 11 million cells for this investigation. In Fig. 18a, the averaged CPU time per time step required for

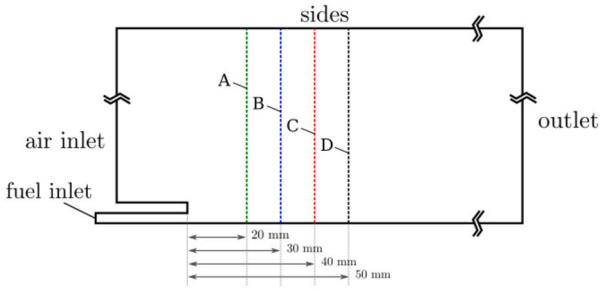


Fig. 16. A 2D sketch of the 3D computational domain of the DLR JHC burner with the positions of the four sampling lines A, B, C, and D. Taken from [1].

load-balanced and unbalanced TDAC models for solving the chemistry sub-step using up to 3072 cores, i.e., 24 computational nodes on HAWK HPC are shown. Here, the mesh size is held constant. It can be seen that for this case the times to solve the ODE with the large chemical mechanism are much higher than for the small hydrogen mechanism of the flame propagation case (compare the orange bars). Such a large computational time can be effectively decreased by the load balancing algorithm (see the blue bars) even in large-scale parallel simulations. In Fig. 18b the ratio of the computational times for unbalanced and load-balanced TDAC models are depicted showing the gained speed-up by the load balancing algorithm. It can be seen that a speed-up of a factor of 5 to 6 compared to the unbalanced model can be achieved in solving the chemistry sub-step.

The strong scaling behavior of the load-balanced and unbalanced TDAC models is plotted in Fig. 19 for 1, 2, 4, 8, 16, and 24 nodes with each 128 cores per node. The load-balanced and default unbalanced TDAC models both show a linear scaling up to 4 nodes or approximately 20 000 cells per core. Below this threshold i.e., by using less cells per core corresponding to using a higher number of cores with fixed mesh size, the strong scaling for the load-balanced and unbalanced model deviates from the ideal line. Thus, for the large reaction mechanism a strong-scaling threshold of about 10 000 cells can be determined which is in the range of best-practice values for the mesh decomposition considering the solution of the transport PDEs. However, even for higher mesh decompositions with fewer than 10 000 cells per core a speed-up is observed up to 5300 cells per core. Beyond this threshold, the MPI communications required for the load-balancing surpass the benefits of an increased number of cores, leading to a constant strong scaling.

4. Conclusion and outlook

Two novel load-balanced chemistry models for OpenFOAM, the standard and the TDAC models, have been developed. The significant load imbalances caused by the large disparities in solving the ODEs, depending on composition, temperature, and pressure, are tackled by balancing the chemistry load without modifying the mesh. Instead, the required information is directly exchanged between the processes via MPI calls. To estimate the load and achieve a high load balance, the time to solve the ODE is tracked on a cell level. In the case of the TDAC model, this information is expanded by adding the time to add the solution to the *in-situ* tables. In addition, the load-balanced TDAC model introduces two key enhancements. Firstly, it implements dual tables on each core, one for local cell solutions and another for solutions computed from received cells. Secondly, the load-balanced TDAC model promptly adds computed solutions to tables post-computation, enabling subsequent cells to benefit from previously computed results within the same time step. Notably, both models offer the flexibility to update load balancing information every n^{th} time step, effectively reducing MPI overheads associated with the required all-to-all communication.

To validate the load-balanced models and to assess their speed-up and strong-scaling behavior, three test cases are investigated. Ranging from a small mesh with a large reaction mechanism on a single node to larger cases incorporating both small and large reaction mechanisms, simulations are conducted on the Hawk supercomputer at HLRS in Stuttgart. Results demonstrate the correct implementation of load-balanced models by reporting identical results to those of unbalanced default versions of OpenFOAM. Noteworthy speed-up factors of up to 6 for the standard model and up to 5 for the TDAC model are observed across various scenarios, while case-dependent, consistent speed-up benefits are evident across all investigated scenarios. Regarding strong-scaling behavior, linear scaling is observed up to approximately 10 000 cells per core, a recognized limit for OpenFOAM applications [5]. Despite non-linear scaling for cell counts below this threshold, load-balanced models still outperform unbalanced counterparts in terms of computational efficiency, establishing them as the preferred choice for all tested scenarios.

Nevertheless, opportunities for refinement persist, including the implementation of automatic detection mechanisms for updating load imbalances and enhancements to load estimation for TDAC models. These avenues constitute part of ongoing and future research work.

CRediT authorship contribution statement

Jan Wilhelm Gärtner: Writing – original draft, Validation, Software, Methodology, Conceptualization. **Ali Shamooni:** Writing – original draft, Methodology, Conceptualization. **Thorsten Zirwes:** Writing – review & editing, Conceptualization. **Andreas Kronenburg:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

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

Data availability

The code repository is available on GitHub: https://github.com/ITV-Stuttgart/loadBalancedChemistryModel – And as a static version on the data repositorium of the University of Stuttgart DaRUS.

Acknowledgements

All HPC resources are provided by HLRS Stuttgart under the project “MULTIPHASE”. The authors acknowledge the financial support by the German Research Foundation (DFG) project No. 513858356.

Appendix A. Comparison of the Load-Balancing to DLBFoam

The DLBFoam [28] and the presented load-balancing library share similarities in the high-level algorithms. Both libraries use the same algorithm to balance the load as presented in Sec. 2.1, both libraries also use some form of a container to send and receive the data between processors. The main differences between the DLBFoam and the new load-balancing libraries are (i) the adaption to the TDAC model of OpenFOAM, which DLBFoam v1.0 does not support (although later versions for the foundation version of OpenFOAM do support it), and (ii) a significant effort made in the current work to improve MPI communications on the low-level implementation. Further, both libraries target different OpenFOAM platforms. The presented load-balancing library is designed for OpenFOAM v2306, whereas DLBFoam focuses on the OpenFOAM Foundation version.

At the time of writing, two main DLBFoam projects exist publicly: the DLBFoam v1.1 project, which uses external libraries for a fully analytical chemistry Jacobian, and the older DLBFoam v1.0, which does not rely on external libraries. Adding the capability to use analytical solutions is quite powerful and useful for the chemistry solution. However, the focus

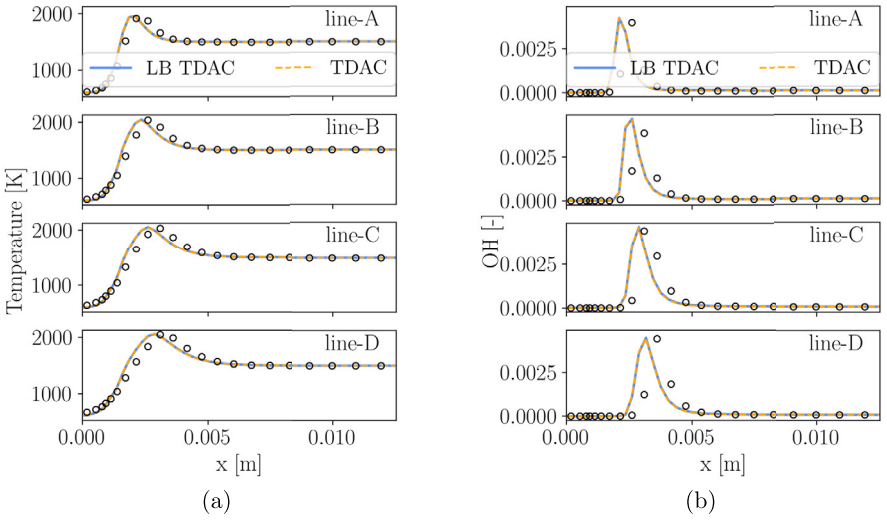


Fig. 17. The DLR JHC burner case: averaged temperature and OH mass fraction at the sample lines A, B, C, and D of Fig. 16.

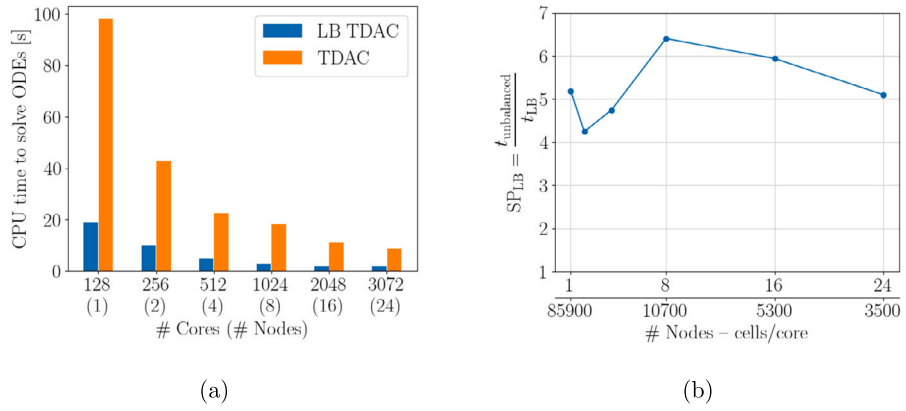


Fig. 18. The DLR JHC burner case: (a) bar plots of the averaged required time (per time step) to solve the ODEs. (b) The ratio of the computational times for unbalanced and load-balanced TDAC models, i.e., the speed-up gained by enabling the load-balancing algorithm compared to the unbalanced TDAC model of OpenFOAM.

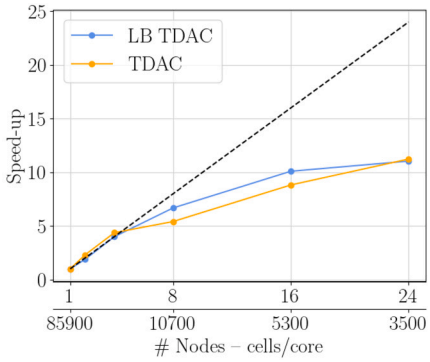


Fig. 19. The DLR JHC burner case: strong scaling of the load-balanced and unbalanced TDAC chemistry models, considering only the time spent on solving the chemistry sub-step. The top x-axis description shows the number of nodes with each 128 cores and the bottom x-axis description denotes the number of cells per core.

here is on the load-balancing aspects. Other features, such as reference mapping or the analytical solution, can be added at a later stage as a new chemistry model, using the load-balancing algorithm as a platform. Hence, as only the load-balancing aspect, and not the chemistry solution, is considered, the DLBFoam v1.0 with reference mapping switched off is used for all the following results. To reiterate, DLBFoam is written with a focus on the OpenFOAM Foundation version, in contrast to the current library, which focuses on the ESI version of OpenFOAM. DLBFoam v1.0 does provide a branch for the OpenFOAM ESI version v2006, which was selected at first to use the same OpenFOAM platform for testing. However, when testing the counterflow benchmark case it crashes with an error in the LoadBalancedChemistryModel::solveSingle() function after the second iteration, which we did not further investigate, as the focus of DLBFoam is on the OpenFOAM Foundation version. Therefore, the main branch of DLBFoam v1.0 for the OpenFOAM Foundation version is selected. However, the inconsistent treatment of the ODE integration of OpenFOAM-8 and OpenFOAM-v2306 does not allow a direct comparison. Therefore, here, the speed-up of the DLBFoam library is calculated as the speed-up compared to the reference case of OpenFOAM-8, not the reference case of OpenFOAM-v2306. Fig. A.20 shows the respective speed-up for the DLBFoam library and the novel load-balancing library for the counterflow case presented in Sec. 3.1. The comparison shows that the present load-balancing library has a higher speed-up of about 35% to 55% compared to the DLBFoam library for this benchmark case. The reduction in speed-up of the DLBFoam library for higher cell counts, and therefore a larger amount of data transferred via MPI, indicates that the improved MPI communications of the novel load-balancing library bear fruit. However, a detailed investigation would require using profilers such as ScoreP, and therefore, it is only one potential, even though likely, theory. Finally, the authors want to point out that only the load-balancing aspect, not the overall workload, is compared. The DLBFoam library can activate features such as reference mapping or efficient ODE integration using third-party libraries to reduce the workload, signifi-

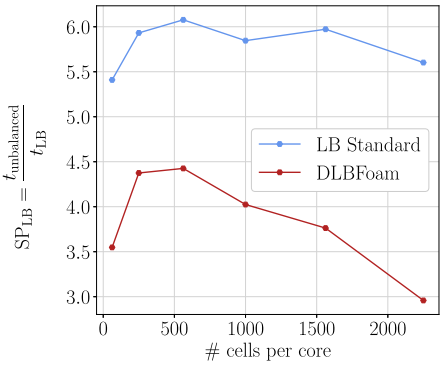


Fig. A.20. 2D counterflow case using a single node with 64-cores: speed-up of the load-balanced standard chemistry model compared to the DLBFoam library [28].

cantly reducing the overall run-time. These features may be added to the current library at a later stage.

References

[1] Grand challenges, exaFoam project, 2024.
[2] GRI-Mech 3.0, the gas research institute, 2024.
[3] C.M. Arndt, W. Meier, Influence of boundary conditions on the flame stabilization mechanism and on transient auto-ignition in the DLR jet-in-hot-coflow burner, Flow Turbul. Combust. 102 (4) (Apr. 2019) 973–993, https://doi.org/10.1007/s10494-018-9991-6.
[4] C.M. Arndt, M.J. Papageorge, F. Fuest, J.A. Sutton, W. Meier, Experimental investigation of the auto-ignition of a transient propane Jet-in-Hot-Coflow, Proc. Combust. Inst. 37 (2) (2019) 2117–2124, https://doi.org/10.1016/j.proci.2018.06.195.
[5] G. Axtmann, U. Rist, Scalability of OpenFOAM with large eddy simulations and dns on high-performance systems, in: High Performance Computing in Science and Engineering '16: Transactions of the High Performance Computing Center Stuttgart (HLRS), 2016, 2016.
[6] J. Chomiak, Combustion a Study in Theory, Fact and Application, Abacus Press, Philadelphia, PA (USA), 1990.
[7] F. Contino, H. Jeanmart, T. Lucchini, G. D'Errico, Coupling of in situ adaptive tabulation and dynamic adaptive chemistry: an effective method for solving combustion in engine simulations, Proc. Combust. Inst. 33 (2) (2011) 3057–3064, https://doi.org/10.1016/j.proci.2010.08.002.
[8] F. Contino, T. Lucchini, G. D'Errico, C. Duynslaegher, V. Dias, H. Jeanmart, Simulations of advanced combustion modes using detailed chemistry combined with tabulation and mechanism reduction techniques, SAE International Journal of Engines 5 (2) (Apr. 2012) 185–196, https://doi.org/10.4271/2012-01-0145.
[9] F. Contino, F. Foucher, P. Dagaut, T. Lucchini, G. D'Errico, C. Mounaïm-Rousselle, Experimental and numerical analysis of nitric oxide effect on the ignition of iso-octane in a single cylinder HCCI engine, Combust. Flame 160 (8) (Aug. 2013) 1476–1483, https://doi.org/10.1016/j.combustflame.2013.02.028.
[10] F. Contino, J.-B. Masurier, F. Foucher, T. Lucchini, G. D'Errico, P. Dagaut, CFD simulations using the TDAC method to model iso-octane combustion for a large range of ozone seeding and temperature conditions in a single cylinder HCCI engine, Fuel 137 (Dec. 2014) 179–184, https://doi.org/10.1016/j.fuel.2014.07.084.
[11] S. Emami, K. Mazaheri, A. Shamooni, Y. Mahmoudi, LES of flame acceleration and DDT in hydrogen–air mixture using artificially thickened flame approach and detailed chemical kinetics, Int. J. Hydrog. Energy 40 (23) (June 2015) 7395–7408, https://doi.org/10.1016/j.ijhydene.2015.03.165.
[12] A. Fiolitakis, C.M. Arndt, Transported PDF simulation of auto-ignition of a turbulent methane jet in a hot, vitiated coflow, Combust. Theory Model. 24 (2) (2020) 326–361, https://doi.org/10.1080/13647830.2019.1682197.
[13] V.N. Gamezo, T. Ogawa, E.S. Oran, Numerical simulations of flame propagation and DDT in obstructed channels filled with hydrogen–air mixture, Proc. Combust. Inst. 31 (2) (2007) 2463–2471, https://doi.org/10.1016/j.proci.2006.07.220.
[14] V.N. Gamezo, T. Ogawa, E.S. Oran, Flame acceleration and DDT in channels with obstacles: effect of obstacle spacing, Combust. Flame 155 (1) (2008) 302–315, https://doi.org/10.1016/j.combustflame.2008.06.004.
[15] V. Hiremath, S.R. Lantz, H. Wang, S.B. Pope, Computationally-efficient and scalable parallel implementation of chemistry in simulations of turbulent combustion, Combust. Flame 159 (10) (2012) 3096–3109, https://doi.org/10.1016/j.combustflame.2012.04.013.
[16] V. Hiremath, S.R. Lantz, H. Wang, S.B. Pope, Large-scale parallel simulations of turbulent combustion using combined dimension reduction and tabulation of chemistry, Proc. Combust. Inst. 34 (1) (2013) 205–215.
[17] Z. Huo, M.J. Cleary, K. Wu, A.R. Masri, X. Fan, A dynamic load balancing model coupled with DAC and ISAT for a stochastic turbulent combustion model, Combust. Theory Model. (ISSN 1364-7830) 27 (3) (Jan. 2023) 1–29, https://doi.org/10.1080/13647830.2023.2165967.
[18] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M.S. Müller, W.E. Nagel, The vampir performance analysis tool-set, in: M. Resch, R. Keller, V. Himmler, B. Krammer, A. Schulz (Eds.), Tools for High Performance Computing, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 139–155.
[19] A. Knüpfer, C. Rössel, D.A. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W.E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, F. Wolf, Score-P: a joint performance measurement run-time infrastructure for periscope, scalasca, TAU, and vampir, in: H. Brunst, M.S. Müller, W.E. Nagel, M.M. Resch (Eds.), Tools for High Performance Computing 2011, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 79–91.
[20] J. Li, Z. Zhao, A. Kazakov, F.L. Dryer, An updated comprehensive kinetic model of hydrogen combustion, Int. J. Chem. Kinet. 36 (2004) 566–575.
[21] L. Liang, J.G. Stevens, J.T. Farrell, A dynamic adaptive chemistry scheme for reactive flow computations, Proc. Combust. Inst. 32 (1) (2009) 527–534.
[22] L. Lu, S.R. Lantz, Z. Ren, S.B. Pope, Computationally efficient implementation of combustion chemistry in parallel PDF calculations, J. Comput. Phys. 228 (15) (2009) 5490–5525.
[23] I. Morev, B. Tekgül, M. Gadalla, A. Shahanaghi, J. Kannan, S. Karimkashi, O. Kaario, V. Vuorinen, Fast reactive flow simulations using analytical Jacobian and dynamic load balancing in OpenFOAM, Phys. Fluids 34 (2) (Feb. 2022) 021801, https://doi.org/10.1063/5.0077437.
[24] T. Poinsot, D. Veynante, Theoretical and Numerical Combustion, 2nd ed. edition, Edwards, ISBN 1-930217-10-2, 2005.
[25] S. Pope, Computationally efficient implementation of combustion chemistry using in situ adaptive tabulation, Combust. Theory Model. 1 (1) (1997) 41–63.
[26] A. Shamooni, K. Mazaheri, M. Timaji, A novel strategy for computationally efficient implementation of detailed chemistry in parallel simulation of reactive flows, in: 8th Mediterranean Combustion Symposium (MCS8), 2013.
[27] L.F. Shampine, Implementation of Rosenbrock methods, ACM Trans. Math. Softw. 8 (2) (1982) 93–113.
[28] B. Tekgül, P. Peltonen, H. Kahila, O. Kaario, V. Vuorinen, DLBFoam: an open-source dynamic load balancing model for fast reacting flow simulations in OpenFOAM, Comput. Phys. Commun. 267 (Oct. 2021) 108073, https://doi.org/10.1016/j.cpc.2021.108073.
[29] A. Yoshizawa, Statistical theory for compressible turbulent shear flows, with the application to subgrid modeling, Phys. Fluids 29 (7) (July 1986) 2152–2164, https://doi.org/10.1063/1.865552.
[30] T. Zirwes, F. Zhang, P. Habisreuther, J.A. Denev, H. Bockhorn, D. Trimis, Optimizing load balancing of reacting flow solvers in OpenFOAM for high performance computing, in: Proceedings of 6th ESI OpenFOAM User Conference, vol. 6, 2018.