A dynamic load balancing method for the evaluation of chemical reaction rates in parallel combustion simulations

J. Muela, R. Borrell[1], J. Ventosa-Molina[2], L. Jofre[3], O. Lehmkuhl[1], C.D. Pérez-Segarra*

Heat and Mass Technological Center (CTTC),Universitat Politècnica de Catalunya - BarcelonaTech (UPC), ESEIAAT, Colom 11, 08222, Terrassa, Barcelona, Spain

Abstract

The development and assessment of an efficient parallelization method for the evaluation of reaction rates in combustion simulations is presented. Combustion simulations where the finite-rate chemistry model is employed are computationally expensive. In such simulations, a transport equation for each species in the chemical reaction mechanism has to be solved, and the resulting system of equations is typically stiff. As a result, advanced implicit methods must be applied to obtain accurate solutions using reasonable time-steps at expenses of higher computational resources than explicit or classical implicit methods. In the present work, a new algorithm aimed to enhance the numerical performance of the time integration of stiff systems of equations in parallel combustion simulations is presented. The algorithm is based on a runtime load balancing mechanism, increasing noteworthy the computational performance of the simulations, and consequently, reducing significantly the computer time required to perform the numerical combustion studies.

*Corresponding author
Email address: `cttc@cttc.upc.edu` (C.D. Pérez-Segarra)
[1]Present address: Barcelona Supercomputing Center (BSC-CNS) , C/ Jordi Girona 31, Nexus II Building, Barcelona, Spain
[2]Present address: Chair of Fluid Mechanics, Technische Universität Dresden, Zeunerbau, George-Bähr-Straße 3, 01069 Dresden
[3]Present address: Center for Turbulence Research, Stanford University, Stanford, CA 94305, USA

Keywords: Combustion, Parallel, Load balancing, Chemical kinetics

1. Introduction

Combustion processes are encountered in a wide spectrum of scientific and engineering applications, ranging from energy conversion and propulsion systems to fires, volcanoes and solar physics. Therefore, the study of combustion processes to better understand its physics and optimize combustion applications is a very relevant and active field of research. The study of combustion processes employing Computational Fluid Dynamics (CFD) simulations has received a wide attention from researchers and engineers over the last decades [1, 2, 3].

In general, combustion simulations are characterized by a wide range of coupled phenomena [4], which makes these simulations complex and challenging. One of these issues is the description of the chemical reactions. The kinetics of combustion chemical reactions are complex processes described by the empirical Arrhenius law [5]. Combustion processes may involve hundreds or even thousands of species and reactions, and the integration of the resulting chemistry system of ordinary differential equations (ODEs) requires a huge amount of computational resources, even for zero-dimensional reactors [6]. Moreover, this set of ODEs is stiff [7]. A stiff equation is characterized for having a smooth solution with a slow variation, whose numerical integration presents instabilities due to the presence of nearby solutions with very fast variations. This characteristic makes these type of equations difficult to integrate numerically. Moreover, this also makes the numerical integration computationally expensive, since very small time-integration steps must be employed in order to avoid oscillatory or diverging solutions. Therefore, non-classical integration methods able to overcome this constraint must be used in order to integrate stiff equations using reasonable integration steps, aiming to obtain accurate solutions in an affordable computing time. Some of these methods are the Rosenbrock method, the Semi-Implicit Extrapolation Method or the Gear's-like methods [8].

Nomenclature

$\mathbf{g}$	Gravity
h	Total enthalpy
h_s	Sensible enthalpy
hc_{it}	*Heavy calculation* iterations
hc_{ss}	*Heavy calculation* system size
hn_{cpu}	*Heavy nodes* per CPU
hn_t	Total number of *heavy nodes*
ms_{hn}	Message size communicated by a *heavy node*
n_{cpu}	Nodes per CPU
n_t	Total number of nodes
N	Number of CPUs
N_{hn}	CPUs with *heavy nodes*
N_s	Number of species
p	Dynamic pressure
P_0	Thermodynamic pressure
$\dot{\mathcal{Q}}$	Energy source term
$\dot{\mathbf{q}}$	Conduction heat flux
S_r	Speed-up ratio
t	Time
$\mathbf{u}$	Velocity
$\mathbf{V_k}$	Species k diffusion velocity
$\dot{w}_k$	Species k mass reaction rate
Y_k	Species k mass fraction
Δt	Simulation time step
Δt_{chem}	*Chemical time step*
$\Delta h^0_{f,k}$	Species k standard formation enthalpy
ρ	Density
τ	Viscous stress tensor
τ_c	Chemical time-scale
τ_t	Turbulent subgrid time-scale
θ_{hn}	Ratio *heavy nodes* per CPU
θ_N	Ratio CPUs with *heavy nodes*
Θ	Ratio of imbalance
ζ	Ratio computing vs. communication effort
$\mathrm{Pr_t}$	Turbulent Prandtl number
Re	Reynolds number
$\mathrm{Sc_t}$	Turbulent Schmidt number
$\widetilde{\cdot}$	Filtered quantity
$\bar{\cdot}$	Favre-averaged quantity

Nonetheless, these numerical time-integration methods can lead to unbalanced executions when running parallel combustion simulations, since only some computational cells of the simulation domain will present active chemical reactions. For example, Velghe et al. [9] reported load balancing problems on 6-cylinder gasoline engine simulation when using more than 128 processors. These problems arise since combustion phenomena are localized and the employed combustion models are only active at some regions of the computational mesh. Another example of load balancing issues is the one reported by Torres et al. [10] in their study about different partitioning strategies for combustion engine simulations employing the Kiva-4 code [11]. In their simulations, computational cells become deactivated and activated during simulation time, originating load balancing problems depending on the initial mesh partitioning, even without considering the possible imbalance generated by chemical reactions.

Therefore, aiming to increase the efficiency and reduce the computing time of parallel combustion simulations, it is of interest the development of numerical methods and parallelization strategies capable to dynamically balance the load of these simulations. With this aim, Thévenin et al. [12] proposed a load balancing strategy based on the transfer of the boundary cells between neighbouring processors. Another load balancing strategy is the one proposed by Shi et al. [13]. In their work, a hybrid CPU/GPU algorithm is presented, where the *highly stiff* nodes are integrated implicitly by the CPUs, while the *moderately stiff* and *non-stiff* nodes are integrated explicitly using GPUs. Hence, they propose a load balancing algorithm that tries to achieve similar computing time between the CPUs and the GPUs at each time-step. In the present work, a novel dynamic load balancing method for parallel combustion simulations is developed and assessed. The algorithm presented is capable of redistributing the unbalanced computational tasks of a parallel computation between all the CPUs involved in the calculation. It is based on a runtime load balancing mechanism which is complementary to the underlying domain decomposition (DD). This load balancing algorithm has been developed using the Message Passing Interface (MPI) standard [14] and in C++ Object Oriented Programming (OOP) language [15], assuring a high-portability on High Performance Computing (HPC) platforms.

The paper is organized as follows. First, in Section 2 the mathematical description of combustion processes is detailed. Section 3 presents the time-integration strategy proposed to integrate the species mass fraction transport equation. The strategy is based on a hybrid explicit/implicit method where the non-stiff cells are integrated explicitly, while the stiff cells are integrated implicitly using a non-classical integration method specially well-suited for stiff equations. The one used in the present work has been the Gear's method [16]. This strategy allows to reduce the computational effort of the simulation, although it creates a load imbalance in parallel simulations. Hence, a dynamic load balancing method capable of efficiently redistribute and rebalance the unbalanced parallel simulations has been proposed. This algorithm developed and implemented within this work is described in Section 4. Next, in Section 5, a performance analysis is presented. Next, in Section 6, the load balancing algorithm has been used to simulate the Cambridge autoignition experiment carried out by Markides and Mastorakos [17]. This benchmark case has been employed to test the speed-up and scalability of the developed method in a real combustion parallel simulation. Finally, conclusions and future work are discussed in Section 7.

2. Mathematical formulation

Combustion processes are globally exothermic chemical reactions which take place in a fluid flow. Furthermore, combustion simulations require solving the dynamics of the flow where combustion occurs as well as the kinetics of the chemical process. In finite-rate combustion simulations, the Navier-Stokes equations together with the energy conservation equation and a mass fraction transport equation per each species have to be solved. The present study is carried out in the framework of Large-Eddy Simulation (LES) modelling. In LES the large scales of the flow are solved, while the subgrid-scales (SGS) are modelled. This scale separation is obtained applying a low-pass filter to the transport equations. Moreover, in non-constant density cases the filtered variables are density-weighted in order to avoid introducing a model for the unclosed term in the continuity equation, as proposed by Favre [18]. Hence, the resulting density weighted filtered governing equations are:

$$\frac{\partial \overline{\rho}}{\partial t} + \nabla \cdot \left(\overline{\rho} \widetilde{\mathbf{u}} \right) = 0, \tag{1}$$

$$\frac{\partial \overline{\rho} \widetilde{\mathbf{u}}}{\partial t} + \nabla \cdot \left(\overline{\rho} \widetilde{\mathbf{u}} \widetilde{\mathbf{u}} \right) = -\nabla \overline{p} + \nabla \cdot \left(\overline{\boldsymbol{\tau}} - \overline{\rho} \left(\widetilde{\mathbf{u}\mathbf{u}} - \widetilde{\mathbf{u}} \widetilde{\mathbf{u}} \right) \right) + \overline{\rho} \mathbf{g}, \tag{2}$$

$$\overline{\rho} \frac{\partial \widetilde{h}}{\partial t} + \overline{\rho} \widetilde{\mathbf{u}} \cdot \nabla \widetilde{h} = \overline{\frac{dP_0}{dt}} - \nabla \cdot \left(\overline{\dot{\mathbf{q}}} + \overline{\rho} \left(\widetilde{\mathbf{u}h} - \widetilde{\mathbf{u}} \widetilde{h} \right) \right) + \overline{\boldsymbol{\tau} : \nabla \mathbf{u}} + \overline{\dot{\mathcal{Q}}}, \tag{3}$$

$$\overline{\rho} \frac{\partial \widetilde{Y_k}}{\partial t} + \overline{\rho} \widetilde{\mathbf{u}} \cdot \nabla \widetilde{Y_k} = -\nabla \cdot \left(\overline{\rho \mathbf{V_k} Y_k} + \overline{\rho} \left(\widetilde{\mathbf{u} Y_k} - \widetilde{\mathbf{u}} \widetilde{Y_k} \right) \right) + \overline{\dot{w}_k}, \tag{4}$$

where t represents time, ρ is the density of the mixture, $\mathbf{u}$ is the velocity vector, p stands for the dynamic pressure, $\mathbf{g}$ the gravity and $\boldsymbol{\tau}$ is the viscous stress tensor. In the energy conservation equation (Eq. (3)) $\dot{\mathbf{q}}$ is the conduction heat flux evaluated using the Fourier's law and $\dot{\mathcal{Q}}$ is a heat source term, including radiation. The energy equation is solved for the total enthalpy h, which includes both sensible and chemical (formation) enthalpy. Additionally, P_0 denotes the thermodynamic pressure, which is spatially constant. In Eq. (4) Y_k is the mass fraction of species k, $\mathbf{V_k}$ the diffusion velocity and $\dot{w}_k$ the mass reaction rate per unit volume of the species k. The term $\overline{\rho} \left(\widetilde{\mathbf{u}\mathbf{u}} - \widetilde{\mathbf{u}} \widetilde{\mathbf{u}} \right)$ in Eq. (2) corresponds to the SGS stress tensor. In the present work this term is closed using an eddy-viscosity-type model following the Boussinesq hypothesis [19]. Regarding the unclosed terms of the energy (Eq. (3)) and species (Eq. (4)) transport equations, these are modelled employing a gradient assumption [5].

The governing equations for the transported scalars are solved in non-conservative form due to the employed time-integration strategy, based on a predictor-corrector scheme for low-Mach number flows further detailed in the work of Ventosa et al. [20]. At each step of the predictor-corrector scheme, first the transported scalars are integrated in non-conservative form in order to compute the density at the new time step, which is afterwards employed in the integration of the momentum equation.

The mass reaction rate per unit volume of species k, $\dot{w}_k$, is the term responsible for possible stiffness when integrating Eq. (4). It is calculated as the sum of the rates $\dot{w}_{k,j}$ for all M reactions present in a chemical reaction mechanism, i.e.:

$$\dot{w}_k = \sum_{j=1}^{M} \dot{w}_{k,j}. \tag{5}$$

The calculation of these rates relies on the empirical Arrehnius law [5]. The values of the terms appearing in the Arrhenius expression are obtained from chemical schemes that are experimentally generated. Deciding which species or how many reactions should be taken into account to properly describe a chemical reaction is not a solved problem. Many different chemical mechanisms can be found in the literature. There exist very detailed mechanisms like the GRI-Mech 3.0 [21], designed to model natural gas combustion, including 53 species and 325 reactions. There are also single-step mechanisms, like the one defined by Lange et al. for methane/air flames [22]. In between, there are the reduced mechanisms, like the chemical scheme derived by Mueller et al. for hydrogen [23], which includes 9 species and 21 reactions. Obviously, as more species and reactions are included in the mechanism, more detailed results are obtained, although the calculation costs increase exponentially. Combustion simulations can be very sensitive to the chemical mechanism employed. Therefore, special care should be taken when selecting it, trying to find the best compromise between accuracy and computational cost.

3. Numerical time integration of finite-rate combustion equations

The set of equations that must be solved in order to calculate the mass reaction rate $\dot{w}_k$ in Eq. (4) are characterized by the presence of a wide range of time-scales in their variables, which may cause that this term becomes stiff. Therefore, this source term is the one requiring a special temporal integration strategy. In the explanation of the numerical time-integration strategy detailed below, Eq. (4) is expressed in its unfiltered form, in order to not haul in all the equations the filtered and density-weighted variables. The time integration strategy is the same for both filtered and non-filtered equations, and therefore, its application to LES cases is straightforward. Hence, Eq. (4) is rearranged and expressed as:

$$\rho \frac{\partial Y_k}{\partial t} = \mathbf{F}(Y_k) + \dot{w}_k, \tag{6}$$

where $\mathbf{F}(Y_k)$ includes the convective and diffusive operators of mass fraction conservation equation for species k, i.e.:

$$\mathbf{F}(Y_k) = -\rho \mathbf{u} \cdot \nabla Y_k - \nabla \cdot (\rho \mathbf{V_k} Y_k). \tag{7}$$

When the term $\dot{w}_k$ is non-stiff, both terms on the right-hand side of Eq. (6) are integrated explicitly. On the other hand, when this mass reaction rate is stiff, a splitting technique is applied. Then, the term $\mathbf{F}(Y_k)$ is integrated explicitly, while the mass reaction rate, $\dot{w}_k$, is integrated implicitly. This implicit integration is done using an integration technique well-suited for stiff equations. Among the several types of operator-splitting techniques reported in the literature [24, 25, 26, 27, 28], the one employed in the present work is based on a pseudo-time splitting procedure similar to the one employed by Vos [26] and Consul [29]. The splitting technique used in the present work has been developed for the predictor-corrector scheme detailed in previous works [20, 30]. In the predictor step scalars are integrated using a second-order Adams-Bashforth time integration scheme. On the other hand, scalars are advanced by means of a Crank-Nicolson time integration scheme in the corrector step. It can be summarized as:

Predictor step.

The mass fraction conservation equations of the N species are fully explicitly integrated in order to obtain the predicted values Y_k^*:

$$\rho^n \frac{{Y_k}^* - {Y_k}^n}{\Delta t} = \frac{3}{2}\left(\mathbf{F}(Y_k^n) + \dot{w}_k^n\right) - \frac{1}{2}\left(\mathbf{F}(Y_k^{n-1}) + \dot{w}_k^{n-1}\right), \tag{8}$$

where the value $\mathbf{F}(Y_k^n)$ is stored in memory.

Corrector step.

In the corrector step, a first value Y_k^{n+1} is calculated fully explicitly for all the nodes of the mesh, according to:

$$\rho^* \frac{Y_k^{n+1} - {Y_k}^n}{\Delta t} = \frac{1}{2}\left(\mathbf{F}(Y_k^*) + \dot{w}_k^*\right) + \frac{1}{2}\left(\mathbf{F}(Y_k^n) + \dot{w}_k^n\right). \tag{9}$$

During this integration loop, it is estimated which nodes present a stiff equation system and which not. The employed criterion is discussed later. If the node is considered non-stiff, species mass fraction equations are integrated explicitly. Then, the mass fraction value at time t^{n+1} is the one obtained from Eq. (9). On the other hand, if the node is detected as stiff, the mass reaction rate $\dot{w}_k$ is treated implicitly, then, Eq. (9) is rearranged as:

$$\rho^* \frac{Y_k^{n+1} - {Y_k}^n}{\Delta t} = \frac{1}{2}\left(\mathbf{F}(Y_k^*)\right) + \frac{1}{2}\left(\mathbf{F}(Y_k^n)\right) + \dot{w}_k^{n+1}, \tag{10}$$

which is split in two parts:

$$\rho^* \frac{Y_k^p - {Y_k}^n}{\Delta t} = \frac{1}{2}\left(\mathbf{F}(Y_k^*)\right) + \frac{1}{2}\left(\mathbf{F}(Y_k^n)\right), \tag{11}$$

$$\rho^* \frac{Y_k^{n+1} - {Y_k}^p}{\Delta t} = \dot{w}_k^{n+1}, \tag{12}$$

where Eq. (11) is integrated explicitly, since the values at the right-hand side are already known, and Eq. (12) is integrated implicitly using an integration method well-suited for stiff equation systems. The value Y_k^{n+1} obtained in Eq. (9) is employed as initial *seed* of the implicit integration method.

This pseudo-time splitting technique allows the employment of any integration method for stiff equations. The one employed in the present work is the Gear's method [16], which is the basis of the vast majority of integration methods developed and used for stiff computations [7]. The Gear's method is based on the Backward Differentiation Formulas (BDFs) coupled together with the Newton's method [8]. This method requires the evaluation of a Jacobian matrix and its inverse at each iteration of the Newton's method. In the present work, the Jacobian matrix is numerically evaluated employing a forward-difference approximation. Depending on the system size (i.e., the number of species N_s), the Jacobian evaluation plus the matrix inversion can be computationally very expensive. Therefore, although this method allows higher time-steps than explicit or classical implicit integration methods, its required computational cost per time-step is higher. Nonetheless, since the allowed time-step is higher, the method allows a remarkable reduction of the total number of iterations, becoming more suitable.

A key aspect of the method presented is the criterion to decide which control volumes (CVs) are integrated explicitly and which ones are integrated employing the implicit strategy by means of the Gear's method. This criterion can be based on a mathematical analysis of the equation system, as the methods presented by Hairer and Wanner [7] for the automatic stiffness detection. Another possibility is to calculate a time-scale for each system of equations and define as stiff the equations with a time-scale below a certain value, as done in the work of Shi et al. [13]. A third option is to define a threshold based on a physical criterion. In this work, the latter option is preferred, since its computational cost is almost negligible and the methodology does not depend on the stiff integration method selected. The idea is to define a *chemical time step* Δt_{chem} sufficient to integrate accurately the mass reaction rates using an explicit time integration method, in some sense a CFL-like condition for the chemical reactions. This *chemical time step* acts as a *detector* of the regions where the chemical process is most reactive, and it is defined as:

$$\Delta t_{chem} = f_r \frac{\rho \left| \left(h - \sum_{k=1}^{N} Y_k \Delta h_{f,k}^0 \right) \right|}{\left| \sum_{k=1}^{N} \dot{w}_k \Delta h_{f,k}^0 \right|}. \tag{13}$$

This criterion is designed trying to limit the increment of energy per unit mass, avoiding rapid increases of enthalpy and temperature, and aiming to follow all the scales of the chemical reaction process. The parameter f_r is a factor that in the present work has been set to $f_r = 5 \times 10^{-5}$. Since this criterion is a key aspect of the method, further discussion about its derivation and behaviour is provided in Appendix A.

The integration time step Δt of the simulation is limited by the requirements of accuracy and stability for both convective and diffusive terms of the governing equations integrated explicitly. This time step Δt can be obtained from the classical CFL-condition. However, a self-adaptive time strategy based on the estimation of the eigenvalues of the system of equations is employed in this work. This method is further detailed in [30]. Hence, in the corrector step of the algorithm, for each CV the *chemical time step* Δt_{chem} is estimated. If the simulation time step is bigger than it, $\Delta t > \Delta t_{chem}$, the mass reaction rates of the CV are integrated implicitly using the Gear's method. Otherwise, if the simulation time step is equal or smaller than the *chemical time step*, $\Delta t \leq \Delta t_{chem}$, all the terms are integrated explicitly.

This integration strategy allows to notably reduce the total amount of computational requirements of the simulation. Only the CVs with *active* chemical reactions are integrated implicitly using Gear's method. The vast majority of the computational domain is integrated explicitly, avoiding the significant computational effort demanded by the Gear's method, since it implies an iterative process involving evaluations of Jacobians and matrix inversions.

4. Dynamic balancing algorithm

The parallelization of the combustion solver presented in the previous section is based on a domain decomposition. The mesh that discretizes the simulation

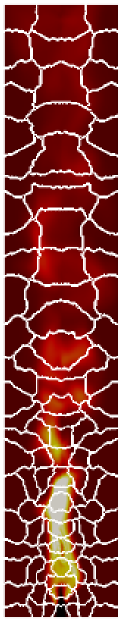


Figure 1: Slice showing a flame and the partitions of the computational mesh.

domain is partitioned into different sub-meshes and each of the resulting subdomains is solved by a different parallel process. To solve the CVs lying in the interface between two subdomains, communications are required between the associated parallel processes. This operation is referred as a halo update. The mesh partition is based on two criteria: i) minimum number of CVs in the interface between subdomains; ii) workload balance, i.e. same number of CVs on each sub-mesh. This partition problem is generally addressed by means of a graph partitioner applied to the adjacency graph. In this paper the k-way partitioning algorithm of the METIS software [31] has been used. An example of mesh partition for a combustor is illustrated in Fig. 1.

Heterogeneity, both in the mesh elements or in the algorithmic approach used across the simulation domain, complicates the fulfilment of the balancing requirement. If this heterogeneity is constant along the simulation process, i.e. neither the mesh elements nor the algorithmic approach used in each part of the domain changes along the simulation, then the partition can be adapted by using a weighted adjacency graph. However, if there is variability in any of the aforementioned aspects, a partition that has been generated *a priori* may result in an unbalanced workload distribution when the initial conditions change. This is the situation considered in this paper: the chemically reactive CVs, which evolve dynamically with the flame shape, are solved with an specific algorithm with higher computing cost. Therefore, there is a dynamism from the algorithmic side related with the physics evolution. This problem has been noted by other authors. For example, Velghe et al. [9] studied different combustion cases and asserted that the parallel efficiency cannot be addressed using static mesh partitioning, and should be tackled by means of a dynamic load balancing strategy.

Different alternatives can be considered to solve this balancing problem. As aforementioned in Section 1, some authors choose to modify the partition when the imbalance exceeds a certain threshold, either by calculating a new one or by moving cells between neighbouring subdomains [12]. This is a valid option but presents various disadvantages: needs to communicate geometrical information between processes and can be inefficient if the workload imbalance rapidly changes at each iteration, since the new *a priori* balanced partitioning is based only in information from the previous iterations. Another option is the one proposed by Shi et al. [13] for hybrid supercomputers. In their work the CVs are sorted by their stiffness. Then, the most stiff CVs are integrated implicitly by the CPUs, while the other ones are integrated explicitly on the GPUs. Their algorithm is designed to achieve a similar computing time between GPUs and CPUs at each time-step. The estimated computing speed of both CPUs and GPUs is based on the performance of the previous time-step. This option accelerates the resolution but it only targets hybrid supercomputers and inter-node balancing is hardly possible.

In this paper an original strategy based on a work redistribution process complementary to the underlying DD is presented. The idea is that in the combustion resolution the overloaded processes transfer the data needed to integrate stiff chemical reactions to the underloaded ones. The latter solve them and send the solution back to the original processes. In this approach the underlying mesh partition is not modified so other phases of the simulation are not affected. The same strategy was used by the authors in the context of multi-fluid flow simulations to overcome the imbalance produced on the reconstruction and advection of two-phase flow interfaces [32, 33].

The load balancing algorithm presented in this work consists on the five main steps outlined in the next items:

1. *Determine workloads:* each parallel process evaluates its number of stiff CVs, and a `MPI_Alltoall` communication is performed to get the workload distribution in all processes.
2. *Define a new balanced distribution:* The same sequential algorithm is run on each parallel process to determine a balanced distribution and to define the required communication scheme. Further details about this algorithm are provided in Section 4.1.
3. *Workload distribution.* All the data required to perform the calculations is packed and distributed according to the communication scheme defined in the previous step. This distribution is done by means of non-blocking point-to-point communications: `MPI_ISend`, `MPI_IRecv`.
4. *Solve stiff CVs.* Once all the data has been transfered, each process solves its assigned tasks. The external tasks, i.e. the ones initially owned by other processes are solved in first place, aiming to return the outgoing results as soon as possible to the original process (`MPI_ISend`).
5. *Collect solutions*: the processes which sent part of their tasks to others, receive the solutions back (`MPI_IRecv`) and store them in the corresponding memory space.

To summarize, the main steps of our load balancing strategy are outlined in Algorithm 1. Note that three communication episodes are required by the algorithm. First a collective communication to obtain the initial workload distribution across the processes. Then two point to point communications are needed to distribute the workload and to collect the solutions back. In the following section further details on the step 2 of the algorithm are provided.

Algorithm 1 Parallel load balancing strategy

```
1: Determine workloads
2: Define a new balanced distribution
3: Workload distribution
4: Solve stiff CVs
5: Collect solutions
```

4.1. Define a new balanced distribution

Algorithm 2 is used to define a balanced workload distribution and the corresponding communication scheme. The input of the algorithm is an array I such that $I[p]$ contains the number of initial stiff CVs owned by process p. The array I is obtained in the first step of Algorithm 1 as a result of a `MPI_Alltoall` communication. The outputs of the algorithm are the arrays $SendTo$ and $RecvFrom$ of dimension P, that contain in position p the number of tasks to send and to receive to/from process p, respectively, by the parallel process running the algorithm. Note than only one of the two output vectors will have non null values.

In the first 10 lines of Alg. 2, the arrays O, S and R of dimension P are evaluated. $O[p]$ are the number of originally owned tasks to be solved by the pth parallel process; and $S[p]$ and $R[p]$, the number of owned tasks to be sent and the number of external tasks to be received by/from the p'th parallel process, respectively. Finally, at line 11, the number of tasks to be reassigned, N_{re}, is evaluated as $N_{re} = \sum_p S[p]$, which equals $\sum_p R[p]$.

In the next loop of the algorithm, which covers lines 12-25, the arrays $SentTask$ and $RecvTask$ of dimension N_{re} are evaluated. In the i'th position, these contain the rank of the process sending and receiving the i'th reassigned task, respectively. Note that this reassignment is independent of the underlying partition.

Finally, once the “sender” and “receiver” of each reassigned task are determined, the evaluation of the vectors $SendTo$ and $RecvFrom$, which determine the communication operations for the process running the algorithm, is straightforward. This is performed in the last loop of the algorithm, corresponding to lines 26-34. The cost of this sequential algorithm, that is replicated in each processor involved in the parallel computation, is almost negligible.

5. Performance analysis

In this section the performance and scalability of the balancing algorithm is analysed. In order to do so, two computing experiments considering the main parameters affecting the efficiency of the algorithm have been carried out. Specifically, the parameters considered in these studies have been: (i) the ratio between the cost of the *heavy calculation* and the size of the message sent per each *heavy node*, and (ii) the imbalance ratio of the parallel computation. Note that *heavy node* denotes the nodes presenting a higher computational load than the average in the domain. *Heavy calculation* refers to the set of mathematical operations resulting in this additional load.

In order to carry out these studies it has been developed a simulation with n_{cpu} nodes per processor, where N_{hn} processors of the N CPUs involved in the parallel simulation have *heavy nodes*. These processors have a variable number of *heavy nodes* (hn_{cpu}) defined by a ratio θ_{hn}, i.e. $hn_{cpu} = \theta_{cpu} n_{cpu}$. The ratio of processors presenting *heavy nodes* is θ_N ($N_{hn} = \theta_N N$). Hence, the total number of nodes in a simulation is $n_t = n_{cpu} N$, and the total number of *heavy nodes* is $hn_t = hn_{cpu} N_{hn} = \theta_{cpu} n_{cpu} \theta_N N$.

The *heavy calculation* performed by the *heavy nodes* consists in a matrix inversion plus a numerical Jacobian evaluation of a system with size hc_{ss} repeated hc_{it} times, mimicking the Newton’s method. The size of the message communicated between processors for each *heavy node* consists in ms_{hn} *doubles*.

All the studies have been carried out in the MareNostrum IV supercomputer, hosted by the Barcelona Supercomputing Center (BSC). MareNostrum is based

Algorithm 2 Define a new balanced distribution and the corresponding communication scheme

```
1: $N = \sum_{p=0}^{P} I[p]$
2: $I_{opt} = \lceil \frac{N}{P} \rceil$
3: for  $0 \leq p < P$ do
4:     $O[p] = I_{opt}$
5:     if $P - p \leq PI_{opt} - N$ then
6:         $O[p] = O[p] - 1$
7:     end if
8:     $S[p] = min(0, I[p] - O[p])$
9:     $R[p] = min\,(0, O[p] - I[p])$
10: end for
11: $N_{re} = \sum_p S[p]$
12: $cont\_send = cont\_recv = 0$
13: for $0 \leq p < P$ do
14:     if $S[p] > 0$ then
15:         for $0 \leq i < S[p]$ do
16:             $SentTask[cont\_send] = p$
17:             $++\,cont\_send$
18:         end for
19:     else
20:         for $0 \leq i < R[p]$ do
21:             $RecvTask[cont\_recv] = p$
22:             $++\,cont\_recv$
23:         end for
24:     end if
25: end for
26: rg = rang of the executing parallel process
27: for $0 \leq i < N_{re}$ do
28:     if $SendTask[i] == rg$ then
29:         $++\,SendTo[RecvTask[i]]$
30:     end if
31:     if $RecvTask[i] == rg$ then
32:         $++\,RecvFrom[SentTask[i]]$
33:     end if
34: end for
```

on Intel Xeon Platinum processors, Lenovo SD530 Compute Racks, a Linux Operating System and an Intel Omni-Path interconnection [34]. The studies have been done from 240 up to 1920 CPUs. The studies analyses the Speed-up ratio (S_r) obtained in the simulations when using the presented dynamic balancing algorithm. S_r is defined as the computing time of an unbalanced simulation (t_{unb}) divided by the computing time of a simulation using the load balancing method (t_{bal}):

$$S_r = \frac{t_{unb}}{t_{bal}}. \tag{14}$$

Hence, S_r will provide a value showing how many times faster is a computation using the load balancing algorithm than the same one not employing it.

The first parametric study has analysed the impact of the ratio between the computing cost of the *heavy calculation* and the size of the message sent per each *heavy node*. In order to characterise this ratio it has been defined the parameter ζ as:

$$\zeta = \frac{hc_{ss}hc_{it}}{ms_{hn}}, \tag{15}$$

where the numerator takes into account the computing cost of the *heavy calculation* while the denominator accounts for the communication effort. Therefore, high values of ζ stand for cases where the computing cost of the *heavy calculation* is big and the communicated message size small. On the other hand, small values of ζ represent cases with a large number of communicated values and *heavy calculations* with a small computing cost.

The set-up for the simulations carried out in this study is as follows: each processor has $n_{cpu} = 200$ nodes, and the percentage of processors with *heavy nodes* is 25 %, hence $\theta_N = 0.25$. The number of *heavy nodes* per CPU is set to $hn = 100$, i.e. $\theta_{cpu} = 0.5$. The studies were done for several values of ζ, ranging from $\zeta = 0.01$ up to $\zeta = 100$. The results are shown in Fig. 2. As can be seen, for higher values of ζ the obtained S_r is better than for lower values of ζ. This is expected, since high values of ζ result in both more imbalance, i.e. larger potential benefits of the balancing process; as well as lower overhead of communications, as the resulting ratio of flops per byte transferred through the network is higher. Nonetheless, for all the cases there is a clear advantage in employing the load balancing algorithm, since all the simulations using it are accelerated, at least by a factor of 1.75. Notice that for higher ζ values the Speed-up ratio

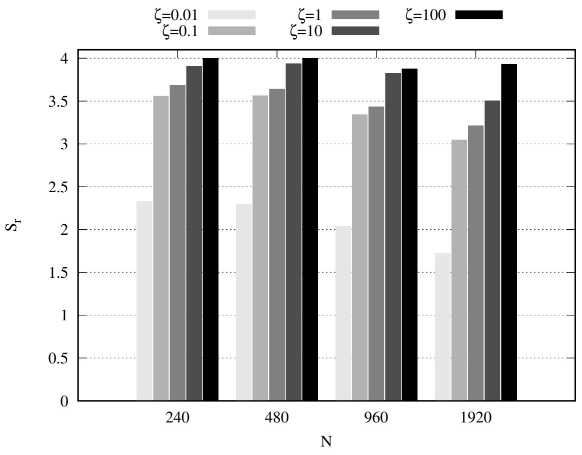


Figure 2: Speed-up ratio (S_r) for different ratios between the computational cost and the size of the message sent by *heavy nodes* (ζ) depending on the number of processors N.

S_r decreases less when increasing the number of CPUs (N) than for small values of ζ. This is because the latter cases imply a higher communication cost with respect to the computing cost of the *heavy calculation*. Additionally, when increasing the number of CPUs of the simulation more communications have to be performed, reducing the Speed-up ratio S_r.

Regarding the second parametric study, it has been analysed the impact of the load imbalance in the performance of the presented algorithm. In order to modify the imbalance on the simulations it has been defined that the total number of *heavy nodes* in the simulation was the 25 % of the total number of nodes, i.e. $hn_t = 0.25n_t$. These *heavy nodes* have been assigned to the different CPUs involved in the computation following a continuous uniform distribution, from the most unbalanced situation, where all the *heavy nodes* are located in the minimum number of CPUs, up to a perfectly balanced simulation, where the *heavy nodes* are uniformly distributed among all the CPUs. In order to quantify this imbalance it has been defined the parameter Θ, ranging from $\Theta = 0$ for the most unbalanced situation up to $\Theta = 1$ for the fully balanced case. A representation of the studied *heavy nodes* distributions is given in Fig. 3. The y-axis shows the ratio of *heavy nodes* per CPU (hn_{cpu}/n_{cpu}). The x-axis

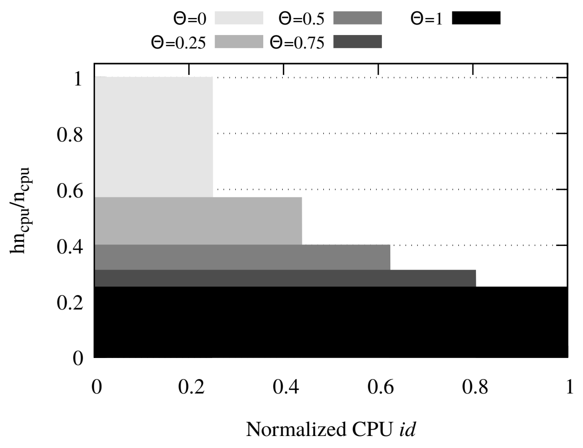


Figure 3: Representation of the heavy node distribution among the CPUs depending on parameter Θ.

corresponds to the normalized CPU *id*. The other parameters of the simulation have been fixed and set to $n_{cpu} = 200$, $hc_{ss} = 5$, $hc_{it} = 5$, and $ms_{hn} = 10$, giving a value of $\vartheta = 2.5$. The results obtained for this study are depicted in Fig. 4. The higher the imbalance, the better the Speed-up ratio S_r. For all the range of Θ, it is observed that the Speed-up ratio S_r slightly decreases when N increases, due to the increment in the number of communications. Since cases with $\Theta = 1$ are perfectly balanced, the algorithm cannot redistribute the load to speed-up the simulation. Hence, ideally a Speed-up ratio $S_r = 1$ is expected. However, values of S_r slightly below unity are observed. This is due to the overhead caused by the load balancing algorithm calculating a new balanced distribution (step 2 of Algorithm 1) on simulations that are already balanced.

Thus, the studies carried out in this section demonstrate the capability of the algorithm to dynamically balance parallel simulations presenting an unbalanced computational load. As observed, this algorithm allows to speed-up the unbalanced simulations. The value of this Speed-up S_r will depend on the conditions of the simulation. Better Speed-up values will be obtained the more unbalanced the workload is, as well as the higher is the ratio between the computing cost of the *heavy calculation* and the size of the message sent by each *heavy node*.

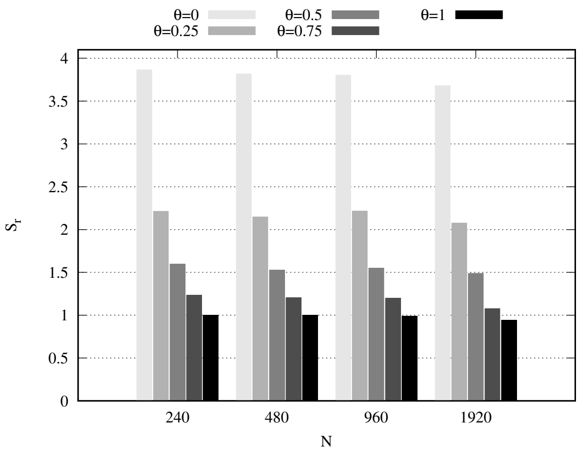


Figure 4: Speed-up ratio (S_r) for different imbalance ratio (Θ) and number of processors N.

6. Benchmark test case: the Cambridge autoignition experiment

After the study carried out in order to analyse the performance and capabilities of the load balancing algorithm developed, this one is tested on a combustion simulation. The reference case chosen is the well-known Cambridge autoignition experiment performed by Markides and Mastorakos [17]. The test case consists of a fuel jet, a mixture of hydrogen H_2 and nitrogen N_2, flowing through a nozzle located at the centre of a co-flowing air stream (see Fig. 5). The air is preheated at different temperatures in order to study its impact in the autoignition behaviour of hydrogen under this geometrical configuration. The co-flowing air is forced to pass through a perforated plate to generate turbulence. The perforated plate (3.0 mm holes and 44 % blockage) is located 63 mm upstream of the fuel nozzle in order to allow turbulence to develop. The fuel nozzle has a diameter of 2.25 mm and is thin-walled (0.32 mm). The main test section consists of a 500 mm long and 25 mm inner diameter vacuum insulated quartz tube.

The experiment was performed over a wide range of operating conditions. Four regimes were identified, namely *no ignition*, *random spots*, *flashback* and *lifted flame*. In the present work, the case with the following operating condi-

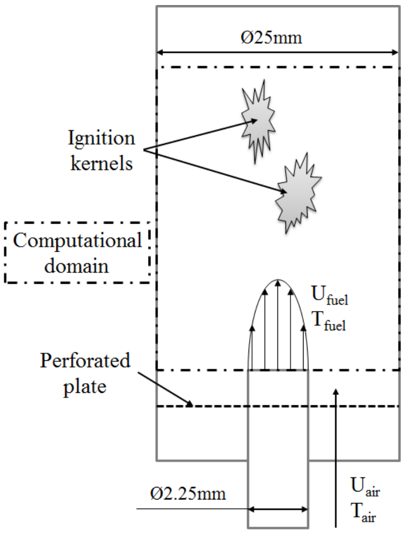


Figure 5: Schematic representation of the experiment presented by Markides and Mastorakos [17].

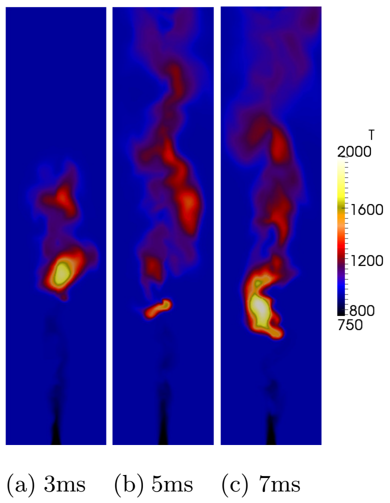


Figure 6: Temperature field at different times in the simulation with $T_{air} = 950$ K.

tions has been chosen: $U_{jet} = 26\ \mathrm{m\,s^{-1}}$, $U_{air} = 26\ \mathrm{m\,s^{-1}}$, $T_{jet} = 750$ K and $T_{air} = 950$ K. For this case, the fuel is a mixture of H_2 and N_2 ($Y_{H_2} = 0.13$, $Y_{N_2} = 0.87$) and the co-flow oxidizer is air ($Y_{O_2} = 0.233$, $Y_{N_2} = 0.767$). The selected case belongs to the *random spots* regime, characterised by the appearance of auto-ignition kernels that are quenched before they could act as a flame anchoring point or a flashback, and are convected out of the domain. This case is chosen because the *random spots* regime is the most challenging one for the workload distribution. This regime is *chaotic* and does not allow to know *a priori* the regions where combustion will occur. Moreover, the workload of the CPUs will change and evolve dynamically during the simulation. The temperature field at three different time instants of the numerical simulation are shown in Fig. 6. In these figures can be clearly seen the described behaviour of the *random spots* regime, with the appearance and quenching of the auto-ignition kernels.

6.1. Simulation set-up

The simulations have been carried out employing the finite-rate chemistry model under the framework of LES modelling. In the present study the Wall-adapting eddy viscosity model (WALE) SGS model [35] is used. The WALE SGS model is based on the square of the velocity gradient tensor. The SGS viscosity obtained with this model takes into account the strain and the rotation rate of the smallest resolved turbulent fluctuations. Some features of this model are its capability of switching off in two-dimensional flows, in laminar flows, and when the length-scale is in the range of $\mathrm{Re}^{-3/4}$. The unresolved scalar fluxes of both energy and species transport equations are modelled using a gradient assumption [5], where a turbulent Prandtl number ($\mathrm{Pr_t}$) and a turbulent Schmidt number ($\mathrm{Sc_t}$) are assumed for the energy and species conservation equations respectively. The values $\mathrm{Pr_t} = 0.4$ and $\mathrm{Sc_t} = 0.4$ have been employed in the present work.

For the current case a perfect mixing at SGS is considered, assuming that the turbulent subgrid time-scale is shorter than the time-scales of the chemical reactions ($\tau_\mathrm{t} \ll \tau_\mathrm{c}$). This assumption is supported in the relatively low-Reynolds number of the case and in the results obtained in the previous study presented by Muela et al. [36]. This approach means that the subgrid chemistry-flow interaction is not modelled, and therefore $\overline{\dot{w}_k} \approx \dot{w}_k$.

The chemical reactions are modelled employing the detailed reaction mechanism for hydrogen of 9 species and 21 reactions developed by Mueller et al. [23].

The pressure-velocity coupling is solved by means of the Fractional Step projection method [37]. The idea behind this technique is to split the momentum in two steps, with a first explicit step where an intermediate velocity $\hat{\mathbf{u}}$ is obtained, followed by a second step where the pressure is solved implicitly and the intermediate velocity is corrected obtaining the physical velocity.

In order to reproduce the turbulence generated by the perforated plate, an auxiliary non-reactive simulation is performed in an annular mesh, recreating the physical domain upstream to the injector lips, where the plate is placed inside the domain using the immerse boundary technique [38]. The unsteady velocity field generated 63 mm downstream of the perforated plate is saved, and later injected in the simulated domain through the co-flowing section. This method allows a significant saving of computational resources during simulation time, and develops a realistic divergence-free velocity field. For the fuel inflow a laminar parabolic velocity profile is employed with $U_{mean} = U_{fuel}$. For the walls, a free-slip boundary condition is used. Therefore, the flow is not well resolved near the wall, but since all the phenomena of interest take places far from the wall, a well resolved shear-layer near the wall is not of interest [39]. Regarding the outlet, a reference atmospheric pressure has been fixed.

The mesh employed for this case has been generated from a 2D plane normal to the streamwise direction of the flow and extruded in this direction. The 2D mesh is unstructured, constructed with 16660 triangular elements, refined in the inner jet and the air-fuel shear layer regions, and coarsened close to the walls. This 2D mesh has been extruded in 675 planes with a height of 0.2 mm each one, giving a total number of more than $11M$ CVs.

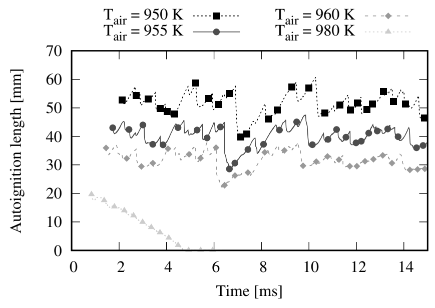
Two studies have been carried out: (i) a strong speed-up test analysing the scalability of the algorithm in a realistic combustion simulation, and (ii) a test comparing two simulations with the same set-up, one employing the dynamic load balancing algorithm and a second one which do not employ it, aiming to study the Speed-up ratio S_r obtained employing the algorithm. In this case a strong speed-up test is employed to study the scalability of the algorithm. A weak speed-up test will imply modify the number of CVs for the different meshes. Since the size of the simulated domain can not be modified (is the one of the experiment), modifying the number of CVs will imply a change in their size, affecting the resolved physics of the problem, and hence, generating differences in the results.

One relevant aspect for the dynamic balancing algorithm is the size of the message that is sent between the processors when distributing the tasks, as well as when recollecting the *outsourced* solutions. Each node that delegates the implicit integration to an external processor sends a message of $4+2N_s$ *doubles*, where N_s is the number of species. This message contains the size of the buffer sent per each node, an ID identifying the node sending the info, the density ρ, the temperature T, and values Y_k^{n+1} plus ${Y_k}^p$, where the former is employed as first guess in the Newton's method. Regarding the recollection step, the size of the message is of $3 + N_s$ *doubles*, including the ID of the node that request the solution, the integrated values Y_k^{n+1}, and two auxiliary values indicating the number of iterations and Jacobian evaluations performed by Gear's method to converge the solution. Although in this case some of the values can be treated as *integers* instead of *doubles*, helping to reduce the message size, the *heart* of the balancing algorithm has been developed seeking generality. Therefore, in this first version all the communicated values are considered as *doubles.* However, it is an aspect that can be improved in upcoming versions of the algorithm.

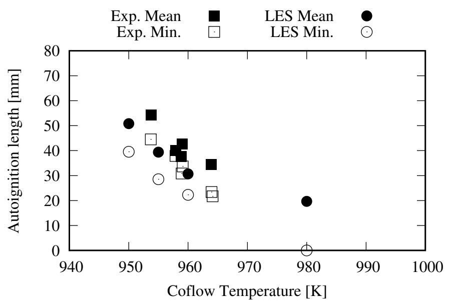
6.2. Results and discussion

Before presenting the results obtained for both studies, some illustrative results of this benchmark combustion case obtained employing the described set-up are presented. The autoignition length obtained for four different temperatures of the co-flowing air stream are depicted in Fig. 7a. The four co-flowing air temperatures are 950 K, 955 K, 960 K and 980 K. The ignition length is determined using as ignition criterion a rise of a 1 % over the initial co-flow temperature. In the three cases with a co-flowing air temperature below $T_{air} = 960$ K, the ignition length oscillates around a mean. This is caused by the appearance of random auto-ignition kernels, which are quenched and convected out of the domain before they can act as an anchoring point or derive in a flashback. This process where an ignition kernel appear, briefly grows, and then is quenched is repeated periodically along the simulation time, resulting in the oscillatory auto-ignition length. Thus, these three cases clearly belong to the *random spots* regime previously described. In contrast, for $T_{air} = 980$ K, once ignited, the ignition length decreases progressively as the flame propagates upstream. Therefore, this case falls inside the *flashback* regime.

These results are in agreement with the ones obtained in the experiment carried out by Markides and Mastorakos [17] where, for the conditions reproduced



(a) Evolution of ignition lenghts.

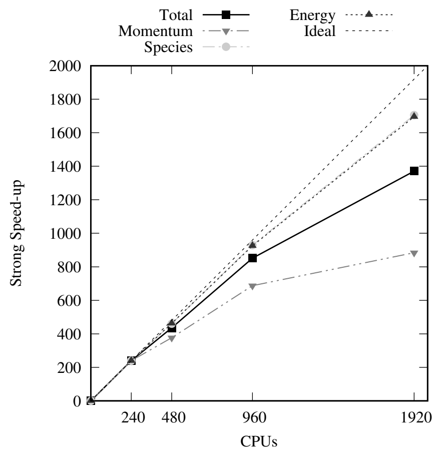


(b) Mean and minimum ignition lengths.

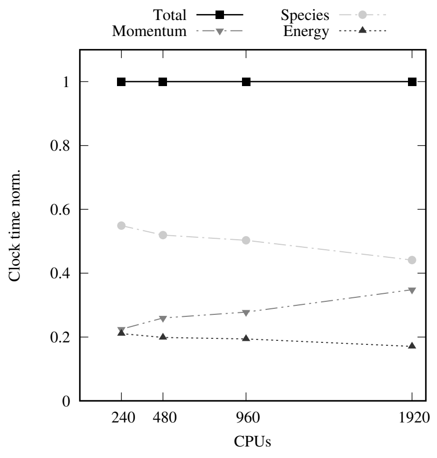
Figure 7: Results for the Cambridge autoignition experiment.

in the present work, the boundary between both regimes was found around a co-flow temperature of 965 K. Hence, the simulations reproduce the physics behind the hydrogen autoignition process taking place in this configuration. In Fig. 7b, the mean and minimum autoignition lengths obtained in the simulations are summarized and compared against the results obtained in the experiment (since case T_{air} = 980 K is a *flashback* flame, the mean autoignition length plotted is the position where the first autoignition event takes place). As can be seen, the obtained results are close to the experimental ones, although the auto-ignition mean and minimum lengths are slightly under-predicted.

The strong speed-up for the momentum, species and energy solvers, as well as the one of the overall simulation are depicted in Fig. 8a. The scalability of both energy and species solvers is similar and very good. However, the global scalability is affected by the momentum solver, which presents a lower scalability. This is due to the solver for the Poisson equation. In this strong speed-up test, the number of CVs per CPU range from $\sim$ 47000 for the case in 240 CPUs to $\sim$ 6000 for simulations in 1920 CPUs. These ratios of CVs per CPU are small and, consequently, the communications overhead becomes more important than the time spent in the solution of the Poisson equation. In order to have a good strong scalability larger workloads per CPU are required. A detailed study of the scalability of TermoFluids code on up to 130K CPU-cores can be found in



(a) Strong speed-up.



(b) Normalized elapsed time.

Figure 8: Results for the transport equations in the strong speed-up test.

[40]. This issue of a low parallel scalability for small ratios of CVs per processor has also been reported in other works and CFD codes, as the one presented by Velghe et al. [9]. Nevertheless, the focus of the current work is placed on the species solver, which exhibits a very good speed-up scalability. Aiming to see the relevance of each solver regarding the total computing time, the normalized elapsed time spent solving each transport equation, as well as the total time of the simulation, are depicted in Fig. 8b. Each simulation has been normalized by its total time. Obviously, the relative cost of the solvers with better scalability (species and energy) reduces, while the relative cost of the momentum solver, with lower parallel efficiency, grows.

Next, the focus is placed on the species solver and specially on the computational step involving the implicit integration of the mass reaction rates. The total time has been split in two parts: one involving the calculations (i.e., the time spent doing the implicit integration by means of the Gear's method), and a second part accounting for the communication of the data and its buffering (although the latter is very small compared to the time spent in communications). Figure 9 shows the evolution of the normalized time spent on each part with

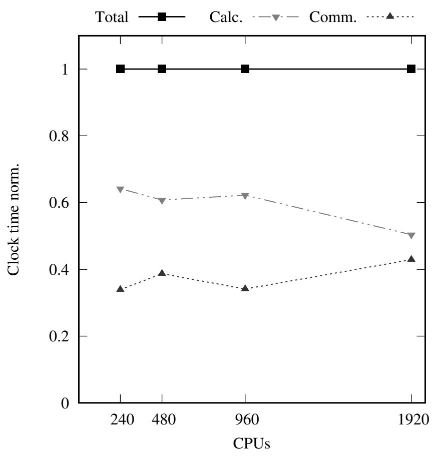
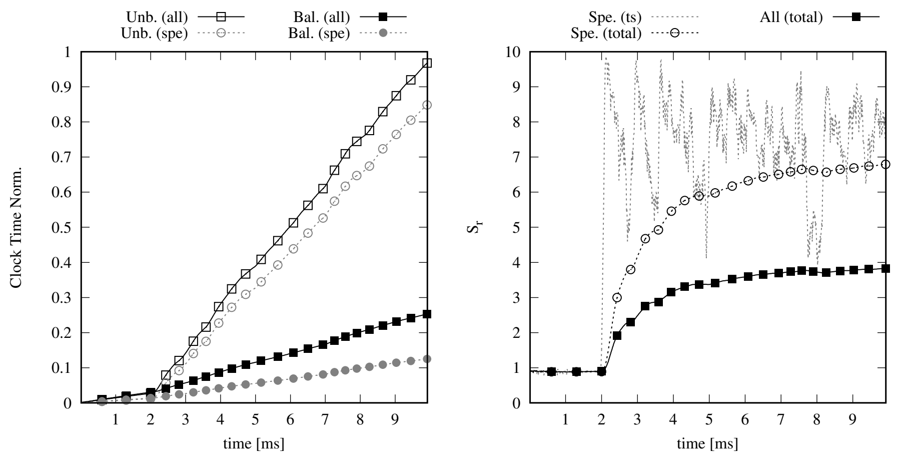


Figure 9: Normalized elapsed time in the strong speed-up test of the implicit integration stage.

the number of CPUs. It can be appreciated a trend that when the number of CPUs increases, more time is devoted to the communications while less time is spent in the calculations. Since the developed load balancing algorithm is able to redistribute the computing load equitably between all the processors, the total amount of calculations are done faster. However, since more processors are involved, the cost in the communications is increased.

Regarding the second study, which compares two simulations with the same set-up but where one employs the load balancing algorithm and a second one which not, the results are depicted in Fig. 10. This study was carried employing 960 CPUs. The evolution of the normalized clock time accumulated by each simulation versus the physical time is plotted in Fig. 10a. The figure shows both the time spent just by the species solver, including the evaluation of the species mass reaction rates, as well as the time spent by the whole simulation, involving all the solvers. It can be seen that the balanced simulation clearly outperforms the unbalanced one, demonstrating the great advantage that comes when the dynamic balancing algorithm is used. But the figure also allows to see other very interesting aspects. As previously explained, the solved case belongs to the *random spots* regime, where auto-ignition kernels appear but are quenched before they could act as a flame anchoring point or a flashback, and are convected out of the domain. It can be seen that up to time ~ 2.1 ms both balanced and unbalanced simulations take a very similar computing time. This is because before this time-step there is no combustion, and only when auto-ignition occurs and the first ignition kernel appears, the simulation begins to have unbalanced CPUs due to the presence of *active* chemical reactions. Notice that the slope of the curve belonging to the balanced simulation is almost equal before and after the first auto-ignition event, i.e., the dynamic balancing algorithm is able to redistribute the additional computing load very efficiently. On the other hand, when the auto-ignition kernels begin to appear, unbalanced simulation become clearly slower. This behaviour can be more clearly seen in Fig. 10b, where three Speed-up ratios S_r are shown. Two of them analyse the S_r obtained specifically by the species solver and the other one shows the S_r achieved by the whole simulation, i.e. accounting for all the solvers. The Speed-up ratios S_r labelled as *total* are obtained taking into account the total elapsed computing time till the i time-step, while the one labelled as *ts* is calculated using only the computing time spent in each i time-step. Up to time $\sim$ 2.1 ms, where there is no combustion and does not exist load imbalance in the simulation, the three S_r present values very close to unity. However, once the first autoignition event takes place, the Speed-up ratio S_r for each i time-step suddenly increase, with values ranging between 4 and 10, depending on the imbalance. The higher the imbalance, the higher the S_r obtained. Moreover, it can be appreciated a relation between this value and the slopes of the curves belonging to the unbalanced case shown in Fig. 10a. Higher values of S_r for i time-step match with more pronounced slopes in Fig. 10a, while smaller values of S_r coincides with decreases in the slopes. These changes in S_r and the slopes are due to the behaviour of the *random spots* regime, where ignition kernels appear but are quenched after a short living time. Higher values of S_r and more pronounced slopes are related to the presence of ignition kernels, while reductions in S_r and the slopes occur after the kernels are quenched and before the ignition of a new combustion kernel. Regarding the *total* Speed-up ratio, it takes values very close to $S_r \approx 7$ for the species solver and $S_r \approx 4$ for the whole simulation



(a) Normalised computing time. (b) Speed-up ratio S_r.

Figure 10: Comparison between an unbalanced and a balanced simulation in 960 CPUs.

in time 10 ms. This means that the balanced case achieved this physical time 4 times faster than the unbalanced computation.

7. Conclusion

A dynamic balancing algorithm well suited for parallel numerical simulations where the computational load of the processors is unbalanced has been presented and assessed. Typically, the partitioning of the meshes for parallel simulations is done assuming a uniform distribution of the computational load assigned to each node where the discretized equations are solved. Nevertheless, in some cases this assumption is not true. If the computational load distribution is known *a priori*, the partitioning of the mesh can be done using this information. However, many times this information is not known before the simulation, and other times the computational load of the nodes changes throughout the simulation, resulting impossible a proper partitioning of the mesh. The implemented dynamic balancing algorithm aims to help in these cases.

In the present work, the dynamic balancing algorithm has been adapted and employed for combustion simulations. Specifically, it has been used to properly distribute the additional computational load that appears due to the implicit integration of the mass reaction rates of the species transport equations. This species mass reaction rate is calculated from a stiff set of equations that requires a special implicit integration method, which creates an imbalance in the simulation.

A deep analysis of the performance of the algorithm has been presented, demonstrating its capacity to properly distribute the computational load of unbalanced simulations for different situations. Moreover, the dynamic balancing algorithm has been tested in a reference combustion case, the well-known Cambridge autoignition experiment. The results show a good scalability of the method, in agreement with the performance analysis carried out in the previous section. Moreover, a comparison between a balanced and an unbalanced simulation has been presented. The results of this comparison show that the dynamic load balancing algorithm allows notably reducing the computing time in parallel simulations. Specifically, for the studied case, the presented methodology allowed to speed up the simulation by a factor of 4. Nevertheless, as showed in the present work, the capacity of the load balancing algorithm to speed up the simulations will depend on their imbalance. The higher the imbalance, the greater the acceleration.

This dynamic balancing algorithm has been designed seeking generality and not case-specificity. Therefore, it can be employed for all kind of parallel simulations presenting unbalanced computational loads, and not only combustion simulations. Then, some future works can involve the adaptation of the algorithm to other physics.

Acknowledgements

This work has been financially supported by the Ministerio de Economía y Competitividad, Spain (ENE2017-88697-R). We acknowledge PRACE for awarding us access to resource MareNostrum IV based in Spain at Barcelona Supercomputing Center.

References

[1] J. Janicka and A. Sadiki. Large eddy simulation of turbulent combustion systems. *Proceedings of the Combustion Institute*, 30(1):537 – 547, 2005.

[2] R.W. Bilger, S.B. Pope, K.N.C. Bray, and J.F. Driscoll. Paradigms in turbulent combustion research. *Proceedings of the Combustion Institute*, 30:21–42, 01 2005.

[3] Heinz Pitsch. Large-eddy simulation of turbulent combustion. *Annual Review of Fluid Mechanics*, 38(1):453–482, 2006.

[4] Denis Veynante and Luc Vervisch. Turbulent combustion modeling. *Progress in Energy and Combustion Science*, 28(3):193 – 266, 2002.

[5] Thierry Poinsot and Denis Veynante. *Theoretical and numerical combustion.* RT Edwards, Inc., 2005.

[6] Federico Perini, Emanuele Galligani, and Rolf D Reitz. An analytical jacobian approach to sparse reaction kinetics for computationally efficient combustion modeling with large reaction mechanisms. *Energy & Fuels*, 26(8):4804–4822, 2012.

[7] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems.* Lecture Notes in Economic and Mathematical Systems. Springer, 1993.

[8] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, 2007.

[9] A. Velghe, N. Gillet, and J. Bohbot. A high efficiency parallel unstructured solver dedicated to internal combustion engine simulation. *Computers and Fluids*, 45(1):116–121, 2011.

[10] David J. Torres, Yuanhong H. Li, and Song-Charng Kong. Partitioning strategies for parallel KIVA-4 engine simulations. *Computers and Fluids*, 39(2):301 – 309, 2010.

[11] David J. Torres and Mario F. Trujillo. Kiva-4: An unstructured ALE code for compressible gas flow with sprays. *Journal of Computational Physics*, 219(2):943 – 975, 2006.

[12] D. Thévenin, F. Behrendt, U. Maas, B. Przywara, and J. Warnatz. Development of a parallel direct simulation code to investigate reactive flows. *Computers and Fluids*, 25(5):485–496, 1996.

[13] Yu Shi, William H Green, Hsi-Wu Wong, and Oluwayemisi O Oluwole. Accelerating multi-dimensional combustion simulations using gpu and hybrid explicit/implicit ode integration. *Combustion and Flame*, 159(7):2388–2397, 2012.

[14] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[15] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, 4th edition, 2013.

[16] C. William Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1971.

[17] C.N. Markides and E. Mastorakos. An experimental study of hydrogen autoignition in a turbulent co-flow of heated air. *Proceedings of the Combustion Institute*, 30(1):883–891, 2005.

[18] A. Favre. Turbulence - Space-time statistical properties and behavior in supersonic flows. *Physics of Fluids*, 26:2851–2863, October 1983.

[19] P. Sagaut. *Large Eddy Simulation for Incompressible Flows: An Introduction.* Scientific Computation. Springer, 2006.

[20] J. Ventosa-Molina, J. Chiva, O. Lehmkuhl, J. Muela, C. D. Pérez-Segarra, and A. Oliva. Numerical analysis of conservative unstructured discretisations for low mach flows. *International Journal for Numerical Methods in Fluids*, 84(6):309–334, 2017.

[21] Gregory P. Smith, David M. Golden, Michael Frenklach, Nigel W. Moriarty, Boris Eiteneer, Mikhail Goldenberg, C. Thomas Bowman, Ronald K. Hanson, Soonho Song, William C. Gardiner, Vitali V. Lissianski, and Zhiwei Qin. GRI-Mech 3.0. `http://www.me.berkeley.edu/gri-mech/`. Accessed: 13/09/2017.

[22] H.C. de Lange and L.P.H. de Goey. Two-dimensional methane/air flame. *Combustion science and technology*, 92(4-6):423–427, 1993.

[23] M.A. Mueller, T.J. Kim, R.A. Yetter, and F.L. Dryer. Flow reactor studies and kinetic modeling of the H2/O2 reaction. *International Journal of Chemical Kinetics*, 31(2):113–125, 1999.

[24] J.A. Miller and R.J. Kee. Chemical nonequilibrium effects in hydrogen-air laminar jet diffusion flames. *The Journal of Physical Chemistry*, 81(25):2534–2542, 1977.

[25] R.J. Kee and J.A. Miller. A split-operator, finite-difference solution for axisymmetric laminar-jet biffusion flames. *AIAA Journal*, 16(2):169–176, 1978.

[26] J.B. Vos. Calculating turbulent reacting flows using finite chemical kinetics. *AIAA journal*, 25(10):1365–1372, 1987.

[27] P.J. Coelho and J.C.F. Pereira. Calculation of a confined axisymmetric laminar diffusion flame using a local grid refinement technique. *Combustion science and technology*, 92(4-6):243–264, 1993.

[28] O. Holm-Christensen, I.P. Jones, N.S. Wilkes, B.A. Splawski, and P.J. Stopford. The solution of coupled flow and chemistry problems. *Progress in Computational Fluid Dynamics, an International Journal*, 1(1-3):43–49, 2001.

[29] R. Cònsul, C.D. Pérez-segarra, K. Claramunt, J. Cadafalch, and A. Oliva. Detailed numerical simulation of laminar flames by a parallel multiblock algorithm using loosely coupled computers. *Combustion Theory and Modelling*, 7(3):525–544, 2003.

[30] Jordi Muela. *Modelling and numerical simulation of combustion and multi-phase flow using finite volume methods on unstructured meshes*. PhD thesis, Universitat Politècnica de Catalunya, 2018.

[31] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[32] Lluís Jofre, Ricard Borrell, Oriol Lehmkuhl, and Assensi Oliva. Parallel load balancing strategy for volume-of-fluid methods on 3-d unstructured meshes. *Journal of Computational Physics*, 282:269–288, 2015.

[33] Lluís Jofre, Oriol Lehmkuhl, Jesús Castro, and Assensi Oliva. A 3-D Volume-of-Fluid advection method based on cell-vertex velocities for unstructured meshes. *Computers & Fluids*, 94:14–29, 2014.

[34] MareNostrum IV (2017) System Architecture (BSC). `https://www.bsc.es/marenostrum/marenostrum/technical-information`. Accessed: 13/03/2019.

[35] Franck Nicoud and Frédéric Ducros. Subgrid-scale stress modelling based on the square of the velocity gradient tensor. *Flow, turbulence and Combustion*, 62(3):183–200, 1999.

[36] J. Muela, O. Lehmkuhl, A. Oliva, and J. Ventosa-Molina. Large eddy simulation of hydrogen autoignition in a preheated turbulent co-flow. In *Proceedings of the 8th Mediterranean Combustion Symposium (MCS8)*, 8-13 September 2013. Cesme-Izmir, Turkey.

[37] J. Kim and P. Moin. Application of a fractional-step method to incompressible Navier–Stokes equations. *Journal of Computational Physics*, 59(2):308–323, 1985.

[38] Rajat Mittal and Gianluca Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.

[39] W.P. Jones and S. Navarro-Martinez. Study of hydrogen auto-ignition in a turbulent air co-flow using a Large Eddy Simulation approach. *Computers and Fluids*, 37(7):802–808, 2008.

[40] R. Borrell, J. Chiva, O. Lehmkuhl, G. Oyarzun, I. Rodríguez, and A. Oliva. Optimising the termofluids CFD code for petascale simulations. *International Journal of Computational Fluid Dynamics*, 30(6):425–430, 2016.

Appendix A. Stiff cells detector

The criterion employed in order to decide which cells are integrated implicitly is detailed in Section 3. The criterion is defined through an estimated *chemical time step* Δt_{chem} which is calculated according to Eq. (13). The mathematical form of this *chemical time step* is derived from the sensible enthalpy transport equation applied to a $0D$ reactor [5, p. 18]:

$$\rho \frac{dh_s}{dt} = \dot{\omega}_T = -\sum_{k=1}^{N} \Delta h^0_{f,k} \dot{\omega}_k. \tag{A.1}$$

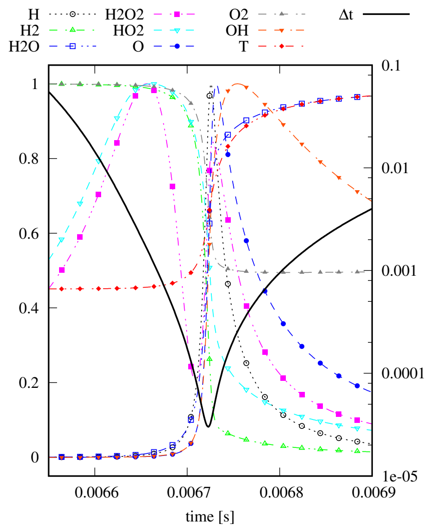
Then, approximating the derivative in the left hand-side as $\frac{dh_s}{dt} \approx \frac{\Delta h_s}{\Delta t}$ and isolating Δt the following expression is obtained:

$$\Delta t = \frac{\rho\,|\Delta h_s|}{\left|\sum_{k=1}^{N} \Delta h^0_{f,k}\dot{\omega}_k\right|}. \tag{A.2}$$

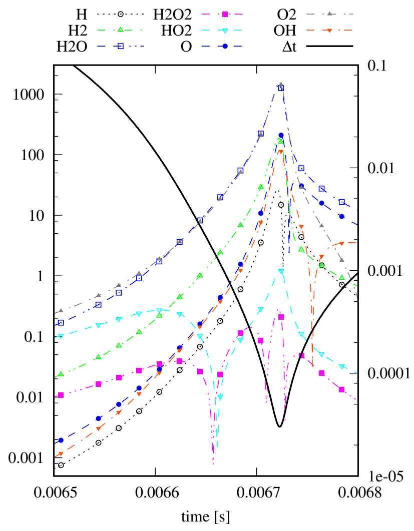
Since Δt is positive by default, the absolute values of both numerator and denominator are taken. It is expected that this expression gives an estimation of the time-scale of the energy release due to chemical reactions. In it, the denominator detects the chemically active regions, while the numerator brings the scaling. In chemically active regions, the value of the denominator increases, leading to a decrease in the estimated chemical time-scale Δt_{chem} and thus, allowing the algorithm to detect that an implicit treatment is required.

Aiming to illustrate the behaviour of this expression as detector of the chemical time-scales, the results obtained in a constant pressure zero-dimensional reactor are shown below. The employed chemical reaction mechanism is the one derived by Mueller et al. [23]. As initial conditions for the constant pressure $0D$ reactor a temperature of $T = 950$ K and an atmospheric pressure $P = 101325$ Pa are set. Several initial values for the mixture fraction have been computed, ranging from $Z = 0.1$ to $Z = 0.4$. For each case two plots are shown in the following; (i) the evolution of the species mass fraction Y_k and the temperature T, each normalized by the maximum value obtained for each field during the simulation, and (ii) the absolute value of the mass reaction rate $\dot{\omega}_k$ for all the species. In both plots the Δt obtained using Eq. (A.2) is also shown. This value is plotted against the right y-axis (note the logarithmic scale). The plots have been zoomed in around the time instant where auto-ignition occurs, aiming to show the region of interest. The obtained results are shown in Figs. A.1 and A.2. Results only for $Z = 0.1$ and $Z = 0.3$ are shown for the sake of brevity. Results for other mixture fraction values yielded the same conclusion.

As it can be seen, the derived time step Δt evolves as expected. Focusing on the evolution of the species mass fraction and the temperature, the figures show that during the time-interval where the most rapid changes take place,

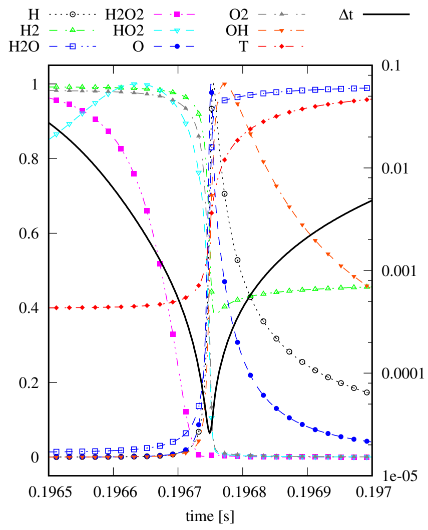


(a) Normalized species mass fractions $\overline{Y_k}$ and temperature $\overline{T}$.

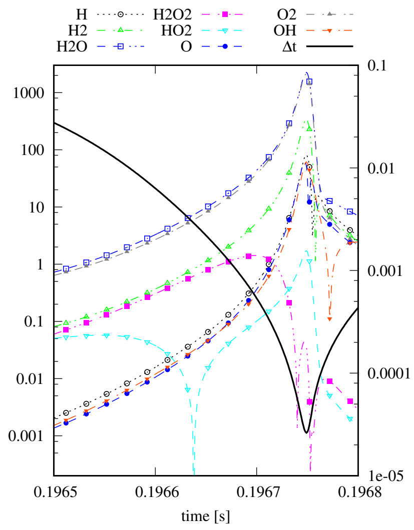


(b) Absolute value of mass reaction rates $|\dot{\omega}_k|$.

Figure A.1: Constant pressure $0D$ reactor results for $Z = 0.1$.



(a) Normalized species mass fractions $\overline{Y_k}$ and temperature $\overline{T}$.



(b) Absolute value of mass reaction rates $|\dot{\omega}_k|$.

Figure A.2: Constant pressure $0D$ reactor results for $Z = 0.3$.

i.e. the ones linked to shorter time-scales, the Δt criteria presents its smallest values. This can be more clearly observed in the plots showing the evolution of the absolute value of the mass reaction rates $|\dot{\omega}_k|$. Higher values of $|\dot{\omega}_k|$, associated to faster changes and shorter time-scales, are strongly correlated with smaller values of the calculated Δt. In fact, the obtained Δt is almost inversely proportional to the highest absolute value of mass reaction rate $|\dot{\omega}_k|$ obtained in the simulation. Thus, the proposed criteria is capable of tracking the chemically active cells and detect the ones presenting stiff systems of equations.

Still, the time step Δt estimated using Eq. (A.2) is larger than the one required to explicitly integrate the species mass reaction rates $\dot{\omega}_k$. Therefore, an additional factor f_r is added to Eq. (A.2). The aim is to obtain a better estimation of the maximum time step that guarantees an accurate and stable integration of the species mass reaction rates if an explicit integration method is employed. With the addition of this coefficient, the final expression used to calculate the *chemical time-scale* Δt_{chem} employed in the present work (see Eq. (13)) is:

$$\Delta t_{chem} = f_r \frac{\rho \left| \left(h - \sum_{k=1}^{N} Y_k \Delta h^0_{f,k} \right) \right|}{\left| \sum_{k=1}^{N} \dot{w}_k \Delta h^0_{f,k} \right|}. \tag{A.3}$$

This parameter f_r allows to adjust the criteria that the algorithm employs to select which cells are integrated implicitly and which ones explicitly. Thus, its value must be selected so that the product of f_r and the time step Δt obtained from Eq. (A.2) results in a *chemical time-scale* Δt_{chem} that would guarantee a stable and accurate explicit integration of the mass reaction rates in species equation.

To some extent, this parameter f_r can be regarded as a kind of threshold. Smaller values of f_r will result in more cells with a *chemical time-scale* Δt_{chem} shorter than the time-step Δt of the simulation used to integrate both convective and diffusive terms for all the transport equations. Hence, this will increase the amount of cells requiring an implicit integration. On the other hand, higher values of f_r will result in larger Δt_{chem} and, consequently, will lead to more cells integrated explicitly. Despite this remark, the authors would like to highlight that f_r must be set according to a mathematical ground, i.e., ensuring a stable integration of the system of equations. Unfortunately, so far has not been found a closed form capable to directly obtain an optimal value for this f_r. Nonetheless, using the same reference to calculate the sensible enthalpy, the order of magnitude of f_r is quite constant. Therefore, based in a small experience and trials with the employed chemical reaction scheme and the configuration set-up of the simulation, it is easy and straightforward to set an optimal value for it.