Received: 27 October 2021 | Revised: 25 March 2022 | Accepted: 28 March 2022

DOI: 10.1002/nme.6990

RESEARCH ARTICLE

WILEY

A parallel finite volume method for incompressible and slightly compressible reactive flows

Henrique C. C. de Andrade[1] | **Fernando L. B. Ribeiro**[1] | **Luiz Carlos Wrobel**[2]

[1]Programa de Engenharia Civil, COPPE/Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil

[2]Departamento de Engenharia Civil e Ambiental, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil

Correspondence
Luiz Carlos Wrobel, Departamento de Engenharia Civil e Ambiental, Pontifícia Universidade Católica do Rio de Janeiro, Gávea, Rio de Janeiro, Brasil.
Email: luiz.wrobel@puc-rio.br

Funding information
Conselho Nacional de Desenvolvimento Cientifico e Tecnologico (CNPq, Brazil); Coordenacao de Aperfeicoamento de Pessoal de Nivel Superior (CAPES, Brazil)

Abstract

In this article, a parallel formulation of the finite volume method is presented for solving three-dimensional, turbulent, mixed, reactive, and slightly compressible flows. It can also be used for incompressible laminar/turbulent flows. The method is designed for nonorthogonal meshes, and oscillations caused by the advective terms are treated by a deferred correction technique. The chosen finite volume scheme is cell centered. The studied fluid is a single-phase multicomponent gas with Newtonian behavior. The focus is mainly on gas mixtures with predominance of N_2, since the chemical reaction of greatest interest is the combustion process in the air. The buoyancy is caused by the gradient of the specified mass, which is a function of the temperature and the composition of the gas mixture. The mathematical model uses an approximation for low Mach numbers, describing slightly compressible flows. The coupling between the fluid dynamic equations is given by the nonlinear Picard's method, with the pressure-velocity coupling treatment given by the SIMPLE algorithm (semi-implicit method for pressure-linked equations). The complete mathematical model includes the sensitive enthalpy equation for the conservation of energy. The LES (large eddy simulation) model is used for modeling the turbulence. The chemical reactions are implemented using the EDC (eddy dissipation concept) and the EDM (eddy dissipation model) approaches. The parallel strategy is based on a subdomain-by-subdomain approach, and uses the MPI and OpenMP standards in a hybrid parallel scheme. Compressed data structures are used to store the matrix coefficients.

KEYWORDS

compressed data structures, finite volume method, message passing interface, parallel computing, slightly compressible reactive flows

1 | INTRODUCTION

In this article, we present a finite volume model for forced natural flows, involving a gas mixture at low velocities with energy release. This type of flow can be classified as turbulent, mixed, reactive and slightly compressible. This model is of great importance to fire simulations, which are based on four main steps: the gas mixture flow, the combustion chemical

reactions, change of phase of the solid/liquid fuel and heat transfer by radiation. The proposed method includes the first two steps of a complete fire simulation. Besides representing the first two steps of fire simulations, this method can be fully exploited in combustion and heat transfer problems. Moreover, it can be used for laminar/turbulent incompressible flows, just by turning off the combustion and the heat transfer models.

Two main formulations of the finite volume method (FVM) can be found in the literature, cell centered and node centered. The most commonly used is the cell centered scheme, which is the one adopted in this article. In cell centered techniques,[1-3] the unknowns are associated to the cells in the average sense. For linear approximations the interpolated values at the centroids coincide with the average values, while in node centered formulation the unknowns are calculated at the vertices of the cells.

As it is well known, the advective terms need special treatment in order to avoid spurious oscillations and thus leading to stable solutions. Pure upwind techniques are excessively diffusive, but there are higher order methods that when allied with the TVD (total variation diminishing) approach provide very good results.[4-6] This procedure optimizes the necessary amount of diffusion to be applied, as it will be seen in Section 3. Other possibilities are WENO (weighted essentially non-oscillatory)[7,8] and hybrid compressible and incompressible.[9]

To obtain the gradients of the primary unknowns it is necessary to use methods for gradient reconstruction. The most commonly used methods for this task are the least square and the Green-Gauss methods. Both can be developed for orthogonal and nonorthogonal meshes.[10-12]

$5 + N$ unknowns are used to describe this type of flow: three velocity components, pressure, energy and N mass fractions, where N is the number of chemical species. Therefore, solving the whole problem with full coupling between the equations might lead to high computational costs. One way to circumvent this problem is to uncouple the equations, observing that special attention should be given to the coupling between pressure and velocity. One way to efficiently solve these equations in an uncoupled manner is to use the semi-implicit method for pressure-linked equations (SIMPLE).[13-15] Initially, the SIMPLE method was developed for steady incompressible flows, and later extended to transient compressible flows.[16]

Concerning turbulence, there are four main families of turbulence models.[17] These methods are the Reynolds averaged Navier–Stokes equations (RANS) family, transport models for the Reynolds tensor, the large eddy simulation (LES) method and lastly, the direct numerical simulation method (DNS).

The combustion model releases energy into the system through chemical reactions. The time and spatial scales in which these reactions occur are largely different from flow scales. Another problem to be dealt with is the high nonlinearity of the chemical reactions, due to the Arrhenius law. To circumvent these problems we use a deterministic approach, in which each cell is divided into two regions; one where the fast chemical reactions occur and another where there are no chemical reactions. The two main models used to implement this idea are the eddy dissipation concept (EDC) and the eddy dissipation model (EDM).[18-24]

The transport of the species present in the gas mixture requires the diffusion velocities of the species. This is highly complex, mathematically speaking, and it also requires great computational costs.[25] In practice, only approximations of this solution are used. The first possibility is to use the Maxwell–Stefan equation, which is close to the exact solution. In this approach the diffusion velocities are explicitly calculated. However, the most commonly used approaches find a correlation between the diffusion velocities and the species gradients, in a way similar to Fick's law. These approaches differ in the definition of the diffusion coefficients and the gradients to be applied, that is, molar or mass fraction gradients.

Some procedures use a single diffusion coefficient for all species through an approximation of the unit Lewis' number and the gradients of the mass fractions. This is a valid approximation for full turbulent flows where the turbulent diffusion is predominant in relation to the molecular diffusion.[1] Another possibility is to use Fick's law for each species. An intermediate possibility, in cases where there is a predominant species, is to use an equivalent diffusion coefficient by means of the Chapman–Enskog's theory. In this case, all coefficients are calculated as a function of the coefficient of the predominant species.[26] Further yet, a first order solution of the Maxwel–Stefan equation relates the diffusion velocities to the molar fraction gradients, known as Hirschfelder and Curtiss approximation. The latter approach was used in the present work.[27]

Our model was designed for 3D nonorthogonal meshes, with a TVD upwind scheme, the SIMPLE method for the equations coupling, the LES model to treat turbulence and the EDM model for the chemical reactions. The gradient reconstruction is performed by the least square method. For the time discretization we use the Adams–Moulton second order scheme.

As it can be observed, the proposed model is highly complex and time consuming. Therefore, parallelization of the code becomes imperative if realistic problems are to be modeled. A state of the art review on parallel strategies for

computational fluid dynamics (CFD) can be found in reference.[28] Many attempts have been made to parallelize CFD codes using, for example, Message Passing Interface (MPI) routines for distributed memory architectures, OpenMP and CUDA computing platforms for shared memory architectures, or even hybrid models mixing these tools for both types of architectures.

The main contribution of this article is the development of a hybrid parallel strategy based on a subdomain-by-subdomain scheme, designed for both distributed and shared memory platforms.[29] This subdomain-by-subdomain approach is an adaptation of that developed for finite element codes,[30] and has been used successfully in many different problems.[31-36] One major improvement presented in this work is the communication mapping, which is highly efficient. This mapping fully optimizes point-to-point nonblocking communications performed by calls to routines of the MPI library. An overlapping partitioning scheme avoids redundancies in terms of floating point operations, when compared to a sequential code. In addition, the OpenMP standard is used to fine tune the speedups, turning the parallelization into a hybrid model. Furthermore, compressed data structures are used to store the matrix coefficients, together with specialized algorithms for matrix-vector and vector-vector operations.[37,38] Two types of solver are used, the CG for symmetric systems (pressure equations) and the BICGSTAB for nonsymmetric systems (Navier–Stokes and transport equations).[39,40] Finally, numerical results demonstrate the efficiency of the proposed method.

2 | MATHEMATICAL MODEL

The mathematical model for reactive flows has $5 + N - 1$ conservation equations, where N is the number of species present in the gas mixture. To simplify the equations notation, the symbols $\overline{()}$ for filtered variable and $\widetilde{()}$ for Favre variable substitution are suppressed. The conservation equations are: mass conservation (Equation 1), three linear momentum equations (Equation 2), energy conservation written in its sensible enthalpy form (Equation 3), and $N - 1$ mass conservation equations for the species (Equation 4). Considering the LES turbulence model, these equations can be written as:

$$\partial_t \rho_g + \nabla \cdot \left[\rho_g \underline{\boldsymbol{v}}\right] = 0, \tag{1}$$

$$\partial_t \left[\rho_g \underline{\boldsymbol{v}}\right] + \nabla \cdot \left\{\rho_g \underline{\boldsymbol{v}}\, \underline{\boldsymbol{v}}\right\} = \nabla \cdot \underline{\underline{\boldsymbol{\tau}}}^L - \nabla p + \left(\rho_r - \rho_g\right) \underline{\boldsymbol{g}}, \tag{2}$$

$$\partial_t \left(\rho_g h_s\right) + \nabla \cdot \left[\rho_g \underline{\boldsymbol{v}} h_s\right] = \dot{\omega}_T - \nabla \cdot \underline{\boldsymbol{q}}_e^L, \tag{3}$$

$$\partial_t \left(\rho_g Y_n\right) + \nabla \cdot \left[\rho_g \underline{\boldsymbol{v}} Y_n\right] = \dot{\omega}_n - \nabla \cdot \underline{\boldsymbol{q}}_y^L, \tag{4}$$

where ρ_g is the density of the gas mixture, $\underline{\boldsymbol{v}}$ is the flow velocity, p is the dynamic pressure, $\underline{\boldsymbol{g}}$ is the gravitational field, h_s is the sensible enthalpy, Y_n is the mass fraction of species n, $\dot{\omega}_T$ is the energy released by the chemical reactions, $\dot{\omega}_n$ is the mass source term for species n, $\underline{\underline{\boldsymbol{\tau}}}^L$ is the molecular turbulent viscous stress tensor, $\underline{\boldsymbol{q}}_e^L$ is the molecular turbulent energy flux, $\underline{\boldsymbol{q}}_y^L$ is the molecular turbulent mass flux and ρ_r is the reference density. The mass fraction of species i is calculated by:

$$Y_i = 1 - \sum_{n=1}^{N-1} Y_n. \tag{5}$$

The turbulent terms are given by:

$$\underline{\underline{\boldsymbol{\tau}}}^L = \left(\mu + \rho_g \nu_r\right) \left\{\nabla \underline{\boldsymbol{v}} + \left(\nabla \underline{\boldsymbol{v}}\right)^T - 2/3 \left(\nabla \cdot \underline{\boldsymbol{v}}\right) \underline{\underline{\boldsymbol{1}}}\right\}, \tag{6}$$

$$\underline{\boldsymbol{q}}_e^L = -\left(\frac{k}{c_p} + \frac{\rho_g \nu_r}{\sigma_h^T}\right) \nabla h_s, \tag{7}$$

$$\underline{\boldsymbol{q}}_y^L = -\left(\rho_g D_n + \frac{\rho_g \nu_r}{\sigma_{sc}^T}\right) \nabla Y_n, \tag{8}$$

where μ is the molecular dynamic viscosity of the gas mixture, n is the molecular thermal conductivity coefficient of the gas mixture, D_n is an approximation for the mass diffusion coefficient of species n, c_p is the specific heat at constant pressure of the gas mixture, σ_h^T is the turbulent Prandtl number, σ_{sc}^T is the turbulent Schmidt number and ν_r is the turbulent viscosity. The turbulent Prandtl and Schmidt numbers are usually adopted as the constants 0.5 and 1.0, respectively.[17,41] The turbulent viscosity is computed by the wall adapting local eddy viscosity (WALE) model.[42] C_w is a constant assuming values between $0.325 - 0.6$ and Δ_f is the LES filter width, taken as the cubic root of the cell volume. The WALE model is given by the following equations, in index notation:

$$\begin{aligned}
\nu_r &= \left(C_w \Delta_f\right)^2 \frac{\left(S_{ij}^d S_{ij}^d\right)^{3/2}}{\left(S_{ij} S_{ij}\right)^{5/2} + \left(S_{ij}^d S_{ij}^d\right)^{5/4}}, \\
S_{ij}^d &= \frac{1}{2}\left(g_{ij}^2 + g_{ji}^2\right) - \frac{1}{3} g_{kk}^2 \delta_{ij}, \\
g_{ij}^2 &= g_{ik} g_{kj}, \\
g_{ij} &= \partial_j v_i, \\
S_{ij} &= \frac{1}{2}\left(\partial_j v_i + \partial_i v_j\right).
\end{aligned} \tag{9}$$

The physical properties of the gas mixture c_p, μ, and k, as well as the mass fraction D_n of each species n, are computed as functions of the mass fractions $\mathbf{Y} = [Y_1, Y_2, \ldots, Y_N]^T$ and the temperature T of the gas mixture:

$$c_p\left(\mathbf{Y}, T\right) = \sum_{n=1}^{N} Y_k c_n^p, \tag{10}$$

$$\mu\left(\mathbf{Y}, \widetilde{T}\right) = \sum_{n=1}^{N} \frac{X_n \mu_n}{X_n + \sum_{j=1, j\neq n}^{N} X_j \phi_{nj}}, \tag{11}$$

$$k\left(\mathbf{Y}, T\right) = \sum_{n=1}^{N} \frac{X_n k_n}{X_n + \sum_{j=1, j\neq n}^{N} X_j \phi'_{nj}}, \tag{12}$$

$$D_n\left(\mathbf{Y}, T\right) = \frac{1 - Y_n}{\sum_{j\neq n}^{N} X_j / D_{jn}}. \tag{13}$$

In the above equations, c_n^p is the specific heat at constant pressure, μ_n is the molecular viscosity, k_n is the molecular thermal conductivity and X_n is the molar fraction, where the subindex n refers to species n. D_{jn} is the binary symmetric mass diffusion coefficient of species j in species n. The coefficients ϕ_{nj} and ϕ'_{nj}, relating species n to species j are given by:

$$\phi_{nj} = \frac{1}{\sqrt{8}}\left(1 + \frac{W_n}{W_j}\right)^{-1/2}\left(1 + \left(\frac{\mu_n}{\mu_j}\right)^{1/2}\left(\frac{W_j}{W_n}\right)^{1/4}\right)^2, \tag{14}$$

$$\phi'_{nj} = \frac{1}{\sqrt{8}}\left(1 + \frac{W_n}{W_j}\right)^{-1/2}\left(1 + \left(\frac{k_n}{k_j}\right)^{1/2}\left(\frac{W_j}{W_n}\right)^{1/4}\right)^2, \tag{15}$$

where W_n and W_j are the molar masses of species n and j. The molar masses are straightforwardly calculated through the atomic mass and the chemical formula of the species. The temperature variation of μ_n, k_n, and D_{jn} is given by Chapman–Enskog's theory.[26] The specific heat c_n^p is given by NASA[43] or Shomate,[44] both lower order polynomials.

The state equations are given by:

$$\rho_g = \frac{P_{th} W_g}{R\ T}, \tag{16}$$

$$h_s = \int_{T_{ref}}^{T} c_p\left(T\right) dT, \tag{17}$$

in which P_{th} is the thermodynamic pressure (usually atmospheric pressure), W_g is the molar mass of the mixture, R is the gas universal constant and T_{ref} is the reference temperature, usually 298.15 K. W_g is given by:

$$\frac{1}{W_g} = \sum_{n=1}^{N} \frac{Y_n}{W_k}. \tag{18}$$

Finally, the chemical reactions complete the mathematical model. For a one-step irreversible chemical reaction, the mass source term of species n is given by:

$$\dot{\omega}_n = (\upsilon_n'' - \upsilon_n')W_n \frac{\rho_g}{\tau_{mix}} \min_{n=1}^{Rea} \left(\frac{Y_n}{\upsilon_n' W_n} \right) A_{edm}, \tag{19}$$

where υ_n'' is the stoichiometric coefficient of species n on the right hand side of the chemical reaction (product), υ_n' is the stoichiometric coefficient of species n in the left hand side of the chemical reaction (reagent), τ_{mix} is the time scale, Rea is the set of species in the left side of the chemical reaction equation, and A_{edm} is a model constant, usually assuming the value of 4. This model is known as EDM.

The time scale τ_{mix} is given by the expression below:

$$\tau_{mix} = \min \left(\frac{1}{\sqrt{2S_{ij}S_{ij}}}, \frac{c_p \Delta^{2/3}}{k} \right), \tag{20}$$

where S_{ij} is defined in Equation (9).

The heat energy released by the chemical reactions is calculated by,

$$\dot{\omega}_T = -\sum_{n=1}^{N} \Delta h_{f,n}^0 \dot{\omega}_n, \tag{21}$$

where $\Delta h_{f,n}^0$ is the enthalpy of formation of species n. This enthalpy is usually calculated using the reference temperature T_{ref}.

3 | NUMERICAL MODEL

The conservation equations can be written in the following integral form,

$$\int_\Omega \partial_t (\rho\phi) + \nabla \cdot \left[\rho \underline{\boldsymbol{v}} \phi\right] dV = \int_\Omega \nabla \cdot \left[\Gamma^\phi \nabla \phi\right] + Q(\phi, t) dV, \tag{22}$$

where Ω is the domain, ρ is the resident fluid density, ϕ is the quantity being transported, $Q(\phi, t)$ is a generic source, $\underline{\boldsymbol{v}}$ is the velocity of the resident fluid and Γ^ϕ is the diffusion coefficient of ϕ. The fluid density ρ_g will be written as ρ, for the sake of simplicity. The integrals over the whole domain can be obtained as a sum of integrals over all the finite volumes and hence the conservation equation for a single control volume (cell) can be written as,

$$\int_{\Omega_C} \partial_t (\rho\phi)\, dV + \int_{\partial\Omega_C} \rho \left(\underline{\boldsymbol{v}} \cdot \underline{\boldsymbol{n}}\right) \phi d\Gamma = \int_{\partial\Omega_C} \Gamma^\phi \left(\nabla\phi \cdot \underline{\boldsymbol{n}}\right) d\Gamma + \int_{\Omega_C} Q(\phi, t) dV, \tag{23}$$

where Ω_C is the volume of the cell and $\partial\Omega_C$ its boundary. The external unit normal to the face of the cell is denoted by $\underline{\boldsymbol{n}}$.

Figure 1 shows a schematic representation of a typical 3D cell and its neighbors in a nonorthogonal mesh. Point P is the centroid of the central cell, points A, B, and C are the centroids of the neighboring cells, points M_{PA}, M_{PB}, M_{PC} are the centroids of the adjacent faces and points F_{PA}, F_{PB}, F_{PC} are the intersection points at the faces given by the lines connecting points A, B, and C to P. For each central cell we can write,

$$a_P^\phi \phi_P = \sum_f^{N(P)} a_f^\phi \phi_f + b_p^\phi, \tag{24}$$

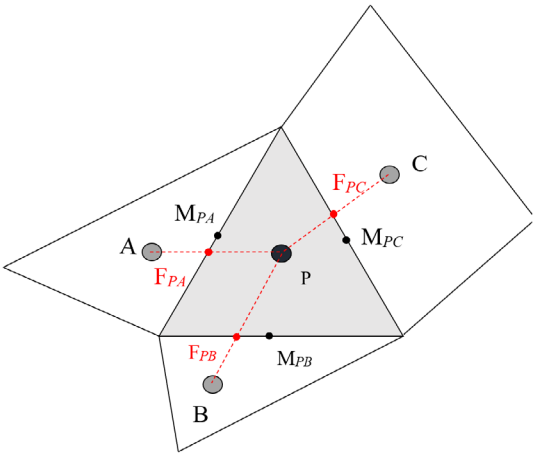

FIGURE 1 Central cell and its neighbors

where the coefficients a_P^{ϕ}, a_f^{ϕ}, and b_p^{ϕ} are given by,

$$a_P^{\phi} = c_1 \rho_P^{i+1} V_P + \left(\sum_f^{N(P)} \dot{m}_{Pf} + \sum_f^{N(P)} D_{Pf}^{\phi} \right)^{i+1}, \tag{25}$$

$$a_f^{\phi} = \left(D_{Pf}^{\phi} - \min(\dot{m}_{Pf}, 0) \right)^{i+1}, \tag{26}$$

$$b_p^{\phi} = c_2 V_P (\rho\phi)_P^i - c_3 V_P (\rho\phi)_P^{i-1} + V_P S_P + \left(\sum_f^{N(P)} S_{Pf}^{CD} - \sum_f^{N(P)} S_{Pf}^{DC} \right)^{i+1}. \tag{27}$$

In the above, ϕ is the unknown, V_P is the volume of the central cell, i is the time step, f indicates the neighboring cells of cell P, $N(P)$ represents the set of all neighbors of P, ρ_P^{i+1} is the density, $\dot{m}_{Pf}$ is the mass flow through the cell face, D_{Pf}^{ϕ} is the discretization of the diffusive term, S_P is the source term, S_{Pf}^{CD} is the correction of the advective term (*deferred correction-DC*), and S_{Pf}^{CD} is the diffusion correction term due to the nonorthogonality of the mesh (*cross diffusion-CD*).

Considering all the cells in Equation (24), we obtain a sparse nonlinear system of equations of the form,

$$A(\phi)\phi = b. \tag{28}$$

The terms c_1, c_2, and c_3 stem directly from the time discretization:

$$\begin{aligned} c_1 &= \frac{1}{\Delta t^i} + \frac{1}{\Delta t^i + \Delta t^{i-1}}, \\ c_2 &= \frac{1}{\Delta t^i} + \frac{1}{\Delta t^{i-1}}, \\ c_3 &= \frac{\Delta t^i}{\Delta t^{i-1}\left(\Delta t^i + \Delta t^{i-1}\right)}. \end{aligned} \tag{29}$$

The above expressions represent the Adams–Moulton technique, also known as second order Euler (SOUE–second order upwind Euler).[16] This time discretization requires the time interval Δt^i for the current time step i and the time interval Δt^{i-1} used in the previous time step $i-1$. The time intervals are calculated according to the Courant–Friedrichs–Lewy (CFL) condition,

$$\frac{|v|\Delta t}{L} \leq 1, \tag{30}$$

where L is the characteristic length of the cell, here taken as the cubic root of the cell volume.

The diffusive terms for orthogonal meshes are approximated by the decomposition of the gradients into orthogonal gradients and a deferred nonorthogonal correction (*cross diffusion-CD*). This approximation is given by the expressions:

$$D_{Pf}^{\phi} = \Gamma_{Pf}^{\phi} \frac{E_{Pf}}{d_{Pf}}, \tag{31}$$

$$S_{Pf}^{CD} = \Gamma_{Pf}^{\phi} \left(\nabla \phi_{Pf} \cdot \underline{\boldsymbol{S}}_{Pf} - \nabla \phi_{Pf} \cdot \underline{\boldsymbol{E}}_{Pf} \right) = \Gamma_{Pf}^{\phi} \nabla \phi_{Pf} \cdot \underline{\boldsymbol{k}}_{Pf}, \tag{32}$$

where Γ_{Pf}^{ϕ} is the diffusion coefficient at the face of the cell, d_{Pf} is the distance between two centroids, and $\nabla \phi_{Pf}$ is the value of the gradient at the faces. The vectors $\underline{\boldsymbol{S}}_{Pf}$, $\underline{\boldsymbol{E}}_{Pf}$, and $\underline{\boldsymbol{k}}_{Pf}$ are represented in Figure 2 considering only the central cell P and its neighbor A. The vector $\underline{\boldsymbol{S}}_{Pf}$ is the product of the area of the face by its external unit normal, $\underline{\boldsymbol{E}}_{Pf}$ is the vector defined by the centroids of the central cell and its neighbor, and $\underline{\boldsymbol{k}}_{Pf}$ is the difference between $\underline{\boldsymbol{S}}_{Pf}$ and $\underline{\boldsymbol{E}}_{Pf}$. Considering the central cell P and its neighbor A, the vector $\underline{\boldsymbol{E}}_{Pf}$ is equal to:

$$\underline{\boldsymbol{E}}_{PA} = \frac{\underline{\boldsymbol{S}}_{PA} \cdot \underline{\boldsymbol{S}}_{PA}}{\underline{\boldsymbol{\xi}}_{PA} \cdot \underline{\boldsymbol{S}}_{PA}} \underline{\boldsymbol{\xi}}_{PA} = E_{PA} \underline{\boldsymbol{\xi}}_{PA}, \tag{33}$$

where $\underline{\boldsymbol{\xi}}_{PA}$ is the unit vector in the direction of $\underline{\boldsymbol{E}}_{PA}$.

The mass flux on the face $\dot{m}_{Pf}$ is given by,

$$\dot{m}_{Pf} = \rho_{Pf} (\underline{\boldsymbol{v}}_{Pf} \cdot \underline{\boldsymbol{S}}_{Pf}) \phi_{Pf}. \tag{34}$$

Here ρ_{Pf} is the density, $\underline{\boldsymbol{v}}_{Pf}$ is the velocity and ϕ_{Pf} is the value of ϕ, computed at the cell faces. The variables ρ_{Pf} and $\underline{\boldsymbol{v}}_{Pf}$ are determined by a linear interpolation, which can be weighted by the distance between the centroids or the volumes of the cells. We use the distance between the centroids of the cells. The variable ϕ_{Pf} is given by upwind schemes as follows.

Figure 3 illustrates the *upwind* technique, where C, D, and U are, respectively, the upstream (*upwind*), downstream (*downwind*) and far upstream (*far upwind*) points. G indicates a dummy node, which does not necessarily coincide with a centroid or vertex of a cell. Figure 3 also shows the two possible scenarios: $\underline{\boldsymbol{v}} \cdot \underline{\boldsymbol{n}} < 0$ and $\underline{\boldsymbol{v}} \cdot \underline{\boldsymbol{n}} > 0$. As it can be observed, each of these scenarios generate different configurations of C, D, and U.

For each face f, ϕ_{Pf} is calculated by the expression,

$$\phi_{Pf} = \phi_C + S_{Pf}^{DC}. \tag{35}$$

The above equation results from the application of the upwind technique with deferred correction. The term S_f^{DC} reduces the numerical diffusion given by the standard upwind technique. Using the TVD approach,

$$S_{Pf}^{DC} = \frac{1}{2} \psi \left(r_{Pf} \right) (\phi_D - \phi_C) \tag{36}$$

with,

$$r_{Pf} = \frac{\phi_C - \phi_D + 2 \nabla \phi_C \cdot d_{CD}}{\phi_C - \phi_D + 10^{-14}}. \tag{37}$$

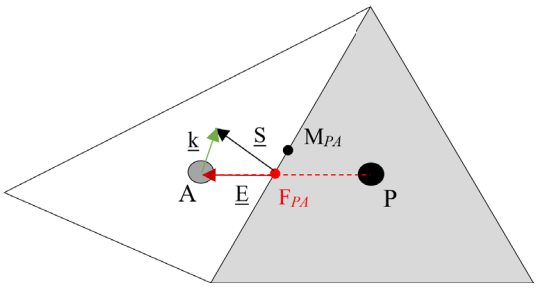

FIGURE 2 Decomposition of the gradients into orthogonal gradients and a deferred nonorthogonal correction

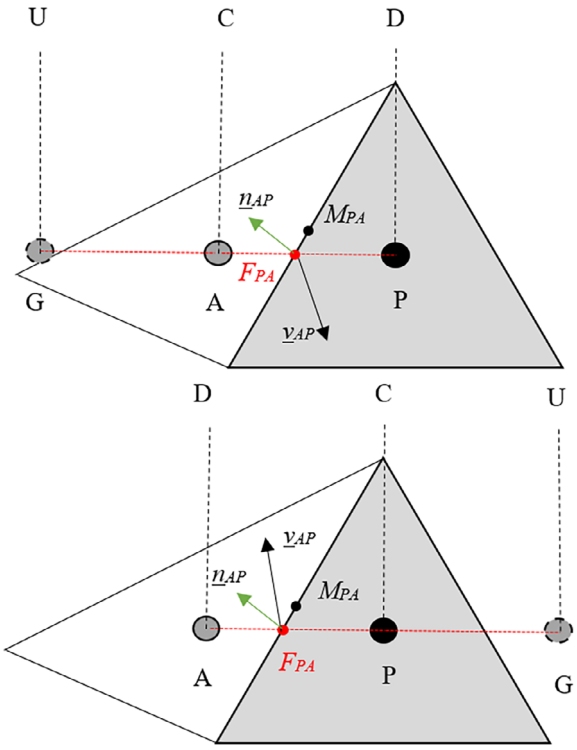

FIGURE 3 Discretization of advective terms for nonorthogonal meshes

In the above equation, d_{CD} is the vector that connects point C to point D. The scalar function $\psi\left(r_{Pf}\right)$ defines the TVD technique to be used. In this article, we use the MINMOD technique, which is given by,

$$\psi(r_{Pf}) = \max(0, \min(1, r_{Pf})). \tag{38}$$

Furthermore, Equation (35) allows for the use of other techniques, such as the linear-upwind stabilized transport (LUST) and the second order upwind (SOUP). For the SOUP technique, the term S_{Pf}^{DC} is calculated as,

$$S_{Pf}^{DC} = \nabla\phi_C \cdot d_{CD}. \tag{39}$$

In this article, the TVD with MINMOD limiter[1,16] technique is used in the conservation equations for the species and also in the conservation of energy. The SOUP approach is used for the conservation of linear momentum.

The set of differential equations that govern the reactive fluid flow is solved in a segregated form, where the coupling pressure-velocity is given by the SIMPLE/SIMPLEC algorithm. One advantage of using a segregated solver is that the whole system can be split into smaller systems, leading to faster iterative solutions with less memory requirements. The pressure-velocity coupling gives origin to oscillations in the pressure field (*checker-board*), which are treated by an interpolation of the Rhie–Chow family.[16]

All conservation equations can be solved directly by applying Equation (24). The only exception is the pressure equation, which corresponds to the Poisson equation. The solution of this equation gives the correction pressure of the SIMPLE/SIMPLEC method. The first step to arrive to this equation is to split the velocity in two parcels,

$$\underline{\boldsymbol{v}} = \underline{\boldsymbol{v}}_c + \underline{\boldsymbol{v}}_e, \tag{40}$$

where $\underline{\boldsymbol{v}}_c$ is the correction velocity, and $\underline{\boldsymbol{v}}_e$ is the best estimate for the velocity obtained by the Navier–Stokes equation. The correction velocity is given by,

$$\underline{\boldsymbol{v}}_c = \mathbf{D}^{\underline{\boldsymbol{v}}} \nabla p_c, \tag{41}$$

where $\mathbf{D}^{\underline{\boldsymbol{v}}}$ is a 3×3 matrix that indicates the pressure-velocity coupling method.

Introducing $\underline{\boldsymbol{v}}_c$ in Equation (1), we obtain the correction pressure p_c,

$$\nabla \cdot \left[\rho \mathbf{D}^{\underline{\boldsymbol{v}}} \nabla p_c\right] = -\left(\partial_t \rho + \nabla \cdot \left[\rho \underline{\boldsymbol{v}}_e\right]\right). \tag{42}$$

The matrix $\mathbf{D}^{\underline{\boldsymbol{v}}}$ is not a physical diffusion tensor. It depends on the discretization and the pressure-velocity coupling method. In the SIMPLE method, and considering a central cell P, $\mathbf{D}_P^{\underline{\boldsymbol{v}}}$ is given by,

$$\mathbf{D}_P^{\underline{\boldsymbol{v}}} = V_P \begin{bmatrix} \frac{1}{a_P^{v_1}} & 0 & 0 \\ 0 & \frac{1}{a_P^{v_2}} & 0 \\ 0 & 0 & \frac{1}{a_P^{v_3}} \end{bmatrix} = \begin{bmatrix} \mathbb{D}_P^{v_1} & 0 & 0 \\ 0 & \mathbb{D}_P^{v_2} & 0 \\ 0 & 0 & \mathbb{D}_P^{v_3} \end{bmatrix}, \tag{43}$$

where $a_P^{v_1}, a_P^{v_2}, a_P^{v_3}$ are the coefficients of the left-hand side of the three linear momentum equations given by Equation (24), replacing the generic unknown ϕ by v_1, v_2, and v_3, which are the three velocity components.

Applying the FVM to Equation (42), we obtain for each cell P the equation below,

$$a_P^{p^c} \phi_P = \sum_f^{N(P)} a_f^{p^c} \phi_f + b_P^{p^c}, \tag{44}$$

where the coefficients $a_P^{p^c}$, $a_f^{p^c}$, and $b_P^{p^c}$ are given by,

$$a_P^{p^c} = \sum_f^{N(P)} a_f^{p^c}, \tag{45}$$

$$a_f^{p^c} = \rho_{Pf} D_{Pf}^{p^c}, \tag{46}$$

$$b_P^{p^c} = -\sum_f^{N(P)} \dot{m}_f^e - V_P \left(\partial_t \rho\right)_{DF}. \tag{47}$$

Here, $(\partial_t \rho)_{DF}$ is a finite difference temporal operator, and $\dot{m}_f^g$ is the mass flux calculated with $\underline{\boldsymbol{v}}_e$. $D_{Pf}^{p^c}$ is related to the anisotropic diffusion tensor. For a central cell P and its neighbor A, $D_{PA}^{p^c}$ is given by,

$$D_{PA}^{p^c} = \frac{E_{PA}}{d_{PD}}, \tag{48}$$

where

$$E_{PA} = \frac{\underline{\boldsymbol{S}}_{PA}^a \cdot \underline{\boldsymbol{S}}_{PA}^a}{\underline{\boldsymbol{\xi}}_{PA} \cdot \underline{\boldsymbol{S}}_{PA}^a} \tag{49}$$

with $\underline{\boldsymbol{S}}_{PA}^a$ being defined by the matrix product,

$$\underline{\boldsymbol{S}}_{PA}^a = \mathbf{D}_{PA}^{\underline{\boldsymbol{v}}} \underline{\boldsymbol{S}}_{PA}. \tag{50}$$

The coefficients of the tensor $\mathbf{D}_{PA}^{\underline{\boldsymbol{v}}}$ are calculated as the average values of $\mathbf{D}_P^{\underline{\boldsymbol{v}}}$ and $\mathbf{D}_A^{\underline{\boldsymbol{v}}}$, which are the diffusion tensors of a central cell P and its neighbor cell A, respectively.

Equation (44) does not need a correction due to the nonorthogonality of the mesh, as it would affect only the convergence rate.

Finally, the pressure p is corrected by,

$$p = p + \alpha_p p_c, \tag{51}$$

where α_p is the under-relaxation factor.

The nonlinear iterative process may present convergence difficulties. To overcome this problem, all dependent variables must have an under-relaxation factor ranging in the interval (0, 1].

Algorithm 1 summarizes the segregated method used to solve the whole set of equations. Steps 7, 9, 17, and 21 correspond to the solution of the following systems:

$$\mathbf{A}^{\underline{\boldsymbol{v}}}\underline{\boldsymbol{v}} = \mathbf{b}^{\underline{\boldsymbol{v}}}, \tag{52}$$

$$\mathbf{A}^{p_c} p_c = \mathbf{b}^{p_c}, \tag{53}$$

$$\mathbf{A}^{Y_n} Y_n = \mathbf{b}^{Y_n}, \tag{54}$$

$$\mathbf{A}^{h_s} h_s = \mathbf{b}^{h_s}. \tag{55}$$

From the above systems, we obtain the velocities $\underline{\boldsymbol{v}}$ (Equation 52), the pressure field p_c (Equation 53), the mass fractions Y_n (Equation 54) and the sensible enthalpy h_s (Equation 55). Except for the pressure equation, all other systems are nonsymmetric. Systems (52) and (54) are solved in a segregated form. The sparse matrices obtained by the FVM discretization are stored in the compressed sparse row column format (CSRC).[30] The conjugate gradients method CG is used for the symmetric systems, whilst the stabilized biconjugate gradients method BICGSTAB is employed for the nonsymmetric systems. For both methods we use a diagonal preconditioner.

Supressing lines 14–23 of Algorithm 1, we obtain the steps for solving incompressible laminar/turbulent flows, shown in Algorithm 2.

Algorithm 1. Main solution steps for reactive flows

1: set $t_0 = 0$
2: **for** $i = 0, 1, \ldots$ **do**
3: $t_{i+1} = t_i + \Delta t$
4: **for** $j = 0, 1, \ldots$, until convergence **do**
5: compute gradients of $\underline{\boldsymbol{v}}$ and p
6: compute coefficients in $\mathbf{A}^{\underline{\boldsymbol{v}}}$, $\mathbf{b}^{\underline{\boldsymbol{v}}}$, and the tensor field $\mathbf{D}^{\underline{\boldsymbol{v}}}$
7: solve system $\mathbf{A}^{\underline{\boldsymbol{v}}}\underline{\boldsymbol{v}} = \mathbf{b}^{\underline{\boldsymbol{v}}}$
8: compute coefficients in $\mathbf{A}^{p_c}$ and $\mathbf{b}^{p_c}$
9: solve system $\mathbf{A}^{p_c} p_c = \mathbf{b}^{p_c}$
10: compute gradients of p_c
11: perform correction for $\underline{\boldsymbol{v}}$
12: compute gradients of $\underline{\boldsymbol{v}}$
13: compute the eddy viscosity ν_r
14: compute the source term $\dot{\omega}_n$ of chemical reactions
15: compute gradients of Y_1, Y_2, ..., Y_N
16: compute coefficients in $\mathbf{A}^{Y_n}$ and $\mathbf{b}^{Y_n}$
17: solve system $\mathbf{A}^{Y_n} Y_n = \mathbf{b}^{Y_n}$
18: compute the energy release $\dot{\omega}_T$
19: compute gradients of h_s
20: compute coefficients in $\mathbf{A}^{h_s}$ and $\mathbf{b}^{h_s}$
21: solve system $\mathbf{A}^{h_s} h_s = \mathbf{b}^{h_s}$
22: compute the temperature T
23: update ρ_g, k, D, μ and c_p
24: perform correction for pressure p
25: **end for**
26: compute new Δt
27: **end for**

Algorithm 2. Main solution steps for incompressible flows

set $t_0 = 0$
for $i = 0, 1, \ldots$ **do**
  $t_{i+1} = t_i + \Delta t$
  for $j = 0, 1, \ldots$, until convergence **do**
    compute gradients of $\underline{\boldsymbol{v}}$ and p
    compute coefficients in $\mathbf{A}^{\underline{\boldsymbol{v}}}$, $\mathbf{b}^{\underline{\boldsymbol{v}}}$, and the tensor field $\mathbf{D}^{\underline{\boldsymbol{v}}}$
    solve system $\mathbf{A}^{\underline{\boldsymbol{v}}}\underline{\boldsymbol{v}} = \mathbf{b}^{\underline{\boldsymbol{v}}}$
    compute coefficients in $\mathbf{A}^{p_c}$ and $\mathbf{b}^{p_c}$
    solve system $\mathbf{A}^{p_c} p_c = \mathbf{b}^{p_c}$
    compute gradients of p_c
    perform correction for $\underline{\boldsymbol{v}}$
    compute gradients of $\underline{\boldsymbol{v}}$
    compute the eddy viscosity ν_r
    perform correction for pressure p
  end for
  compute new Δt
end for

TABLE 1 Node specifications

Processor	**Intel Xeon E5-2630 v4**
Processors	2
Cores	2×10
Cache size (L3)	25 MB
Clock	2.2 GHz
QPI	8 GT/s
Memory	256 GB RAM (DDR4–2.134 MHz)
Motherboard	Intel S2600CWR
LAN	Ethernet (2×10 Gb)
OS	CentOS 7

4 | PARALLEL IMPLEMENTATION

Algorithm 1 shows the sequential solution of the problem. We used three different parallel schemes to solve this problem: shared memory using OpenMP directives, distributed memory using the MPI library and a hybrid implementation, which is a combination of these two paradigms. Although the MPI library was designed for distributed memory platforms in its strict sense, that is, each node possessing its own private memory, it behaves very well on multicore platforms, simulating a distributed architecture where the communication between processor and memory is performed internally. Therefore, the MPI implementation can run on both systems, multicore platforms and on clusters with each node having its own memory, connected through a local network. While the bottleneck in multicore platforms is the internal communication bandwidth, the efficiency of the implementation on clusters is highly dependent on the speed of the local network. Another potential disadvantage of running MPI based implementations on multicore machines is that the code is replicated in memory for each process. On the other hand, the OpenMP paradigm is based on threads opened in a single code, and the main drawback is how to avoid the problem of memory race condition.

In this article, we test the possibilities discussed above on a cluster with four nodes connected by a local Ethernet network of 10 Gb/s. Each node has two processors, with a cache coherent nonuniform memory access (cc-NUMA). Table 1 shows the hardware specifications of each node, and Figure 4 shows a schematic representation of two nodes of the

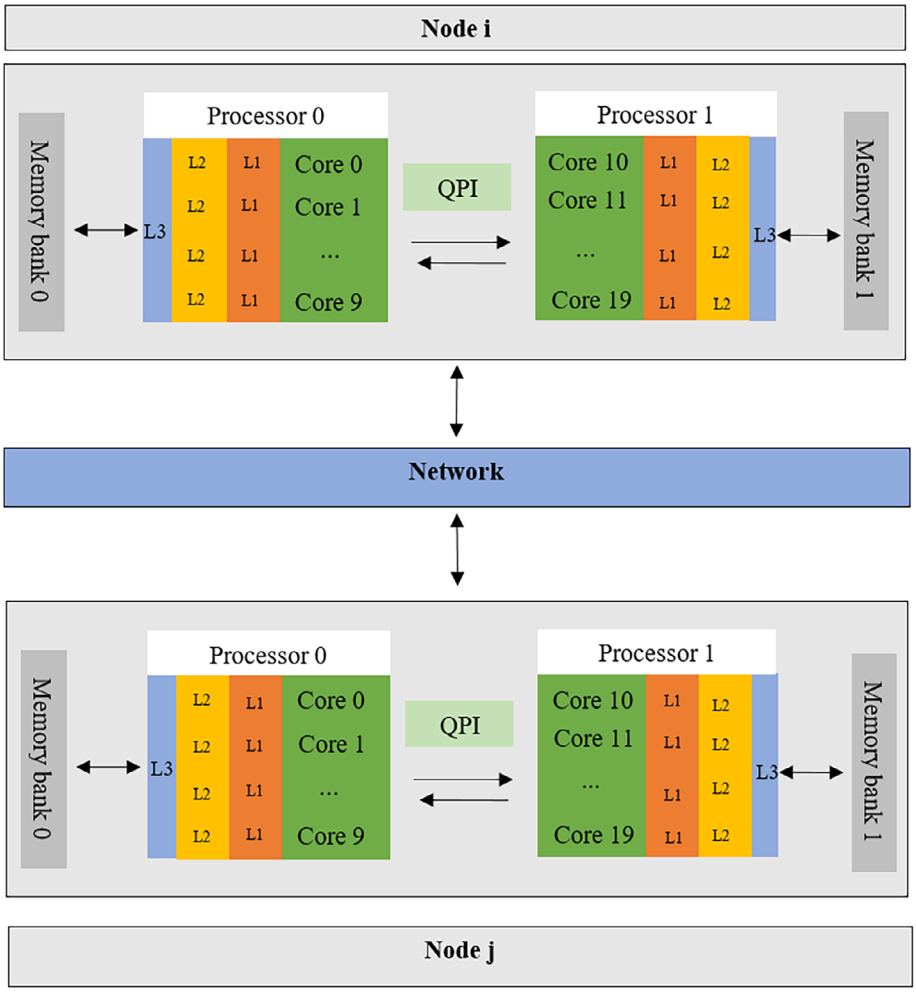

FIGURE 4 Schematic representation of two nodes of the computational platform

cluster. As can be seen, each processor has 10 cores and its own memory bank. Both memory banks are visible by the two processors. Communication between nodes is performed by a local network, whilst a quick path inter-connect (QPI) circuit is responsible for the communication between the two processors within a same node. Although processor 0 can use memory bank 1, and vice-versa, communication between one processor and its own memory bank is faster.[45] Each processor has three levels of intermediate cache memory, that is, L1, L2, and L3. The cache memories L1 and L2 are exclusive for each core, whilst L3 is a common level for all cores of a processor. Concerning the software, we used the Intel C/C++ compiler version 18.0.3, which has both the MPI library and the OpenMP directives.

In what follows we describe the parallelization of Algorithm 1. All steps between lines 5–24 involve a loop over the cells, except steps 7, 9, 17, and 21, which correspond to the solution of linear systems. The FVM matrices possess a higher degree of sparsity than the finite element method. Figure 5 shows the example of a 2D orthogonal mesh of quadrilaterals, where the resulting matrix has the maximum of five nonzero coefficients per line. This is also true for non-structured/nonorthogonal meshes. For all types of meshes, the discretization by a cell centered FVM results in matrices having a maximum number of nonzero coefficients equal to the number of faces of the cell plus 1. Although the number of nonzero coefficients per line is constant (except for boundary cells), it is important to renumber the cells in such a way that these coefficients are clustered to the main diagonal as much as possible, for the sake of data locality.[37]

The data structure used to store these matrices is CSRC, due to its high efficiency in both symmetric and nonsymmetric systems. With this data structure, the algorithm to perform matrix-vector products (matvec) is the same for both types of system.

Examining Algorithms 1 and 2, we observe that the two main tasks to be parallelized are the loop over the cells and the solvers. It is important to note that, differently from the finite element method, the assembly of the matrices performed

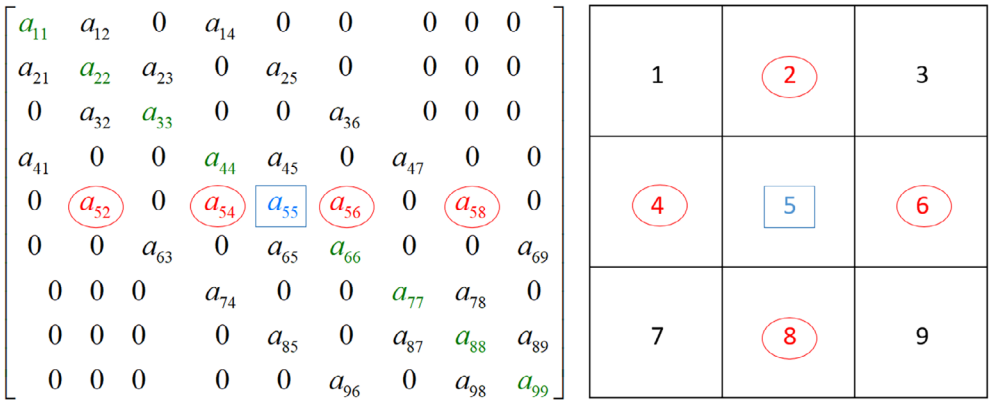

FIGURE 5 Mesh of nine cells and its resulting system of equations

in a loop over the cells does not involve accumulation of coefficients, which means that there is no coupling between the cells. On the other hand, the solvers are strongly coupled. In the case of iterative solvers, three main operations must be parallelized: vector updates, internal products, and matvec operations, as can be seen in Algorithms 3 and 4, which show the implementations of the CG and the BICGSTAB.

As described in Reference 30, the CSRC format consists of three real and second integer arrays: $ad(nEq)$ for the diagonal coefficients, $al(nAd)$ for the lower triangular part, $au(nAd)$ for the upper triangular part, $ia(nEq+1)$, which is the position in al of the first nonzero coefficient of each line, and $ja(nAd)$, which tells the column of each coefficient in al. nEq is the number of equations, and nAd is the number of nonzero coefficients in each triangular part of the matrix. Due to the symmetric topology of the matrix, ia also indicates the position in au of the first nonzero coefficient in each column, and ja also indicates the line of each coefficient in au. Note that the total number of nonzero coefficients is equal to $nEq + 2 * nAd$. Therefore, taking advantage of the topological symmetry, we can write the matvec product shown in Algorithm 5, which can be used for both symmetric and nonsymmetric systems. In what follows, we discuss these points in detail.

4.1 | OpenMP parallelization

The loop over the cells can be easily parallelized with the OpenMP directives, as all the operations can be performed without causing the problem of memory race. Algorithm 6 is a typical example of a parallel loop over the cells. A single parallel region is opened for the entire loop. The variables i, $rho1$, and $rho2$ are declared as private and the variables rho,

Algorithm 3. CG

1: $r = b - Ax$
2: $p = z = M^{-1}r$
3: $d = (r, z)$
4: **for** $j = 0, 1, \ldots,$ until convergence **do**
5: $\quad z = Ap$
6: $\quad \alpha = d/(z, p)$
7: $\quad x = x + \alpha \cdot p$
8: $\quad r = r - \alpha \cdot z$
9: $\quad z = M^{-1}r$
10: $\quad di = (r, z)$
11: $\quad \beta = di/d$
12: $\quad p = r + \beta \cdot p$
13: $\quad d = di$
14: **end for**

Algorithm 4. BICGSTAB

$r_0 = b - Ax$
$p = r = r_0$
$z = M^{-1}p$
for $j = 0, 1, \ldots,$ until convergence **do**
  $v = Az$
  $rr_0 = (r, r_0)$
  $\alpha = \frac{rr_0}{(v,r_0)}$
  $x = x + \alpha \cdot z$
  $r = r - \alpha \cdot v$
  $z = M^{-1}r$
  $t = Az$
  $w = \frac{(t,r)}{(t,t)}$
  $x = x + w \cdot z$
  $r = r - w \cdot t$
  $\beta = \left(\frac{(r,r0)}{rr_0}\right) \cdot \left(\frac{\alpha}{w}\right)$
  $p = r + \beta \cdot (p - w \cdot v)$
  $z = M^{-1}r$
end for

Algorithm 5. Matvec product (sequential version)

for $i = 1,\ nEq$ **do**
  $xi = x(i)$
  $t = ad(i) * xi$
  for $k = ia(i),\ ia(i+1)$ **do**
    $jk = ja(k)$
    $t = t + al(k) * x(jk)$
    $y(jk) = y(jk) + au(k) * xi$
  end for
  $y(i) = t$
end for

Algorithm 6. Parallel loop over the cells to compute the densities

$omp parallel for *private(i, rho1, rho2) shared(rho, T, alpha, nCell) numthreads*(4)
for $i = 1,\ nCells$ **do**
  $rho1 = rho(i)$
  $rho2 = density(T(i))$
  $rho(i) = alpha * rho2 + (1 - alpha) * rho1$
end for
$end omp parallel

T, *alpha*, and *nCells* are shared. In line 5, the thread safe function *density*($T(i)$) computes the density as a function of temperature $T(i)$. All other loops over the cells follow these same rules.

The efficiency of an OpenMP parallelization depends on the number of parallel regions, due to the overhead to open a thread. The lower is the number of parallel regions, the more efficient is the code. Fortunately, both Algorithms 3 and 4 can be entirely implemented with just one parallel region. Vector update operations are implemented according to Algorithm 7. Algorithm 8 shows the parallel code for inner products. In this code, the variable *dot* is shared and can be initialized in just one thread, using the directive *single*. The directive *reduction* tells the compiler that the shared variable *dot* accumulates the results, preventing memory conflicts.[46-48]

Algorithms 7 and 8 can be implemented in parallel without any major difficulty. However, the same is not true for the matvec operation described by Algorithm 5. The problem with this algorithm is the memory race that happens in line 7, when accumulating the contributions of the upper triangular part. To resolve this issue, we create a buffer $thY(nEq * nTh)$ to store the product results in each thread, for a number of threads equal to *nTh*. Each thread has three key integer variables, $thBegin(nTh)$, $thEnd(nTh)$, and $thHeight(nTh)$. The variables $thHeight(i)$ and $thEnd(i)$ represent the range in the result vector $y(nEq)$ addressed by thread *i*. The variables $thBegin(nTh)$ and $thEnd(nTh)$ are previously calculated in order to balance the loads between the threads, in a way that each thread works approximately over the same number of nonzero coefficients. The variable $thBegin(i)$ indicates the first line with a nonzero coefficient in the lower triangular part addressed by thread *i*.

The first step in Algorithm 9 is to initialize the buffers $thY(i)$ with zeros between $thHeight(i)$ and $thBegin(i)$, in lines 3–10. The second step is to compute the matvec product, in lines 13–24. The third and last step, lines 27–34, starts after a *barrier* directive, and accumulates the results stored in all buffers in the final result vector $y(nEq)$. An alternative to the creation of these memory consuming buffers is to use a coloring algorithm to distribute the lines between threads in a way that the memory conflict is avoided. However, this latter approach has proven to be less efficient than the use of buffers, in terms of speedups.[38] Figure 6 schematically illustrates the buffers for a system of nine equations, parallelized with three threads. Colors red, green and blue stand, respectively, for threads 0, 1, and 2.

4.2 | MPI parallelization

Our shared memory parallel implementation is performed using calls to the MPI standard functions. It is based on an adaptation of a subdomain-by-subdomain approach, previously designed for finite element codes.[30] Unlike the shared memory parallelization paradigm, which is based on the concept of threads opened within the same code, the distributed

Algorithm 7. Parallel vector update (OpenMP version)

```
1: $omp for
2: for i = 1, nEq do
3:     x(i) = x(i) + alpha * p(i)
4: end for
5: $end omp for
```

Algorithm 8. Parallel inner product (OpenMP version)

```
1: $omp single
2: dot = 0.0
3: $end omp single
4: $omp for reduction(+:dot)
5: for i = 1, nEq do
6:     dot = dot + x1(i) * x2(i)
7: end for
8: $end omp for
```

Algorithm 9. Matrix vector product (OpenMP version)

```
1: id = omp_get_thread_num()
2: /*Step 1 - buffer initialization:*/
3: for i = 1, nTh do
4:     inc = (i − 1) ∗ nEq
5:     $omp for
6:     for k = thHeight(i) + inc, thBegin(i) − 1 do
7:         thY(k) = 0.0
8:     end for
9:     $end omp for
10: end for
11: inc = id ∗ nEq
12: /*Step 2 - Matrix vector product:*/
13: for i = thBegin(id), thEnd(id) do
14:     y(i) = 0.0
15:     xi = x(i)
16:     t = ad(i) ∗ xi
17:     for k = ia(i), ia(i + 1) do
18:         jk = ja(k)
19:         t = al(k) ∗ x(jk)
20:         jk = ja(k) + inc
21:         thY(jk) = thY(jk) + au(k) ∗ xi
22:     end for
23:     thY(i + inc) = t
24: end for
25: $omp barrier
26: /*Step 3 - Accumulation in y:*/
27: for i = 1, nTh do
28:     inc = (i − 1) ∗ nEq
29:     $omp for
30:     for k = thHeight(i), thEnd(i) do
31:         y(k) = y(k) + thY(k + inc)
32:     end for
33:     $end omp for
34: end for
```

memory programming paradigm is completely different, and implies that each process works over its own set of data. This means that each process executes Algorithms 1 and 2 for its own mesh partition. These partitions can be obtained by a graph partitioner for unstructured meshes, thus balancing the loads assigned for each process. A good choice for this is the METIS software.[49,50] In our subdomain-by-subdomain approach, the sequential code is identical to its parallel version, with the addition of calls to the MPI library, whenever needed. Small modifications are necessary in the matvec product, with some adjustments in the data structure.

The efficiency of the parallelization depends on the optimization of communication between processes. This can be achieved by creating a communication mapping designed for point to point nonblocking communication. Most of the communication in the code is performed using this mapping, except in a few cases where a collective communication cannot be avoided. For example, in the case of inner product operations.

To clarify these ideas, let us consider the mesh of Figure 7. This mesh, with 16 cells, is divided into four overlapping partitions, as shown in Figure 8. In this figure, it can be seen that for each partition (solid lines), all cells on the internal boundaries are replicated (dashed lines), forming an overlapping partition. These overlapping cells are necessary in order to compute the contributions between neighboring cells. The cell labels in black refer to the local numbering of each partition, whilst the ones in red refer to the global numbering of the unpartitioned mesh (Figure 7). A

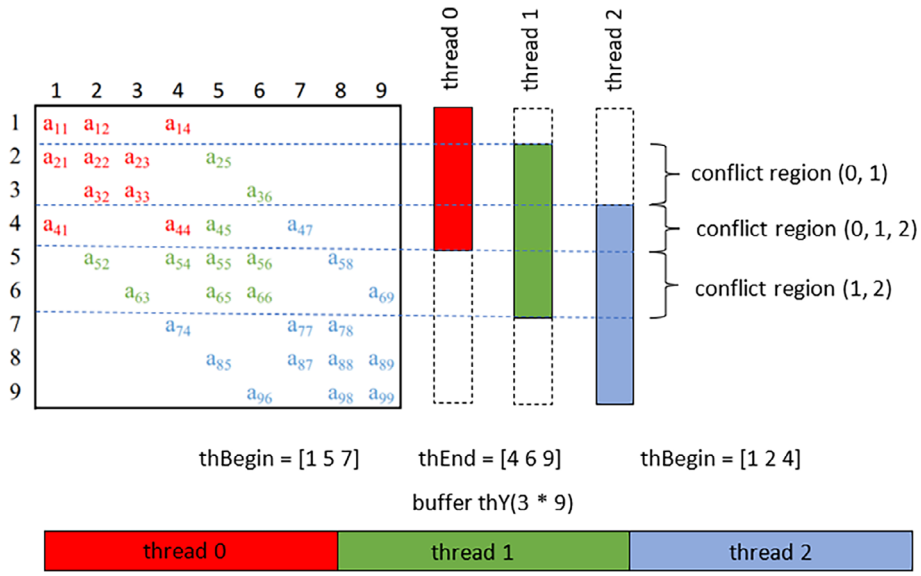

FIGURE 6 Use of the *buffer* in a system of nine equations, with three threads (red, green, and blue)

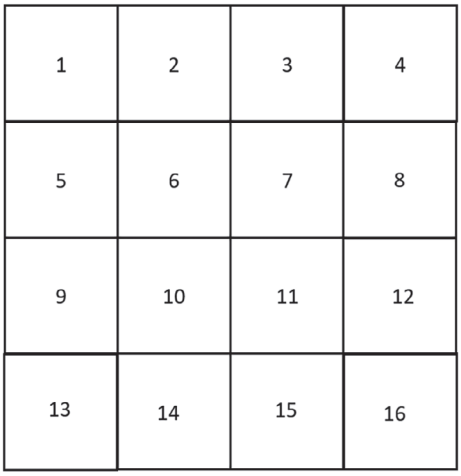

FIGURE 7 Original unpartitioned mesh with 16 cells

reverse Cuthill Mckee reordering[40] is applied to the local numbering of each partition. Only the nonoverlapping cells are renumbered.

Figures 9 and 10 show the global matrix referred to the original unpartitioned mesh and the corresponding rectangular local matrices of each partition. The colors in Figure 9 refer to each partition. In Figure 10, subscripts refer to the local numbering, and superscripts refer to the global numbering of the original unpartitioned mesh. Each rectangular matrix is divided into a square matrix and a rectangular part, as shown in Figure 10. The coefficients of the square matrix are related to the nonoverlapping cells, and a rectangular part includes coefficients belonging to the overlapping cells.

The loops over the cells do not require communication, except for when computing the gradients, where a single point to point communication instruction is required. As well as in the case of shared memory models, the MPI parallelization of the solvers requires special attention to three operations: vector updates, inner products and matvec operations, which will be described next.

Algorithm 10 presents the parallel code for inner product operation. The loop in lines 1–3 is performed over the equations of each partition ($nEq1$), without the overlapping cells. The sum of the contribution of each partition is

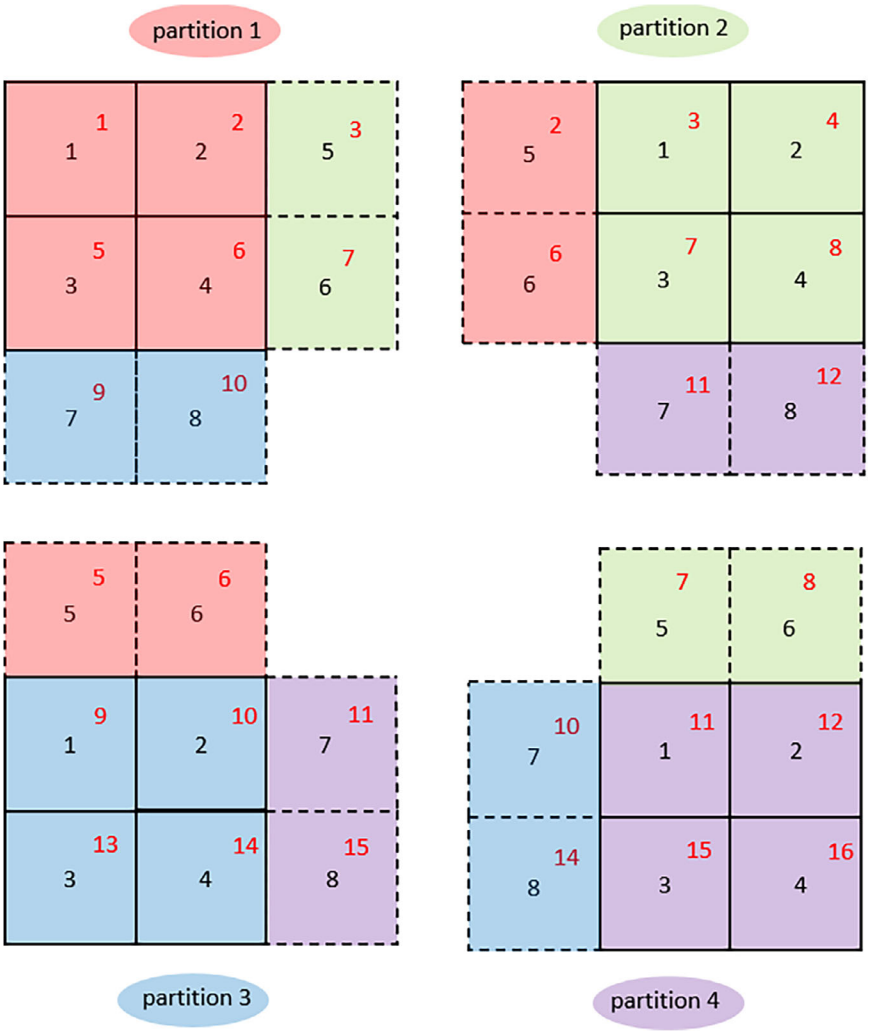

FIGURE 8 Mesh divided into four overlapping partitions

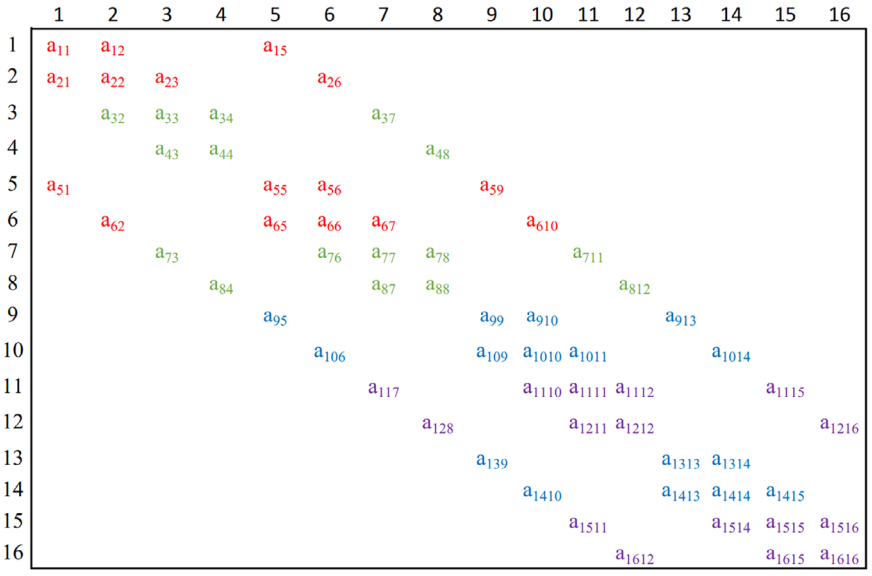

FIGURE 9 Coefficients matrix for the mesh of Figure 7. The colors red, green, blue, and purple refer, respectively, to partitions 1, 2, 3, and 4

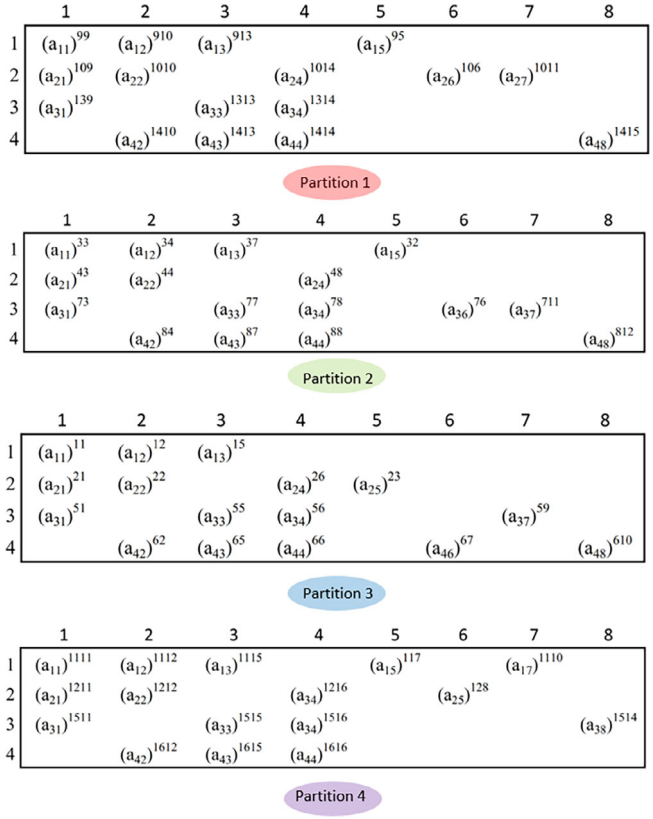

FIGURE 10 Coefficients rectangular matrix for each partition

Algorithm 10. Inner product (MPI version)

1: **for** $i = 1$, $nEq1$ **do**
2: $tmp = tmp + x1(i) * x2(i)$
3: **end for**
4: **MPI_ALLREDUCE**(tmp, dot, 1, **MPI_DOUBLE**, **MPI_SUM**)

computed in line 4, with a call to the collective function MPI_ALLREDUCE. Vector updates can be straightforwardly implemented, with a loop over $nEq1$, requiring no communication.

Figure 11 shows schematically the matvec product $y = Ax$ to be computed in each partition. As mentioned before, matrix A is rectangular, and is divided into a square matrix (in blue) and a rectangular part (in red). The red color refers to the coefficients of the overlapping cells. Matrix A has dimensions ($nEq1$, $nEq1 + nEq2$), vector x has dimension ($nEq1 + nEq2$) and the resulting vector y has dimension ($nEq1$). $nEq2$ is the number of equations corresponding to overlapping cells. In most real applications, $nEq1 \gg nEq2$.

To compute the matvec operation, the first step is to guarantee that all partitions have a copy of the entire vector x, including all $nEq1 + nEq2$ positions. This requires point to point communication between interface nonoverlapping and overlapping cells. Algorithm 11 presents the matvec operation code. In line 2, the function $com(x, fMap)$ uses a communication mapping to perform the necessary communication in a way that all partitions have their entire copy of x. The matvec product is performed in lines 4–19, where the main loop acts over the equations $nEq1$. The inner loop in lines 8–12 is responsible for the square matrix, stored in the CSRC format. Lines 14–17 compute the product corresponding to the rectangular part of the matrix, stored in the traditional CSR format, that makes use of the arrays iar, jar, and ar, similarly to the CRSC format.

Figure 12 presents the data structure corresponding to the communication traffic shown in Figure 13. The field R stands for receive operation, whilst the field S indicates a send operation. The first part of the array $fMap$ of partition i

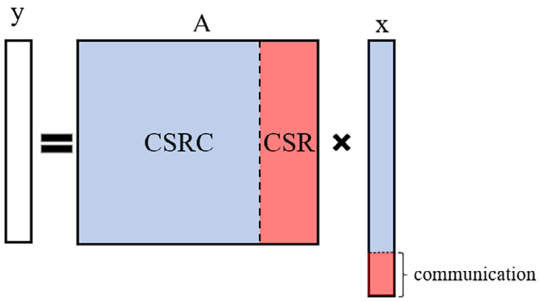

FIGURE 11 Matvec product computed in each partition

Algorithm 11. Matrix vector product (MPI version)

```
1:  /*Communication:*/
2:  com(x, fMap)
3:  /*Matrix vector product:*/
4:  for i = 1, nEq1 do
5:      xi = x(i)
6:      t = ad(i) * xi
7:      /*Square part:*/
8:      for k = ia(i), ia(i + 1) do
9:          jk = ja(k)
10:         t = al(k) * x(jk)
11:         y(jk) = y(jk) + au(k) * xi
12:     end for
13:     /*Rectangular part:*/
14:     for k = iar(i), iar(i + 1) do
15:         jk = jar(k)
16:         t = ar(k) * x(jk)
17:     end for
18:     y(i) = t
19: end for
```

contains all cells from other partitions that send information to partition i. The second part indicates the cells in partition i that send information to other partition cells. The cells in *fMap* are ordered in an ascending order, within an ascending order of partitions. At this stage, the global cell numbering is used. After reordering, each array *fMap* stores the cells in local numbering for partition i.

Figure 14 shows the array *fMap*, referred to the local numbering of partition i. Each array *fMap* has 3 integer auxiliary arrays: *ngh*, *rcvs*, and *send*. The array *ngh* stores the partitions involving communication for partition i. The array *rcvs* is a pointer in *fMap* indicating a starting position of the information to be retrieved from partition i. The array *send* is a pointer in *fMap* indicating the starting position of the information to be sent to each partition listed in *ngh*.

Algorithm 12 shows the code of the function $com(x, fMap)$, that performs all communication using the array *fMap* and its auxiliary data structure. In input, the array x is the incomplete multiplier array in the product $y = Ax$ of each partition, still lacking information from other partitions. In output, the array x is completely updated for all partitions. In lines 2–5, a send/receive buffer *xb* retrieves, using *fMap*, the coefficients to be sent by each partition. Lines 7–17 perform all necessary communication using the nonblocking point to point functions MPI_ISEND and MPI_IRECV. In lines 21–24, the values in *xb* are recovered and stored in the multiplier vector x. A synchronization function must be invoked before this recovering operation, in line 19.

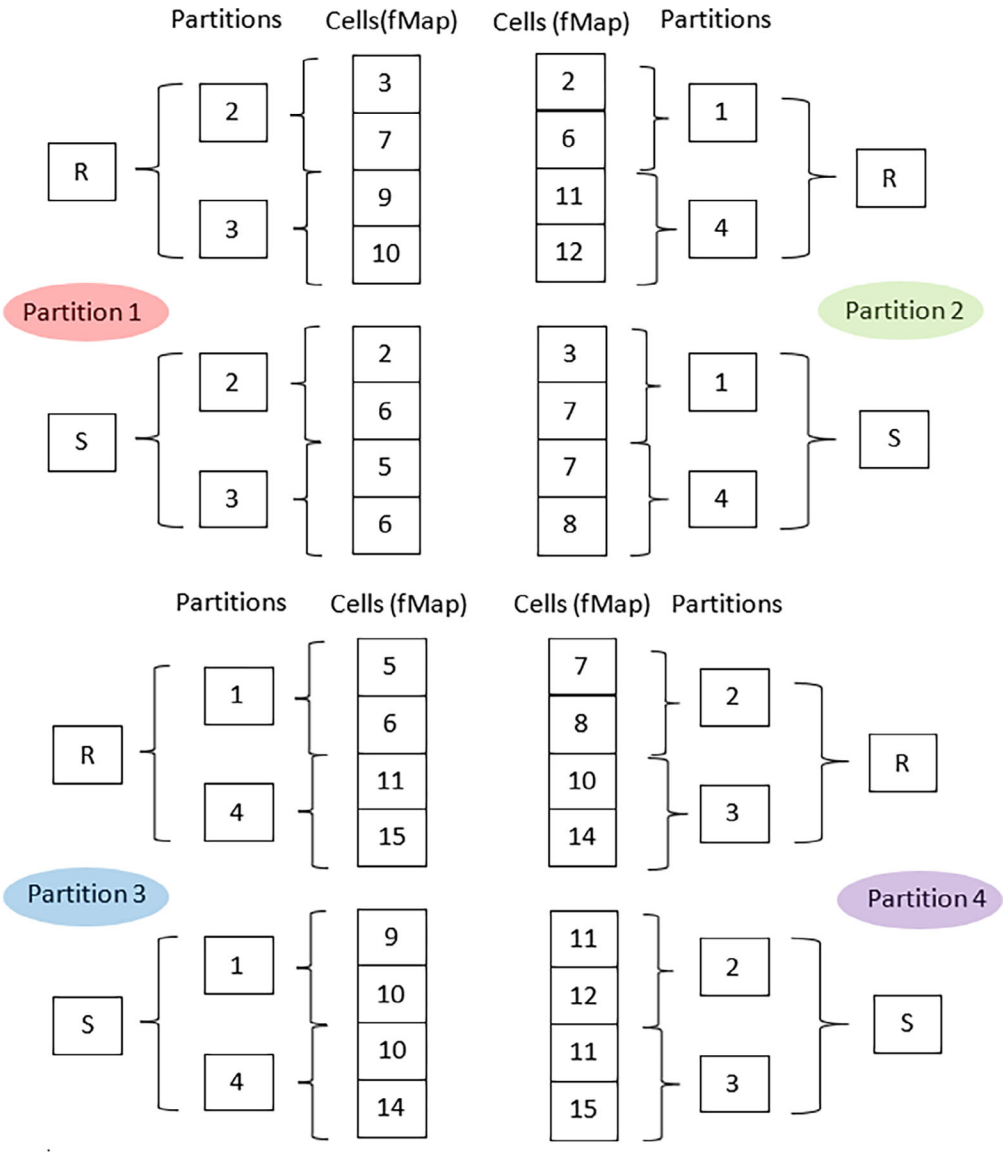

FIGURE 12 Communication mapping in global cell numbering

4.3 | Hybrid parallelization

In a hybrid parallelization, both paradigms for distributed and shared memory are used at the same time. Fortunately, the MPI tool is built with support to multi-threaded codes, which allows for the simultaneous use of OpenMP directives. The MPI has an internal variable that defines the way threads can behave. By default, the MPI initializes with the variable MPI_THREAD_SINGLE set, which means that only one thread can make calls to MPI routines. When the variable MPI_THREAD_FUNNELED is set, only the master thread can make MPI calls. There are other possibilities, setting the variables MPI_THREAD_SERIALIZED or MPI_THREAD_MULTIPLE, both allowing multiple threads to make MPI calls. Our hybrid parallelization model is based on the MPI's default model. Therefore, we need to rewrite the codes for inner product and matvec operations in their hybrid versions, shown in Algorithms 13 and 14. As can be noted, these algorithms are a combination of Algorithms 8–10 and Algorithms 9–11. In these hybrid versions, the directive *omp single* must be used in order to guarantee that just one thread calls MPI routines. In this case, this thread is the first one reaching that line of the code.

Therefore, our hybrid parallel model can run n MPI processes, each of these opening m threads. For each node, $n * m$ must be less than or equal to the total number of cores.

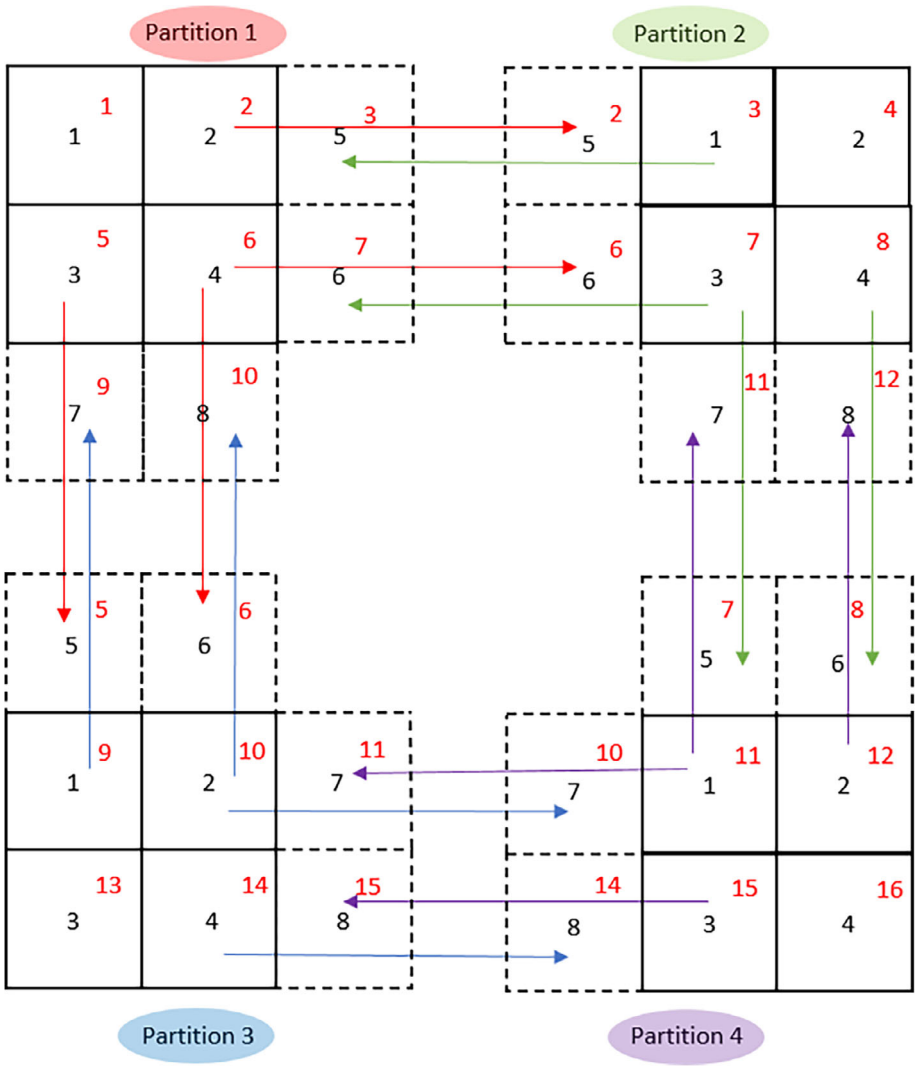

FIGURE 13 Comunnication traffic between partitions

	fMap								rcvs			send			ngb	
partition 1	5	6	7	8	2	4	3	4	1	3	5	5	7	9	2	3
partition 2	5	6	7	8	1	3	3	4	1	3	5	5	7	9	1	4
partition 3	5	6	7	8	1	2	2	4	1	3	5	5	7	9	1	4
partition 4	5	6	7	8	1	2	1	3	1	3	5	5	7	9	2	3

FIGURE 14 Communication mapping data structure

Algorithm 12. Function *com*

```
/*Gathering send information*/
for i = 1, nSends do
    lCel = fMap(i)
    xb(i) = x(lCel)
end for
/*Communication*/
for i = 1, nPart do
    partID = ngb(i) − 1
    /*Send:*/
    k = send(i)
    kk = send(i + 1) − send(i)
    MPI_ISEND(xb(k), kk, MPI_DOUBLE, partId, myID)
    /*Recv:*/
    k = rcvs(i)
    kk = rcvs(i + 1) − rcvs(i)
    MPI_IRECV(xb(k), kk, MPI_DOUBLE, partId, partID)
end for
/*Wait for communications to finish*/
MPI_WAITALL()
/*Recovered and store recv information*/
for i = 1, nRvcs do
    lCel = fMap(i)
    x(lCel) = xb(i).
end for
```

Algorithm 13. Parallel inner product (hybrid version)

```
$omp single
dotPar = 0.0
$end omp single
$omp for reduction(+:dotPar)
for i = 1, nEq1 do
    dotPar = dotPar + x1(i) ∗ x2(i)
end for
$end omp for
$omp single
MPI_ALLREDUCE(dotPar, dot, 1, MPI_DOUBLE, MPI_SUM)
$end omp single
```

Algorithm 14. Matrix vector product (hybrid version)

```
1: $omp single
2: com(x, fMap)
3: $end omp single
4: id = omp_get_thread_num()
5: for i = 1, nTh do
6:     inc = (i − 1) ∗ nEq1
7:     $omp for
8:     for k = thHeight(i) + inc, thBegin(i) − 1 do
9:         thY(k) = 0.0
10:    end for
11:    $end omp for
12: end for
13: inc = id ∗ nEq1
14: for i = thBegin(id), thEnd(id) do
15:    y(i) = 0.0
16:    xi = x(i)
17:    t = ad(i) ∗ xi
18:    for k = ia(i), ia(i + 1) do
19:        jk = ja(k)
20:        t = al(k) ∗ x(jk)
21:        jk = ja(k) + inc
22:        thY(jk) = thY(jk) + au(k) ∗ xi
23:    end for
24:    for k = iar(i), iar(i + 1) do
25:        jk = jar(k)
26:        t = ar(k) ∗ x(jk)
27:    end for
28:    thY(i + inc) = t
29: end for
30: $omp barrier
31: for i = 1, nTh do
32:    inc = (i − 1) ∗ nEq
33:    $omp for
34:    for k = thHeight(i), thEnd(i) do
35:        y(k) = y(k) + thY(k + inc)
36:    end for
37:    $end omp for
38: end for
```

5 | NUMERICAL EXAMPLES

In this section we run our tests using two 3D examples, the lid-driven cavity problem for pure incompressible flow (test case 1) and other for reactive flow (test case 2). Figure 15 shows the geometry and boundary conditions. The density is $\rho = 1.0$ kg/m^3, and the dynamic viscosity is $\mu = 10^{-3}$ Pa s. The pressure and velocity are set to 0, as initial conditions.

The second test case is a reactive flow in the domain shown in Figure 16. The domain has two outlets and two inlet surfaces. The other two surfaces are impermeable with no-slip conditions and are also adiabatic. A pure fuel ($Y_{CH4} = 1.0$) is injected in one inlet whilst a flow of air ($Y_{O2} = 0.23$ and $Y_{N2} = 0.77$) is prescribed at the opposite inlet surface. In both inlets, the speed and temperature are, respectively, 0.1 m/s and 25°C. The density is implicitly defined by the temperature and the mass fractions. In the outlet, all normal derivatives are considered to be equal to 0, except for the pressure, which

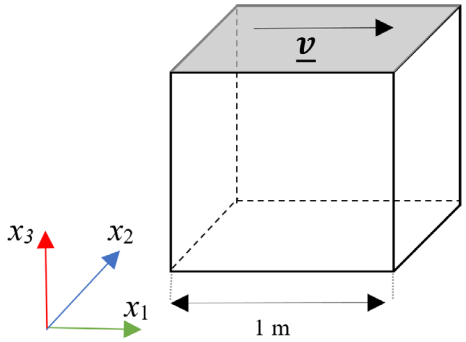

FIGURE 15 Lid-driven cavity problem: boundary conditions are: $|\underline{\boldsymbol{v}}| = 1$ m/s at the top, and $\nabla p \cdot \underline{\boldsymbol{n}} = 0$ and $\underline{\boldsymbol{v}} = \underline{\boldsymbol{0}}$ at all other faces

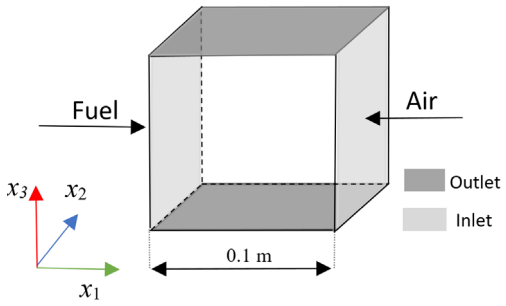

FIGURE 16 Reactive flow: geometry and boundary conditions

TABLE 2 Meshes for cases 1 and 2

Mesh	Cells	Nodes	ndf_i	ndf_r
A	125,000	132,651	530,604	1,125,000
B	571,787	592,704	2,287,148	5,146,083
C	1,000,000	1,003,301	4,000,000	9,000,000
D	2,352,637	2,406,104	9,410,548	21,173,733

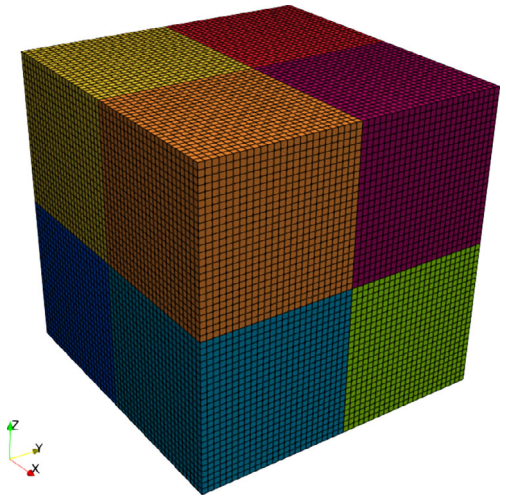

FIGURE 17 Typical mesh of hexahedrons with eight partitions

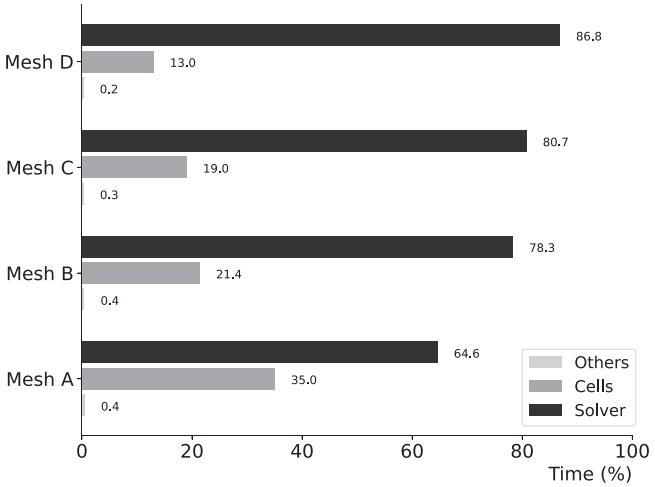

FIGURE 18 Time percentages in a sequential run for test case 1

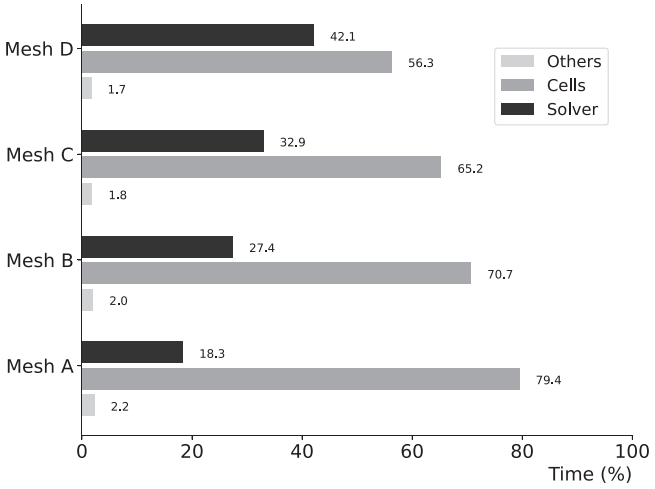

FIGURE 19 Time percentages in a sequential run for test case 2

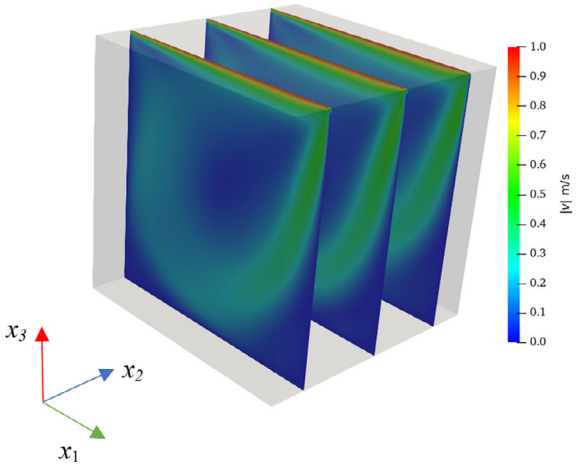

FIGURE 20 Velocity field at time $t = 120$ s for case 1, mesh A

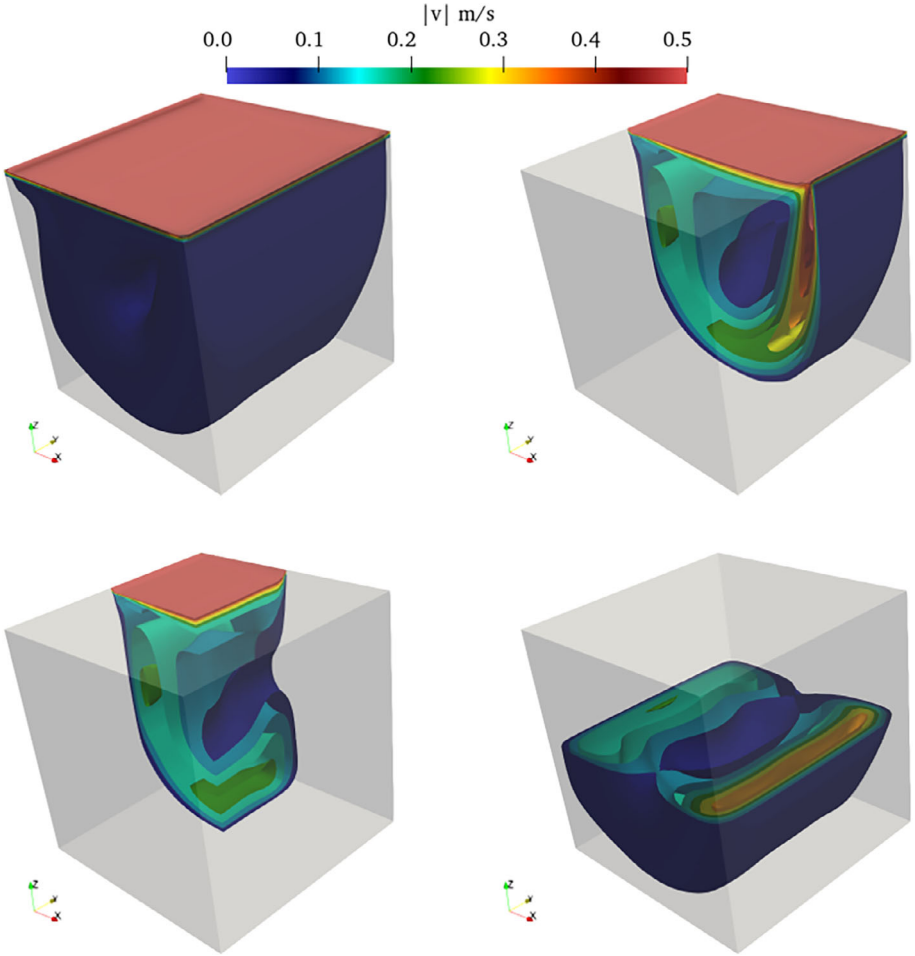

FIGURE 21 Isosurfaces of the velocity field at time $t = 120$ s for case 1, mesh A

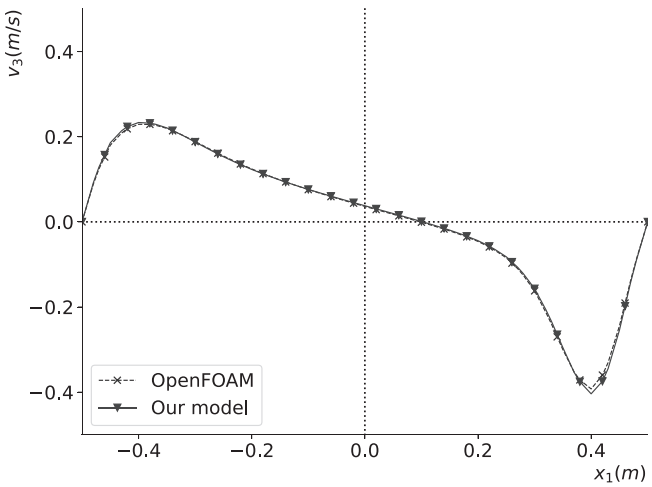

FIGURE 22 Velocity field at time $t = 120$ s across a line passing through the center of the cube, parallel to x_1. Comparison OpenFOAM x our model, case 1, mesh A

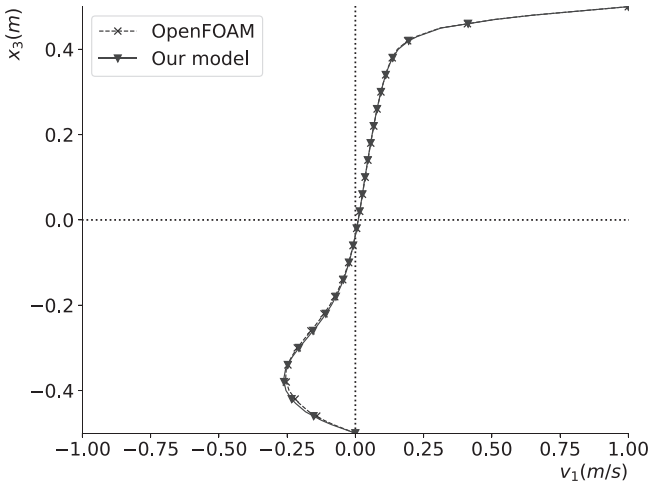

FIGURE 23 Velocity field at time $t = 120$ s across a line passing through the center of the cube, parallel to x_3. Comparison OpenFOAM x our model, case 1, mesh A

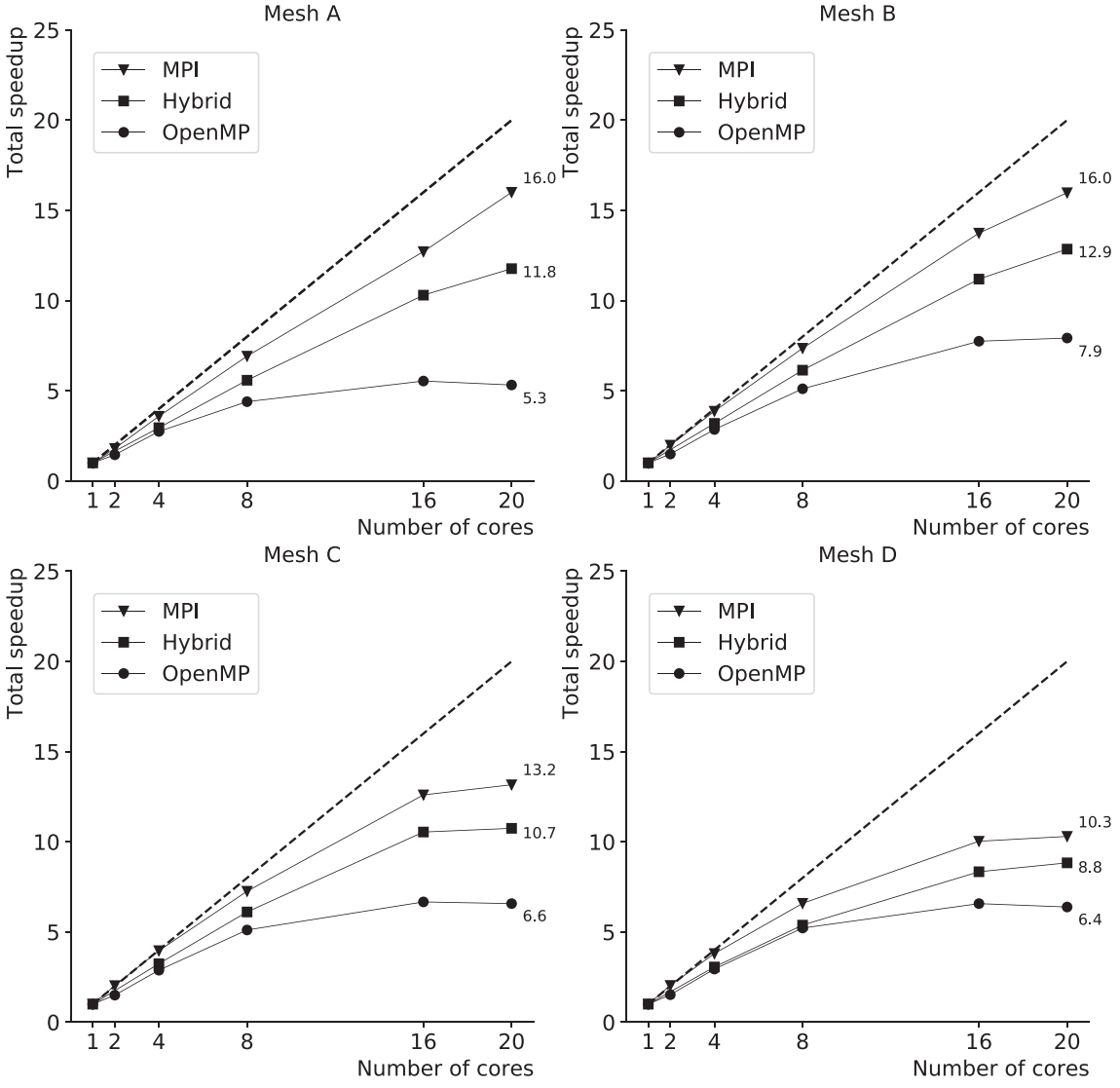

FIGURE 24 Total speedup curves for one node, case 1

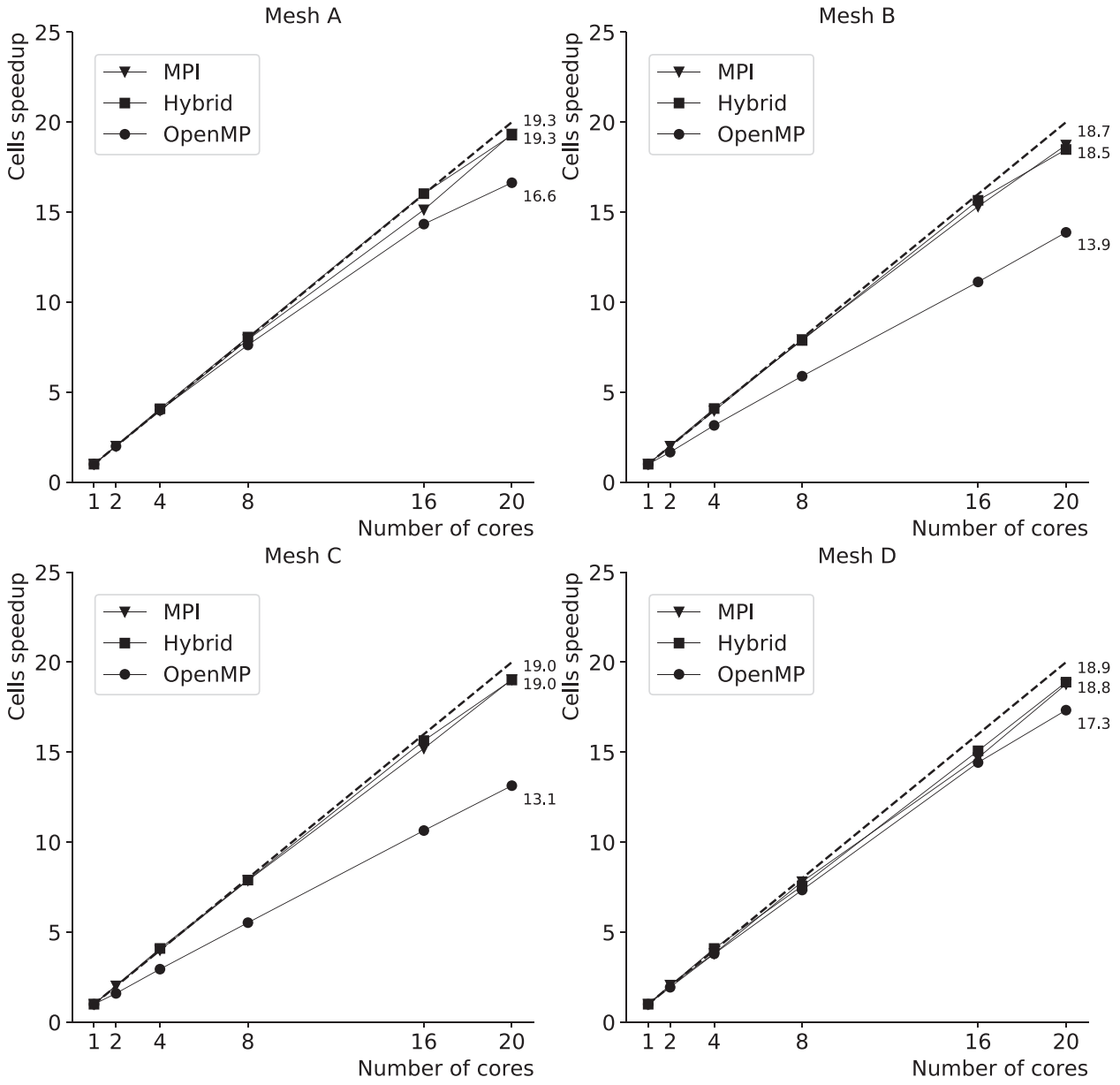

FIGURE 25 Cell loop speedup curves for one node, case 1

is $p = 0$. As initial conditions, all variables are set to 0, except for $Y_{N2} = 1.0$. The variation of the physical properties c_p, μ, k, and D_n are given by Equations (10), (11), (12), and (13). The chemical reaction is given by the methane (CH_4) global combustion,

$$CH_4 + 2O_2 \rightarrow CO_2 + 2H_2O. \tag{56}$$

Both problems were discretized by four meshes, A, B, C, and D, composed by hexahedrons, as shown in Table 2. In this table, ndf_i is the number of degrees of freedom for test case 1 and ndf_r is the number of degrees of freedom for test case 2. Figure 17 shows a typical mesh with eight partitions.

Figures 18 and 19 present the time percentages of the solver, loop over the cells, and other operations, for test cases 1 and 2, in a sequential run. As it can be observed from these figures, the solver is predominant in test case 1 whilst the loop over the cells is the most time consuming task in test case 2, for the chosen meshes. However, the solver time percentage tends to increase as the mesh is refined, in both problems. In what follows, we discuss the results for both problems. The

MPI function MPI_WTIME was used to perform time measurements. This function measures the overall time, rather than the net CPU time.

5.1 | Test case 1

The velocity field obtained at time $t = 120$ s is shown in Figures 20 and 21. Figures 22 and 23 compare our results with those obtained by running the problem with the OpenFOAM software, for mesh A. As can be seen, the results are practically identical across two lines passing through the center of the cube, one parallel to the axis x_1, and the other parallel to x_3.

In order to compare the three implementations, MPI, OpenMP, and hybrid, we show the results in terms of speedups for 1 node of the cluster (Figures 24–26). For the hybrid version, only two threads are opened for each MPI process. The performance of the MPI and hybrid implementations are almost the same for the speedups of the loop over the cells. It is also clear that the MPI overall results are better than the other two approaches. It is important to note that in this case there is no network communication. These results were measured for one time step.

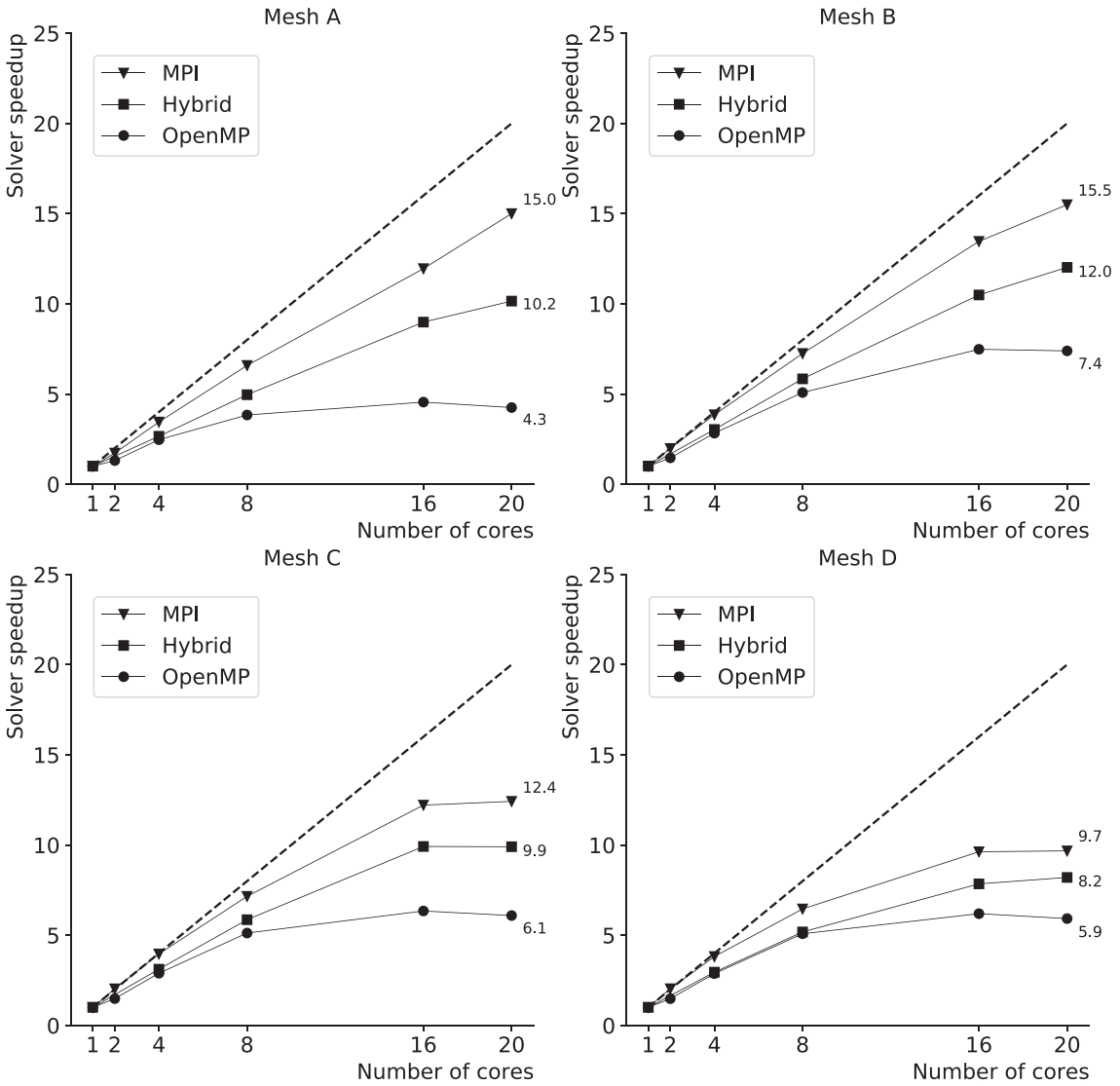

FIGURE 26 Solver speedup curves for one node, case 1

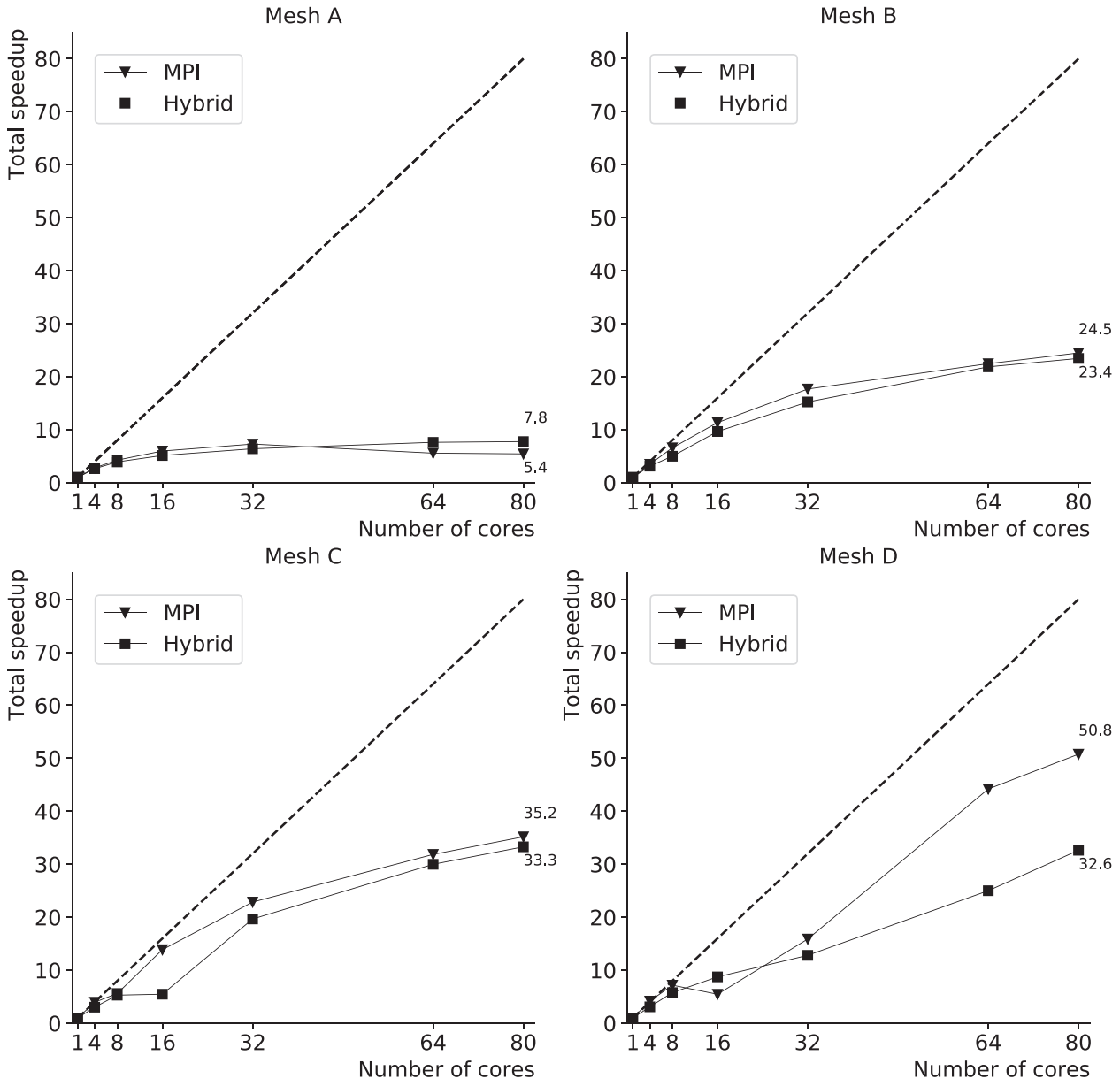

FIGURE 27 Total speedup curves: MPI × hybrid, case 1

For a comparison MPI × hybrid, we can use all nodes of the cluster, reaching its maximum capacity with 80 cores. This comparison is shown in Figures 27–29. The hybrid version presents a slightly superior performance considering the loop over the cells, but the gain in the solver phase makes the MPI the best choice for all meshes, A, B, C, and D. Figure 30 presents the total solution times for one time step, using meshes A, B, C, and D. For mesh D (2,352,637 cells and 9,410,548 equations), considering 80 cores, the total solution times for one time step were equal to 45 s in the MPI implementation and 70s for the hybrid approach.

Figures 31 and 32 compare the MPI results of our model to the OpenFOAM software (version 1906), in terms of total speedups and the ratio OpenFOAM total solution time/our model total solution time. These graphs show a significant difference in favor of our model, in all situations. For a fair comparison, these measurements were carried out considering the same number of solver and nonlinear iterations for both models, using the same solvers. Also, both codes were compiled with the Intel C/C++ compiler. The optimization level is O3.

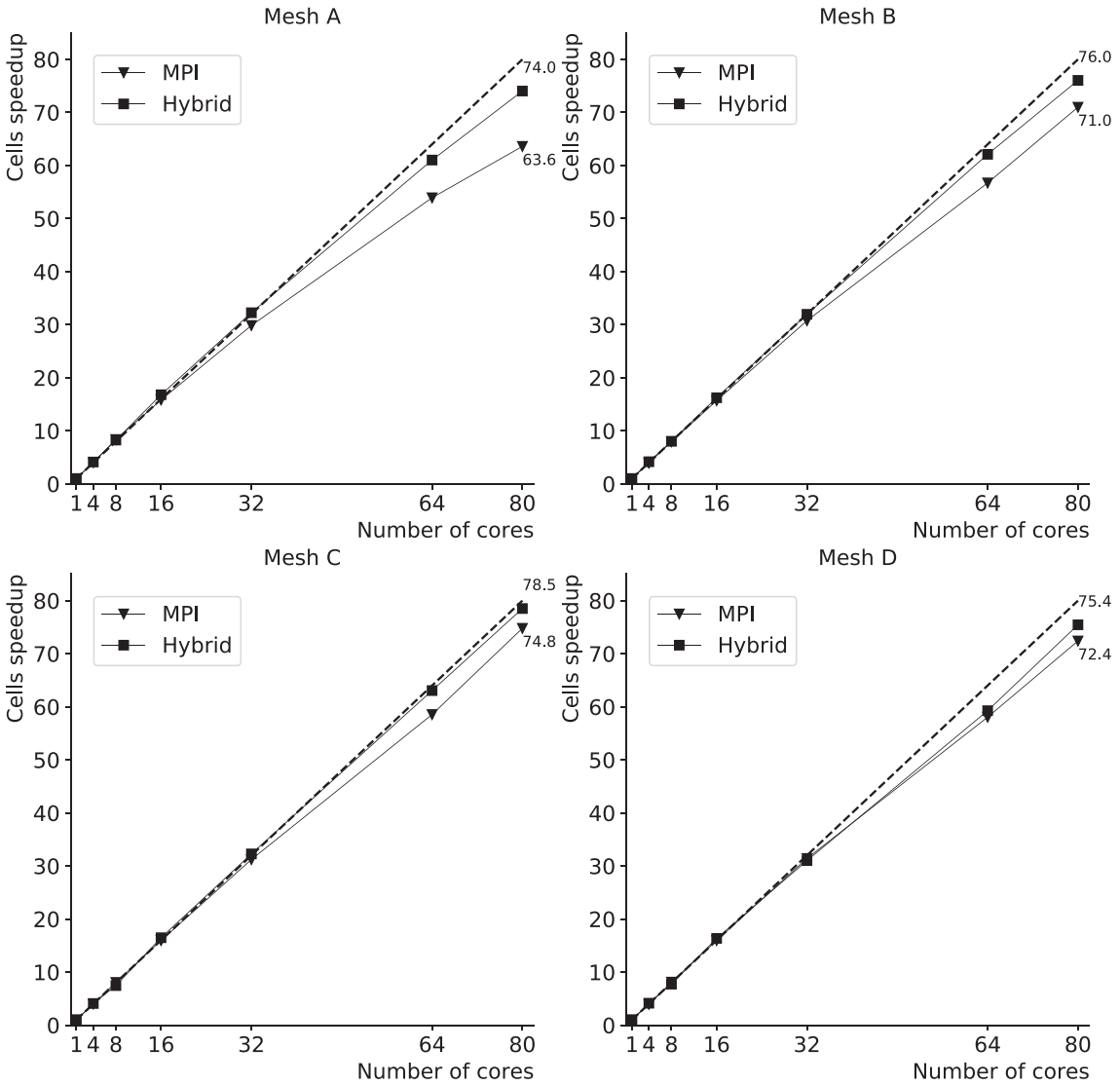

FIGURE 28 Cell loop speedup curves: MPI × hybrid, case 1

5.2 | Test case 2

Figure 33 shows the heat energy release ($\dot{\omega}_T$) by the chemical reaction between the gases CH_4 and O_2 at time $t = 2$ s. Figures 34 and 35 present the velocity field and temperature for time $t = 2$ s obtained by using both our model and the OpenFOAM software.

As in case 1, to compare the MPI, the pure OpenMP, and the hybrid parallel versions it is necessary to run the problem in just a single machine. These results, in terms of speedups, are shown in Figures 36–38, for meshes A, B, C, and D. As can be seen in these graphs, the results are very similar to the ones obtained for case 1, except for the solver, which presents lower speedups. This is due to the fact that the CG is called 1 time and the BICGSTAB is called 8 times in Algorithm 1, whilst Algorithm 2 has one call to the CG solver and 3 calls to the BICGSTAB solver. The BICGSTAB has 2 matvec operations per iteration, against 1 of the CG. This means that the amount of communication in the BICGSTAB doubles the communication in the CG matvec product. The loss in the solver speedups is compensated by the fact that in case 2 the most time consuming phase is the loop over the cells, thus leading to similar total speedups obtained for case 1.

A comparison between the MPI and the hybrid versions using all nodes can be seen in Figures 39–41. Figure 42 presents the total solution times for one time step, using meshes A, B, C, and D. For mesh D (2,352,637 cells and 21,173,733

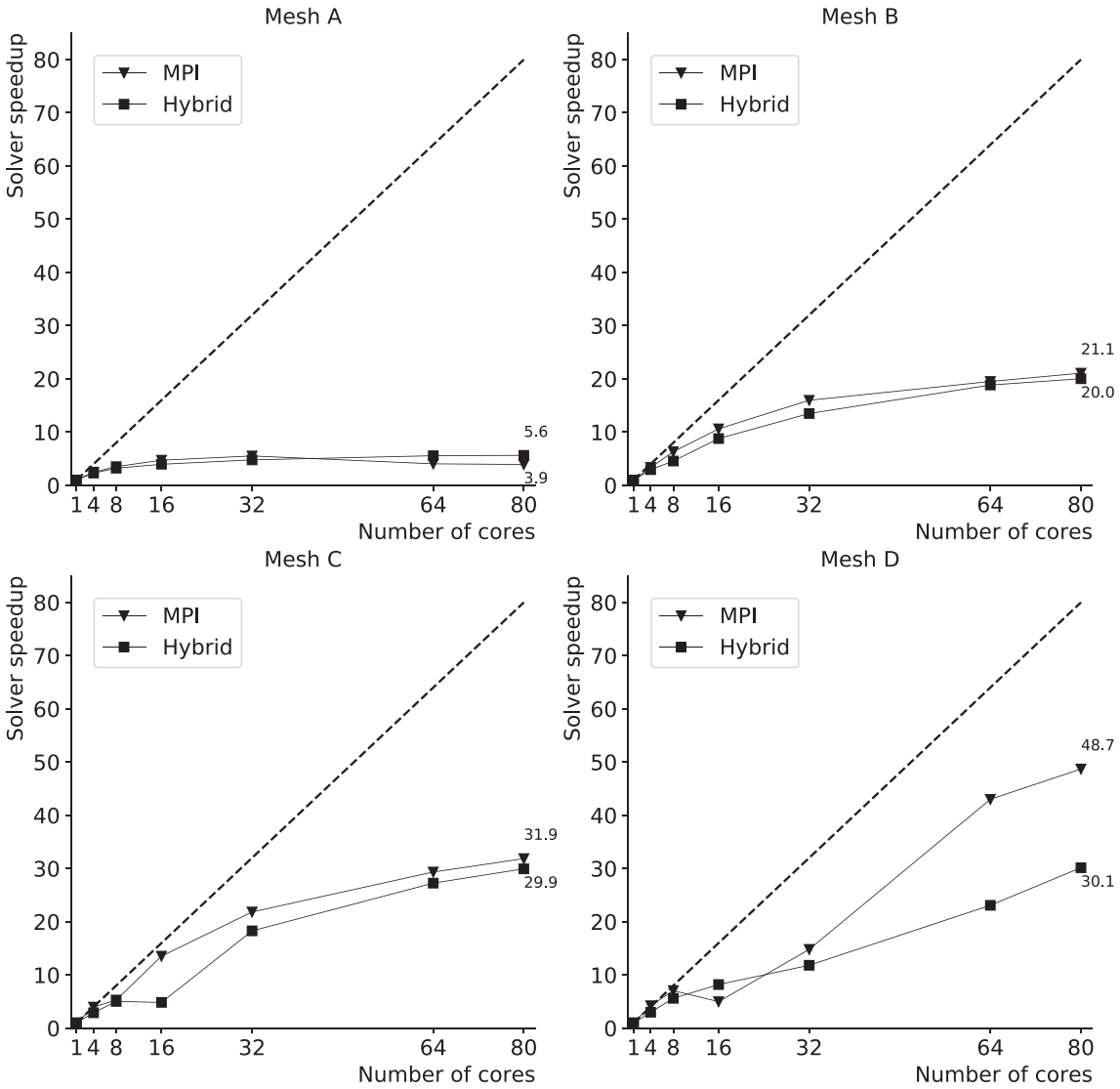

FIGURE 29 Solver speedup curves: MPI × hybrid, case 1

equations), considering 80 cores, the total solution times for one time step were equal to 16 s in the MPI implementation and 18 s for the hybrid approach.

5.3 | Scalability

One of the most important parameters for measuring the efficiency of a parallel implementation is its scalability. Scalability is defined, for a fixed size problem, as being the rate in which the speedups increase as the number of processes or threads increase. Finite volumes, as well as finite elements, are typically coarse grain applications, due to the strong coupling of the solver. In both methods, the scalability is ruled by the ratio r = number of nonoverlapping/overlapping cells. This ratio is also related to the ratio noninternal boundary cells/internal boundary cells. The scalability can be measured by the efficiency, defined as the actual speedup divided by the ideal speedup. Figure 43 shows the ratio r, with an increasing number of partitions. The average ratio r and its standard deviation, for meshes A, B, C, and D, can be seen in Table 3. The standard deviation is an important parameter that measures the load balance between partitions. Figures 44 and 45 present the efficiency for test cases 1 and 2, considering the MPI implementation. We observe in these figures that the points in the curves where there is a loss in efficiency correspond to points of higher standard deviations. This is clearly the case of mesh D with 16 partitions.

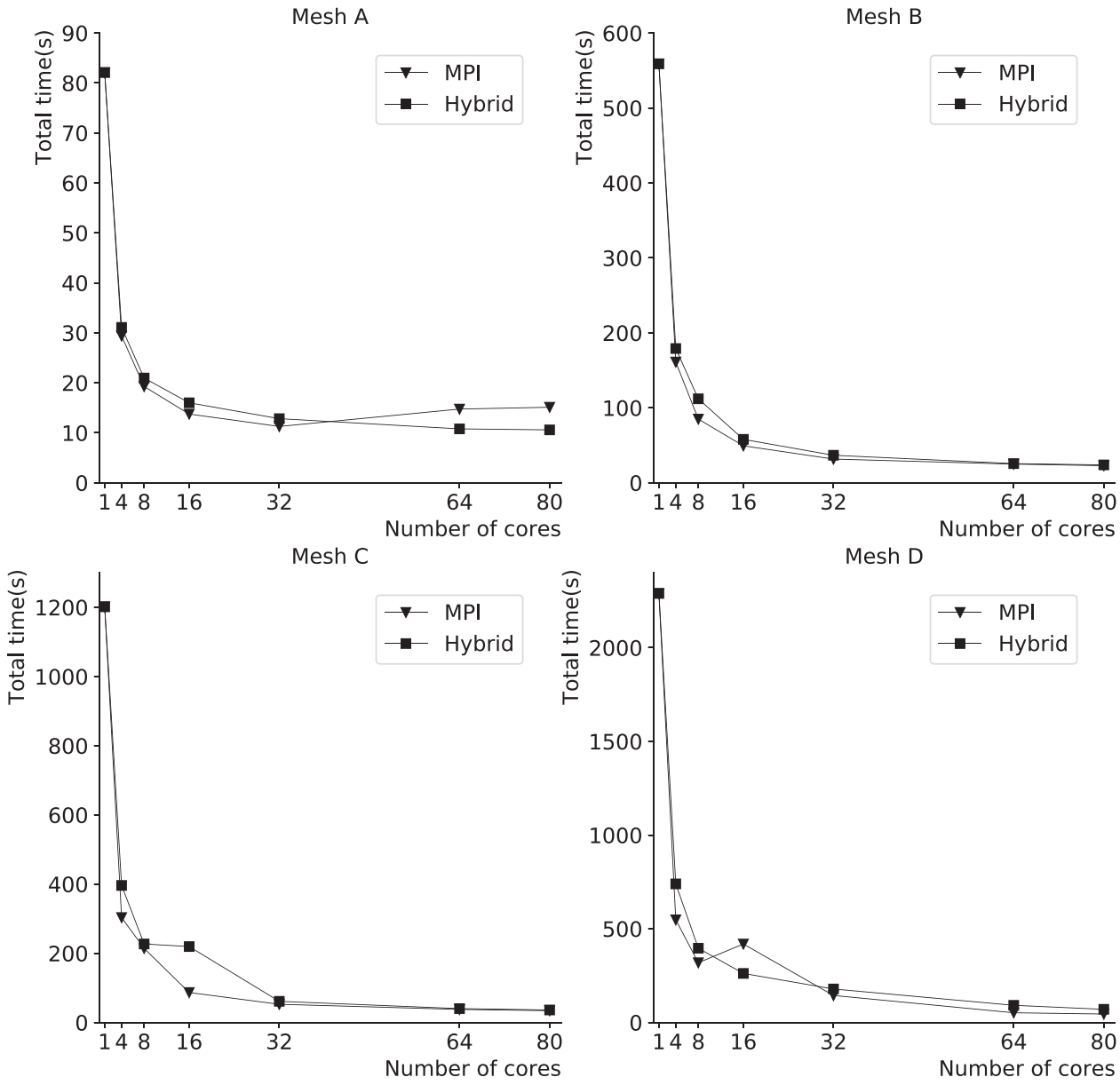

FIGURE 30 Total times for one time step: MPI × hybrid, case 1

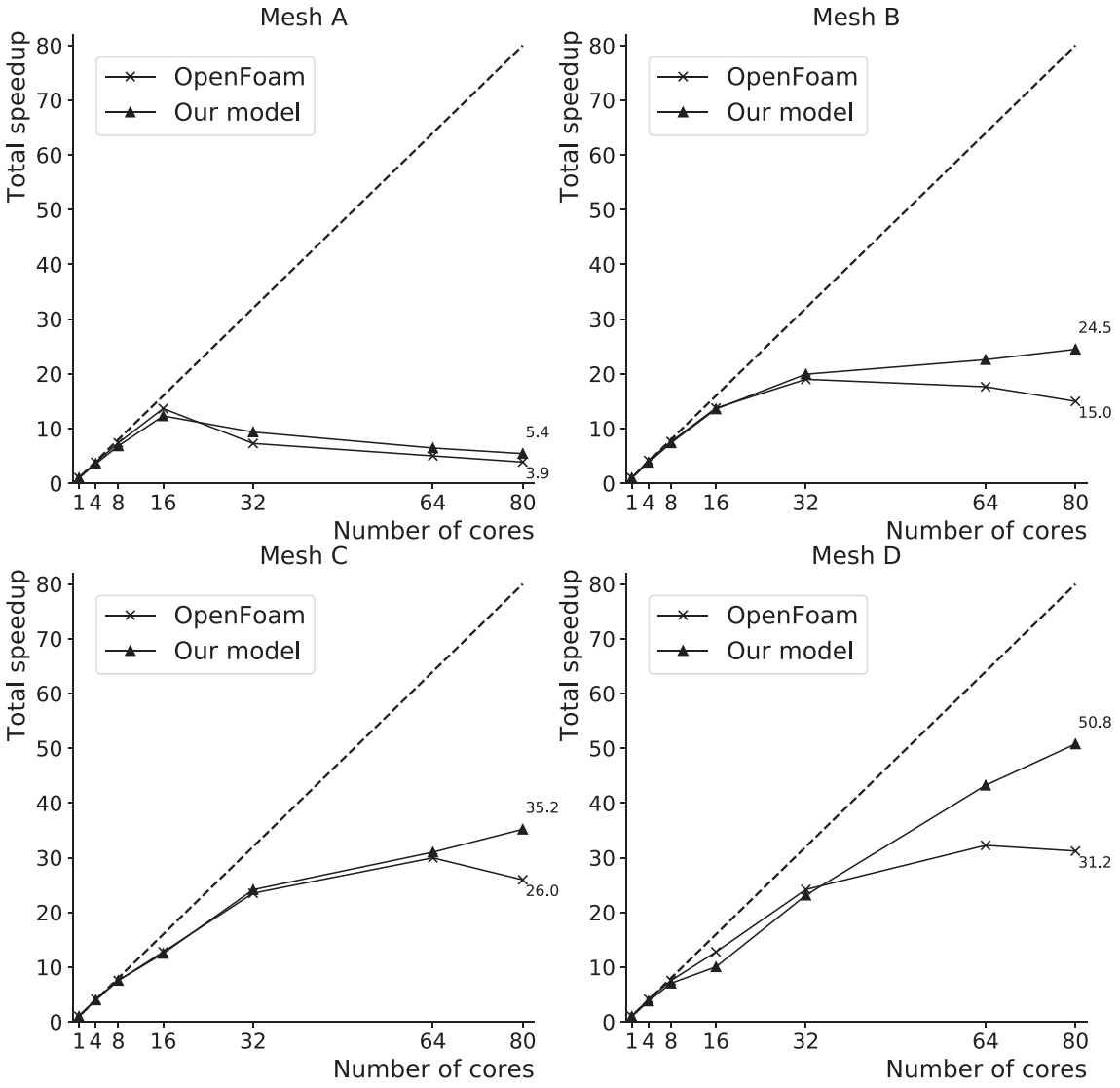

FIGURE 31 MPI total speedups, OpenFOAM × our model, case 1

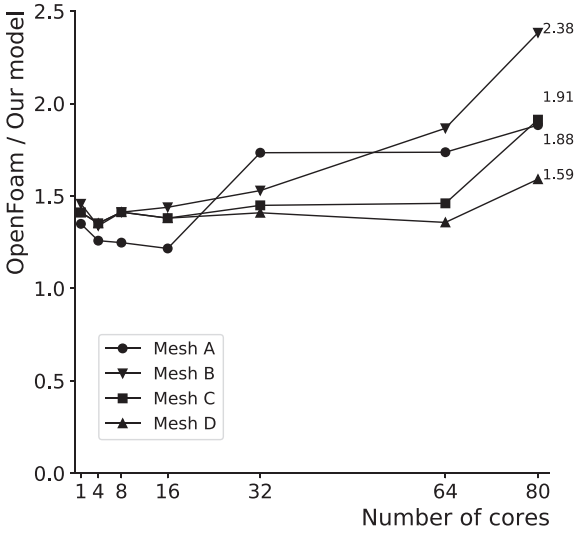

FIGURE 32 Ratio OpenFOAM total time/our model total time, case 1

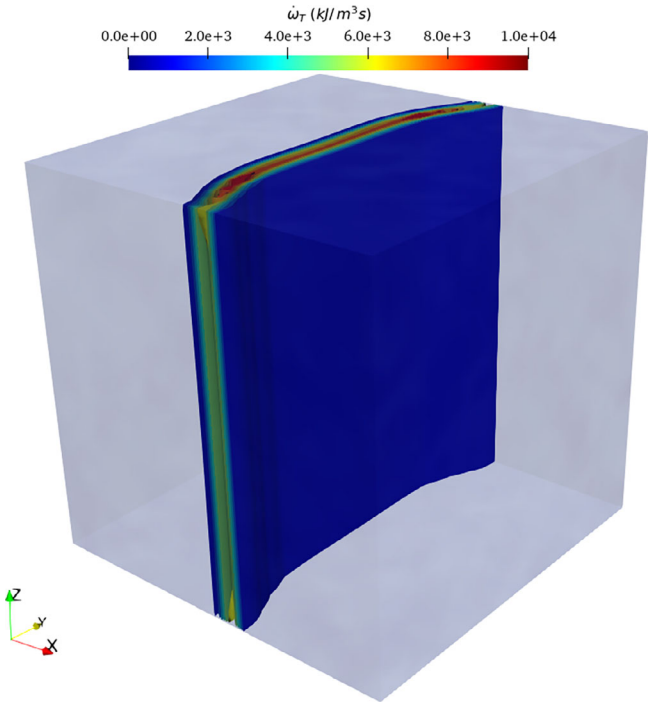

FIGURE 33 Isosurfaces for the energy release at time $t = 2$ s, for case 2, mesh A

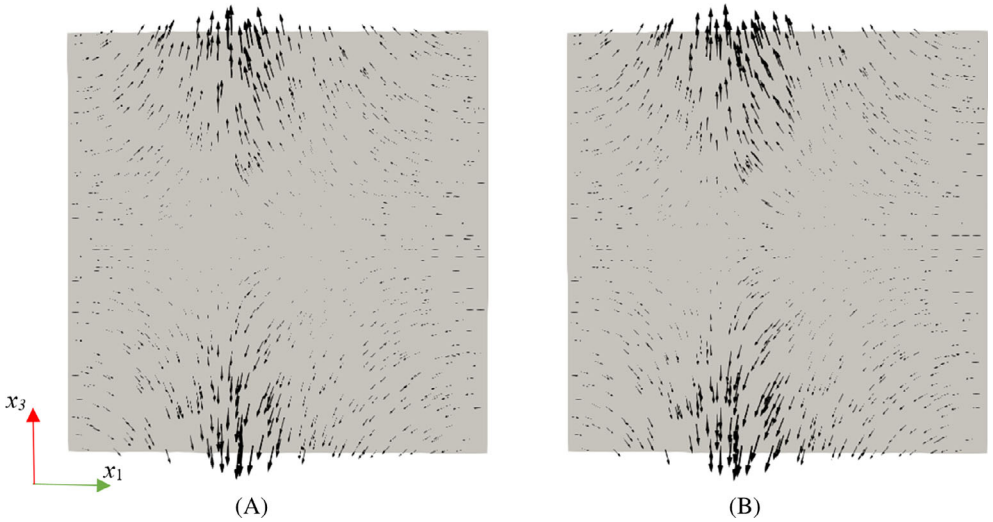

FIGURE 34 Velocity fields at time $t = 2$ s for case 2, mesh A: (A) our model and (B) OpenFOAM

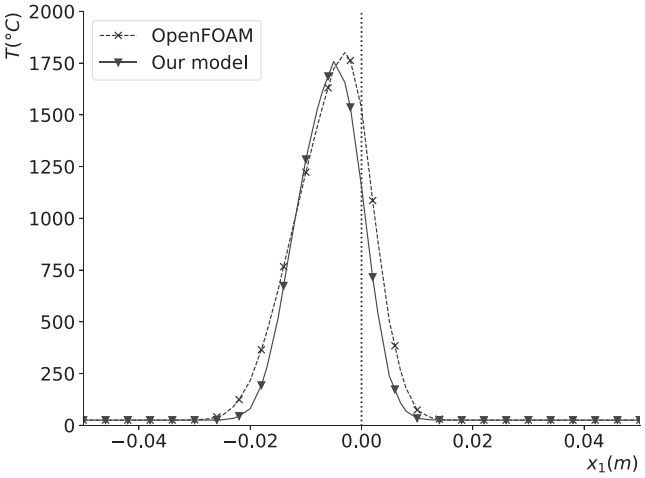

FIGURE 35 Temperature at time $t = 2$ s for case 2, mesh A

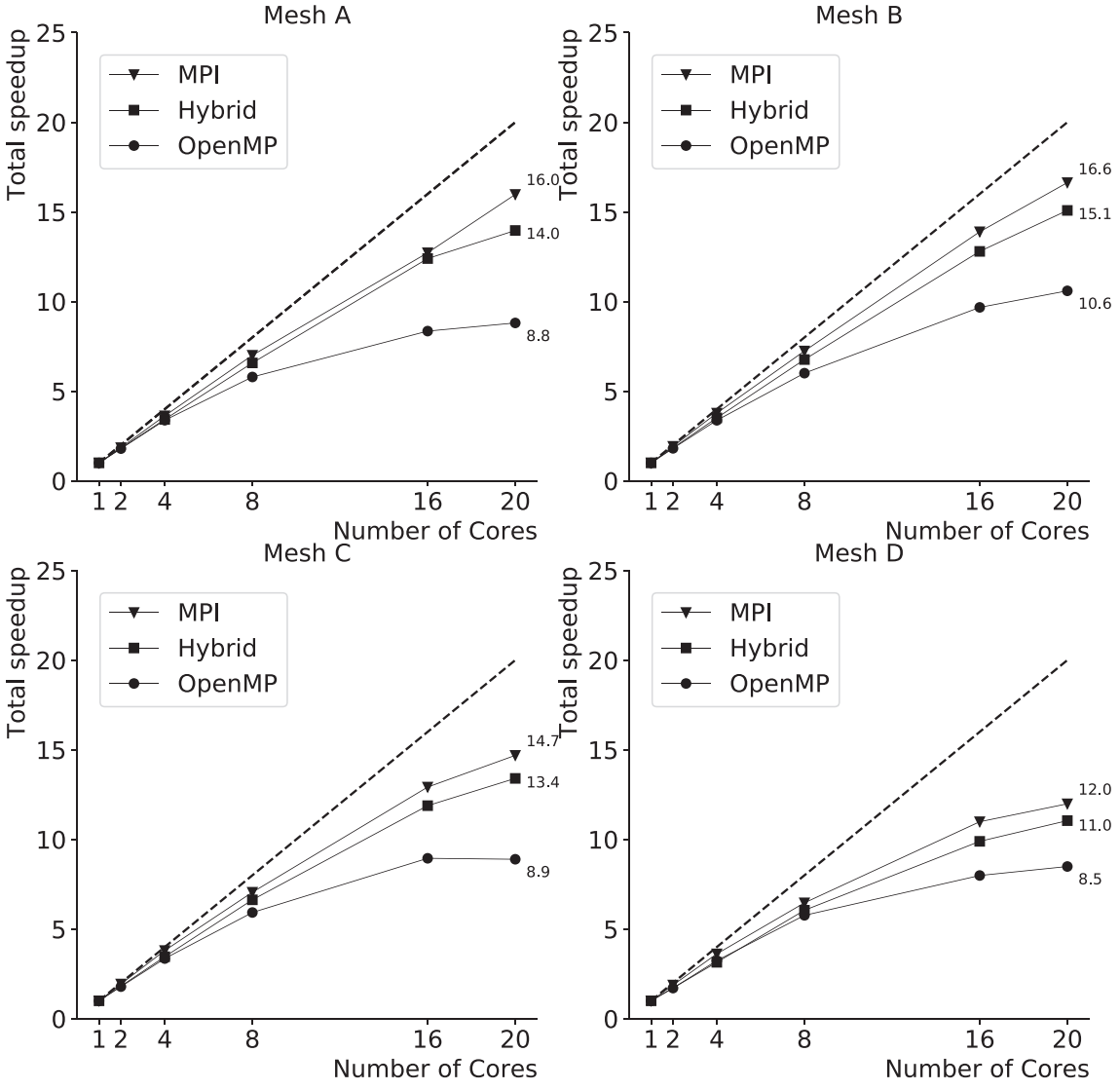

FIGURE 36 Total speedup curves for one node, case 2

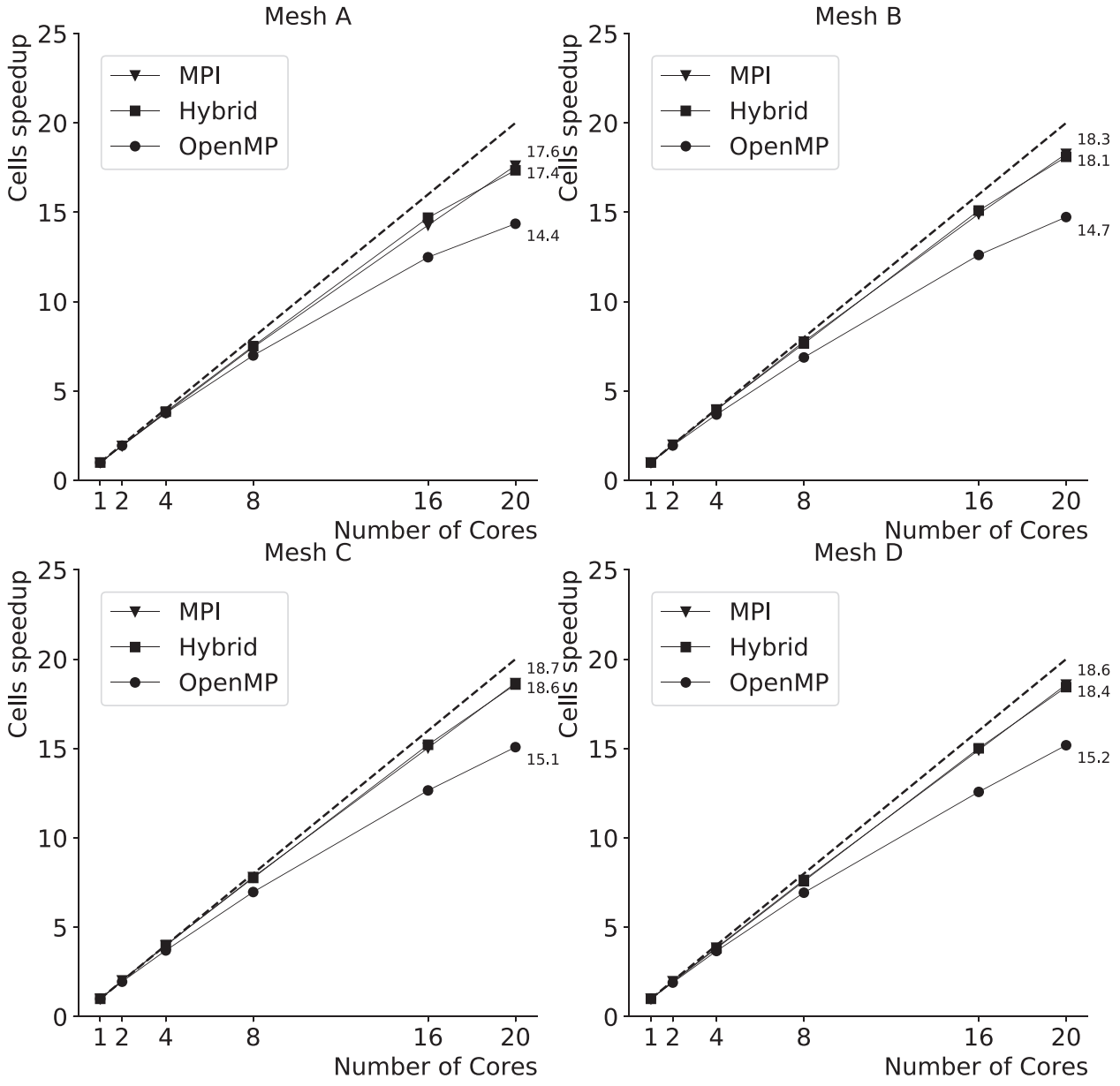

FIGURE 37 Solver speedup curves for one node, case 2

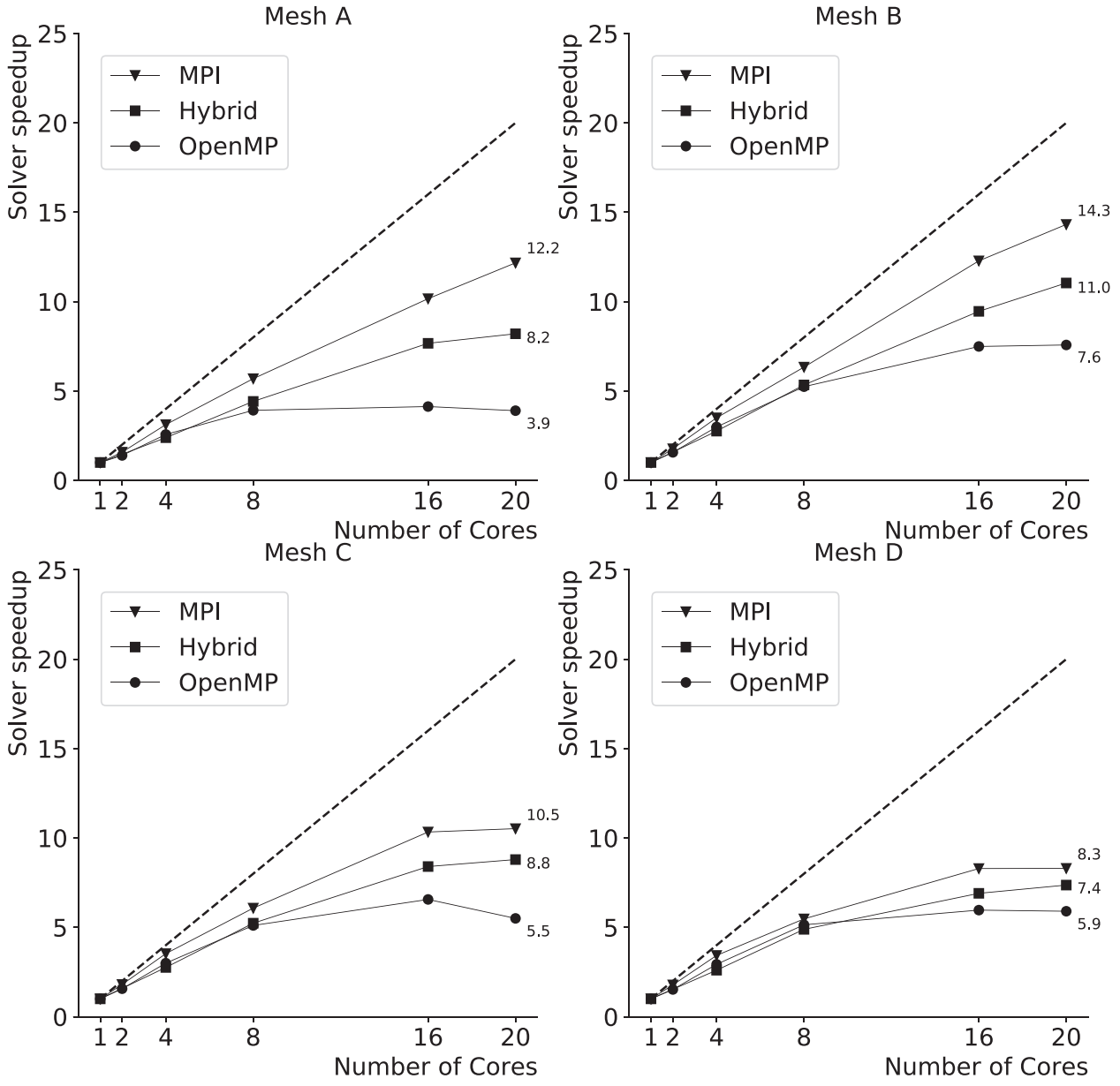

FIGURE 38 Cell loop speedup curves for one node, case 2

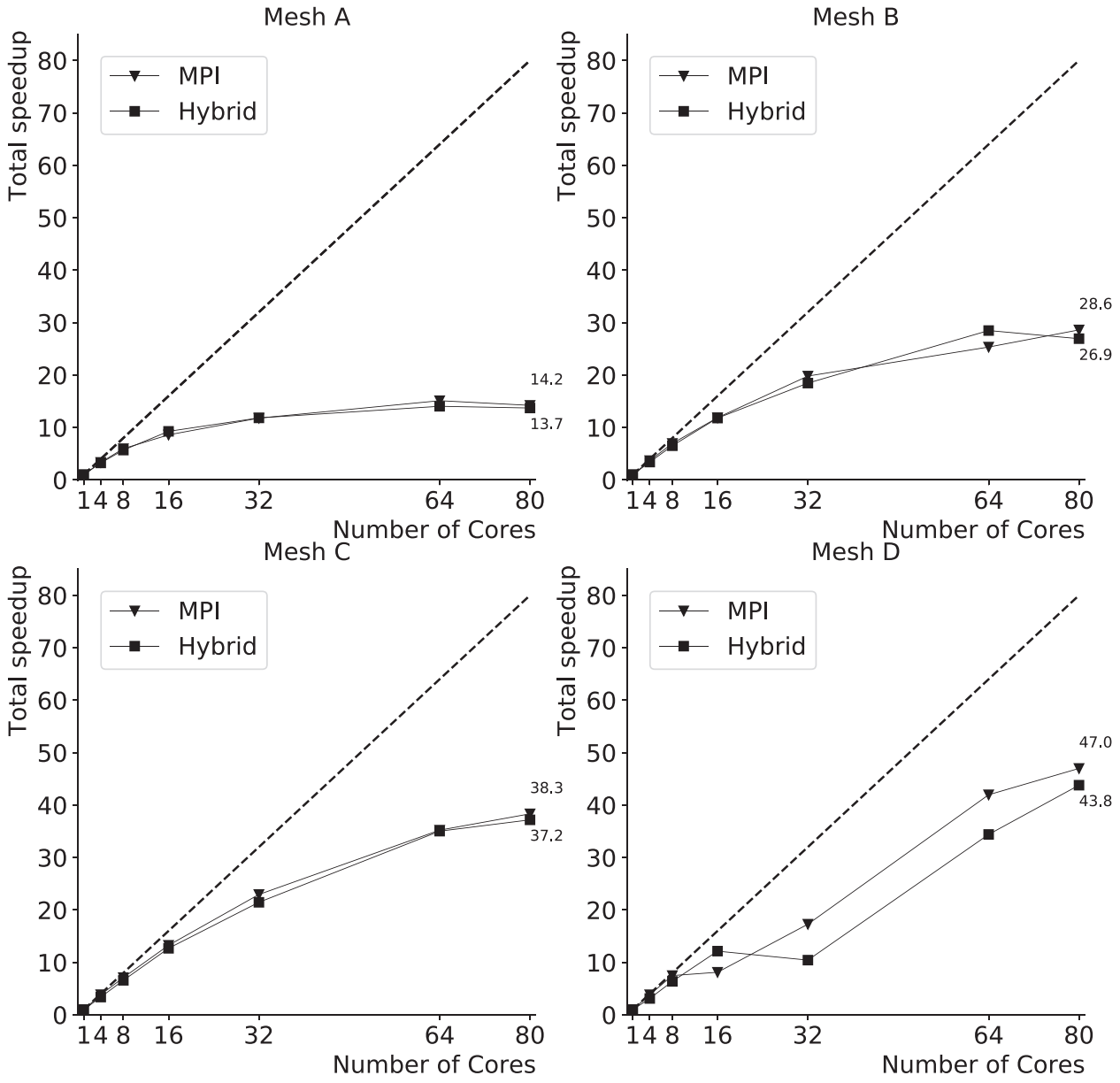

FIGURE 39 Total speedup curves: MPI × hybrid, case 2

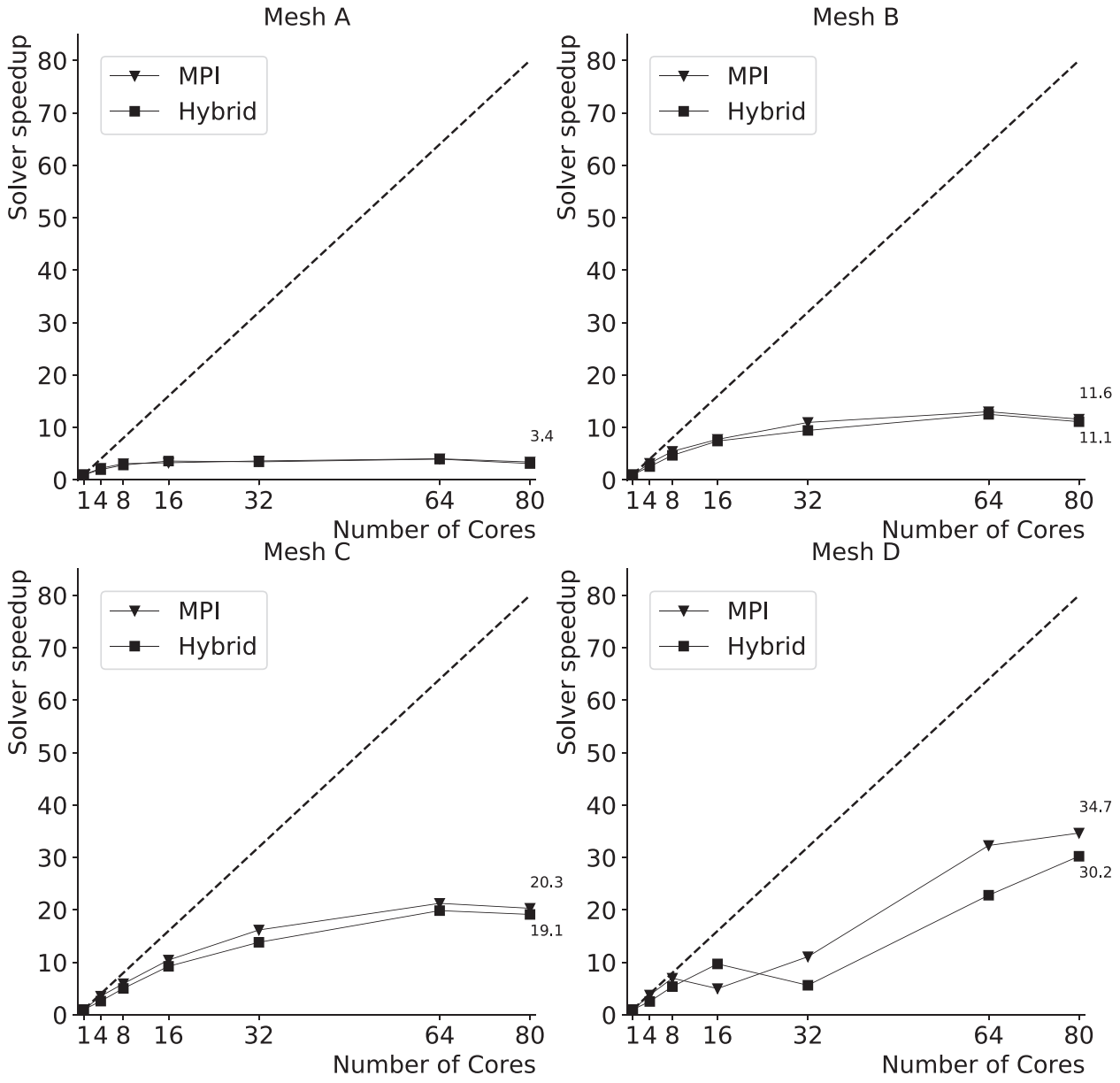

FIGURE 40 Solver speedup curves: MPI × hybrid, case 2

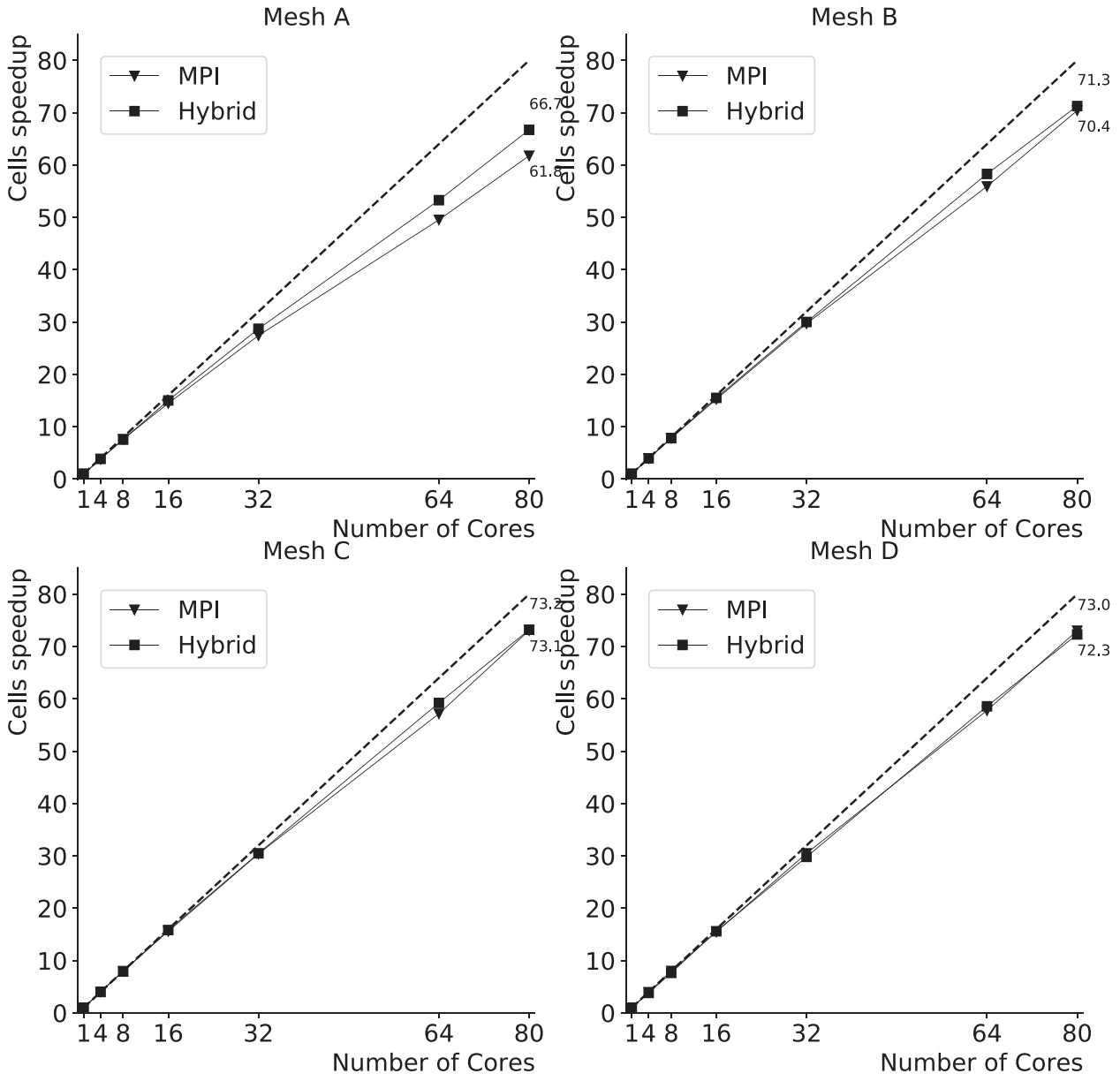

FIGURE 41 Cell loop speedup curves: MPI × hybrid, case 2

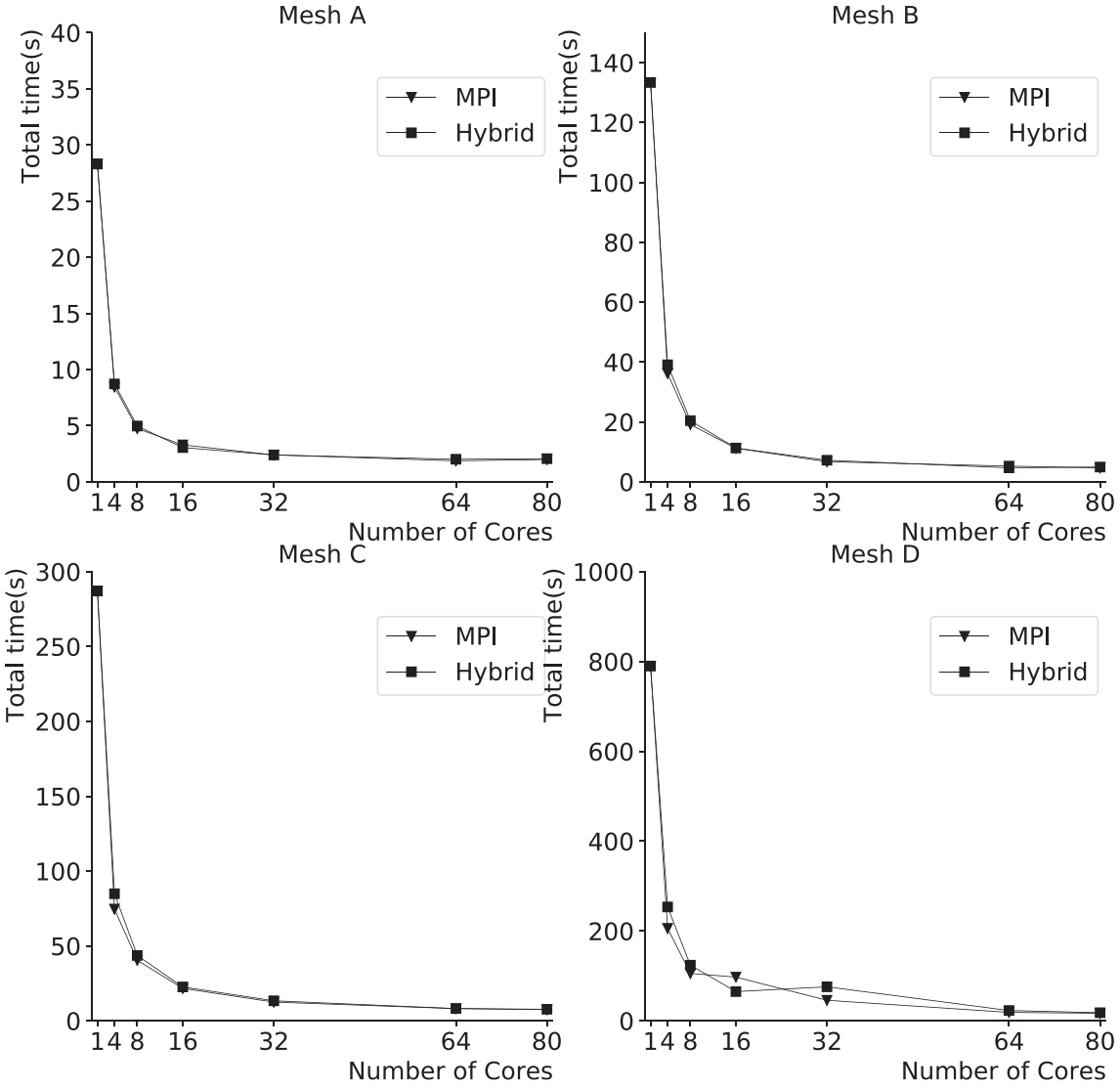

FIGURE 42 Total times for one time step: MPI × hybrid, case 2

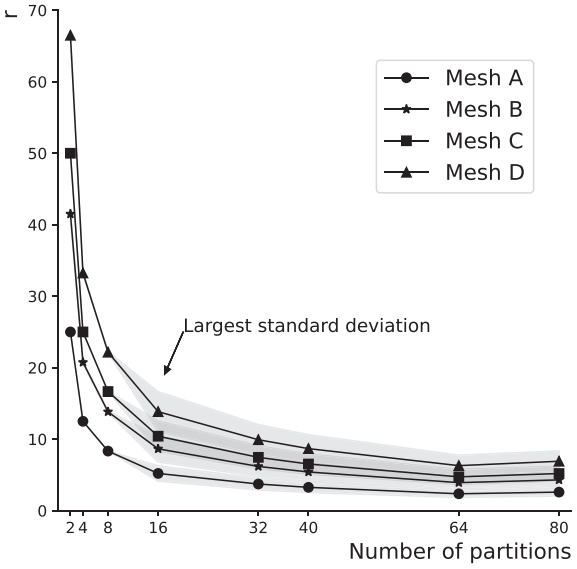

FIGURE 43 Ratio r = number of nonoverlapping/overlapping cells

TABLE 3 Average ratios r and standard deviations for meshes A, B, C, and D

	Avg	Std	Avg	Std	Avg	Std	Avg	Std
No. of partitions	**Mesh A**		**Mesh B**		**Mesh C**		**Mesh D**	
2	25.0	0.0	41.5	0.7	50.0	0.0	66.5	0.6
4	12.5	0.0	20.7	0.2	25.0	0.0	32.2	0.1
8	8.3	0.0	13.8	0.1	16.7	0.0	22.2	0.1
16	5.2	1.1	8.6	1.8	10.4	2.2	13.8	2.8
32	3.7	0.8	6.2	1.4	7.5	1.6	9.9	2.1
40	3.3	0.7	5.4	1.3	6.5	1.4	8.7	2.0
64	2.4	0.5	3.9	0.8	4.7	1.1	6.3	1.5
80	2.6	0.5	4.3	0.9	5.2	1.1	6.8	1.4

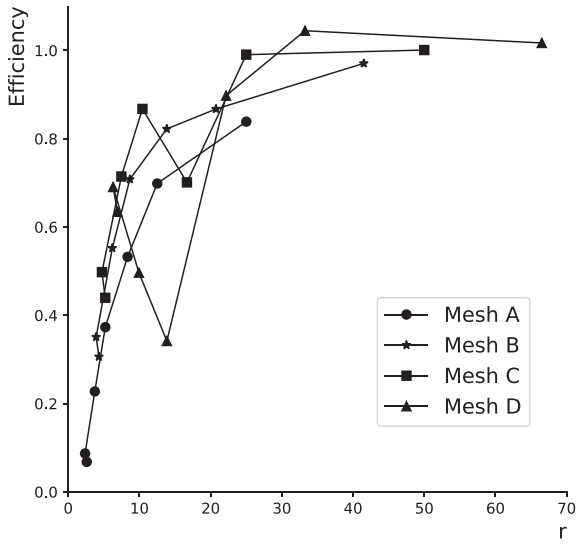

FIGURE 44 Efficiency for test case 1 as function of r, MPI

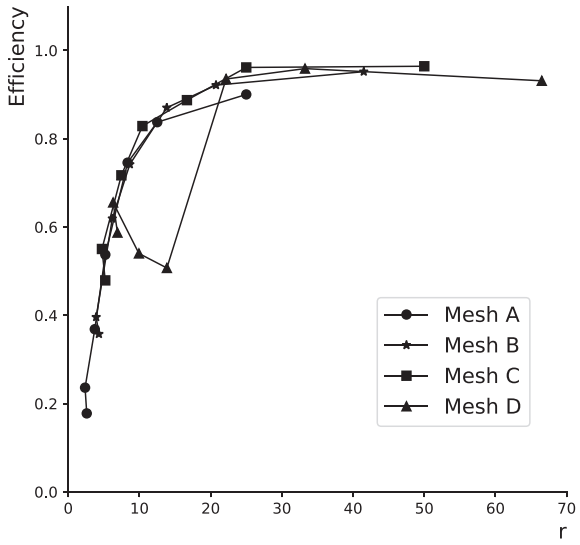

FIGURE 45 Efficiency for test case 2 as function of r, MPI

6 | CONCLUSIONS

We presented a parallel implementation of the finite volume method for incompressible and slightly compressible reactive flows in this work. Three approaches were discussed for distributed/shared memory architectures: pure MPI, pure OpenMP and a mix of these tools, resulting in a hybrid implementation. As it is well known, the bottleneck of finite volume, as well as finite element methods, is the solver phase, due to the intrinsic coupling characteristic present in the solution algorithms. Although the MPI library has been designed for distributed memory platforms, it is clear that it works very well on multicore shared memory machines, simulating a distributed memory environment. The main challenge in an MPI implementation is to minimize communication, whilst in an OpenMP code the problem is concerned with the occurrence of memory conflicts. For minimizing communication, we developed a highly efficient method, based on a communication mapping that uses point to point nonblocking functions. This mapping was designed for a subdomain-by-subdomain strategy, where the sequential code is equal to the parallel code, apart from communication calls and the algorithm that performs matvec operations. Using overlapping cells at the internal partition boundaries, the resulting parallel code has the same number of floating point operations as the sequential code. Also, a special matvec algorithm was presented for rectangular matrices, stemming from the overlapping partitioning method.

Our MPI implementation was compared to the OpenFOAM/MPI software, in test case 1. The results not only have a reasonable match in terms of solution response but also presented better results, in terms of execution time and speedups, in all situations. Furthermore, our model presented a better scalability.

Concerning the parallel model to be used, that is, the choice between pure MPI, pure OpenMP and hybrid, the first conclusion that can be inferred from the results is that the MPI model is superior to the OpenMP implementation. The hybrid implementation might present slightly better results in the cell loop, in a few cases, but the pure MPI is always superior in an overall sense. It is important to note that a low band inter-connect network was used (Ethernet, 2×10 Gb/s). However, the use of faster communications protocols would obviously improve speedups when communication between nodes is necessary.

Finally, we believe that this article can be an important reference for researchers working in the field of high performance computational fluid dynamics.

ACKNOWLEDGEMENT

The authors are indebted to the Brazilian scientific agencies CNPq and CAPES, for the financial support.

ORCID

Luiz Carlos Wrobel https://orcid.org/0000-0001-6702-0178

REFERENCES

1. Versteeg H, Malalasekera W. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. 2nd ed. Pearson Education Limited; 2007.
2. Ferziger JH, Perić M. *Computational Methods for Fluid Dynamics*. Springer; 2002. doi:10.1007/978-3-642-56026-2
3. Jasak H. *Error Analysis and Estimation for the Finite Volume Method with Applications to Flows*. PhD thesis. Imperial College London, 1996.
4. Darwish M, Moukalled F. TVD schemes for unstructured grids. *Int J Heat Mass Transf*. 2003;46(4):599-611. doi:10.1016/S0017-9310(02)00330-7
5. Kong J, Xin P, Shen CJ, Song ZY, Li L. A high-resolution method for the depth-integrated solute transport equation based on an unstructured mesh. *Environ Model Softw*. 2013;40:109-127. doi:10.1016/j.envsoft.2012.08.009
6. Delis AI, Nikolos IK, Kazolea M. Performance and comparison of cell-centered and node-centered unstructured finite volume discretizations for shallow water free surface flows. *Arch Comput Methods Eng*. 2011;18:57-118. doi:10.1007/s11831-011-9057-6
7. Tsoutsanis P, Titarev V, Drikakis D. WENO schemes on arbitrary mixed-element unstructured meshes in three space dimensions. *J Comput Phys*. 2011;230(4):1585-1601. doi:10.1016/j.jcp.2010.11.023
8. Tsoutsanis P, Dumbser M. Arbitrary high order central non-oscillatory schemes on mixed-element unstructured meshes. *Comput Fluids*. 2021;225:104961. doi:10.1016/j.compfluid.2021.104961
9. Oggian T, Drikakis D, Youngs DL, Williams RJR. A hybrid compressible–Incompressible computational fluid dynamics method for Richtmyer–Meshkov mixing. *J Fluids Eng*. 2014;136(9):091210. doi:10.1115/1.4027484
10. Diskin B. Accuracy of gradient reconstrucion on grids with high aspect ratio. NIA report no. 2008-12; 2008.
11. Mavriplis D. Revisiting the least-squares procedure for gradient reconstruction on unstructured meshes. Proceedings of the 16th AIAA Computational Fluid Dynamics Conference; 2003
12. Syrakos A, Varchanis S, Dimakopoulos Y, Goulas A, Tsamopoulos J. A critical analysis of some popular methods for the discretisation of the gradient operator in finite volume methods. *Phys Fluids*. 2017;29(12):127103. doi:10.1063/1.4997682

13. Patankar S, Spalding D. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *Int J Heat Mass Transf*. 1972;15(10):1787-1806. doi:10.1016/0017-9310(72)90054-3
14. Patankar S. *Numerical Heat Transfer and Fluid Flow*. CRC Press; 1980.
15. Doormaal JPV, Raithby GD. Enhancements of the simple method for predicting incompressible fluid flows. *Numer Heat Transf*. 1984;7(2):147-163. doi:10.1080/01495728408961817
16. Moukalled F, Darwish M, Mangani L. *The Finite Volume Method in Computational Fluid Dynamics. An Advanced Introduction with OpenFOAM® and Matlab*. Springer International Publishing; 2015.
17. Pope SB. *Turbulent Flows*. Cambridge University Press; 2000.
18. Magnussen B. On the structure of turbulence and a generalized eddy dissipation concept for chemical reaction in turbulent flow. Proceedings of the 19th Aerospace Sciences Meeting; 1981
19. Magnussen B. Modeling of "NOx" and soot formation by the eddy dissipation concept. Proceedings of the International Flame Research Foundation, First Topic Oriented Technical Meeting; 1989.
20. Gran IR, Magnussen BF. A numerical study of a bluff-body stabilized diffusion flame. Part 2. Influence of combustion modeling and finite-rate chemistry. *Combust Sci Technol*. 1996;119(1-6):191-217. doi:10.1080/00102209608951999
21. Magnussen BF. The Eddy dissipation concept a bridge between science and technology. Proceedings of the ECCOMAS Thematic Conference on Computational Combustion; 2005.
22. Panjwani B. Subgrid combustion modeling for large eddy simulation (LES) of turbulent combustion using eddy dissipation concept. Proceedings of the 5th European Conference on Computational Fluid Dynamics, ECCOMAS CFD; 2010.
23. Lewandowski MT, Ertesvåg IS. Analysis of the eddy dissipation concept formulation for MILD combustion modelling. *Fuel*. 2018;224:687-700. doi:10.1016/j.fuel.2018.03.110
24. Ertesvåg IS. Analysis of some recently proposed modifications to the eddy dissipation concept (EDC). *Combust Sci Technol*. 2020;192(6):1108-1136. doi:10.1080/00102202.2019.1611565
25. Williams FA. *Combustion Theory*. 2nd ed. Westview Press; 1985.
26. Poling BE, Prausnitz JM, Connell JPO. *The Properties of Gases and Liquids*. 5th ed. McGraw-Hill Education; 2001.
27. Poinsot T, Veynante D. *Theoretical and Numerical Combustion*. 2nd ed. R.T. Edwards; 2005.
28. Afzal A, Ansari Z, Faizabadi AR, Ramis MK. Parallelization strategies for computational fluid dynamics software: state of the art review. *Arch Comput Methods Eng*. 2017;24(2):337-363. doi:10.1007/s11831-016-9165-4
29. de Andrade HCC. *Modelagem De Escoamentos Reativos Levemente Compressíveis Com Um Modelo Computacional Paralelo De Volumes Finitos*. PhD thesis. Universidade Federal do Rio de Janeiro; 2020.
30. Ribeiro FLB, Ferreira IA. Parallel implementation of the finite element method using compressed data structures. *Comput Mech*. 2007;41(1):31-48. doi:10.1007/s00466-007-0166-x
31. Ainsworth GO, Ribeiro FLB, Magluta C. A parallel subdomain by subdomain implementation of the implicitly restarted Arnoldi/Lanczos method. *Comput Mech*. 2011;48(563):1-15. doi:10.1007/s00466-011-0607-4
32. Silva ABC, Laszczyk J, Wrobel LC, Ribeiro FL, Nowak AJ. A thermoregulation model for hypothermic treatment of neonates. *Med Eng Phys*. 2016;38(9):988-998. doi:10.1016/j.medengphy.2016.06.018
33. Andrade HCC, Silva ABCG, Ribeiro FLB, Maghous S. A parallel poromechanics fem model. Conference Proceedings 38th Iberian Latin American Congress on Computational Methods in Engineering; 2017.
34. Rita M, Fairbairn E, Ribeiro F, Andrade H, Barbosa H. Optimization of mass concrete construction using a twofold parallel genetic algorithm. *Appl Sci*. 2018;8(3). doi:10.3390/app8030399
35. Silva ABC, Wrobel LC, Ribeiro FL. A thermoregulation model for whole body cooling hypothermia. *J Thermal Biol*. 2018;78:122-130. doi:10.1016/j.jtherbio.2018.08.019
36. Andrade HCC, Silva ABCG, Ribeiro FLB, Wrobel LC. A parallel implementation of a thermoregulation model using the finite element method. Proceedings of the XL Ibero-Latin-American Congress on Computational Methods in Engineering; 2019.
37. Ribeiro FLB, Coutinho ALGA. Comparison between element, edge and compressed storage schemes for iterative solutions in finite element analyses. *Int J Numer Methods Eng*. 2005;63(4):569-588. doi:10.1002/nme.1290
38. Batista VHF, Ainsworth GO Jr, Ribeiro FLB. Parallel structurally-symmetric sparse matrix-vector products on multi-core processors. In: Topping BHV, Ivanyl P, eds. *Proceedings of the Third International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Paper no. 22*. Stirlingshire, UK: Civil-Comp Press; 2013.
39. van der Vorst HA. *Iterative Krylov Methods for Large Linear Systems*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press; 2003.
40. Saad Y. *Iterative Methods for Sparse Linear Systems*. 2nd ed. Society for Industrial and Applied Mathematics; 2003.
41. McGrattan K, Hostikka S, McDermott R, Floyd J, Weinschenk C, Overholt K. Fire Dynamics Simulator: Technical Reference Guide: Volume 1: Mathematical Model. . NIST Special Publication; 2017.
42. Nicoud F, Ducros F. Subgrid-scale stress modelling based on the square of the velocity gradient tensor. *Flow Turbul Combust*. 1999;62(3):183-200. doi:10.1023/A:1009995426001
43. McBride BJ, Zehe MJ, Gordons S. NASA Glenn coefficients for calculating thermodynamic properties of individual species. NASA/TP-2002-211556; 2002.
44. NIST. *Chemistry WebBook*. National Institute for Standards and Technology (NIST), US Department of Commerce. Gaithersburg, MD, USA; 2019.
45. Lameter. NUMA (Non-Uniform Memory Access): an overview; 2013.

46. Chandra R, Menon R, Dagum L, Kohr D, Maydan D, McDonald J. *Parallel Programming in OpenMP*. Morgan Kaufmann; 2000.
47. Chapman B, Jost G, van der Pas R. *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press; 2007.
48. Pas RVD, Stotzer E, Terboven C. *Using OpenMP - The Next Step: Affinity, Accelerators, Tasking, and SIMD*. The MIT Press; 2017.
49. Karypis G, Kumar V. Multilevelk-way partitioning scheme for irregular graphs. *J Parallel Distrib Comput*. 1998;48(1):96-129. doi:10.1006/jpdc.1997.1404
50. METIS - serial graph partitioning and fill-reducing matrix ordering. Accessed January 10, 2021. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

How to cite this article: deAndrade HCC, Ribeiro FLB, Carlos Wrobel L. A parallel finite volume method for incompressible and slightly compressible reactive flows. *Int J Numer Methods Eng*. 2022;1-47. doi: 10.1002/nme.6990