Optimizing Load Balancing of Reacting Flow Solvers in OpenFOAM for High Performance Computing

Thorsten Zirwes*, Feichi Zhang, Peter Habisreuther, Jordan A. Denev, Henning Bockhorn, Dimosthenis Trimis

**Karlsruhe Institute of Technology, Steinbuch Centre for Computing, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, +49 721 608-29278, thorsten.zirwes@kit.edu*

In solvers for reacting flows like `reactingFoam`, most of the simulation time is spent on computing chemical reaction rates if detailed reaction mechanisms are used. In parallel simulations, the time required for the chemistry computations can vary drastically between processes. In this work, an optimization method for High Performance Computing (HPC) of reacting flow solvers is presented. The method consists of a dynamic load balancing approach, which takes advantage of the chemistry computations not being dependent on neighboring cell values or other spatial relations. The approach is based on forming groups of processes that share their chemistry workload and can be used together with adaptive mesh refining and additional load balancing techniques. It requires little changes to the existing code but helps to reduce the total simulation time significantly as well as increases the utilization of hardware resources on HPC clusters. It is shown that for the validation case, the total simulation time is reduced by 30%. The optimized solver along with a self-implemented, highly-efficent approach for detailed calculation of chemical reaction and molecular transport is applied to simulate a turbulent jet flame. For this flame, the computational grid consists of 150 million cells and the simulation has been run on Germany's fastest supercomputer Hazel-Hen at HLRS Stuttgart on 28,800 CPU cores. It is shown that the total simulation time for this case is reduced by 10 %, which corresponds to over one million saved CPU hours.

1. Introduction

Accurate simulations of flames are computationally expensive due to the large number of chemical reactions and species relevant during the combustion. For example, the oxidation of methane in air can be described by 325 reactions and 53 chemical species [1]. Although simplified models are available, a detailed insight into the realistic flame dynamics and pollutant emission can only be achieved by using

these detailed reaction mechanisms. At present, highly resolved simulations of 3D turbulent flames without turbulence or chemistry models employing complex reaction mechanisms are possible on today's fastest supercomputers [2,3]. It is therefore mandatory that the simulation software is suitable for high performance computing (HPC) achieving the best possible performance. If detailed reaction mechanisms are utilized in the simulation, the most computing time is spent on calculating the chemical reaction rates [3]. In previous works, serial approaches for speeding up the chemistry computations have been discussed [4]. In this work, the parallel performance of OpenFOAM's `reactingFoam` class of solvers is optimized, which often suffers from load balancing issues when detailed reaction mechanisms are used.

The reason for this imbalance is that OpenFOAM, like many other CFD tools, computes the chemical reaction rates with an operator splitting approach. This means that instead of computing the reaction rates $\dot{\omega}_k$ directly from the Arrhenius laws of the chemical reactions, they are averaged over the CFD time step:

$$\bar{\dot{\omega}}_k = M_k \frac{\partial C_k}{\partial t} \approx M_k \frac{C_k^{n+1} - C_k^n}{\Delta t_{CFD}} \tag{1}$$

$$C_k^{n+1} = \frac{1}{M_k} \int_{\Delta t_{CFD}} \dot{\omega}_k \, \mathrm{d}t \tag{2}$$

Here, C_K^n is the current concentration of the kth species, M_k its molar mass, $\bar{\dot{\omega}}_k$ the averaged reaction rate, t the time and Δt_{CFD} the CFD time step. During this integration over the CFD time step, each cell is considered as a closed batch reactor. This allows setting the CFD time step according to the Courant number without taking the smallest time scale of the chemical reactions into account, which is typically many orders of magnitude below the CFD time step. Eq. (2) represents a system of highly non-linear, stiff and coupled ordinary differential equations (ODE) because $\dot{\omega}_k$ in general depends on all species concentrations and temperature. The system of ODEs has to be solved in each cell in order to get an approximation of the concentration C_k^{n+1} at the next CFD time step. The chemical sub-time steps are adaptively chosen during the integration. OpenFOAM offers different integrators for this. The number of time steps or iterations, however, depends on the numerical stiffness for the current conditions in each cell. Therefore, some cells require more time than others. Only the process with the highest workload determines the overall simulation time. In the case of flames, the chemical reactions take place only in a thin layer within the flame front. When the mesh is decomposed to the different processes, some processes might have more cells in the reacting zone than others. Since all processes have to synchronize after computing the reaction rates, they have to wait for the slowest process or the process, which has the most cells within the flame's reaction zone, respectively.

There are some factors that influence how much the processes are imbalanced:

- The ratio of cells with small chemical time scales (i.e. cells within the flame's reaction zones) to cells without chemical reaction: for example if the flame constitutes only a small part of the computational domain while the largest part is an inert flow.
- The size of the reaction mechanism: The more complex the system of chemical reactions is, the more expensive are the iterations of the ODE integrator leading to more imbalance.
- The stiffness of the reaction mechanism: The smaller the chemical time scales become and the larger the gap between the smallest and largest chemical time scales among the reactions within one cell is, the more work has to be performed by the integrator.
- The chemistry ODE solver tolerances: OpenFOAM allows specifying absolute and relative tolerances for the ODE integrator. The smaller the tolerances become, the more accurate is the result but the more work has to be done by the integrator for cells within the reaction zone.
- Adaptive mesh refinement: If adaptive mesh refining is used to improve the resolution of the reaction zone, processes that require the most time for the ODE integration are assigned even more cells in the reaction zone.
- Time steps: the higher the CFD time steps are, the more chemical sub-time steps have to be performed by the ODE integrator.
- Using a threshold temperature: In OpenFOAM, the computation of chemical reaction rates can be disabled if the temperature is below a threshold value (`Treact`). This usually does not lead to performance improvements in parallel simulations. Some processes might not performing any chemistry computations at all, but they have to wait until the slowest processes are finished, leading to even greater load imbalances.

If the setup for the reacting flow tends toward a steady-state and the locations with small chemical time scales or fast reactions, where the ODE integrator has to perform the most iterations, are known beforehand, the mesh can be decomposed manually to assign less cells to processes in these regions. This approach however has several downsides: The user has to perform the mesh decomposition manually, requiring additional work. The assignment of different numbers of cells to each process might result in overall better performance. However, as shown in the next section, this introduces new load balancing problems for the solution of the transport equations. Lastly, in many cases, the locations with small chemical time scales are not known beforehand or the setup is transient, e.g. a turbulent flame that propagates transiently within the domain.

In this work, a load balancing approach is presented, which dynamically rebalances only the computation of chemical reaction rates. This approach is computationally cheap, easy to implement, can be combined with adaptive mesh refining and leads to better HPC hardware utilization as well as shorter simulation times.

2. Performance Profiling

In order to quantify the parallel load imbalances in OpenFOAM's `reactingFoam` class of solvers, a simple 2D flame setup is used to perform performance profiling with different tools. The setup consists of a methane-air flame using the GRI3.0 reaction mechanism with 325 reactions and 53 chemical species [1] (see Fig. 1). The mesh consists of 500,000 cells and the simulation is performed with second order time discretization and cubic interpolation for spatial discretizations. The OpenFOAM version used in the tests is OpenFOAM v1612+; however, the discussed results apply to all modern versions of OpenFOAM. The solver is a modified version of `reactingFoam` [4]. The simulation is performed on 6 nodes (120 CPU cores) of the "JURECA" cluster at the Jülich Supercomputing Centre [5]. The performance profiling has been done with the tracing tool Extrae [6]. Figure 2 shows the profiling results visualized in Paraver [7].

Each horizontal line represents one of the 120 processes. Depicted is one full time step from the simulation with time passing from left to right. The black color denotes useful work done by the solver. The blue areas show the time spend in MPI communication and waiting, leading to unused computing power. In this case, about 60 % of the computing time is spent on computing chemical reaction rates, denoted by the red brace on the bottom left. During this time, there are only a few processes, which need the whole time for the chemistry computations (red box). The rest of the processes spends almost 30 % of the time waiting, not performing any work. This is due to the thin flame front (region with small time scales) in Fig. 1 being distributed only to a subset of the 120 processes.

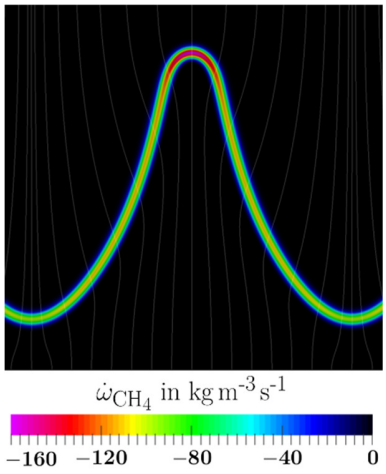

Fig. 1: Reaction zone of a 2D flame depicted by the methane reaction rate in the computational domain. The domain is decomposed into 120 parts.

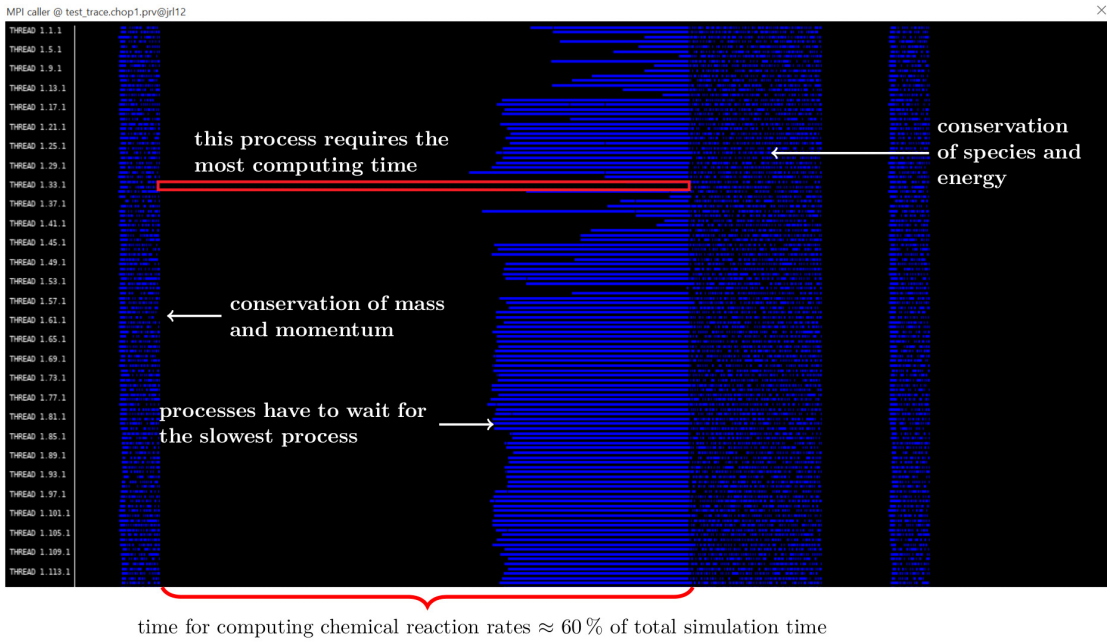

Fig. 2: Profiling results from Extrae, showing the time each process has to wait (blue) during one time step.

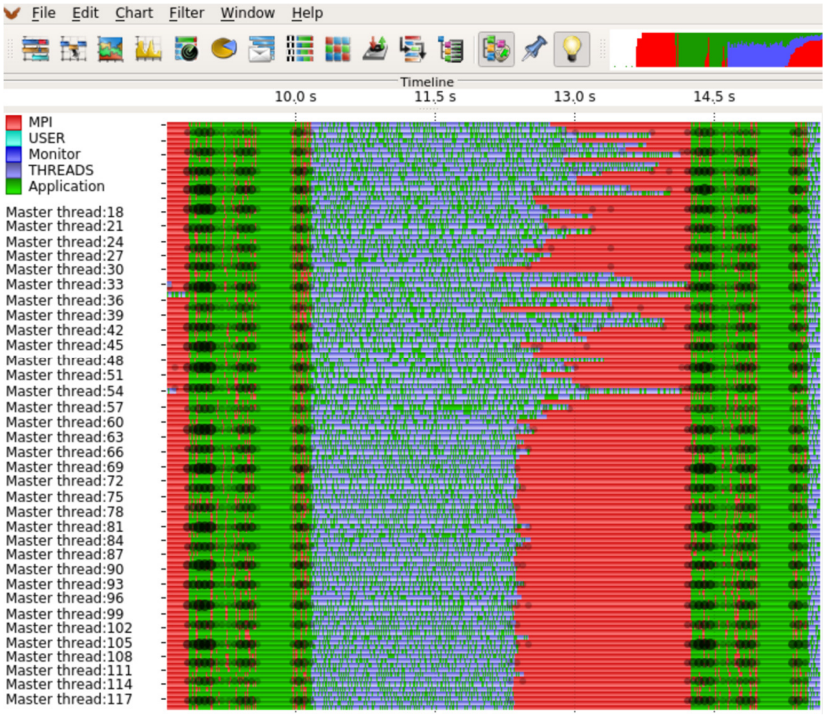

Fig. 3: Profiling results for the same case with Score-P. Green denotes useful work done by the solver, grey shows the chemistry computations and red is time spent waiting.

Figure 3 shows profiling results for the same case, recorded with the performance tool Score-P [8] and visualized with Vampir [9]. Again, one time step is depicted with each horizontal line representing one process. Green shows useful work done by the solver, grey is the time spent in chemistry computations and red shows the waiting time. Since the chemistry computations require most of the total computing time, there are a lot of resources being wasted by the waiting processes. It can also be seen from Fig. 3 that the solution of the transport equations (green areas) is very well balanced because it scales with the number of cells. Dividing the mesh into parts with equal number of cells is therefore optimal for the solution of the transport equations. It is however apparent, that the computation of chemical reaction rates lead to large imbalances among the processes.

3. Dynamic Load Balancing Implementation

In order to improve the load balancing, an approach is presented which dynamically rebalances the workload of chemistry computations during the simulation between groups of processes. Since this load balancing only affects the chemistry part of the solver, the mesh can still be decomposed into an equal number of cells, thereby achieving the best load balance for the chemistry while keeping the best load balance for the transport equations.

The dynamic load balancing method is based on the fact, that the chemical reaction rates can be computed from the local cell values, without considering neighbouring cells. This means, that the workload can be freely shared between processes without considering spatial relations, connectivities or processor boundary patches. It can therefore be used together with adaptive mesh refining and other load balancing techniques.

The load balancing method consists first of all processes exchanging the time needed for computing the chemical reaction rates in their part of the decomposed mesh with all other processes, shown below in pseudo code:

```
Every N time steps:

   1) Create a list containing the process ID and time for the
      chemistry
   2) gather/scatter the list to all processes
   3) Sort the list by chemistry time
   4) Assign pairs of processes that will share their workload
```

This is illustrated in Fig. 4: First, a list containing the time required for the chemistry computations of each process and the process ID is shared between all processes. Because step 2) involves a potentially expensive all-to-all communication, it is only performed every N CFD time steps, where N is chosen by the user. If the location of the reaction zone changes fast, then N should be a low value (e.g. 1000) to ensure a good load balancing. After step 2), each process has the same copy of the list of process IDs and chemistry times, as shown in Fig. 4a. In step 3), each process sorts its copy of the list by the chemistry time (Fig. 4b). In step 4), each process looks up its position p in the list and is assigned the process at position $(N_p - p - 1)$ as its partner, where N_p is he number of processes. For example, this means that the process with the process ID 0 from Fig. 4c knows, that it needs the most time for the chemistry, and therefore forms a pair with process 2, which requires the least time. Likewise, process 1 is the process with the second highest time and therefore forms a pair with process 4, having the second lowest time.

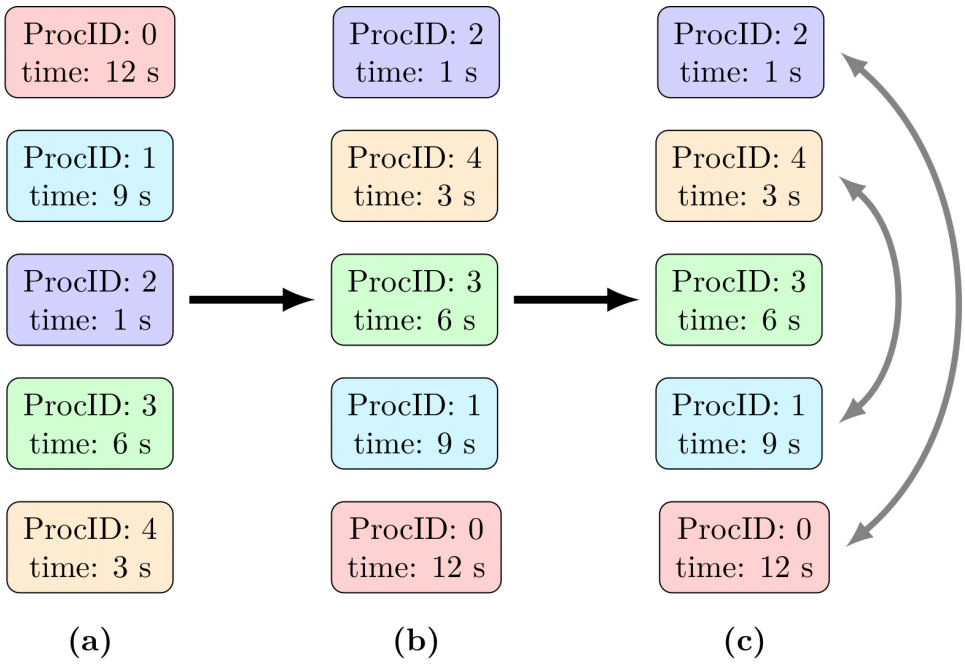

Fig. 4: The method first consists of: (a) sharing a list containing all process IDs and time spent computing the chemical reaction rates. (b) Sorting the list by the chemistry time. (c) Assigning pairs of processes depending on the chemistry time.

After step 4), each process knows the process ID of its partner and it also knows if it is the "sender" (requiring more time for the chemistry) or the "receiver" (requiring less time). This information is then used in the dynamic load balancing step:

```
Each time step:

If (sender)

   1) Create buffer containing a list of species mass fractions,
      temperatures and pressures of the last N_shared cells
   2) Send the buffer to the partner process
   3) Compute the reaction rates for cells 0 to (N_cells-N_shared)
   4) Receive the reaction rates for cells with index N_shared to
      N_cells
   5) Send the value of N_shared for the next time step to receiver

If (receiver)

   1) Compute the reaction rates for all cells on own mesh
   2) Receive the buffer from sender process
   3) Compute the reaction rates for all cells in the buffer
   4) Send the additional reaction rates back to the sender process
   5) Receive the value of N_shared for the next time step
```

At each CFD time step, the sender process, i.e. the process that needs more time for the chemistry computations than its partner process, computes the reaction rates not for all its cells, but only for the first `(N_cells-N_shared)` cells. `N_cells` is the total number of cells of the sender process and `N_shared` the number of cells communicated to the receiver process. However, before that, it creates a buffer containing a list of the mass fractions, temperatures and pressures from the rest of its cells, and sends the buffer to its partner process.

The partner process ("receiver") computes the reaction rates for all of its own cells. It then receives the buffer from its partner process and uses the information within the buffer to compute the additional reaction rates from the sender process. They are then sent back to the sender process, along with the time it took to compute all of its own reaction rates and the time to compute the additional reaction rates.

The number of exchanged cells `N_shared` is then dynamically recomputed for the next time step, so that the amount of work in the sender and receiver process is equal based on the current time measurements. The sender process keeps track of how much time is needed for each cell of its own domain, to decide the correct number of cells required to achieve an equal distribution of computing time.

In summary: Every N CFD time steps it is decided, which two processes form a pair and which one of the two is the sender and the receiver. These pairs stay connected for the next N time steps. Then, new pairs are formed based on the current load imbalances. Every CFD time step, the process designated as sender sends some of its cells, meaning a list of mass fractions, temperatures and pressures, to the receiver process it currently forms a pair with. The receiver computes the reaction rates for the additional cells and sends them to the sender, along with the time it took to compute the reaction rates. Based on these timings, the sender tells the receiver, how many cells will be transferred in the next time step.

With this method, only one send/receive pair for the buffer and one send or receive of the `N_shared` value per process are required for the load balancing and most of the communication overhead can be hidden by non-blocking communication. Since this method is only affecting the chemistry computations, it can easily be implemented into the `chemistryModel` without affecting other parts of the code or other load balancing methods. Because the workload is rebalanced every time step, the dynamic load balancing is efficient even for cases where the reaction zone moves rapidly during the simulation and only a few of the processes have a lot more work to do than the others.

4. Evaluation of Performance Gain

The method described in the last section is implemented and applied to the case from section 2, using 100 CPU cores from the ForHLR II cluster at KIT [10] and OpenFOAM version 1712+. Figure 5 on the left shows the time for computing the chemical reaction rates on each process during one time step. The blue bars show the time for chemistry computations without load balancing and the orange bars with the load balancing approach from section 3. The dashed lines show how much time the chemistry computations take during the simulation, which is solely determined by the slowest process. By using the load balancing approach, the time reduces from about 12 seconds to 8 seconds, therefore saving more than 30% of the time. The computational resources wasted by waiting processes also has drastically reduced. The ratio of time of the slowest process to the fastest process is reduced from a ratio of 2 to a ratio of 1.3.

By increasing the CFD time step from 0.1 µs to 1 µs (Fig. 5 middle), the computation time for the chemistry increases (see section 1). Again, the time is reduced by 35% of the original time (from 20 s to 13 s) when the dynamic load balancing is used, and the ratio of chemistry times from the slowest to the fastest process reduces from 3.3 to 1.3. Similar results are obtained for the same case, when the ODE integrator tolerances are lowered by an order of magnitude (Fig. 5 on the right).

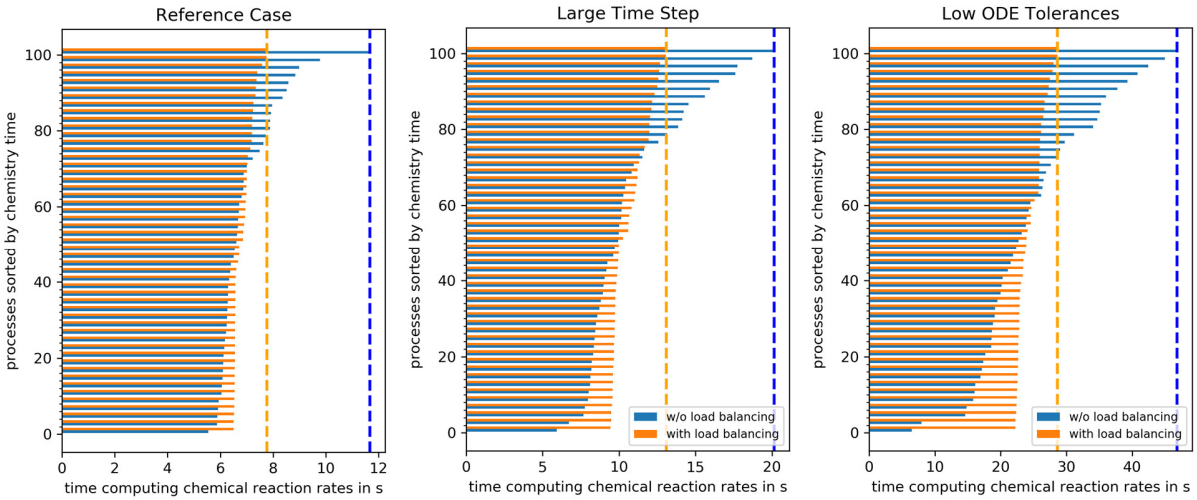

Fig. 5: Time for the chemistry for all processes for the reference case from section 2 (left), the same case with 10 times larger CFD time step (middle), and the same case with ten times lower absolute and relative solver tolerances (right).

5. Application to a Turbulent Flame on a HPC Cluster

As explained in the introduction, performing highly resolved simulations of turbulent flames with detailed chemistry can only be run on modern supercomputers. As an example of a case benefitting from the presented load balancing approach, a simulation of a turbulent flame [11] is briefly presented in this section.

Due to the complex interplay of the turbulent flow and the combustion chemistry, a modified version of `reactingFoam` is used which employs detailed molecular transport for each chemical species [4]. This is achieved by coupling OpenFOAM with the open-source library Cantera [12]. An optimized method for computing the reaction rates is also used, which is able to use a reaction mechanism including quasi-steady state species [13]. The case consists of 150,000,000 cells and considers 19 different chemical species. The simulation is performed on Germany's fastest supercomputer "Hazel Hen" at the High Performance Computing Center Stuttgart [14] with 28,800 CPU cores. Figure 6 shows a 2D slice through the temperature field of the flame. On the left, unburnt fuel and oxidizer enter the domain.

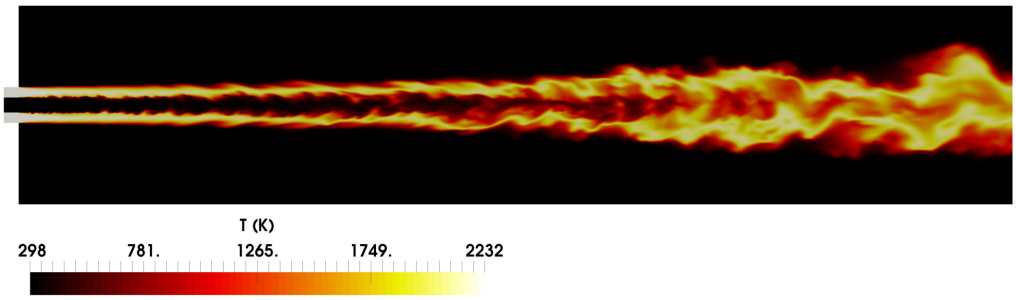

Fig. 6: 2D cutting plane of the temperature field.

In this simulation, there is also an imbalance of the chemistry time between the processes because a complex reaction mechanism is used. Compared to the validation case in section 4, the imbalances are smaller (ratio of chemistry time from the slowest to the fastest process is 1.6) due the employed reaction mechanism [13] being smaller than GRI 3.0 used in section 4. Applying the presented load balancing approach to this case leads to a reduction of total computing time of 10 %. Because of the large scale of this simulation, this reduction in simulation time corresponds to over 1 million core hours that can be saved.

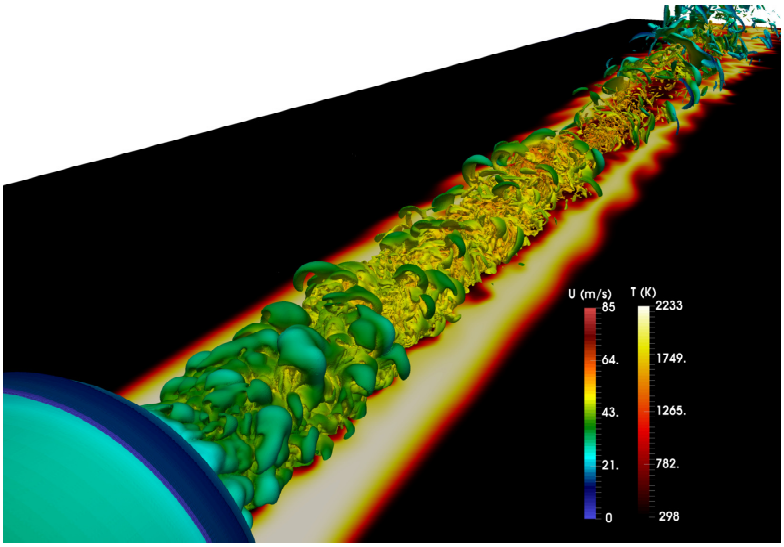

Fig. 7: 2D temperature cutting plane and iso-surface of vorticity colored by the gas velocity.

Due to the highly resolved nature of the simulation in space and time together with the complex reaction mechanism, the results will help to gain deeper insights into the effect of turbulent flow structures on the flame properties. In Fig. 7 on the bottom left, the burner nozzle can be seen. An iso-surface of vorticity colored by the gas velocity shows the turbulent flow structures of the cold, unburnt fuel-oxidizer mixture

coming from the nozzle. As the flow heats up (shown by the 2D temperature cutting plane in the background), the turbulent flow structures are dampened by the increased viscosity.

6. Summary

In this work, a HPC solver optimization for parallel load balance is presented. The current load balancing issues of the `reactingFoam` class of solvers are discussed in cases where highly accurate results with detailed chemistry reaction mechanisms are desired. Two profiling tools are used for the analysis.

The load balancing approach is computationally cheap and easy to implement. The method is based on the fact that chemical reaction rates can be computed without any spatial information or dependence because they only depend on the local cell values. Therefore, a list of values of mass fractions, temperatures and pressures from the cells can freely be distributed among processes without preconditions, side effects or mesh redistribution. The implemented load balancing method forms pairs of processes that share their workload. The rebalancing between the pairs is done dynamically at each CFD time step, thus adapting even to highly transient cases. Only one send/receive pair of a buffer and an additional send/receive per process are required for this so that the overhead is low. The method can also be extended to groups of more than two processes sharing their workload at the expense of higher communication overhead. The benefits of this method are not only the reduction in total computing time but also that it can be combined with adaptive mesh refinement and different load balancing techniques targeting the mesh, thereby ensuring optimal load balancing not only for the chemistry but also for the solution of the transport equations. The method is validated for a case run with 100 processes with OpenFOAM 1712+. The simulation times are reduced by about 30 %, saving many core hours, and the HPC hardware is more efficiently used due to less waiting times among the processes.

As an example of a real application benefitting from this optimization, the simulation of a turbulent flame is presented using complex reaction chemistry and detailed molecular transport. The simulation is performed on up to 28,800 CPU cores on Germany's largest supercomputer "Hazel Hen". The detailed results will help to improve industrially relevant combustion models.

Acknowledgments

This work was performed on the national supercomputer Cray XC40 Hazel Hen at the High Performance Computing Center Stuttgart (HLRS) and on the computational resource ForHLR II funded by the Ministry of Science, Research and the Arts Baden-Württemberg and DFG ("Deutsche Forschungsgemeinschaft").

The performance profiling was done with the help of the Energy oriented Centre of Excellence for computing applications on the Jureca cluster by the Jülich Supercomputing Centre.

References

[1] G. Smith, D. Golden, M. Frenklach, N. Moriarty, B. Eiteneer, M. Goldenberg et al., “Gri 3.0 reaction mechanism.”
[2] T. Poinsot and D. Veynante, Theoretical and Numerical Combustion. R.T. Edwards, 2001.
[3] T. Zirwes, F. Zhang, J. Denev, P. Habisreuther, and H. Bockhorn, “Automated code generation for maximizing performance of detailed chemistry calculations in OpenFOAM,” in High Performance Computing in Science and Engineering, pp 189-204
[4]] T. Zirwes, F. Zhang, J. Denev, P. Habisreuther, and H. Bockhorn, “Improved Vectorization for Efficient Chemistry Computations in OpenFOAM for Large Scale Combustion Simulations,” in High Performance Computing in Science and Engineering
[5] Jülich supercomputing Centre, http://www.fz-juelich.de/ias/jsc/EN/Home/home_node.html
[6] Extrae instrumentation package, http://www.vi-hps.org/Tools/Extrae.html
[7] Paraver performance tool, http://www.vi-hps.org/Tools/Paraver.html
[8] Scopre-P tracing tool, http://www.vi-hps.org/tools/score-p.html
[9] Vampir visualization tool, http://www.paratools.com/vampir/
[10] Steinbuch centre for computing, https://www.scc.kit.edu/dienste/forhlr2.php.
[11] R. Barlow, S. Meares, G. Magnotti, H. Cutcher, and A. Masri, “Local extinction and near-field structure in piloted turbulent CH4/air jet flames with inhomogeneous inlets,” Combust. Flame, vol. 162, no. 10, pp. 3516–3540, 2015.
[12] D. Goodwin, H. Moffat, and R. Speth, “Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes. version 2.3.0b,” 2017, software available at www.cantera.org.
[13] Lu, T., Law, C.K.: Toward Accommodating Realistic Fuel Chemistry in Large-Scale computations. Prog. Energy Combust. Sci. 35(2) 192–215 (2009)
[14] High performance computing center Stuttgart,” www.hlrs.de/systems/cray-xc40-hazel-hen, 2018.