Dynamic multi-level load balancing for scalable simulations of reacting multiphase flows

van den Oord, Gijs; Azizi, Victor; Fathi, Mohamad; Hickel, Stefan

DOI
10.1177/10943420251329199

Publication date
2025

Document Version
Final published version

Published in
International Journal of High Performance Computing Applications

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

Research Paper

Dynamic multi-level load balancing for scalable simulations of reacting multiphase flows

The International Journal of High Performance Computing Applications
2025, Vol. 0(0) 1–13

DOI: 10.1177/10943420251329199
journals.sagepub.com/home/hpc
Sage

Gijs van den Oord[1], Victor Azizi[1], Mohamad Fathi[2] and Stefan Hickel[2]

Abstract

Simulations of reacting multiphase flows tend to display an inhomogeneously distributed computational intensity over the spatial and temporal domains. The time-to-solution of chemical reaction rates can span multiple orders of magnitude due to the emergence of combustible kernels and thin turbulent reaction zones. Similarly, the time to solve the equation of state (EoS) for non-ideal fluid mixtures deviates substantially between the grid cells. These effects result in a performance profile that is unbalanced and rapidly changing for transient simulations, and therefore beyond the capabilities of traditional (quasi-) static mesh partitioning methods. We analyse this loss of parallel efficiency for large-eddy simulations of the ECN Spray-A benchmark with the multi-physics solver INCA and propose to mitigate the problem by introducing two independent repartitioning stages in addition to the classic domain decomposition for fluid transport: one for the EoS and one for chemical reactions. We explore various scalable repartitioning strategies in this context and observe that rebalancing computational load yields a significant speedup that is robust for various mesh resolutions and process numbers. The dynamic multistage load-balancing thus effectively removes obstacles towards good parallel scaling of INCA and similar solvers for reacting and/or multiphase flows.

Keywords

Load balancing, parallel computing, large-eddy simulation, reacting flow, multiphase flow, ECN spray-A

1. Introduction

High-fidelity simulations of reacting multiphase flows at high pressures are essential to the development and optimization of rocket engines, gas turbines, and modern energy conversion systems. Driven by the demand for higher efficiency and emission reduction, the operating pressure of these devices exceeds the critical values for fuel and oxidizer, but is usually lower than the cricondenbar pressures of their mixtures. States that locally occur during the mixing can thus fall within the subcritical two-phase region. This leads to a *transcritical* regime, where the multi-species mixture may locally condensate and develop liquid-gas interfaces Oschwald et al. (2006). Simulating such conditions requires a realistic representation of the complex multi-physics and multi-scale interactions of turbulence, non-linear variations of fluid properties including phase change, mixing, heat transfer, and chemical reactions. A proven successful approach is the combination of large-eddy simulation (LES) with a multiphase equilibrium model Matheis and Hickel (2014); Fathi et al. (2022) and a suitable (cubic) equation of state (EoS) reflecting the departure from ideal gas behavior at high pressures Müller et al. (2016). Matheis and Hickel (2018) pioneered this approach and demonstrated that LES with real-gas EoS including vapor liquid equilibrium (VLE) calculations can accurately reproduce experimental results for the Spray-A benchmark case Engine Combustion Network (ECN), (2021), where cold n-dodecane is injected into a warm nitrogen reservoir at a pressure of 6 MPa. The method can capture jet breakup, subcritical condensation and evaporation of species with different rates, as well as real-fluid effects such as dissolution of the ambient gas in the liquid or liquid-like fuel (Matheis and Hickel, 2018). The resulting flow displays combustible

[1]Natural Sciences and Engineering Section, Netherlands eScience Center, Amsterdam, The Netherlands
[2]Aerodynamics Group, Faculty of Aerospace Engineering, Delft University of Technology, Delft, The Netherlands

Corresponding author:
Gijs van den Oord, Natural Sciences and Engineering Section, Netherlands eScience Center, Science Park 402, Amsterdam 1098XH, The Netherlands.
Email: g.vandenoord@esciencecenter.nl

kernels and reaction zones within the two-phase flow regions in the outer shear layer, spreading out into a large gaseous cooling zone away from the injection point Skeen et al. (2015); Ma et al. (2019); Fathi et al. (2022).

The two-phase regions and reaction zones represent locations of high computational intensity for the numerical integration scheme. Solving the EoS to obtain pressure and temperature within a cell is computationally more expensive if the cell contains a vapor-liquid mixture that requires phase-splitting (VLE) calculations with an iterative procedure to solve the isochoric-isoenergetic flash problem. Calculating the species mass fraction evolution proves to be time-consuming whenever local conditions enable combustion. This evidently introduces load imbalance in a computational fluid dynamics (CFD) solver that has been parallelized via traditional domain decomposition of the coordinate grid.

Classic domain decomposition strategies implicitly assume a homogeneous computational cost proportional to the number of grid cells plus communication overhead with neighboring partitions. Numerical algorithms with non-homogeneous and non-stationary computational cost, such as the chemistry and multi-phase thermodynamics of fuel-injection simulations, require more sophisticated strategies to mitigate the emerging imbalance between processors. A straightforward method keeps a weight w^i for every single cell i, which represents the computational cost of the integration time step of that grid cell, and regularly applies a re-partitioning of the constructed weighted graph. This method assumes the weights are sufficiently auto-correlated over the integration period between re-partitionings, such that the principle of persistence Kalé (2002) holds. The strategy is typically combined with adaptive mesh refinement Berger and Colella (1989) to improve accuracy of turbulent or chemically reacting zones. In multi-physics simulations with synchronization points, the partitioning algorithm can be refined to include weights from individual processes Karypis and Kumar (1998).

The methods above always use a single universal parallel decomposition throughout the integration time step. This has the benefit that communication overhead from switching between partitionings is absent. The residual load imbalance, however, remains nonzero for each physical process because the partitioner has to make a compromise between the different (sequential) phases of the computation. Furthermore, the unbalanced computational phases often operate point-wise and the edge-cut objective is not relevant for such constraint. If data migration overhead is small with respect to such point-wise function evaluation, load *rebalancing* may yield better performance. Watts et al. (1997) have implemented a diffusion scheme to redistribute the uneven load of computing particle movement in electromagnetic fields in plasma physics. Lu et al. (2009) addresses the imbalanced fractional time stepping for chemistry in reactive flows within the framework of in situ adaptive tabulated (ISAT) chemical kinetics. Hiremath et al. (2012) examine some heuristic balancing strategies such as random reassignment of particles or partitioned versions thereof. Recently, Wu et al. (2019) deployed a more sophisticated parallel iterative balancing Aggarwal et al. (2003) algorithm that minimizes data migration by prioritizing grid cells where the 4th-order Rosenbrock-Krylov solver converges much slower than the average. Tekgül et al. (2021) applied a combination of dynamic load balancing and tabulation to reacting flows in the `OpenFOAM` compressible PIMPLE scheme, noting that the impact of the reference mapping optimization is substantial for transient problems like the ECN Spray-A case.

In this paper, we develop and analyse a dynamic load balancing method in a similar spirit as Refs. Wu et al. (2019); Tekgül et al. (2021). To accommodate the various phases in the time stepping, a flexible multi-stage approach is introduced. This novel method has been implemented in the multi-physics CFD solver `INCA` as two additional and independent work redistribution stages for (1) the thermodynamics and (2) chemistry, in addition to the classical domain decomposition for the fluid transport (stage 0). The methodology is described in Section , where we explore several load-balancing strategies and discuss their implementation. We evaluate the effect of various parameters and test the resulting parallel efficiency and scaling for the ECN Spray-A benchmark case in Section , and summarize and discuss our conclusions in Section .

2. Methodology

2.1. Fluid physics model and solver

The `INCA` CFD[1] code integrates the compressible multi-species Navier-Stokes equations with an explicit third-order Runge-Kutta time-marching method and a finite-volume space discretization; interested readers are referred to Hickel et al. (2014) for more details. To avoid prohibitively small time steps, stiff chemical source terms are computed with an implicit fifth-order time-marching scheme implemented in the `VODE` library (Brown et al., 1989) and coupled to the fluid dynamics through second-order Strang splitting. The real-fluid thermodynamics model is based on VLE calculations for cubic EoS which are solved by Newton-Raphson iteration in an effectively reduced space based on the molar specific value of the volume function derived from the Helmholtz free energy, see Fathi and Hickel (2021). This method prevents the quadratic growth of the computational time with the number of species observed with the conventional method (Matheis and Hickel, 2018), such that the cost of VLE solutions becomes essentially independent of the number of species (Fathi and Hickel, 2021). A detailed description of the physical models and numerical methods for thermodynamics and chemistry calculations is beyond

the scope of this paper and can be found in Refs. Fathi and Hickel (2021); Fathi et al. (2022). `INCA` operates on block-Cartesian meshes that are generated by an Adaptive Mesh Refinement (AMR) method. The mesh blocks are distributed over the available processors using the multi-constraint k-way domain partitioning algorithm provided by the `ParMETIS` library (Karypis and Kumar, 2013, 1998). Communication between the processors is implemented through the Message Passing Interface (`MPI`) library. We remark here that a single MPI task can therefore operate on multiple blocks, but blocks cannot me divided between multiple MPI tasks.

2.2. Characterization of load imbalance

Since neither the chemistry nor the thermodynamics solver wall clock time play any role in the baseline domain partitioning, the computational load imbalances within these physical models will naturally result into an overall decrease in parallel efficiency. This is illustrated for the Spray-A setup on a mesh with 1.8 million grid cells distributed over 2727 mesh blocks, for which a snapshot of the temperature field is shown in Figure 1. The reaction zone spans several mesh blocks and the VLE calculations result in localized spots of high computational load, as one can see in Figure 2. These cells populate the long tail in the timings distribution in Figure 3, where we have aggregated the timings of 100 time steps. The chemistry step, which also involves additional evaluations of the equation of state, is seen to be consistently slower than the thermodynamics. Spikes above the median load often, but not always, coincide for the two computational phases in the time step.

2.3. Objectives and limitations

Our aim is to redistribute the clustered cells of high computational load over available processors within each time step. For a given set of processes p and wall-clock times T_n spent on the computation within process of rank n, we intend to minimize the total load imbalance

$$\begin{aligned} L &= \frac{\max_{n\in P}(T_n)}{\langle T\rangle_P} - 1 \\ &= \max_{n\in P} L_n \end{aligned}$$

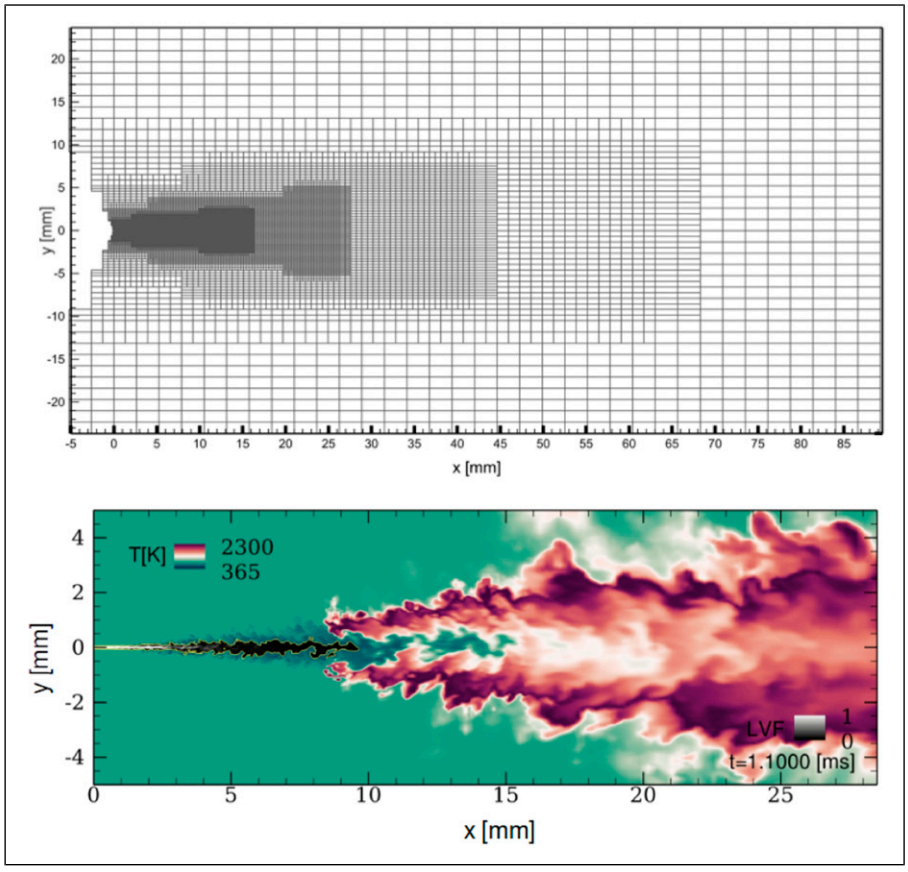

Figure 1. Adaptive mesh for the full domain of the Spray-A coarse grid setup and fluid temperature contours in and around the developed jet on a slice through the simulation domain for the reacting ECN Spray-A case.

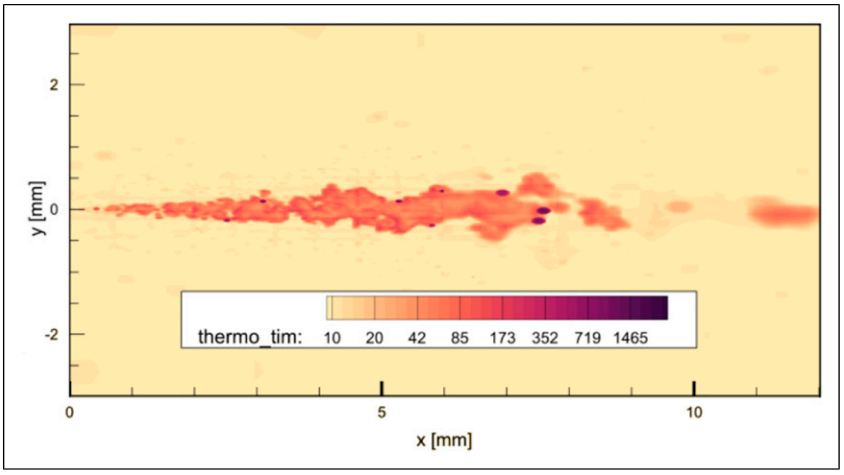

Figure 2. Time spent per grid point on the thermodynamics calculations for a snapshot of the developing jet.

where $L_n = T_n/\langle T \rangle_p - 1$ denotes the *fractional* load imbalance per process and the brackets $\langle \cdot \rangle_p$ indicate the mean over all processors. The sorted fractional load imbalances shown in Figure 4 display a sharp peak that leads to a large L and signals inefficiency in the parallellization of the problem.

Ignoring vectorization, memory bandwidth and cache effects, we can assume the timespan per processor is the sum of the compute times per grid cell,

$$T_n = \sum_{i \in I_n} t_i, \tag{1}$$

where I_n denotes the set of grid cells belonging to processor n within the initial partitioning. In the INCA application, we measure the t_i by clocking the iteration loop that solves the reaction rates or vapor-liquid equilibrium differential equations in each grid cell. Now, if we allow the transfer of grid cell data and the associated workload from one processor to another, we can dynamically minimize L, and the above formula becomes

$$T_n = \sum_{i \in I_n \setminus O_n} t_i + \sum_{m \neq n} \sum_{i \in P_{mn}} (s_i + t_i), \tag{2}$$

with P_{mn} being the set of grid cells transferred from processor m to n and

$$O_n = \bigcup_{k \neq n} P_{nk} \tag{3}$$

denotes the cells offloaded to other processors. In the formula above, s_i denotes the communication time to transfer the state of cell i from processor m to n and to return the results.

There exist many strategies for the selection of the partitions P_{mn}; our present methods are all based upon the principle of *persistence*. This is the assumption that the measured set $\{t_i\}$ at a rebalancing time step provides a sufficient indicator for the timings at the subsequent time step. Abandoning this principle would require a radically

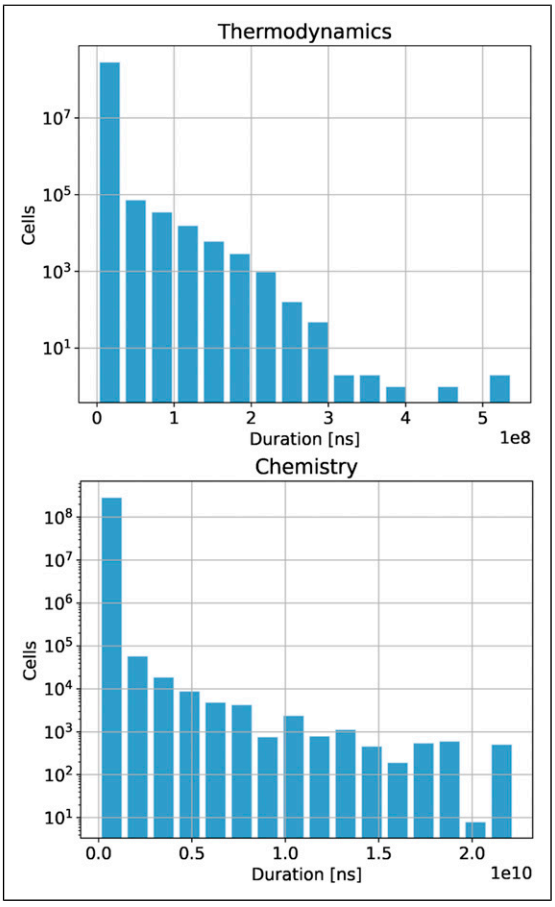

Figure 3. Characterization of load imbalance for ECN Spray-A benchmark. Histogram of time spent per grid cell on the thermodynamics calculations (top) and chemistry (bottom), for 700 consecutive time steps.

different load balancing approach, such as *work stealing*, which is challenging to implement over `MPI` whilst keeping low communication overhead (Klinkenberg et al., 2020). In a combined approach between work stealing from persistent task configurations was found to effectively mitigate the overhead of processors polling for work for high core counts, using HPC middleware to support task-based parallelism via active messages over MPI. In Micale et al. (2024), a hybrid approach was chosen, where OpenMP dynamic scheduling was used to balance chemistry calculations within partitioning domains.

Predictability of the timings per grid cell is a key limiting factor to the achievable increase in efficiency by dynamic load balancing and depends critically on the case at hand. A non-stationary case like the fuel injection in Spray-A contains a substantial number of locations where chemistry and thermodynamics compute times are uncorrelated between time steps. Throughout the development of the jet, combustion cores may appear or extinguish, leading to a significant unpredictable fraction of cells. The correlation of the chemistry calculation time of two consecutive time steps is analysed in Figure 5. Cells with unpredictable timing are represented by the horizontal and vertical arms in the left plot and the peaks at ± 1 in the histogram on the right in Figure 5. These grid cells may deteriorate the effectiveness of persistence-based algorithms and we will discuss the effect of this limitation based on our results in Section .

2.4. Load balancing methods

Given the persistence assumption, there are plenty of strategies one can follow to obtain the P_{mn} in order to reduce the load imbalance L. However, in order to select appropriate candidates for a large-scale `MPI`-parallel CFD application such as `INCA`, we face a trade-off between various

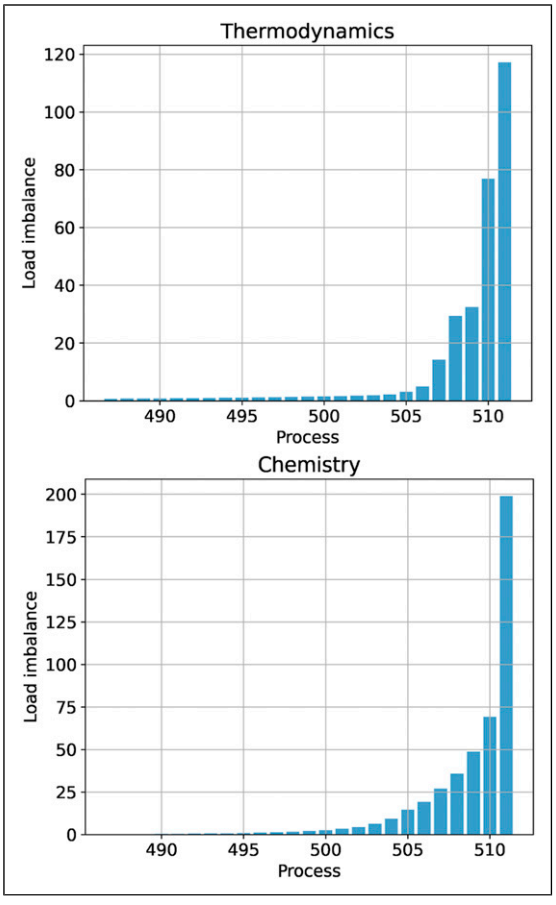

Figure 4. Characterization of load imbalance for ECN Spray-A benchmark. Bar charts of the average load imbalance of the top 5% most unbalanced processors, for a configuration of 512 MPI tasks.

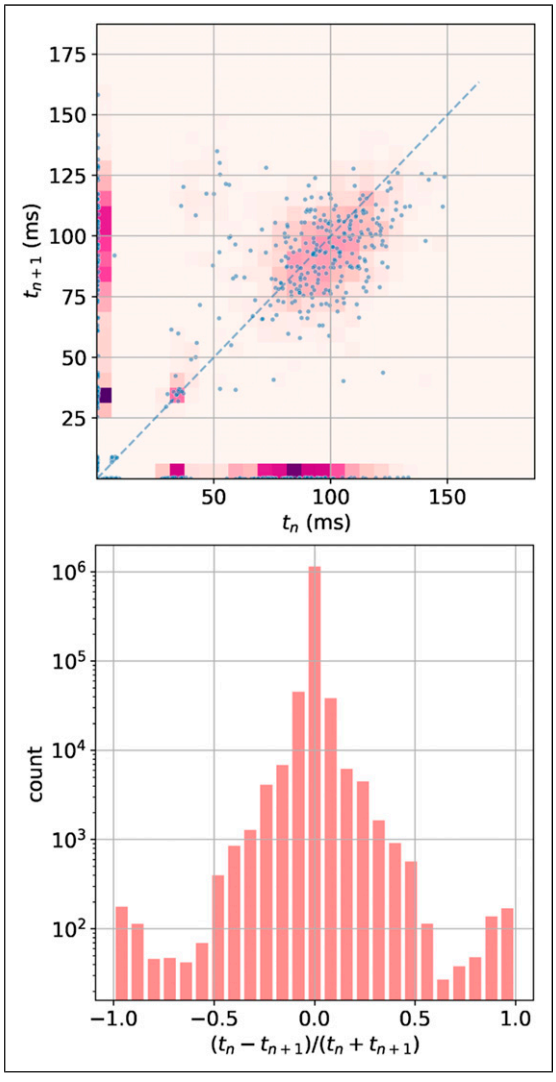

Figure 5. Top: the auto-correlation of the chemistry calculation time of two consecutive time steps per grid cell. Color contours show a 2d histogram of the derived density. Bottom: histogram of the normalized difference of the chemistry calculation time.

aspects of dynamic load balancing. First of all, the (single-threaded) wall-clock time of the partitioning computation itself should not become a dominant factor in the performance profile, meaning that the strategy should be either relatively fast and iterative methods should be controlled with a tolerance parameter to avoid carrying out many passes that do not provide significant performance benefits. Secondly, we favor parallel and scalable partitioning algorithms that minimize collective communication and do not require a large-memory master process gathering timing information from all grid cells. This has lead us to dismiss the so-called 1.5-order algorithm described in Ref. Aggarwal et al. (2003) and examined in Ref. Wu et al. (2019). Finally, the algorithm should produce balanced partitions that not only minimize processor idle time, but also data movement across compute nodes, in order to keep the communication overhead s_i terms in equation (2) small.

The package ZOLTAN Devine et al. (2009) contains a useful set of routines to compute the task distribution based upon *hypergraph repartitioning*. We can assign the 'baseline' distribution of grid cells in INCA (domain decomposition used for fluid dynamics) as the original partition and let the ZOLTAN library compute an optimization of this using the measured timings. Since the cell-wise computations involve no communications with neighbors, the hyperedges in this setup contain the grid cells residing on the same compute node, where task migration involves only copying memory and no networking. Because the mesh topology in this case is extremely simple, the hypergraph partitioning tends to be too heavyweight for our purpose. We have therefore developed several simpler alternatives with better overall efficiency. These methods share the method of offloading blocks in one-to-one communication, but differ in how they select which processes should communicate to offload work.

2.4.1. One-to-one communication. The one-to-one communication between processes proceeds only if the pair (m, n) has an overloaded and underloaded processor,

$$L_n < 0 < L_m. \tag{4}$$

Then the list of surplus grid cells of the overloaded processor is determined by taking the maximal value of k such that

$$\sum_{i>k} t_i - \langle T \rangle_P > 0, \tag{5}$$

where the timing t_i denote the grid cell weights in the overloaded process m. The weights $\{t_1, \ldots, t_k\}$ are sent to process n, which determines how many points it can accommodate without becoming overloaded, that is computing the minimal value of ℓ for which

$$\frac{1}{\langle T \rangle_P} \sum_{i<\ell} t_i + L_n \geq 0. \tag{6}$$

Then the cells one to ℓ (including ℓ) are flagged to be offloaded from process m to n and the respective load imbalances are updated to

$$L_n \to 0, \quad L_m \to L_m + L_n. \tag{7}$$

If the overloaded processor m already had offloaded cells to other processors in a preceding iteration, these are discarded in the above work exchange determination. Furthermore, there is a slight discrepancy between the updated load imbalances above and the 'true' loads carried by n and m since equation (6) generally corresponds to a small positive value for the updated L_n. In practice, this means that imposing a very small L-threshold as a stopping condition for the iterations may result in a configuration that slightly exceeds this value.

2.4.2. Load balancing strategies. The load-balancing strategies now differ in the sequence in which these point-to-point communications are being issued:

- GREEDY: Simple round-robin-like balancing algorithm. During each iteration i, every processor (say with rank n) engages in the above offloading dialogue with processor $n + i$. After each iteration, the approximate quality of the partitioning is checked by determining if $\max_{n \in P}(L_n)$ is below the maximum allowed absolute load imbalance, or if i is equal to the number of processors. If either is true the partitioning is done.
- SORT: This strategy distributes the partial load imbalances L_i to all processes and locally sorts the processes according to descending load. If we represent this sorting by the mapping $S: \{1, \ldots, N\} \to P$, then the above work offloading procedure is started from $S(1)$ to $S(N)$, $S(2)$ to $S(N-1)$ and so forth. This algorithm is repeated until the load imbalance is below the maximum allowed absolute load imbalance or the redistributed load becomes less than a user-specified tolerance $\Delta L_{\min}$. We use $\Delta L_{\min} = 0$ in this paper. This is the same algorithm as described in Ref. Tekgül et al. (2021) and does not provide special treatment of node-crossing task offloading.
- SORT2: This algorithm is a node-aware version of the above SORT method; it applies SORT first within all process groups on shared compute nodes, favoring intra-node offloading. After that, it continues identically to SORT to resolve the final imbalances.

One may notice that we use several limiting criteria in the above methods. For GREEDY, the user needs to provide a

desired global threshold for L, which may not be known in advance. For the sorted algorithms we also can use a convergence parameter, halting the whenever L no longer significantly decreases. The latter is unsuitable for the GREEDY method because it may perform intermediate iterations where L does not decrease significantly due to the pre-assignment of the processors. In any case, we additionally impose a maximum number of allowed iterations after which the partitioning halts.

2.5. Implementation

The load-balancing functionality has been implemented in a modular and object-oriented design. To support multiple physics routines which need re-balancing within an application, the design supports an arbitrary number of independent load-balancing tasks in a straightforward and transparently programmable way. The infrastructure is implemented at a high level of abstraction, is unaware of the underlying mesh, and supports work sharing via data transfers of any size and type. Within the `INCA` solver, the same work-offloading and partitioning code is executed for both the chemistry calculations and the thermodynamics. Figure 6 schematically illustrates the time stepping of the simulation of a reacting flow: during each Runge-Kutta substep, the fluid transport terms are evaluated as first part of the operator splitting. Then the thermodynamic equation of state, viscosity, diffusivity and conductivity are evaluated for the interior (non-boundary) cells. After integrating the boundary conditions, the same quantities are computed for the boundary cells. In the last Runge-Kutta sub-step, the stiff system of chemical reaction rates is solved with a Strang-splitting procedure; again this process involves evaluating the equation of state twice in another operator splitting.

At the heart of the implementation is the load balancer object, a Fortran type `t_loadbalancer`. The load balancer instance keeps an administration of process IDs that receive data from and offload data to its current `MPI` task. The data associated with the compute task is subdivided into chunks comprising one or several grid cells. Upon creation, the caller must provide the load balancer object how large the outgoing requests should be per grid cell and how many bytes the incoming results will take. For example, if the load balancer is used for distributing a double-precision calculation of pressure, temperature and the speed of sound from density and internal energy, then the requests and results buffers should be 16 and 24 bytes long, respectively. The (de-)serialization of the in- and output fields to these buffers have to be implemented by the user application and, together with the to-be-balanced routine, have to be supplied to the load balancer object as procedure pointers. Once these implementations are present and the load balancer instance contains a valid, non-trivial

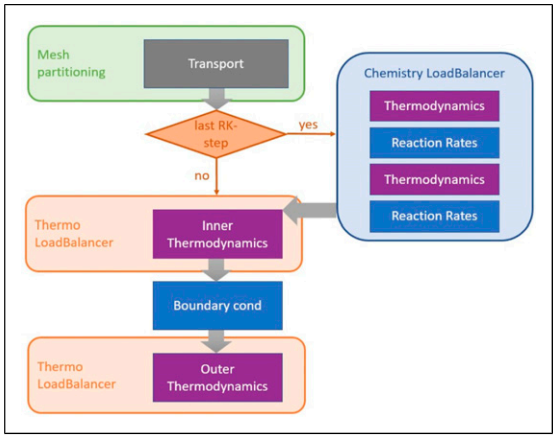

Figure 6. Runge-Kutta time step flow diagram. The various load-balanced functional blocks are indicated. The chemistry calculations are only executed the last step, using a double Strang splitting with a thermodynamics update.

Table 1. Details of the Spray-A simulations.

Case	Spray-A-coarse	Spray-A-fine
Grid points	1.8 million	10.6 million
AMR blocks	2727	2864
Time steps	700	400
Rebalancing frequency	1 step	1step
Load balancing chunk size	4 cells	4 cells

partitioning, the computation is redistributed by the following sequence of calls:

```
! Send work to accepting processes
! and/or receive work from offloading
! processes:
call load_balancer% COMMUNICATE_DATA
! Compute 'native' grid cells:
call load_balancer% LOCAL_COMPUTATION
! Compute migrated grid cells:
call load_balancer% IMPORT_COMPUTATION
! Receive results from accepting
! processes and/or return results to
! offloading processes:
call load_balancer% COMMUNICATE_RESULT
```

During the computations, the load balancer keeps track of the time spent per grid cell to construct a partitioning weight. Periodically, the load balancer can be instructed to recompute the partitioning following one of the strategies listed in the previous section.

We provide the load balancing methods implemented in this paper as an open-source package called `QUICKL`,[2] which includes the original Fortran implementation as a standalone library, a *Python* interface, and documentation with examples.

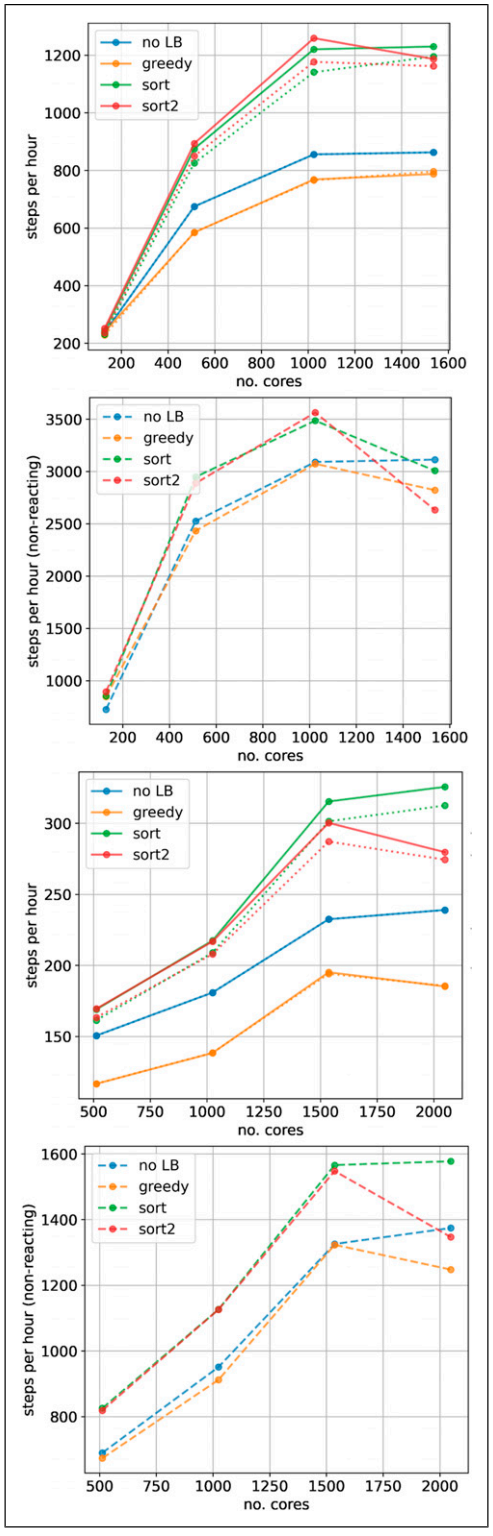

Figure 7. Strong scaling of the load-balanced INCA program for the Spray-A-coarse case (top), Spray-A-coarse without chemistry (2nd from above), Spray-A-fine case (3rd from above) and Spray-A-fine without chemistry (bottom). The dotted lines in top and 3rd plot denote the performance using only chemistry load balancing.

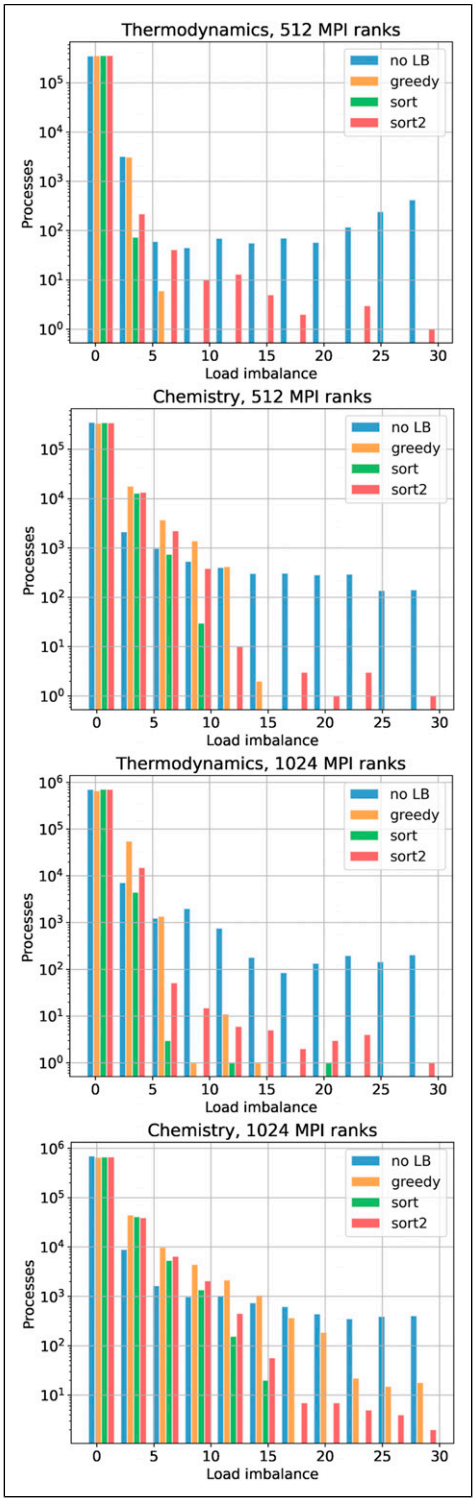

Figure 8. Load imbalance histograms for thermodynamic EoS solving of the Spray-A-coarse simulation with 512 processors (top), chemistry solver of Spray-A-coarse (second from above), thermodynamics solver of Spray-A-fine using 1024 processors, and chemistry solver of Spray-A-fine at the bottom. All computed load imbalances for all time steps are aggregated. The baseline histogram tail has been cut off for visualization purposes.

3. Results

Our results are based upon a low-resolution and a high-resolution version of the reacting Spray-A benchmark (see Table 1) simulation with the INCA solver. For the low-resolution run, we use a mesh containing 1.8 million cells distributed over 2727 blocks, which was generated by AMR but remains fixed throughout all experiments. For the high-resolution case, the mesh contains 2864 AMR blocks and about 11 million cells. Results for the coarse mesh are obtained using 700 time steps starting from a state with a well-developed reacting jet. The fine-resolution results are based upon 400 time steps to keep the computational cost limited.

For the load-balancing strategies that we explore, we use chunk sizes of four grid cells, which was observed as an optimal value from a preliminary parameter space scan using 10 time steps of the Spray-A-coarse benchmarks. We have examined the impact of the repartitioning frequency using a few full simulations and found that re-balancing each time step yields the best overall performance. This is because the persistence reduces when re-balancing is only performed every few time steps and the partitioning process causes only a small overhead with respect to the possibly large load imbalance of chemistry calculations in rapidly evolving flows. As for convergence criteria, the objective is always a global maximum load imbalance of 1%, but we impose a maximum of 100 iterations for each balancing strategy. The load-balancing of thermodynamics and chemistry calculations are performed independently from another with the same load-balancing method and the same parameters unless stated otherwise.All simulations have been executed on the Dutch supercomputer Snellius, which is a cluster of dual-socket nodes containing two 64-core 2.6Ghz AMD Rome 7H12 processors, connected by an Infiniband HDR100 (100Gbs) network. For each experiment, we have used a single core per MPI task, and no OpenMP parallelization has been used. Note that the number of AMR blocks provides an upper limit to the amount of MPI tasks that can be used by INCA, and ideal parallel speedup for the fluid transport can be expected only

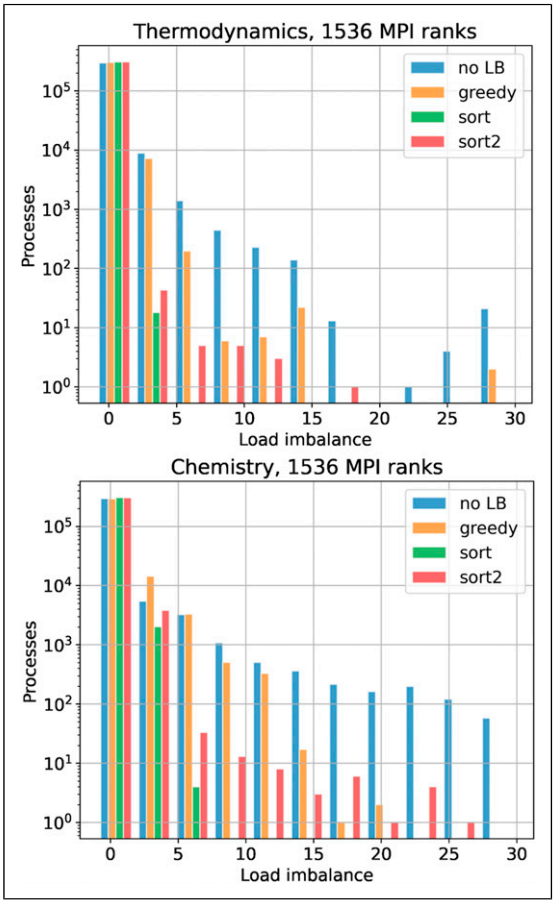

Figure 9. Load imbalance histograms for the fine-mesh SprayA simulation for 1536 processors. All computed load imbalances for all time steps are aggregated. The baseline histogram tail has been cut off for visualization purposes.

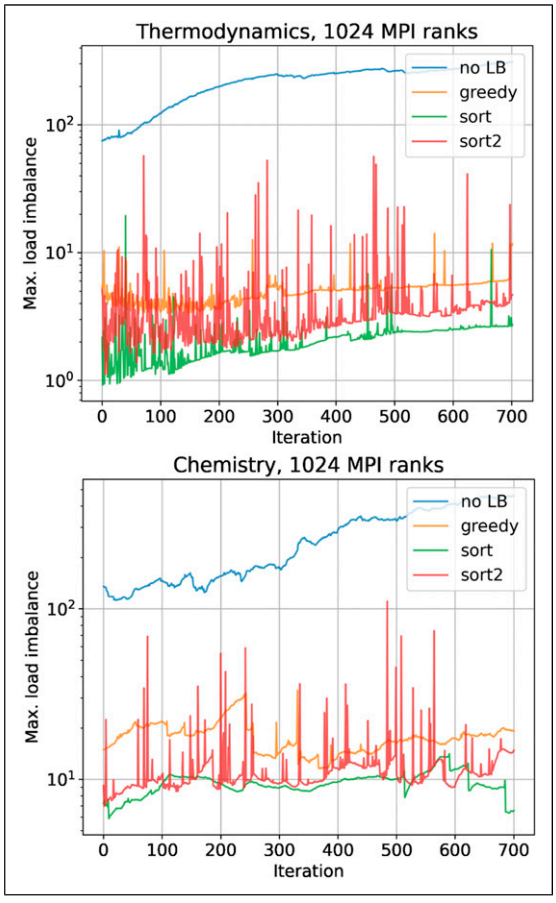

Figure 10. Time series for the value of $\max_{n \in p}(L_n)$ produced by the load balancers for the coarse Spray-A case with 1024 processes.

up to about half this maximum number of MPI tasks as the AMR blocks have unequal cell numbers in both considered cases. All benchmarks have been performed without dynamically refining the mesh.

3.1. Parallel scaling

The resulting performance of the load-balanced application and its strong scaling is displayed in Figure 7 for all methods. In these charts we have also plotted lines where we substituted the equation of state solver with the timings of the baseline run (dotted lines in the left column) as a proxy for the scaling behavior in the absence of thermodynamics load balancing. Likewise, the charts in the right column of Figure 7 are obtained by subtracting the chemistry solver time as a proxy for a non-reacting case. These are plotted in separate charts because the solution to the thermodynamic EoS is much faster than the chemistry and its load-balancing impact is almost unnoticeable in a reacting flow profile. The performance characteristics of the various load-balancing strategies are very similar for both tested resolutions, available cores and process at hand; the sorting algorithms (`SORT` and `SORT2`) both result in a substantial speedup of the solver, reaching around 45% at higher core counts, where the program ceases to scale any further. The `GREEDY` algorithm is observed to be consistently slower than the baseline, especially in the high-resolution reacting case. The improved balance of the computational load fails to offset the overhead of repartitioning. Reducing the frequency of the `GREEDY` repartitioning has been observed to lift its performance above the imbalanced case, but the algorithm cannot match the speedups achieved by the sorted variants.

The `SORT2` method performs similarly to the regular sorting, except at high core counts, where the parallel scaling is sub-optimal compared to the `SORT` strategy. This is due to the fact that the constraint of re-balancing within individual nodes leads to many small communication requests and reduces the quality of the partitioning especially for a high number of nodes.

3.2. Rebalancing efficiency

The partitions resulting from the methods described in Section display different characteristics and quality. In Figures 8 and 9 we depict the distribution of processor loads $\{L_i\}_{i\in P}$ for various methods and values of $|p|$. Roughly speaking, the tail of these distributions determine the performance of the load-balanced solver. We see that overall, `SORT` produces the shortest tail and is, in agreement with the scaling numbers above, the best performing algorithm. Note that the tail of the `SORT` distribution extends well beyond the target maximal load imbalance of 0.01, which is due to the fact that the maximal number of 100 iterations is usually reached while $\max_{n\in p}(L_n)$ is larger than this threshold. The `GREEDY` and `SORT2` tend to produce a thin tail of high processor loads. Here it seems like `SORT2` is the worse performer due to a few highly loaded processes spread thinly across the plotted range, which would be contradictory to the global performance numbers of the two strategies in Figure 7. A closer inspection of the time evolution of the maximum load imbalance shown in Figure 10, however, reveals that these tails of high loads result from incidental spikes, and the `SORT2` strategy performs systematically better than `GREEDY`, and close to the efficiency of `SORT`.

We can crudely explore weak scaling by comparing the 2048-core high-resolution case with a 512-core coarse-resolution setup, which leads us to conclude that, from these results, the weak scaling is in line with the unbalanced version over the tested range of core counts.

3.3. Predictability of process loads

In Section , we pointed out that the autocorrelation of chemistry and thermodynamics solver time at the grid-cell

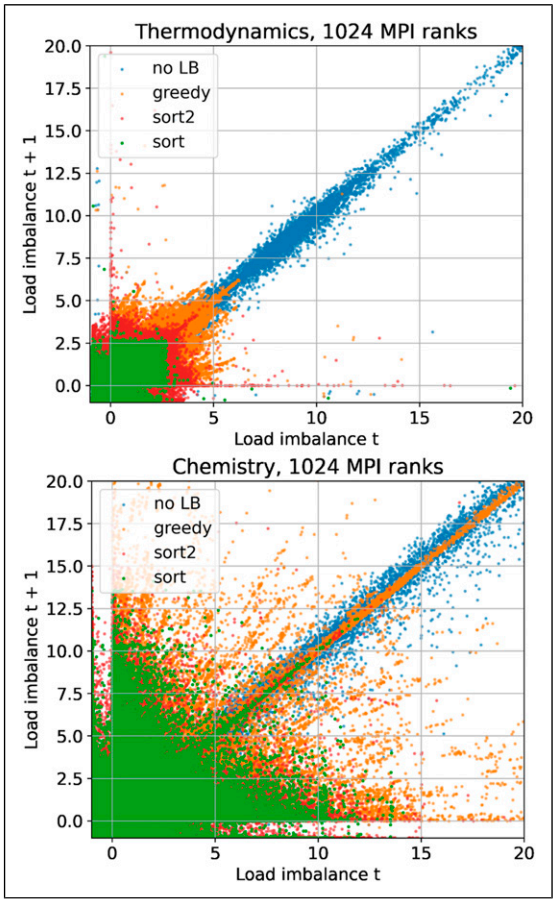

Figure 11. Measured load per processor plotted against load in previous repartitioning for the equation of state (left) and chemistry solver (right). The plotting range has been limited to 20 to make the load-balanced data visible.

level displays highly off-diagonal clusters, limiting the efficiency of load balancing strategies based upon the principle of persistence. Fortunately at the process level, the load is a lot more predictable because the cells of high computational load tend to stay within the same AMR blocks. Hence, the blue dots with higher load imbalance in Figure 5 are grouped along the diagonal axis. Load balanced methods, on the other hand, tend to erase this correlation; the algorithm redistributes costly cells across processes in an ad hoc fashion, which results in an uncorrelated cloud of points near the origin. The sub-optimal `GREEDY` algorithm retains a tail of highly loaded processes that are persistently slow, especially for the chemical reactions. We hypothesize that these essentially are processes of high computational load that were not redistributed due to the fact that the algorithm has exceeded its maximal number of iterations. Allowing the slow-converging `GREEDY` to take more iterations indeed results in an improved distribution of loads, but it comes at the cost of more repartitioning overhead. A separate coarse-resolution experiment on 1024 MPI ranks, where we allowed `GREEDY` repartitioning to take 1000 iterations resulted in a performance that was 10% faster than the baseline and substantially reduced maximum load imbalance (Figure 11).

4. Conclusions and outlook

In recent years, progress in HPC hardware has been driven by a steady increase of the number of logical cores per compute node. Software seeking to effectively leverage this computational power should therefore implement the underlying algorithm as a parallel workflow that is balanced across the available resources. Traditionally in CFD, optimal parallelization of the fluid dynamics requires a balanced partition of the mesh that minimizes edge-cuts. The situation, however, drastically changes when chemical reactions and multi-phase thermodynamics enter the problem and dominate the performance profile. These are simulation aspects for which the computational intensity is usually highly nonuniform across the physical domain. Moreover, these physical processes lead to strong dynamic fluctuations of computational load within a given grid cell in transient simulations, making a partitioning that is (quasi-)static with respect to the flow evolution by definition sub-optimal. We have signalled this problem for reacting multi-phase flows, and illustrated this by measured performance metrics for simulations of the ECN Spray-A benchmark with the `INCA` solver.

We have mitigated the uneven computational burden of chemical reactions and multi-phase thermodynamics in `INCA` by dynamically offloading work to under-utilized cores. The implementation is done in a highly abstract way, allowing us to use independent dynamic load balancers for the chemistry and thermodynamics in `INCA`. We have designed and implemented multiple repartitioning strategies, which are parallel in nature and bring the number of `MPI` collective calls to a minimum. The relevant parameters for these repartitioning methods are the granularity of the offloaded work, the stopping criterium for the iterative repartitioning loop and the repartitioning frequency. Due to the small number of collective communications in the implementation of these algorithms, we can safely choose a small granularity, a strict stopping criterium and high frequency of repartitioning, in order to achieve a high-quality repartitioning of the work load.

For sorting-based load balancing strategies, which assemble the partitioning by pairing highest- with lowest-cost processors, we see a reduction in the maximum processors load by an order of magnitude. This translates in a 45% reduction of the overall wall-clock time of the application, a speedup that is more or less independent of the number of available resources. A sorting algorithm designed to favor intra-node communication does not seem to have significant benefit, and actually scales worse at high core counts due to its constrained nature, which leads us to conclude that the communication latency and bandwidth limitations are small factors compared to the local computational cost. The basic round-robin-based `GREEDY` method slows down the program. This is likely due to the slow convergence of the iterative rebalancing, resulting in sub-optimal partitions. However, increasing the maximal number of rebalancing iterations could likely still provide a speedup with the `GREEDY` method in many cases.

We have observed substantial decorrelation of the computational cost between time steps at the grid cell level for both the chemistry and thermodynamics for Spray-A. This is a consequence of the unpredictable nature of the interactions between a turbulent velocity field and thin gas-liquid interfaces, small ignition kernels, and thin flame fronts in transient flows. This unpredictability can be a limiting factor to the effectiveness of the dynamic load balancing we have adopted here. At the aggregated process level however, the correlation is much higher for high-cost processes.

Obviously, other types of boundary conditions, geometries, or flow conditions may impact these findings: for example, we expect cases with fewer chemically reacting species or injection pressures beyond or below the trans-critical regime to be computationally less demanding and less imbalanced. In such situations, the optimal load balancer settings could well change, since the ratio of communication overhead and computation increases. Also, more stationary cases (such as the reactive shear layer configuration in Tekgül et al. (2021)) may improve the speedup of the load balanced code as the compute intensity of the grid cells are more predictable and partitionings based on measurements of the preceding time step are more effective. An interesting direction for future research is finding a good predictor of the subsequent run time for a

given grid cell, based upon its neighborhood and current physical state. With novel machine-learning approaches, the persistence assumption could be improved upon, paving the way for more efficient partitions for which efficiency levels of work-stealing methods could be achieved.

The load-balancing core is completely unaware of the spatial discretization, the variables being computed, or the time stepping; it is therefore highly abstract and re-usable in any other MPI-parallel code. Wherever the workload becomes highly unbalanced in the most time-consuming section of the algorithm with respect to the original parallelization, it makes sense to consider work redistribution, which can subsequently be fine-tuned per use case by altering block sizes, repartitioning intervals, quality threshold or work distribution strategy. Hence this method can be applied to many multi-physics problems dealing with localized computational loads with a relatively modest modification of the code structure.

5. Supplementary material

The load-balancing library has been developed as part of `INCA` and is also available as a stand-alone library, `QUICKLB`, for use in other HPC codes. The open-source package `QUICKLB` includes the original Fortran implementation and a *Python* interface and is available at https://zenodo.org/record/6598303. Additional documentation is available at https://quicklb.readthedocs.io. The benchmark data and plotting scripts are publically available on Zenodo at https://doi.org/10.5281/zenodo.14846663

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Dutch Research Council (Nederlandse Organisatie voor Wetenschappelijk Onderzoek - NWO) grant number 680.91.082 and the Netherlands eScience Center project number 027.017.G08. This work was carried out on the Dutch national e-infrastructure with the support of SURF Cooperative.

ORCID iDs

Gijs van den Oord https://orcid.org/0000-0001-8367-1333
Stefan Hickel https://orcid.org/0000-0002-7463-9531

Notes

1. https://www.inca-cfd.com.
2. https://zenodo.org/record/6598303 and https://quicklb.readthedocs.io.

References

Aggarwal G, Motwani R and Zhu A (2003) The load rebalancing problem. In: *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 258–265.

Berger MJ and Colella P (1989) Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics* 82(1): 64–84.

Brown PN, Byrne GD and Hindmarsh AC (1989) VODE: a variable-coefficient ODE solver. *SIAM Journal on Scientific and Statistical Computing* 10(5): 1038–1051.

Engine Combustion Network (ECN) (2021) Spray-A operating condition. https://ecn.sandia.gov/diesel-spray-combustion/target-condition/spray-ab

Fathi M and Hickel S (2021) Rapid multi-component phase-split calculations using volume functions and reduction methods. *AIChE Journal* 67(6): e17174.

Fathi M, Hickel S and Roekaerts D (2022) Large eddy simulations of reacting and non-reacting transcritical fuel sprays using multiphase thermodynamics. *Physics of Fluids* 34: 085131.

George K and Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 20(1): 359–392. https://www.cs.umn.edu/~metis

George K and Kumar V (2013) ParMETIS: parallel graph partitioning and fill-reducing matrix ordering. https://www.cs.umn.edu/~metis

Hickel S, Egerer CP and Larsson J (2014) Subgrid-scale modeling for implicit large eddy simulation of compressible flows and shock-turbulence interaction. *Physics of Fluids* 26(10): 106101.

Hiremath V, Lantz SR, Wang H, et al. (2012) Computationally-efficient and scalable parallel implementation of chemistry in simulations of turbulent combustion. *Combustion and Flame* 159(10): 3096–3109.

Jan M and Hickel S (2018) Multi-component vapor-liquid equilibrium model for LES of high-pressure fuel injection and application to ECN Spray A. *International Journal of Multiphase Flow* 99: 294–311.

Klinkenberg J, Samfass P, Bader M, et al. (2020) CHAMELEON: reactive load balancing for hybrid MPI+OpenMP task-parallel applications. *Journal of Parallel and Distributed Computing* 138: 55–64.

Lu L, Lantz SR, Ren Z, et al. (2009) Computationally efficient implementation of combustion chemistry in parallel pdf calculations. *Journal of Computational Physics* 228(15): 5490–5525.

Ma PC, Wu H, Jaravel T, et al. (2019) Large-eddy simulations of transcritical injection and auto-ignition using diffuse-interface method and finite-rate chemistry. *Proceedings of the Combustion Institute* 37(3): 3303–3310.

Micale D, Bracconi M and Maestri M (2024) Increasing computational efficiency of cfd simulations of reactive flows at catalyst surfaces through dynamic load balancing. *ACS Engineering Au* 4(3): 312–324.

Müller H, Niedermeier CA, Matheis J, et al. (2016) S> Hickel. Large-eddy simulation of nitrogen injection at trans- and supercritical conditions. *Physics of Fluids* 28(1): 015102.

Oschwald M, Smith JJ, Branam R, et al. (2006) Injection of fluids into supercritical environments. *Combustion Science and Technology* 178(1-3): 49–100.

Skeen SA, Manin J and Pickett LM (2015) Simultaneous formaldehyde PLIF and high-speed schlieren imaging for ignition visualization in high-pressure spray flames. *Proceedings of the Combustion Institute* 35(3): 3167–3174.

Tekgül B, Peltonen P, Kahila H, et al. (2021) DLBFoam: an open-source dynamic load balancing model for fast reacting flow simulations in OpenFOAM. *Computer Physics Communications* 267: 108073.

Watts J, Rieffel M and Taylor S (1997) A load balancing technique for multiphase computations. *Proc. of High Performance Computing* 97: 15–20. https://surface.syr.edu/lcsmith_other/18

Wu H, Ma PC and Ihme M (2019) Efficient time-stepping techniques for simulating turbulent reactive flows with stiff chemistry. *Computer Physics Communications* 243: 81–96.

Author biographies

Gijs van den Oord studied theoretical physics and mathematics at Utrecht University and obtained a PhD on Monte Carlo simulation for particle physics at Radboud University Nijmegen. He joined the Netherlands eScience Center in 2016 where he is primarily focused on high performance computing for geosciences and fluid dynamics. In 2024 Gijs became head of the Natural Science and Engineering section of the Netherlands eScience Center.

Victor Azizi completed his master's degree in computer science in 2016 and worked at the computational science lab of the University of Amsterdam on lattice Boltzmann solvers for blood flow simulation. He currently works at the Netherlands eScience Center on a wide range of topics in computational science and high-performance computing for complex numerical modelling.

Mohamad Fathi is postdoctoral researcher at Delft University of Technology. He earned his MSc in aerospace engineering in Delft, graduating summa cum laude in the thermo-fluids track. His PhD research involved developing high-fidelity simulations of reacting and non-reacting transcritical fuel sprays using multiphase thermodynamics.

Stefan Hickel is head of the aerodynamics section at Delft University of Technology and director of the Dutch graduate school of aerospace engineering. His main areas of interest are theoretical and computational fluid dynamics with an emphasis on turbulence phenomena in aerodynamics and flight propulsion as well as turbulence in fluids with complex thermodynamics. Professor Hickel has over fifteen years experience in the field of turbulence modelling and the development of highly accurate and computationally efficient numerical methods for Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS).