Parallel Load Balancing Strategies for Mesh-Independent Spray Vaporization and Collision Models

Federico Perini, Wisconsin Engine Research Consultants LLC
Stephen Busch, Sandia National Laboratories
Rolf D. Reitz, Wisconsin Engine Research Consultants LLC
Angela Wu, Sandia National Laboratories

Abstract

Appropriate spray modeling in multidimensional simulations of diesel engines is well known to affect the overall accuracy of the results. More and more accurate models are being developed to deal with drop dynamics, breakup, collisions, and vaporization/multiphase processes; the latter ones being the most computationally demanding. In fact, in parallel calculations, the droplets occupy a physical region of the in-cylinder domain, which is generally very different than the topology-driven finite-volume mesh decomposition. This makes the CPU decomposition of the spray cloud severely uneven when many CPUs are employed, yielding poor parallel performance of the spray computation. Furthermore, mesh-independent models such as collision calculations require checking of each possible droplet pair, which leads to a practically intractable $O(n_p^2/2)$ computational cost, n_p being the total number of droplets in the spray cloud, and additional overhead for parallel communications. This problem is usually overcome by employing O'Rourke's same-cell collision condition, which, however, introduces severe mesh dependency. In this work, we introduced two strategies to achieve optimal load balancing for fast spray calculations with mesh-independent models. Both methods were implemented in the FRESCO CFD code. For drop collisions, a mesh-independent collision detection algorithm with high parallel efficiency was developed. This method pre-sorts eligible collision pairs using a high-performance three-dimensional clustering algorithm similar to what is used for on-the-fly chemistry model reduction; these are then filtered again based on deterministic impact parameters and assembled in parallel into a global sparse adjacency structure. For the particle-in-cell vaporization/multiphase solver, we developed a solution-preserving load balancing algorithm. At each timestep, the parallel cell-ownership-based spray cloud structure is re-sorted into cell-owner bins, which are used to distribute the spray parcels across all CPUs along with their cell thermodynamic states; the distributed solution results are then sent back to the cell owners. The combination of both methods achieved more than one order of magnitude speed-up in spray solution for diesel engine simulations with a full and sector cylinder geometry.

Introduction

Lagrangian particle modeling using the Lagrangian-Drop/Eulerian Fluid (LDEF) method [1] represents a critical boundary condition in the multidimensional modeling chain of direct-injection engines, as spray drop scales are orders of magnitude smaller than one's engine multidimensional grid. Eulerian fluid modelling in the cylinder is not feasible when the engine scale is of interest, and Lagrangian trackers representing liquid parcels or drop distributions are preferred. Lagrangian parcels act as moving boundary conditions to the Eulerian CFD solver, as mass, energy, and momentum are transferred to the Eulerian phase as source terms due to spray in the Navier-Stokes equations [2]. Achieving an accurate representation of the liquid phase in terms of both size and momentum distributions is hence necessary to achieve a good representation of gas-phase fuel-air mixing and combustion in the engine.

In the recent past models have been developed to improve several aspects of the computational representation of the Lagrangian spray cloud: for breakup [3], injection and drop dynamics [4, 5], drop-to-drop collisions and their outcomes [6, 7], and vaporization [8, 9]. These models address deficiencies in both physical modeling and the computational implementation; the latter efforts being mainly devoted at reducing the dependency of the aforementioned algorithms on the local mesh size and structure.

All these models increase the computational burden of the Lagrangian spray calculation; and, as they have not been developed with execution on parallel computers in mind, they typically do not scale well in parallel on many CPUs. In this work, we tackle the parallel scaling of spray algorithms, and develop methods for the efficient scalability of previously developed vaporization and mesh-independent collision methods [5]. For vaporization/multiphase source terms, we develop a solution-preserving method. Their calculation depends on the whole set of particles inside one computational cell and its local thermodynamic state. Therefore, a greedy algorithm for load-balancing of the vaporization solver, based on solution "bins", was developed to scatter drops and their gas-phase cell owners across the CPUs. Each bin contains both one cell's field and geometry information and all drops inside of it. In this way, the owner cell composition is sequentially updated in the same way as in the serial solver, and an identical solution is achieved as with the serial case.

For collisions, we developed a parallel method for mesh-independent estimation of eligible drop-to-drop collision pairs in the whole mesh. The whole spray structure is copied onto each CPU; and a high-performance three-dimensional (3d) clustering algorithm [10] is run to pre-sort eligible collision pairs based on previously assessed eligibility conditions [5]. Additional filtering based on a deterministic impact parameter is applied, and all eligible collision pairs are eventually stored into a parallel, lower-triangular sparse adjacency structure, achieving a fast, parallel, and mesh-independent collision solution.

The new load balancing algorithms were assessed against two well-suited diesel engine simulation setups: one features a conventional diesel combustion (CDC) operating point with a near-TDC pilot-main injection strategy, and is run with a full

engine cylinder model and all injector nozzle holes; another features a sector mesh representation and an Reactivity-Controlled Compression Ignition (RCCI)-like single injection pulse, early during the compression stroke, with a longer liquid phase lifetime. Both cases highlighted the capability of the two load balancing methods to achieve more than one order of magnitude speed-up for the spray solution when compounded.

Numerical setup

The FRESCO CFD simulation platform was employed to model the engine. The code implements an unstructured, parallel volume-of-fluid solver for the Navier-Stokes equations with automatic domain decomposition for boundary-fitted variable-topology meshes. More details about FRESCO are given in [11]. Turbulence is modeled using a generalized re-normalization group (GRNG) turbulence closure model that has been validated with engine flows, as well as for impinging and reacting jets [12]. Fuel injection and spray phenomena are modeled with a Lagrangian-Droplet/Eulerian-Fluid (LDEF) approach. Table 1 is a summary of the sub-models used to simulate turbulence and sprays for the current simulations.

Table 1. Computational model setup employed for the current study.

Phenomenon	**Sub-model**
Turbulence	Generalized re-normalization group (GRNG) k-ε [13, 12]
Injection	Blob model with dynamic blob allocation [5]
Spray angle	Reitz and Bracco [14]
Spray breakup	Hybrid KH-RT instability, Beale and Reitz [3]
Near-nozzle flow	Unsteady gas-jet model with implicit momentum coupling [5]
Drop drag	Analytical with Mach number effects [5]
Droplet collision	Deterministic impact; bounce, coalescence, reflexive separation, and stretching separation [6]; dynamic radius of influence [5]
Evaporation	1D discrete multi-component fuel [8]
Piston compressibility	Static, Perini et al. [15]

Parallel Collision Detection algorithm

Grid-independent collision model. Droplet collision physics are modelled using a deterministic impact parameter and extended collision outcomes **[6]**, and a mesh-independent radius-of-influence (ROI) based collision detection algorithm is employed **[5]**. The ROI model was developed to simplify the computational burden of collision detection estimates. In general, collisions may occur between any pair of computational parcels in the computational domain, yielding a computational cost $O(n_p^2)$. As the number of parcels can grow rapidly during injection (up to $n_p > 10^5$ if a multi-hole injector is employed), this cost quickly becomes too high. In the ROI model, a 'region of influence' concept is introduced to define a spatial region surrounding each droplet, where no collisions can take place outside of its reach. This region was originally defined as a fixed radius in **[16]**, and later extended with the tetrahedralization model in **[5]**. The radius of influence is assumed to be equivalent to the radius of a sphere with the same volume as that of the physical region containing all N_p drops in a computational parcel's drop distribution. As represented in two dimensions in Figure 1, all N_p drops in the parcel are assumed to be equally spaced, hence located such that they form an exact tetrahedralization. A user-defined liquid/gas volume fraction parameter is employed:

$$\boldsymbol{k_V = d_c / r_p}, \tag{1}$$

where d_c represents center-to-center distance of neighbor drops, and r_p the drop radius. This parameter was assumed to be constant and equal to $k_V=10$ as it provided a Pareto-optimal tradeoff between number of modeled collisions and computational cost **[5]**; though being an approximation, this approach does not rely on the local grid features and is thus fully mesh-independent. Furthermore, as recent research is enabling fast multi-phase liquid-vapor volume fraction calculations **[17]**, future efforts will be able to include more realistic estimates of the parcel's cloud volume based on the local thermodynamic conditions.

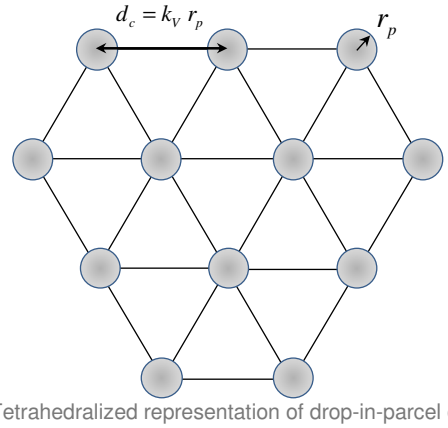

Figure 1. Tetrahedralized representation of drop-in-parcel distribution.

The tetrahedralized drop-in-parcel representation provides a fast, analytical formulation for the volume-of-influence (VOI_p) of the computational parcel, which is then equaled to a ROI as

$$ROI_p = \left[\frac{3}{4\pi} VOI_p\right]^{1/3}. \tag{2}$$

The availability of an ROI estimate for each parcel is needed to estimate a number of collisions between parcel pairs: according to O'Rourke [18], the probable number of drop-to-drop collisions in a parcel-to-parcel collision event, where the parcels are denoted as S (having smaller drops) and L (having larger drops), is:

$$n_{col} = \frac{\pi}{4} \frac{(ROI_S + ROI_L)^2}{VOI_S + VOI_L} |\boldsymbol{\theta}_S - \boldsymbol{\theta}_L|, \tag{3}$$

where $\boldsymbol{\theta}_S$ and $\boldsymbol{\theta}_L$ represent parcel velocity vectors. This formula is at the basis of the widely used, conventional 'same-cell' O'Rourke collision approach [18]: there, same-cell filtering was employed to avoid any expensive spatial search. However, this

algorithm has been widely known to introduce severe grid dependency and produce inaccurate results with non-physical spray structures [7].

To mitigate the $O(n_p^2)$ cost of estimating all potential collision pairs, we employ a two-staged parabola-function-based pre-filtering strategy to rule out all collisions which cannot be possible according to the local collision geometry. First, a pre-screening based on a simple geometric overlap condition is evaluated, as represented in Figure 2:

$$d\big(\boldsymbol{x}_L(t_n), \boldsymbol{x}_S(t_n)\big) \leq ROI_L + \Delta t\, \boldsymbol{\theta}_L, \tag{4}$$

i.e., all parcel pairs far enough from each other that their spheres cannot physically overlap during one timestep are immediately discarded. This initial search is made $O(n_p \log n_p)$ by employing a kd-tree data structure, while the rest of the search is again $O(n_p^2)$. In practice, the initial screening dramatically reduces the number of eligible collision pairs, making the second stage viable and fast. During the second stage, the instantaneous squared distance between two computational parcels during a time-step is estimated as a parabolic function:

$$d^2(\boldsymbol{x}_S(t_n), \boldsymbol{x}_L(t_n), t) = p_a t^2 + p_b t + p_c, \tag{5}$$

where

$$\begin{aligned} p_a &= \|\boldsymbol{\theta}_S - \boldsymbol{\theta}_L\|^2, \\ p_b &= 2\langle \boldsymbol{x}_S(t_n) - \boldsymbol{x}_L(t_n), \boldsymbol{\theta}_S - \boldsymbol{\theta}_L \rangle, \\ p_c &= \|\boldsymbol{x}_S(t_n) - \boldsymbol{x}_L(t_n)\|^2. \end{aligned} \tag{6}$$

By exploiting the analytical features of this parabolic function, it is possible to compute a minimum distance d_{min} and a minimum-distance-time, t_{min}, which computes when the two parcels, with these velocities and initial locations, will be closest to each other, and how much that distance is. Based on these metrics, collisions are deemed impossible for those parcel pairs where:

$$\begin{aligned} &p_b > 0, t_{min} < 0, && \rightarrow \text{moving far away from each other;} \\ &d_{min}^2 > (ROI_S + ROI_L)^2, && \rightarrow \text{never close enough to collide;} \\ &t_{min} \geq \Delta t, && \rightarrow \text{too far to collide during current step.} \end{aligned} \tag{7}$$

Once a restricted set of eligible collision pairs is eventually found, the individual collision probabilities and the corresponding collision outcomes are based on the local impact parameter b (Figure 3), collisional Weber number We_c, and drop diameter ratio Δ:

$$b = \sin\beta = \sqrt{1 - \frac{\langle \boldsymbol{\theta}_S - \boldsymbol{\theta}_L, \boldsymbol{x}_L(t_n) - \boldsymbol{x}_S(t_n) \rangle}{\|\boldsymbol{\theta}_S - \boldsymbol{\theta}_L\|^2 \cdot \|\boldsymbol{x}_L(t_n) - \boldsymbol{x}_S(t_n)\|^2}},$$

$$We_c = \frac{\rho_S r_S^3 + \rho_L r_L^3}{r_S^3 + r_L^3} \|\boldsymbol{\theta}_S - \boldsymbol{\theta}_L\|^2 \frac{r_S}{\sigma},$$

$$\Delta = r_S / r_L, \tag{8}$$

where ρ represents liquid density, σ the surface tension, and β the relative impact angle as represented in Figure 3.

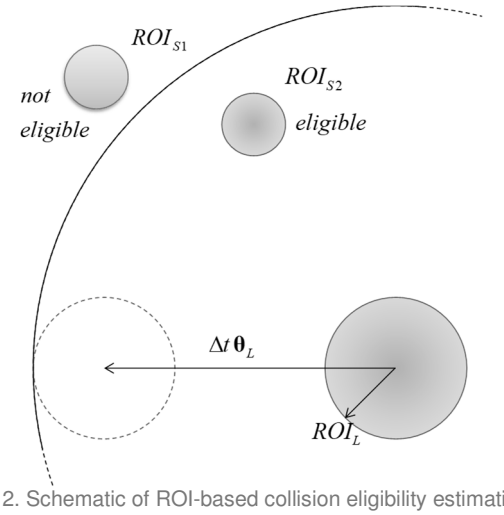

Figure 2. Schematic of ROI-based collision eligibility estimation.

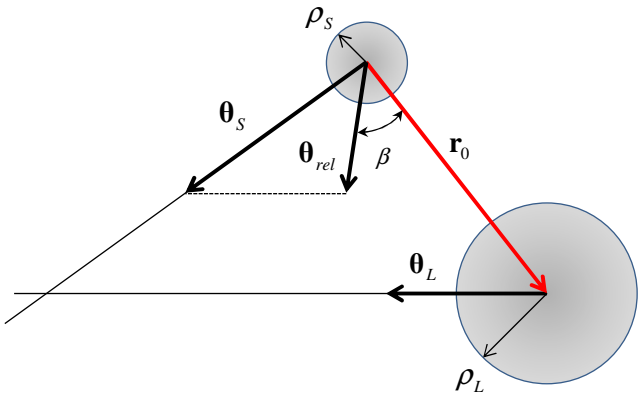

Figure 3. Deterministic collision impact parameter evaluation.

Parallel collision algorithm. Despite the sequential filtering operations, collision eligibility estimation can still take a relevant amount of CPU time in real-world scenarios, for example, in full engine simulations where multiple injector nozzles are present and the total number of drops is of the order of a few hundred thousand. Also, in parallel simulations, when collisions can happen across CPUs, the same algorithm with the global spray cloud data has to be solved identically on each CPU, making for a very inefficient and actually non-parallel usage of the multiple CPUs. Hence, a parallel algorithm can further speed-up collision eligibility detection.

Instead of evaluating binary collision eligibility for each collision pair, a *collision detection* adjacency matrix is built first. With no pre-processing of the eligible collision pairs according to the two-stage procedure outlined in the previous section, the matrix would be fully symmetric, which would render its storage as a matrix object unfeasible due to memory constraints. However, in practice, the filtering procedure leads to a highly sparse structure, because only parcels sufficiently close to each other can actually collide. Hence, a sparse, square, symmetric structure was employed. It has size $(n_p \times n_p)$, and its sparsity represents the graph of eligible collision pairs. The structure of this matrix is represented in Figure 4. In our

implementation, the object-oriented Compressed Sparse Row (CSR) class is employed [19].

Similar to the non-parallel procedure, the algorithm starts from a parallel gather of the whole spray cloud structure (parcel locations, velocities, and ROIs).

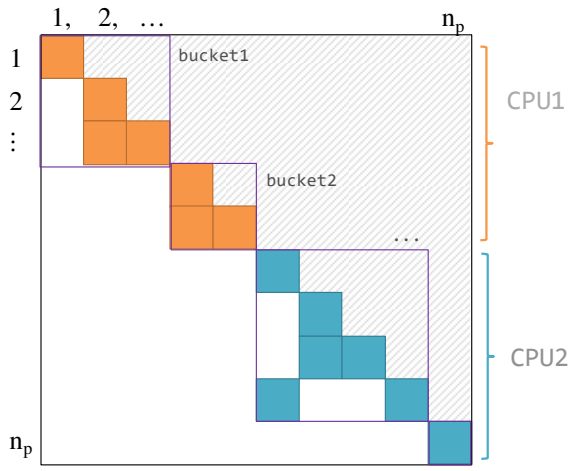

Figure 4. Parallel collision eligibility matrix structure.

Then:

1) A kd-tree clustering procedure is applied to the whole spray cloud according to the algorithm of [10], with a final bucket size equal to the maximum parcel ROI in the spray cloud. As a result, all parcels are clustered into *buckets*, sufficiently close to each other, achieving ROI-based grouping similar to the case represented in Figure 2. This accounts for a first filtering of the eligible collision pairs: parcels not inside the same bucket are not considered for collision. As this operation is fast and requires full knowledge of the tree structure, it is run identically on each CPU.

2) Second, the parallel part of the algorithm is run. To maximize parallel scalability, the workload for building the collision detection matrix is partitioned by rows. At the end of the procedure, each CPU must have a whole copy of the matrix. Hence, there is no need for complex CPU partitioning: each CPU will process approximately the same number of buckets based on an even distribution.

3) For all parcels in each bucket, the collision eligibility constraints of Equation 7 are evaluated, and eligible pairs are stored as nodes of the lower sparse adjacency graph.

4) After each CPU has processed all rows of its buckets, its horizontal band of the matrix is ready; a final global data gathering is the last operation needed to reconstruct the whole matrix on each CPU.

Parallel Vaporization Load-Balancing

In the FRESCO code, the spray solution is intrinsically parallel: the code's approach is to store the computational parcels on the CPU that owns the cells that contain them. From a coding standpoint, this is convenient, as the solution to the parcel equations and the spray sub-models always requires knowledge of the gas-phase thermodynamic and turbulence properties surrounding them, so, no parallel communications are involved when solving for Lagrangian-related terms. However, this approach carries over an intrinsically severe computational efficiency drawback: if the drop cloud is geometrically located in a few compact regions of the domain, then only a few CPUs will contain the majority of the parcels – those where most of the dense spray core is located – leading to poor load balancing.

Among the Lagrangian models, vaporization plays the most important role, especially in FRESCO, where complex physics with multicomponent fuels and an internal 1D drop discretization [8], or a full multiphase solver, are available.

The spray vaporization algorithm is not an "*embarrassingly" parallel* problem, i.e., a problem where the solution to each individual component is completely independent of any others. In fact, following the method of [2], vaporization of all spray drops is solved sequentially, as represented in Figure 5. After each parcel lying inside a cell is solved for its vaporization rate, the cell's gas-phase properties are sequentially updated before solving for the next parcel. This introduces dependency of the vaporization algorithm on the order the parcels are fed to the vaporization algorithm: if a parcel is being vaporized after several other ones, its vaporization rate would likely be slower than if it was vaporized first, since the gas-phase has already received much fuel vapor, and underwent significant cooling from the vaporization of the previous parcels. Hence, a parallel vaporization algorithm must take this concept into account. If each drop was solved for independently, as represented in Figure 5 (below), all drops in a same cell would be 'first', and a greater vaporization rate would be predicted. Also, the results from a parallel solution would be different than those from a single-CPU one.

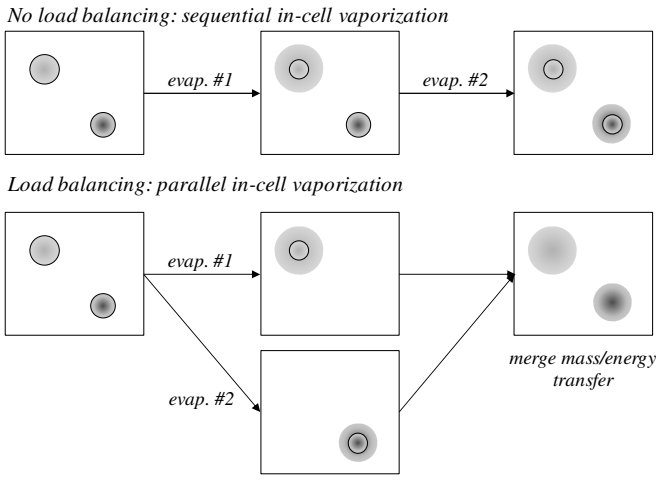

Figure 5. Schematic of sequential vs. parallel drop vaporization solution. (top) sequential: the vaporized amount of next drops is affected by already-vaporized fuel from the previous drops; (bottom)

parallel: each drop's vaporization rate responds to a 'scratch' underlying gas phase state.

To overcome this issue, we introduce the idea of 'buckets'. A bucket is a data structure which contains one cell's gas phase, and all drops geometrically within it. So, the vaporization solution within each bucket will be identical regardless of the CPU it is being solved upon. A bucket-based greedy algorithm for load balancing was hence developed to maximize parallel efficiency while not affecting the sequential parcel-in-cell solution constraint. To atychieve identical solution results between the serial and the parallel cases, the greedy algorithm starts by binning all parcels into buckets by cell ownership, as represented in Figure 6: each bucket contains all parcels in the same computational cell. Within each bucket, parcels are sorted for increasing drop radius, such that the smallest-drop distributions (which have the shortest timescales) will be solved for vaporization first. The target number of parcels per CPU is the algebraic average:

$$\overline{n_p} = \frac{1}{N_{cpu}} \sum_i^{N_{cpu}} n_{p,i}. \quad (9)$$

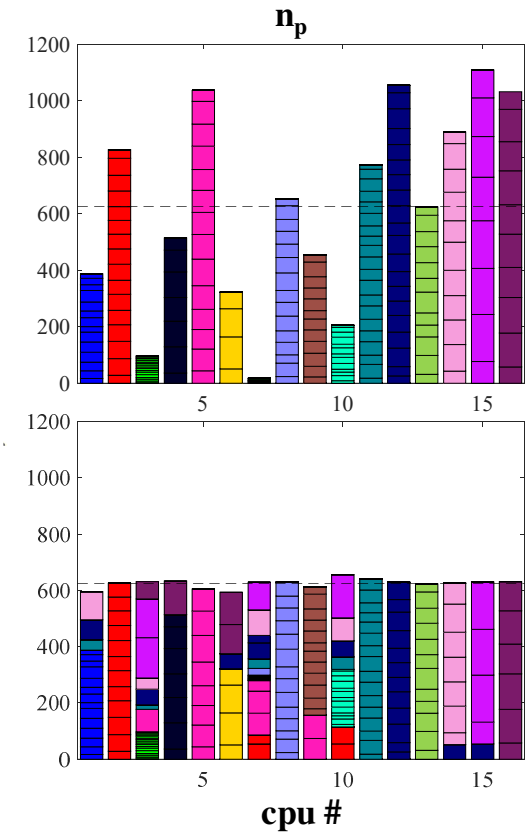

Figure 6. Greedy algorithm bucket redistribution for a randomly-sampled set with 10,000 parcels in 16 CPUs. (Top) before redistribution; (bottom) after redistribution. Dashed line: target size; colors: initial CPU owner.

At each greedy iteration, those CPUs still having more than the target number of parcels will re-distribute one or more buckets to other CPUs that have less than the target. As any greedy algorithms, the procedure stops when the best redistribution operation at any iteration count would 'make things worse', i.e., would cause the final number of parcels in any CPUs to be farther from the target than the current distribution.

Figure 6 represents the results of the greedy redistribution from a randomly sampled set of 10,000 parcels scattered across 16 CPUs; Figure 7 represents redistribution from a randomly sampled set of 150,000 parcels across 64 CPUs. In both figures, the original bucket CPU is represented by color. Note that each CPU is either a sender or a receiver based on its initial number of parcels; its state cannot be mixed and cannot change during the iterations. This constraint is actually needed for the vaporization parallel load balancing, as the memory range above the currently stored parcels is used as temporary storage for the parcels being received. If a CPU was both a sender and a receiver, memory would be overwritten causing information loss, while using a specific additional memory allocation would render the algorithm less efficient, both CPU- and memory-wise.

The greedy algorithm defines the optimal drop distribution such that all parcels-in-cell are kept together, and the number of drops solved on each CPU is as close as possible to being even. The implementation also includes data structures and methods to communicate these pieces of information across the CPUs:

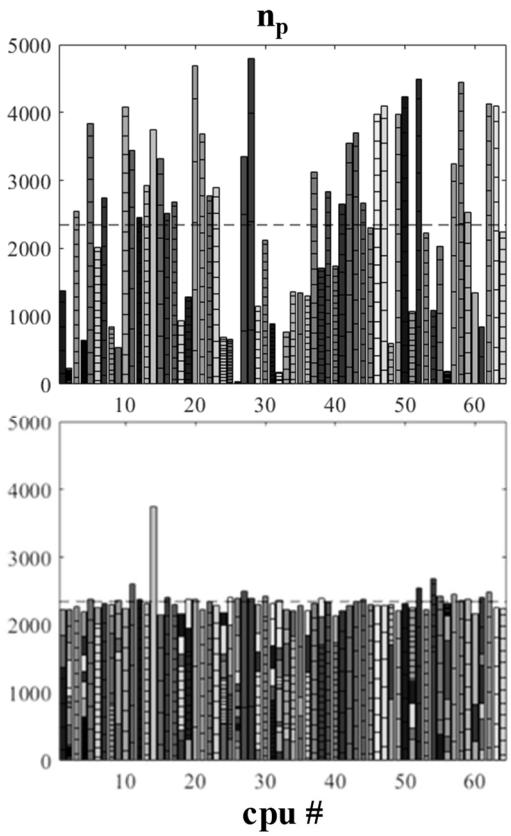

Figure 7. Greedy algorithm bucket redistribution for a randomly-sampled set with 150,000 parcels across 64 CPUs. (Top) before redistribution; (bottom) after redistribution. Dashed line: target size; colors: initial CPU owner. Each CPU is either a sender or receiver.

1) The optimal bucket distribution is computed by the greedy algorithm;
2) Sparse adjacency structures for data exchange of both parcel and gas-phase data across CPUs are created and temporary storage is allocated;
3) Data is transferred using MPI non-blocking communications;

4) Vaporization rates for all local parcels are computed;
5) Information is sent back to the actual CPU owners: updated drop size, internal distribution, composition and temperature; and species mass and energy exchanged with the gas phase;
6) Cell source terms are updated back on the actual CPU owners;
7) Temporary storage is deallocated.

It should be noted that the current algorithm minimizes the gas-phase cell status information having to be exchanged through the network, as each set of parcels being sent is clustered, so gas-phase information about each cell does not have to be duplicated. Finally, the mesh-independent form of FRESCO's Lagrangian source terms (Figure 8) is enforced: during the solution of each parcel-in-bucket, only the volume fraction of the vaporized amount, corresponding to its geometric parcel-cell overlap volume, is actually used to update the bucket cell's gas phase composition. The full Eulerian source term field is computed at the end, after the vaporization rates from all parcels are available.

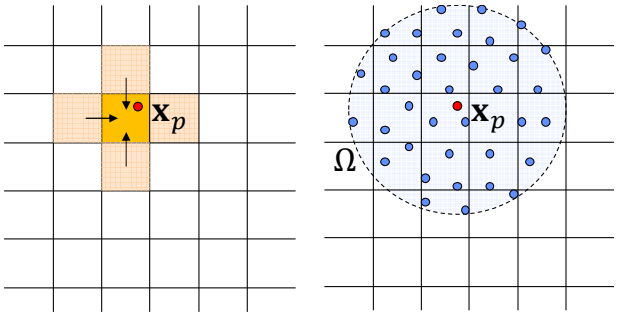

Figure 8. Comparison of (left) KIVA-like [2] vs. FRESCO's mesh independent Lagrangian source term calculation approaches.

Results and discussion

Two Diesel engine simulation test cases were run to assess the computational efficiency of both collision and vaporization load balancing methods. The first test case features a conventional Diesel combustion strategy, featuring relatively short-lived liquid spray clouds close to TDC, and uses a full engine mesh with 7 nozzle holes, hence with a relatively well distributed spatial occupation of the spray jets. The second one employs a sector mesh, and features a single-pulse early diesel RCCI injection, which has longer lifetime and worse usage of the cylinder sector volume. For both of them, 24-case simulation sweeps were run to systematically analyze the behavior of both submodels and the impact of the number of computational parcels in the spray cloud on them:

- *Collision detection method* - 3 methods were used:
  - O'Rourke's same-cell collision detection method ("ORK") [18];
  - Parabola-based pre-filtering method, operated on the full spray cloud ("PAR") [5];
  - Parallel kd-tree based cloud clustering method, followed by parabola-based pre-filtering ("KDT") – current work;
- *Vaporization solution method*:
  - With no load balancing;
  - With greedy load balancing – current work;
- *Number of injected parcels:*
  - Variable, according to full blob model [5]
  - Fixed, Np = 2000, 5000, 10000 per nozzle

Two pulse Diesel spray injection case (CDC9). The Sandia optical diesel engine platform was used as the first test case. A full engine geometry model was used, which includes the intake and exhaust ports and runners up to the surge tanks in the optical facility. The optical piston featured a stepped-lip bowl design (See [21, 22]). Engine data is summarized in Table 2 [23]. The model uses an unstructured, body-fitted hexahedral mesh with 724,000 cells at bottom dead center. The mesh is depicted in Figure 9. Details of the engine's 7-hole injector are given in Table 2. Spray targeting in the simulation matches the spray targeting used in corresponding engine experiments: both spray targeting and measured injection rate profiles are available on the ECN website [25]. The engine operating point represents a part-load (9 bar IMEP), conventional diesel combustion strategy (CDC9) with a split pilot-main injection strategy, with the main pulse's SOI approximately at TDC. Injection takes place in a non-reacting environment with 100% N_2, as the experimental results have been evaluated using fuel tracer planar laser-induced fluorescence (PLIF) images. All simulations for this case were performed on the same node equipped with 48 CPUs, from -20 deg aTDC to +40 deg aTDC.

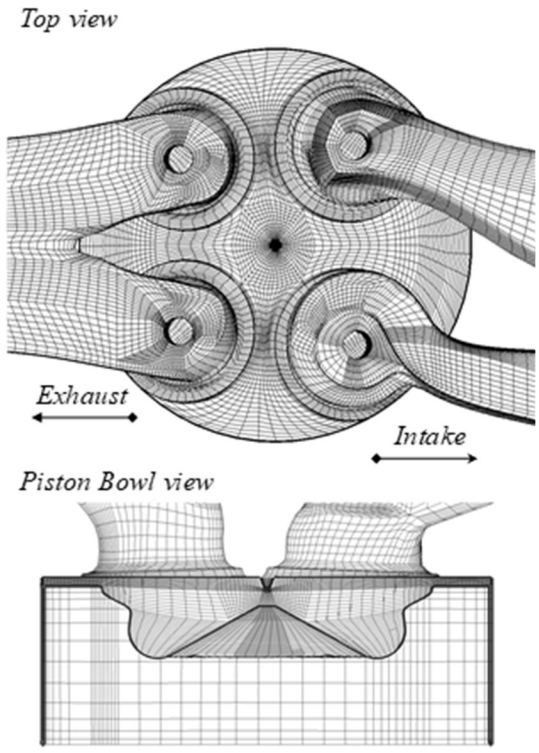

Figure 9. Cylinder detail of the full-engine CFD mesh used for the engine spray calculation.

Table 2: Engine and fuel injector geometry data

Bore	82.0 mm
Stroke	90.4 mm
Connecting rod length	166.7 mm
Squish height	1.36 mm
Geometric compression ratio	15.8 : 1
Injector nozzle holes x diameter	7 x 139 µm
Nozzle hole conicity (k_s)	1.5
Injector opening angle	149°

RCCI simulation. The second validation test case featured a different configuration, with a sector mesh model (Figure 10) and an early Diesel injection pulse. The testcase mimicked an RCCI-like injection pulse in the same engine, equipped with an RCCI piston [26], and modeled using a sector mesh approach. The choice of an RCCI operating point was motivated by the need to assess computational efficiency at the upper end of the liquid fuel's lifetime range: this leads to more parcels and more complex collision and vaporization scenarios than in near-TDC injections. The sector mesh has 74,000 cells at bottom dead center. A single injection pulse, from -60 to -53.35 deg aTDC, was modeled using the CRI2.2 rate of injection model of [27]. A single-component fuel model using n-dodecane as the physical surrogate for vaporization was employed. Simulations for each tested setup were run on the same single node with 20 CPUs, with total simulation times ranging from 1h17' to 1h50'.

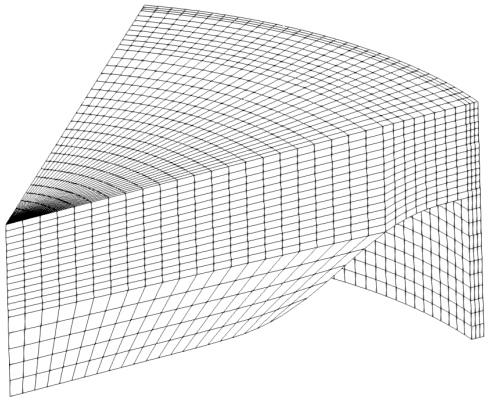

Figure 10. View of the computational mesh employed for the RCCI calculation at -20 degrees aTDC.

Collision model impact

CDC9 simulation. The number of instantaneous parcels alive during a conventional diesel combustion case is relatively low, as the time histories of parcel count show in Figure 11. All curves exhibit two separate events with a first, low peak slightly earlier than TDC due to the pilot injection, and a bi-modal distribution during the main injection with two peaks: one shortly after SOI_{main}, and one close to EOI_{main}. In all cases, the parcel count quickly decays after the end of injection. Choice of the collision method significantly affected the parcel count. The O'Rourke (ORK) model consistently exhibited lower parcel numbers than the parabola (PAR) and kd-tree (KDT) methods, which instead showed very similar parcel histories. This suggests that the kd-tree method does not significantly change the collision detection matrix. With KDT, collisions can only take place with other parcels from the same cluster; hence, the kd-tree based parallel clustering produced accurate partitions of the spray cloud. All models showed consistent behavior with different numbers of injected parcels, with greater parcel numbers when more parcels were injected.

In the O'Rourke model, a slightly lower number of instantaneous in-cylinder parcels was observed. As shown in Figure 12, this did not affect the spray structure significantly, as no meaningful differences could be observed in the main jets with any of the collision detection methods. However, as a drawback, that would affect CPU time dramatically. As all same-cell parcels are forced to be eligible for collisions regardless of the collision geometry, the number of simulated collisions with the O'Rourke model is significantly higher than with the mesh-independent models, such as reported in Figure 13. With 10,000 injected parcels, the O'Rourke method simulated slightly more than 840,000 collisions, as opposed to 202,000 for the parabola method, and just 52,000 for the kd-tree based method. It should be noted that the parabola method provides an exact filtering of the eligible collisions based on the impact geometry, even outside of the cell's reach, so it would be expected to predict a larger amount of collisions. However, the same-cell method does not use the impact geometry as a filtering step; hence, it will simulate collisions also between slightly diverging drops, such as those found in conically-shaped spray jets.

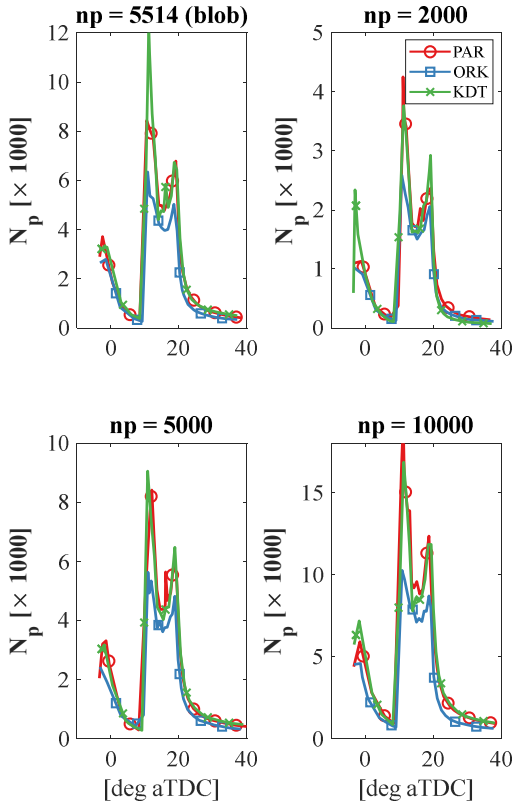

Figure 11. Instantaneous in-cylinder number of spray parcels vs. crank angle for the RCCI simulation case. Total parcels injected: clockwise from the top: blob model (5514), 2000, 10000, 5000.

This counterintuitive fact suggested that the O'Rourke method unnecessarily simulated a large number of highly improbable collisions. This negatively affected the CPU time performance of the method, which, despite being the simplest one, suffered from large CPU times due to the large number of computed collisions (red bars in Figure 14). In Figure 14, CPU times for the parallel collision calculations are split between evaluation of the collision adjacency matrix, and actual collision computation involving all eligible pairs.

The parabola method exhibited similar CPU times as O'Rourke, even if most of the time was actually spent detecting eligible collisions, instead of performing them. The kd-tree method, on the other hand, dramatically reduced the CPU times for collision detection due to the parallel clustering step, and the time for collision simulation due to its additional filtering of the least likely collisions. Overall, the kd-tree method achieved speed-ups from 1.74 times with as few as 2000 injected parcels, to 18.6 times with 10000 injected parcels. The full blob model (5514 injected parcels) had a speed-up of 3.25 times.

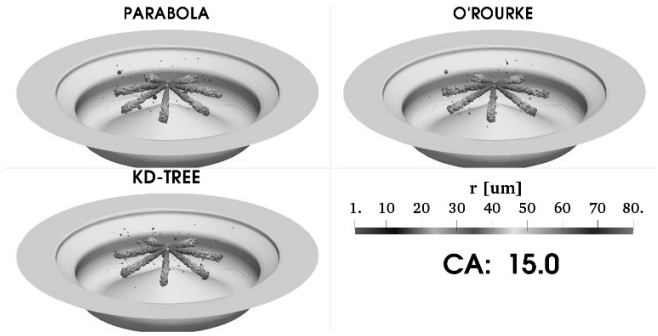

Figure 12. View of predicted spray structure for different collision detection methods during fuel injection (main injection at CA = 15.0 deg aTDC) for the CDC9 case.

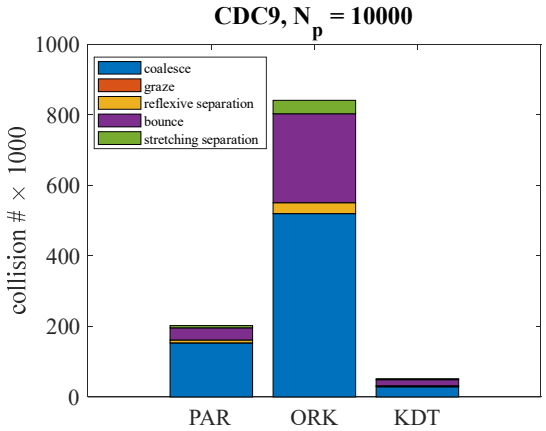

Figure 13. Number of collisions simulated in the CDC9 cases with 10,000 injected parcels. PAR = Parabola-based detection; ORK = O'Rourke same-cell detection; KDT=kd-tree enhanced detection.

RCCI simulation. With a more long-lived spray injection, the choice of the collision detection affected the instantaneous number of spray parcels in the cloud more noticeably, as represented in Figure 15. In all cases, the number of parcels reached a peak (due to both injection and breakup) shortly after SOI, and then decayed until all liquid fuel had vaporized, approximately after -20 degrees aTDC. Again, all models exhibited consistent behavior with different numbers of injected parcels, and the most significant differences were observed with the detection model. O'Rourke's same-cell model always predicted much lower numbers of parcels than the parabola and kd-tree clustering models. Again, parabola and kd-tree models predicted very similar numbers of parcels, which suggested that the final bucket size of the kd-tree method was large enough that the algorithm only loosely affected collision eligibility, while guaranteeing good parallel scalability.

This was confirmed as shown by predicted spray structure in Figures 16 and 17: the parabola and kd-tree methods yielded nearly identical spray structures, while the O'Rourke collision method exhibited a noticeably thicker spray core region, with larger drops present. In particular, the O'Rourke model predicted smaller drops than the others early after the start of injection (Figure 17, top), while drop size significantly increased later on, in contrast with intuitive atomization-driven processes, which lead to the formation of finer drops as the jet penetrates into the gaseous phase.

It should also be noted that collision types were also affected by the O'Rourke model. Since the RCCI simulations were run on a relatively coarse mesh with ~75,000 cells at BDC, the number of

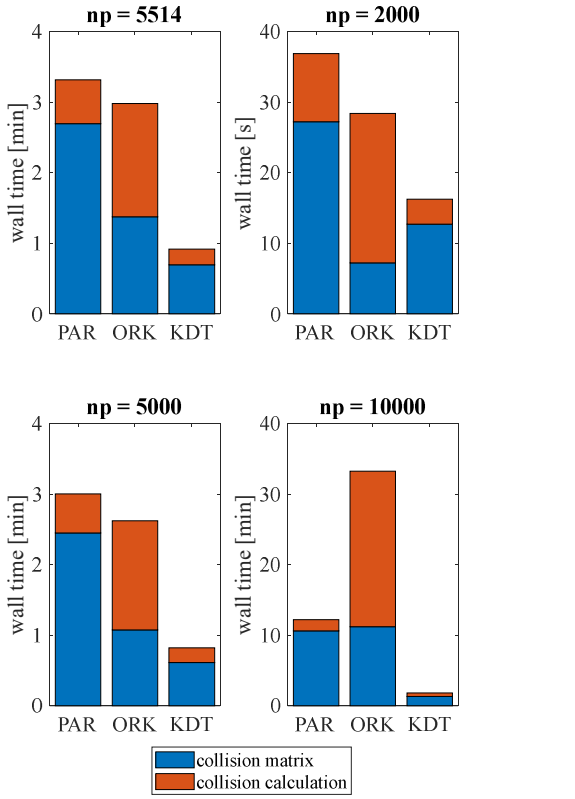

Figure 14. Collision wall times for CDC9 cases. PAR = Parabola-based detection; ORK = O'Rourke same-cell detection; KDT=kd-tree enhanced detection.

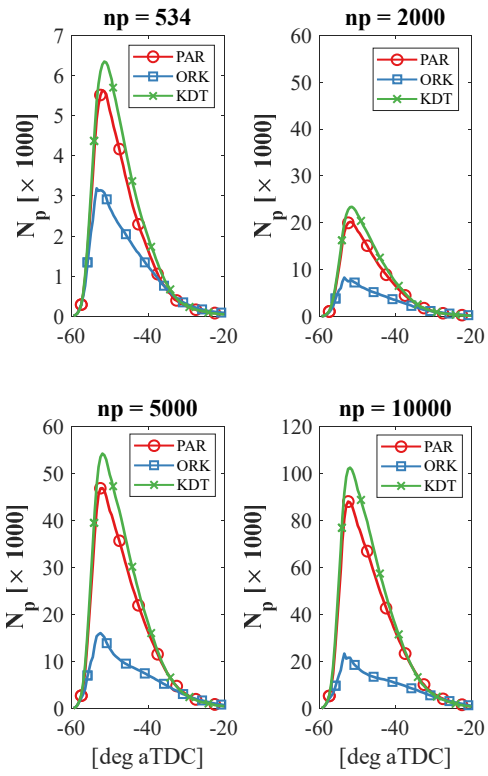

Figure 15. Instantaneous in-cylinder number of spray parcels vs. crank angle for the RCCI simulation case. Total parcels injected: clockwise from the top: blob model (534), 2000, 10000, 5000.

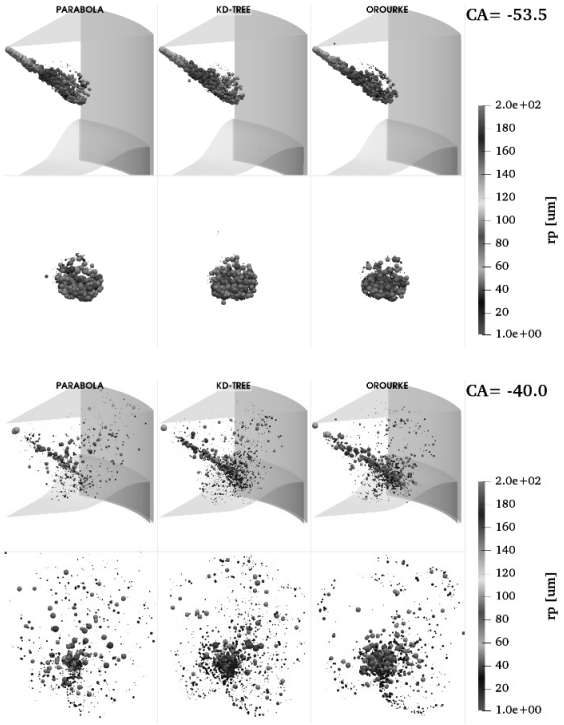

Figure 16. View of predicted spray structure for different collision detection methods during fuel injection for the RCCI case. In each image, the top row represents the in-cylinder sector; the bottom row represents the spray footprint as seen from below the injection axis.

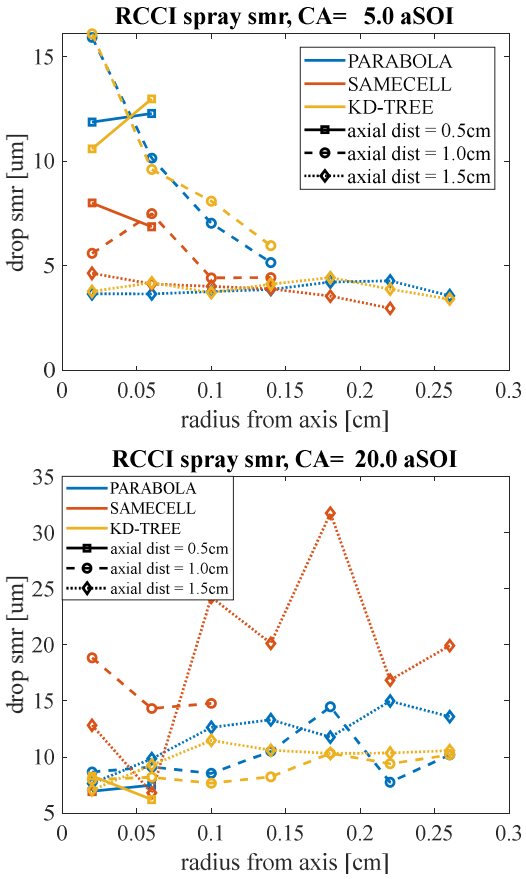

Figure 17. Predicted liquid Sauter Mean Radius (SMR) distributions for the RCCI case vs. radial distance from the injection axis.

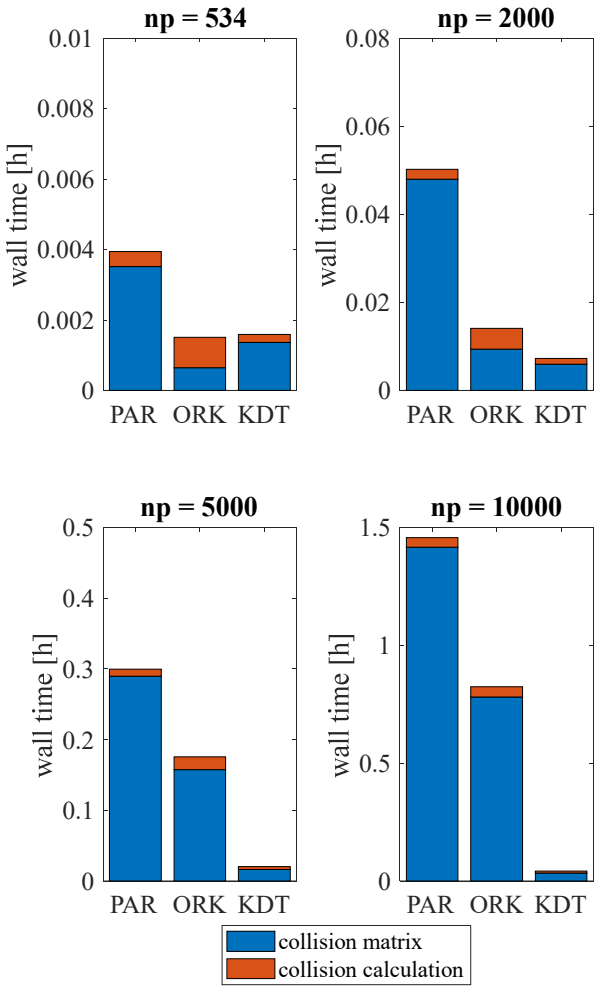

Figure 18. Collision wall times PAR = Parabola-based detection; ORK = O'Rourke same-cell detection; KDT=kd-tree enhanced detection.

collisions was again pretty large, as shown in Figure 19. Also: bouncing and coalescing collisions were the far dominant collision types for all models, even though this phenomenon is more evident with the mesh-independent methods. This is to be attributed to the deterministic impact parameter selection: while possible collision outcomes have to cover the whole Weber/impact parameter space (including, for example, head-to-head collisions), in a high-pressure fuel injection event we have a slightly diverging spray, so the domain range covered by the actual collisions appears to be narrower.

With all collision detection models, the dominating collision types were coalescence and bouncing, which are respectively characterized by low collision Weber numbers (similar drop velocities or very small drop sizes involved) and high impact parameter values (head-to-head type of impact). But, the total number of simulated collisions varied significantly. With O'Rourke's model, the number of simulated collisions was almost double as with the Parabola model, which again suggests for the usefulness of introducing a more reliable and mesh independent collision detection method.

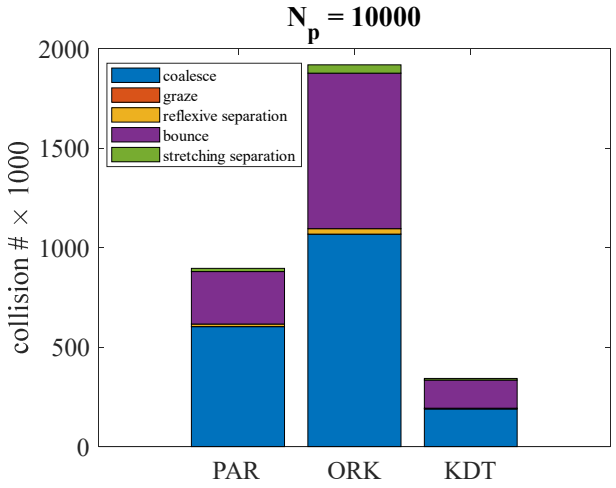

Figure 19. Number of collisions simulated in the RCCI cases with 10,000 injected parcels. PAR = Parabola-based detection; ORK = O'Rourke same-cell detection; KDT=kd-tree enhanced detection.

Since RCCI cases involve long-living liquid drops, the barriers among clusters imposed by the parallelized kd-tree method appeared to effectively filter out collision pairs, which, while theoretically eligible based on impact geometry only, involve drops that are too far off from each other, such that these pairs may already have had a much lower collision probability than all others in the same cluster.

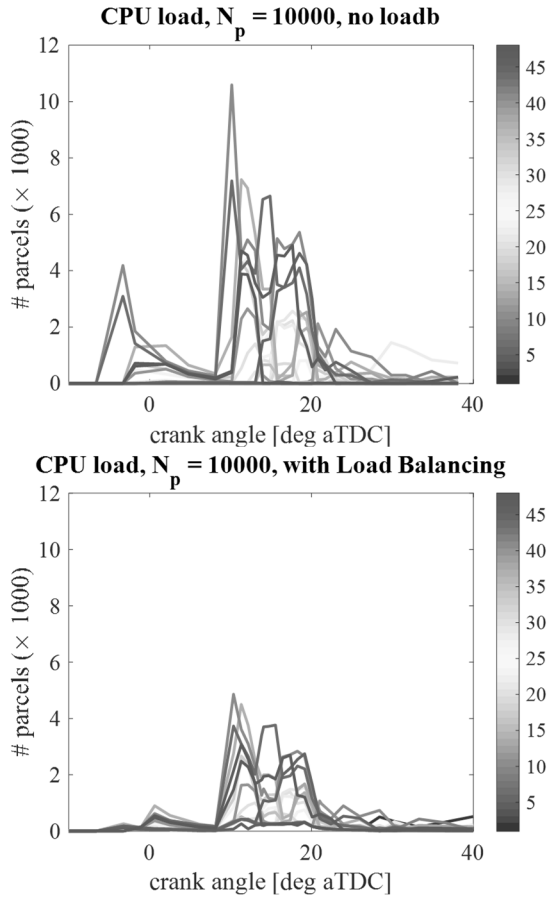

Figure 20. CPU load (number of vaporizing parcels) with (above) and

without (below) vaporization load balancing, for the RCCI case with 10000 injected parcels. Line color represents the CPU identifier.

As a result, simulation CPU times for collisions significantly benefited from the adoption of the kd-tree clustering method, such as reported in Figure 18. The benefits were even stronger than with the full-cylinder CDC9 case. As Figure 18 shows, the most computationally demanding task was collision detection, i.e., building the collision eligibility graph, due to the large number of drops surviving for longer times inside the cylinder.

In the RCCI cases, regardless of the number of injected parcels, the parabola-based method was always the most computationally demanding case, which is justified by its additional geometry-driven deterministic parameters whose natural scaling is $O(n_p^2)$. The same-cell collision method, while simple, still needed some CPU time to evaluate all particle-in-cell buckets, with growing effort at larger numbers of parcels. The kd-tree based method, instead, provided excellent performance in all cases: from being at least as effective as the same-cell method with the lowest number of parcels; and being by far the fastest collision detection method when at least 5,000 parcels had been injected.

Speed-ups of about 33.9 times versus the parabola method, and 19.2 times versus the same-cell method, were observed for the 10,000 parcels case.

Vaporization load balancing impact

CDC9 simulation. The impact of enabling the greedy algorithm for load balancing for the vaporization calculation is reported in Figure 20 first. Here, the vaporization workload for each CPU of the N_p=*10000* case is shown, either with or without load balancing enabled. When no load balancing is present, the workload distribution among processors appeared to be much worse, with 1) several CPUs exhibiting flat-zero curves, i.e., just sitting idle until all others have completed vaporizing their parcels; and 2) the number of parcels count exhibiting a huge standard deviation within the CPU set, with a peak value of ~11,000 parcels handled by one single CPU at ~10 degrees aTDC. When load balancing is present, 1) a non-zero minimum workload was always present for all CPUs; and 2) the peak CPU count was much lower on average, and adding up to ~4,850 parcels during the same peak count region.

The CPU-based standard deviation of the instantaneous number of parcels handled for vaporization is shown in Figure 21, providing a quantitative outlook on the effectiveness of vaporization load balancing. Across all cases, 2121load balancing approximately halved CPU dispersion, as the standard deviation was reduced by approximately a factor of 2. The resulting standard deviation was very close to the mean value, equal to the total number of vaporizing parcels divided by the number of CPUs.

Peak effectiveness was obtained during the pilot injection, before TDC: here, load balancing brought the standard deviation of the CPU load close to zero for all cases, meaning that the vaporization task was almost perfectly balanced across CPUs. No differences could be observed in simulation output by enabling vaporization load balancing. For example, Figure 22 highlights identical predictions of instantaneous in-cylinder liquid and vaporized fuel mass.

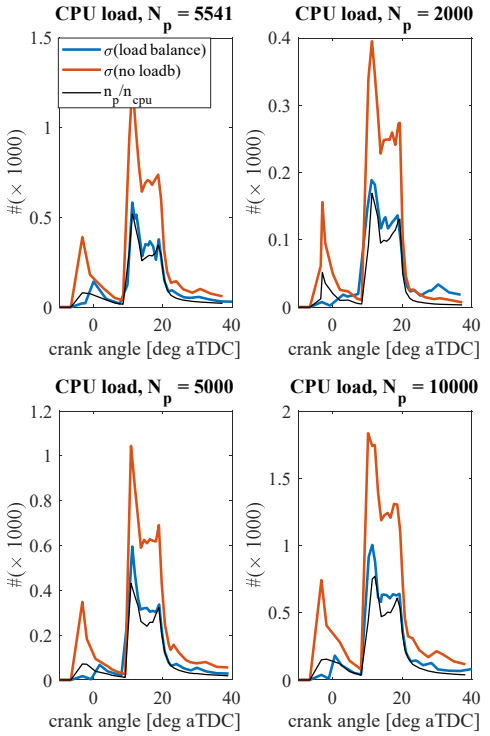

Figure 21. Standard deviation of the instantaneous CPU load for vaporization in the CDC9 cases, expressed as number of parcels. (red) no load balancing; (blue) with vaporization load balancing; (black) mean number of vaporizing parcels per CPU.

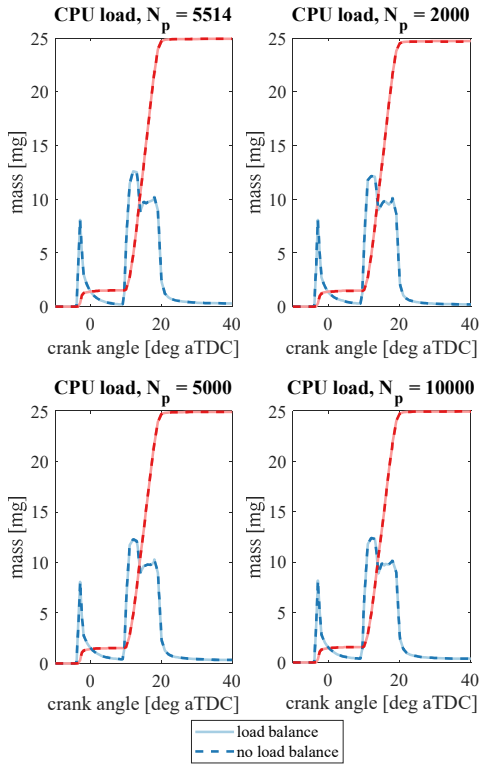

Figure 22. Predicted in-cylinder fuel vapor mass (red) and

instantaneous liquid mass (blue), for several numbers of injected parcel cases, with (solid lines) and without (dashed lines) vaporization load balancing.

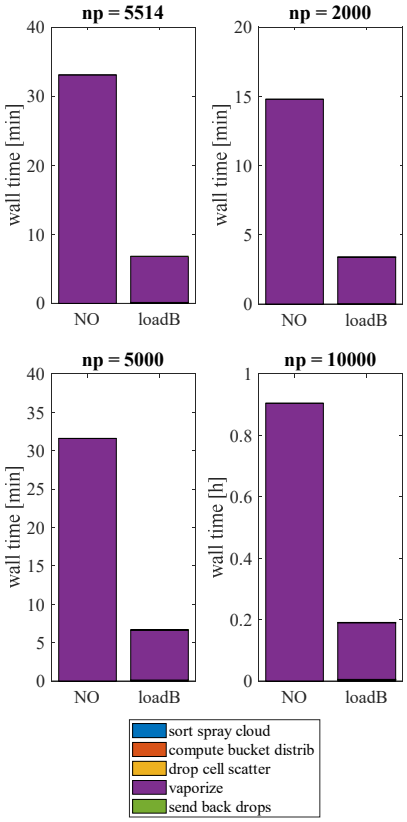

Figure 23. CPU times during vaporization computation of the CDC9 cases, with or without load balancing.

The results also highlighted that, different from the collision algorithm, whose load balancing impact was greater for spray clouds with more parcels, the dependency of vaporization speed-up with the number of injected parcels is much less. Figure 23 reports cumulative CPU times for all steps of the vaporization calculation, with or without load balancing. First of all, all MPI-related tasks of the load balancing term, such as sending/receiving drop, as well computing the optimal partitioning with the greedy algorithm, used almost no additional overhead; global speed-ups of 4.88× with 5514 parcels, 4.36× with 2000, 4.74× with 5000, 4.78× with 10000, exhibiting a slightly monotonic increase with more injected parcels were seen.

RCCI simulation. Similarly, for the RCCI simulations, the standard deviation of CPU workload is shown in Figure 24. In this case, as there is one single injection pulse, the standard deviation reaches a maximum shortly after SOI, then starts decreasing. Again, the effect of the load balancing algorithm is such as to approximately halve the standard deviation of CPU load. It should be noted that in this case, the shape of the standard deviation curve is apparently meandering. This is caused by the mesh topology changes successively introduced during the compression stroke, as layers of squish cells are removed to keep mesh resolution approximately constant. Every time this happens, the CPU decomposition of the finite volume domain is updated, which may lead regions of the cylinder with spray parcels to suddenly change CPU owner.

The wall time results reported in Figure 25 also highlighted that it is much less. Speed-ups were of 5.23× with 534 parcels, 5.96× with 2000, 5.70× with 5000, 5.67× with 10000. These speed-ups were slightly larger than those observed with the CDC9 cases, despite the lower number of CPUs involved (20 against 48). Also, in this case, over-head due to parallel communication operations was negligible in the load balancing case.

As highlighted in Figure 26, the cumulative impact of spray load balancing is directly transferred to the total simulation times, as all other operators (mainly chemistry, diffusion, advection terms) are not affected by the spray load balancing method.

Concluding remarks

In this work, we presented two methods to achieve efficient load balancing of parallel spray calculations in multi-dimensional engine simulations. The first addressed mesh-independent collision estimation algorithms, by employing a parallel kd-tree based clustering algorithm to classify eligible collision pairs, independent of the underlying mesh. The second achieved vaporization model load balancing by using a greedy bucket algorithm: buckets represents containers with cells and drops; these are assigned an owner CPU based on a greedy procedure, which optimizes the parcel-weighted CPU load.

Tests against sector and full meshes, with CDC or RCCI operating spray strategies, led to the following conclusions:

- Identical results were obtained with or without the vaporization load balancing algorithm;

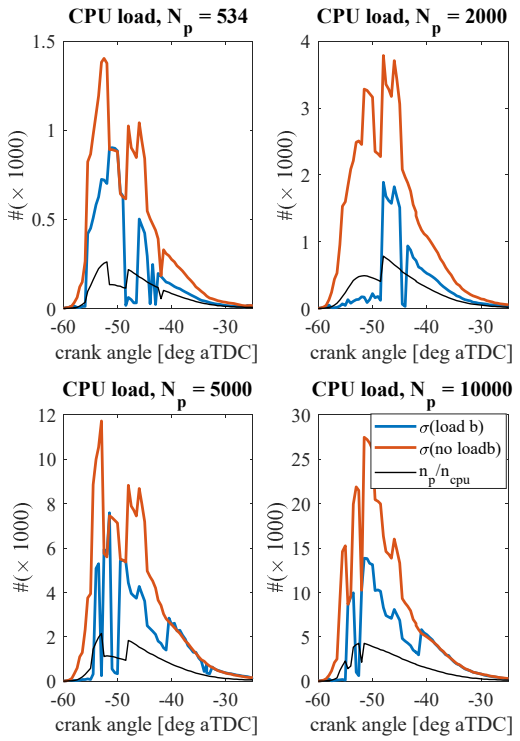

Figure 24. Standard deviation of the instantaneous CPU load for vaporization in the RCCI case, expressed as number of parcels. (red) no load balancing; (blue) with vaporization load balancing.

- The vaporization load balancing greedy algorithm yielded speed-ups between 4-6 times both on 20 and 48 CPUs, and on different spray configurations, suggesting that the amount of speed-up achievable depends on the instantaneous spray parcel distribution in the finite volume domain;
- Mesh independent collision algorithms (with or without parallel partitioning) yielded nearly identical spray structure results, but a higher number of parcels than with O'Rourke's same-cell method; this suggested that the same-cell method, while discarding potentially likely collisions outside of the same cell, also counterintuitively overestimates the number of collisions that can take place among parcels of the same cell;
- Most CPU time in both mesh-independent collision algorithms was spent evaluating the collision detection graph;
- Usage of the parallel kd-tree clustering method allowed speed-ups of more than one order of magnitude, up to 19.2× versus O'Rourke's method, and up to 33.8× versus the full mesh-independent method.

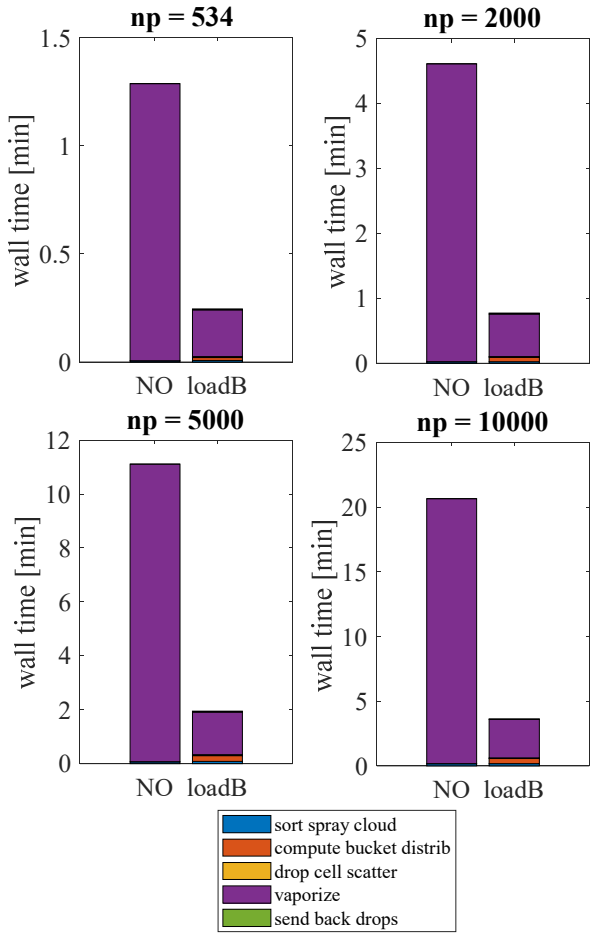

Figure 25. CPU times during vaporization computation of the RCCI case, with or without load balancing.

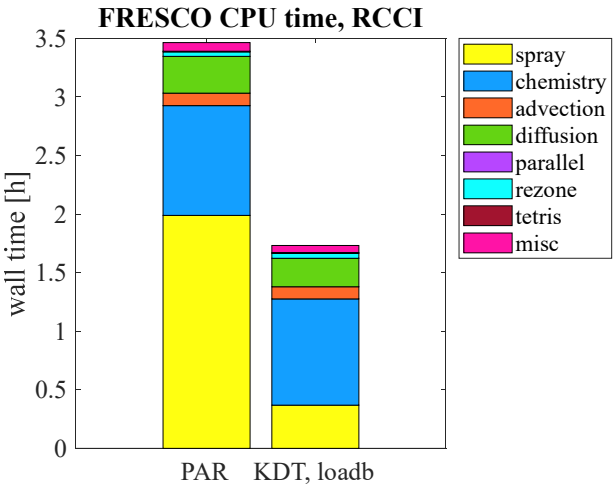

Figure 26. Cumulative CPU time impact of the mesh-independent load balancing methods on the RCCI case: (left) Parabola collisions, no vaporization load balancing; (right) kd-tree collisions; vaporization load balancing.

Future work will be devoted to extending the application of the greedy vaporization load balancing algorithm to more sub-

models, such as the multiphase solver in the Equilibrium-Phase (EP) framework [28].

References

[1] J. K. Dukowicz, "A particle-fluid numerical model for liquid sprays," *Journal of Computational Physics,* vol. 35, no. 1, pp. 229-253, 1980.

[2] D. J. Torres and M. F. Trujillo, "KIVA-4: An unstructured ALE code for compressible gas flow with sprays," *Journal of Computational Physics,* vol. 219, no. 2, pp. 943-975, 2006.

[3] J. C. Beale and R. D. Reitz, "Modeling Spray Atomization with the Kelvin-Helmholtz/Reyleigh-Taylor Hybrid Model," *Atomization and Sprays,* vol. 19, no. 7, pp. 623-650, 1999.

[4] N. Abani and R. R. Reitz, "Unsteady turbulent round jets and vortex motion," *Physics of Fluids,* vol. 19, no. 1, p. 125102, 2007.

[5] F. Perini and R. D. Reitz, "Improved atomization, collision and sub-grid scale momentum coupling models for transient vaporizing engine sprays," *International Journal of Multiphase Flows,* vol. 79, pp. 107-123, 2016.

[6] A. Munnannur and R. D. Reitz, "Comprehensive Collision Model for Multidimensional Engine Spray Computations," *Atomization and Sprays,* vol. 9, no. 6, pp. 597-619, 2009.

[7] D. P. Schmidt and C. J. Rutland, "A New Droplet Collision Algorithm," *Journal of Computational Physics,* vol. 164, no. 1, pp. 62-80, 2000.

[8] D. J. Torres, P. J. O'Rourke and M. F. Trujillo, "A Discrete Multicomponent Fuel Model," *Atomization and Sprays,* vol. 13, no. 2&3, p. 42, 2003.

[9] Y. Ra and R. D. Reitz, "A vaporization model for discrete multi-component fuel sprays," *International Journal of Multiphase Flow,* vol. 35, no. 2, pp. 101-117, 2009.

[10] F. Perini, "High-dimensional, unsupervised cell clustering for computationally efficient engine simulations with detailed combustion chemistry," *Fuel,* vol. 106, pp. 344-356, 2012.

[11] F. Perini and R. D. Reitz, "FRESCO - an object-oriented, parallel platform for internal combustion engine simulations," in *28th International Multidimensional Engine Modeling User's Group Meeting at the SAE Congress*, Detroit, 2018.

[12] F. Perini, S. Busch, K. Zha and R. D. Reitz, "Comparison of Linear, Non-linear and Generalized RNG-based k-epsilon models for turbulent diesel engine flows," in *SAE Technical Paper 2017-01-0561*, Detroit, MI, 2017.

[13] B.-L. Wang, P. C. Miles, R. D. Reitz and Z. Han, "Assessment of RNG Turbulence Modeling and the Development of a Generalized RNG Closure Model," in *SAE Technical Paper 2011-01-0829*, Detroit, MI, 2011.

[14] R. D. Reitz and F. V. Bracco, "On the Dependence of Spray Angle and Other Spray Parameters on Nozzle Design and Operating Conditions," in *SAE Technical Paper 790494*, 1979.

[15] F. Perini, A. B. Dempsey, R. D. Reitz, D. Sahoo, B. Petersen and P. C. Miles, "A Computational Investigation of the Effects of Swirl Ratio and Injection Pressure on Mixture Preparation and Wall Heat Transfer in a Light-Duty Diesel Engine," in *SAE Technical Paper 2013-01-1105*, Detroit, MI, 2013.

[16] A. Munnannur and R. D. Reitz, "A new predictive model for fragmenting and non-fragmenting binary droplet collisions," *International Journal of Multiphase Flow,* vol. 33, pp. 873-896, 2007.

[17] F. Perini, S. Busch and R. D. Reitz, "An Investigation of Real-Gas and Multiphase Effects on Multicomponent Diesel Sprays," in *SAE Paper Offer 20PFL-0263, under review*, Detroit, MI, 2020.

[18] P. J. O'Rourke, "Collective drop effects on vaporizing liquid sprays," Princeton University Ph.D. Dissertation, Princeton, NJ, 1981.

[19] F. Perini, E. Galligani and R. D. Reitz, "An analytical Jacobian approach to sparse reaction kinetics for computationally efficient combustion modelling with large reaction mechanisms," *Energy&Fuels,* vol. 26, no. 8, pp. 4804-4822, 2012.

[20] F. Perini, S. Busch and R. D. Reitz, "Investigation of post-injection strategies for diesel engine Catalyst Heating Operation using a vapor-liquid-equilibrium-based spray model," *The Journal of Supercritical Fluids,* vol. 167, p. 105042, 2020.

[21] S. Busch, K. Zha, E. Kurtz, A. Warey and R. C. Peterson, "Experimental and Numerical Studies of Bowl Geometry Impacts on Thermal Efficiency in a Light-Duty Diesel Engine," in *SAE Technical Paper 2018-01-0228*, 2018.

[22] S. Busch, K. Zha, F. Perini, R. D. Reitz, E. Kurtz, A. Warey and R. Peterson, "Bowl Geometry Effects on Turbulent Flow Structure in a Direct Injection Diesel Engine," in *SAE Technical Paper 2018-01-1794*, Heidelberg, Germany, 2018.

[23] "Small-Bore Diesel Engine," Sandia Mational Laboratories, 11 8 2017. [Online]. Available: https://ecn.sandia.gov/engines/engine-facilities/small-bore-diesel-engine/. [Accessed 2018].

[24] F. Perini, R. D. Reitz and P. C. Miles, "A comprehensive modeling study of in-cylinder fluid flows in a high-swirl, light-duty optical diesel engine," *Computers and Fluids,* vol. 105, pp. 113-124, 2014.

[25] S. Busch and P. C. Miles, "Parametric Study of Injection Rates With Solenoid Injectors in an Injection Quantity and Rate Measuring Device," in *ASME Paper No. ICEF2014-5583*, Columbus, Indiana, USA, 2015.

[26] R. Hanson, S. Curran, R. Wagner, S. Kokjohn, D. Splitter and R. Reitz, "Piston Bowl Optimization for RCCI Combustion in a Light-Duty Multi-Cylinder Engine," *SAE International Journal of Engines,* vol. 5, no. 2, pp. 286-299, 2012.

[27] F. Perini, S. Busch and R. D. Reitz, "A phenomenological rate of injection model for predicting fuel injection with application to mixture formation in light-duty diesel engines," *Proc. IMechE Part D: Journal of Automobile Engineering,* vol. 0, no. 0, p. 29, submitted.

[28] Z. Yue and R. D. Reitz, "An equilibrium phase spray model for high-pressure fuel injection and engine combustion simulations," *International Journal of Engine Research,* vol. 20, no. 2, pp. 203-215, 2019.

Contact Information

Federico Perini
Federico.Perini@w-erc.com

Acknowledgments

This work was performed under Sandia Subcontract 1890589, 1, sponsored by the United States Department of Energy, Office of Vehicle Technologies, with program managers Gupreet Singh and Michael Weismiller.

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. The views expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the United States Government.