International Multidimensional Engine Modeling User's Group Meeting at the SAE Congress
Detroit, MI, 9 April 2018

FRESCO – an object-oriented, parallel platform for internal combustion engine simulations

F. Perini[1], R.D. Reitz[1]

[1]Engine Research Center, University of Wisconsin-Madison
Madison, WI, USA

Abstract: We present the development of FRESCO, a new, object-oriented simulation platform for multidimensional modeling of internal combustion engines. FRESCO has an unstructured solver for field operations on body-fitted moving meshes, where all finite-volume and parallel communications machinery is embedded. On top of that, state-of-the-art models were implemented. Detailed chemistry is solved with a sparse analytical Jacobian chemistry solver combined with a high-dimensional, on-the-fly dynamic adaptive chemistry method. Spray employs an enhanced blob injection and Mach-dependent drop dynamics model, the KH-RT breakup model, an unsteady SGS flow model for near-nozzle dynamics, advanced parallel collision algorithms for grid-independent collision probability estimations with extended outcomes, and a multicomponent vaporization model. Good parallel scalability was seen up to 256 CPUs.

Introduction

The frontier of engine combustion technologies blends combustion strategies in a range of increasing in-cylinder reactivity gradients; being able to predict local turbulence, flow and mixing for combustion development is crucial to their success. As engine design workflows are being put under pressure by competition from hybrid and electric powertrains, multidimensional combustion modeling can support and simplify the design process provided that the right answer is produced in a reasonable amount of time for the combustion engineer (ideally, engine simulations should take no longer than what needed to run overnight).

We developed FRESCO with the aim to provide a robust and accurate platform for internal combustion engine modeling, where a solid finite-volume flow solver produces accurate and reproducible results in a limited amount of time, and where general tools for sprays, flames, etc. are available, so that additional models can be developed, tested and accessed with ease by the engine modeler.

The FRESCO platform

FRESCO (*a Fast, Robust Engine Simulation COde*) is written in object-oriented Fortran: most of this coding paradigm has been available since the Fortran 2003 standard and as of today most Fortran compilers support it nearly completely. We chose Modern Fortran to be able to exploit all the best of object-oriented programming (encapsulation, polymorphism, etc.), which is mostly useful for input/output, settings and code maintainability, and still focus on execution speed, both thanks to full vectorization and to a 60-year legacy of high-performance math libraries.

FRESCO employs body-fitted meshes for maximum near-wall accuracy. An unstructured volume-of-fluid solver includes all *field* operations, where the solution of the Navier-Stokes equations (spatial operators, linear-system solution, etc.) is performed. Scalar, vector and tensor fields are available both as cell-centered and node-centered (Voronoi) storage. The solution follows the successful ALE scheme [1] where operators are split and solved partially on a Lagrangian perspective (explicit: combustion, implicit: momentum, energy, mass conservation), then remapped back to the Eulerian mesh locations by means of a conservative fluxing scheme. A staggered grid is employed with node-centered velocity field and cell-centered scalar fields.

Parallelism. Parallelism and domain decomposition are embedded within the *field* formalism. Each CPU contains a subset of cells, nodes, faces, edges each one defined as *interior* (self-owned), *boundary* (in the halo surrounding the subdomain) or *ghost* (globally deactivated). A recursive domain decomposition procedure as from Figure 1 is applied once an arbitrary cell-based CPU ownership is provided, and corresponding vectorized list objects are generated for cells, nodes, faces, edges.

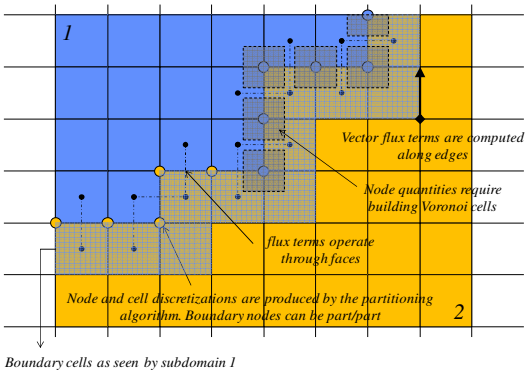

Figure 1. Recursive subdomain definition.

The METIS and ParMETIS library are currently coupled to FRESCO to provide accurate domain partitioning [2]. The domain decomposition is not supervised, and self-updating whenever the mesh topology changes (e.g., when layers of cells are being added/removed during piston movement). Figure 2 reports sample ParMETIS-defined domain decompositions during a sector mesh engine simulation.

The *boundary* parts of each subdomain are kept always synchronized with their owner CPUs: parallel communications across the CPUs are only requested to synchronize their information whenever spatial operators have to be computed, while non-spatial operators are directly run through both interior and boundary of each subdomain. In order to keep the CPU time spent on MPI communications to a minimum, inter-CPU data exchange is performed using a data_exchange sparse matrix-based structure, which stores vectorized lists of element data to be exchanged across CPUs (from *interior* to *boundary*). Non-blocking MPI operations are employed to allow for overlap between communication and computation times.

Figure 3 reports the parallel efficiency of a full-engine simulation of the Sandia 1.9L light-duty engine, featuring a 700k-cells mesh, run for 100 fluid cycles starting from bottom dead center. The test was run flow-only without spray or chemistry, which would bias the parallel efficiency estimates. Good scalability was achieved up to 256 processors (~3k cells/CPU).

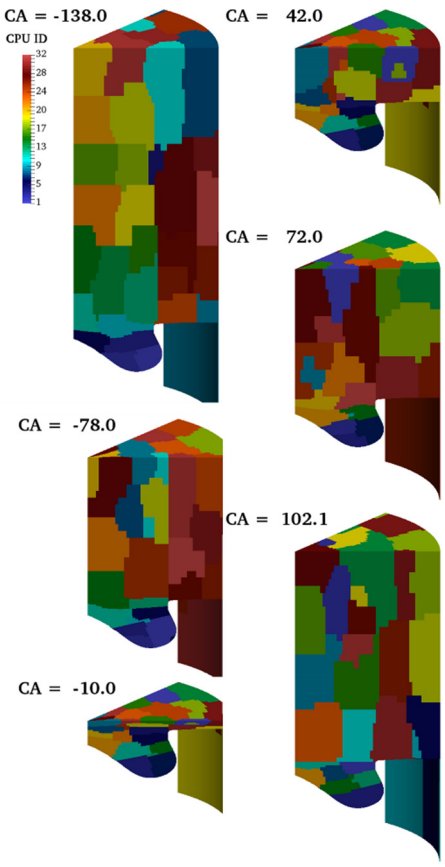

Figure 2. Self-updating domain decomposition.

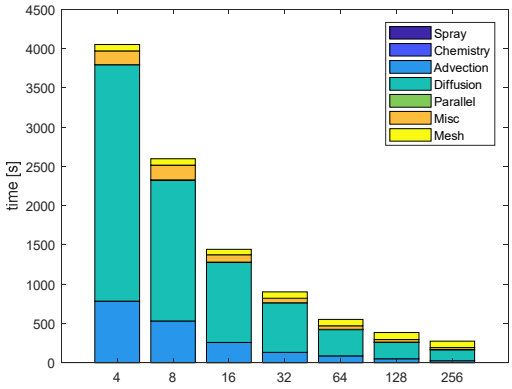

Figure 3. CPU times vs. number of CPUs for a 100-timestep flow field calculation. Parallel efficiency

Numerics. A first-order time integration approach is employed in FRESCO, with a variable time-stepping strategy similar to what employed in the KIVA family of codes [2]. Second-order accurate spatial operators are instead employed for both diffusion and advection terms in the Navier-Stokes equations. For the gradient and Laplacian operators, we employ both linear interpolation of face quantities, and a least-squares based gradient reconstruction procedure for boundary faces.

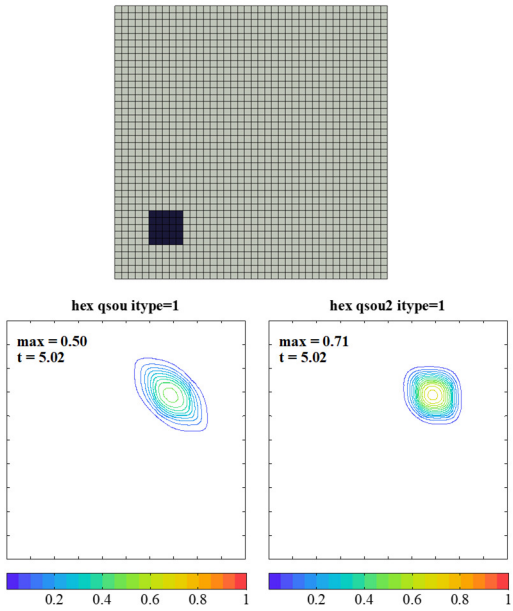

Figure 4. Square translation: quasi-second order upwind [2] vs. FRESCO 2nd-order upwind scheme.

For the fluxing terms, we developed a second-order upwind method with a least-squares based gradient reconstruction method for the flux derivatives instead than the directional derivatives employed in conventional upwind methods, together with a vanLeer flux limiter (Figure 4).

Moving mesh handling

Besides conventional mesh motion methods for sector and full-engine meshes, a universal unstructured mesh optimization and rezoning procedure is also implemented in FRESCO [3]. This procedure produces an optimal, high-quality mesh shape at all simulation timesteps, that does not affect near-wall cell quality, for meshes with any combinations of cell types and unstructured topology.

As large displacements and complex geometries interact in engine simulations, diffusion-based methods such as those solving a Poisson equation may fail, requiring strategies to fix inverted cell regions. The FRESCO mesh rezoning method is based on an optimization problem instead, where the rezoned mesh is the solution to a global mesh quality scalar functional:

$$\mathbf{X_r} = \arg\min F(\mathbf{X_r}|\mathbf{X_b}) ,$$

i.e., node locations $\mathbf{X_r}$ subject to a boundary node discretization $\mathbf{X_b}$ are sought. An 'exponential sum' formulation is employed for the global functional, which provides a continuous and smooth representation for a minimax optimization, which optimizes the whole domain while focusing on the worse element in the set.

Tetrahedron quality measures are used for the element level, which can model effectively unstructured meshes with tetrahedra, hexahedra, prisms and pyramids (Figure 4). A novel tetrahedron quality formulation, which combines aspect ratio (rms edge length) and untangling features (tetrahedron volume) into a same formula, is employed:

$$q_{tet} = \frac{1}{2} - \frac{\det \mathbf{J}}{c \cdot \ell_{rms}} \in [0,1] .$$

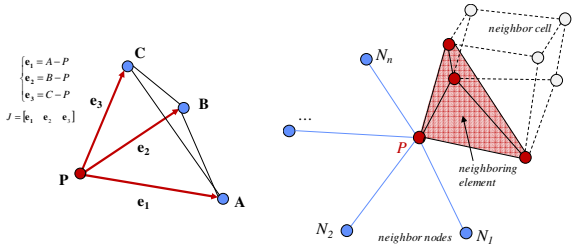

Figure 4. (left) tetrahedron topology. (right) Neighboring element stencil for node-based optimization problem.

The large-scale optimization resulting from the mesh functional is solved using a L-BFGS method [4] with proper accelerators to exploit the functional's gradient and Hessian matrix symmetry, also accounting for the functions' partial separability properties. Figure 5 reports snapshots during the rezoning process for a hexahedral mesh of a cylinder which had been severely tangled initially: the optimization moves all nodes simultaneously, to untangle all inverted cells and bring them towards their optimal configuration in less than 10 iterations. Figure 6 instead shows selected snapshots during a full-cycle (720-degree) simulation of a 4-valve SI engine simulation including valve motion.

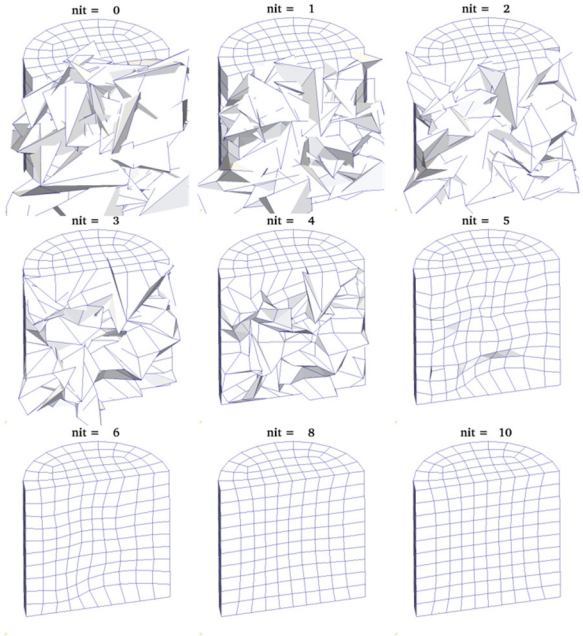

Figure 5. Untangling and optimization of an artificially untangled hexahedral cylinder mesh.

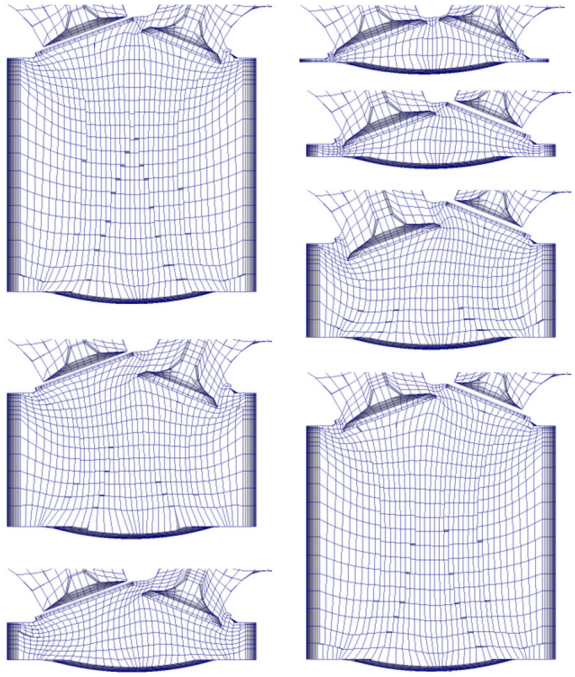

Figure 6. Rezoning of a 4-valve engine geometry through the engine cycle at -180, -90, -40, 0, +20, +70, +180 degrees after top dead center.

Spray modelling

State-of-the-art spray models are employed to describe liquid phase development with the Lagrangian-Drop /Eulerian-Fluid (LDEF) approach: injection, dynamics, breakup, vaporization.

Injection. The spray cloud is decomposed across the subdomains according to a particle-in-cell principle: each particle is owned by the CPU which owns the cell it's contained in; i.e., all parcels are always in the *interior* part of the subdomain. At each timestep, a tracking algorithm is queried to check if any cells has moved to the *boundary* part, in which case the parallel data exchange structure for the parcel cloud is called.

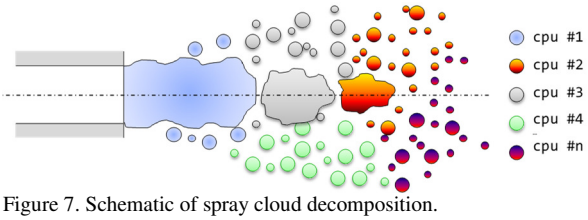

Figure 7. Schematic of spray cloud decomposition.

A fully resolved blob injection model is employed. In general, a computational parcel contains a distribution of droplets, characterized by an SMR value. In the resolved blob model, the number of injected parcels is such that, at the nozzle outlet, each computational parcel contains exactly one liquid blob: a droplet distribution will only develop within each computational parcel only if breakup and collisions occur. Using this approach, for a full engine calculation with a multiple-nozzle injector, usually a few hundreds thousands parcels are injected during the simulation.

Dynamics. Droplet evolution employs an unsteady gas-jet flow model for particle transport in the under-resolved near-nozzle region (Figure 8). In the resolved region, the particle momentum equations are solved implicitly coupled with the gas-phase momentum equation, using an enhanced, Mach-dependent drop drag formulation (Figure 9). Spray breakup features the hybrid KH-RT model of Beale and Reitz [5].

Collisions. Several binary droplet collision outcomes (coalescence, elastic bouncing, reflexive/stretching separation, grazing) were implemented. A deterministic algorithm for grid-independent, radius-of-influence based collision probability estimates was developed. Each parcel collision volume is based on a tetrahedralization of the droplet-in-parcel distribution, which assumes that all droplets contained in a computational parcel are distributed at the vertices of regular tetrahedra, as reported in Figure 10.

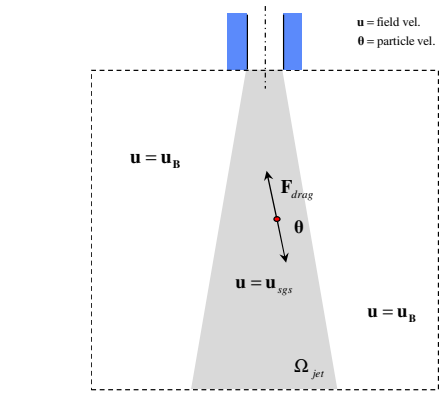

Figure 8. Schematic of the unsteady near-nozzle flow model.

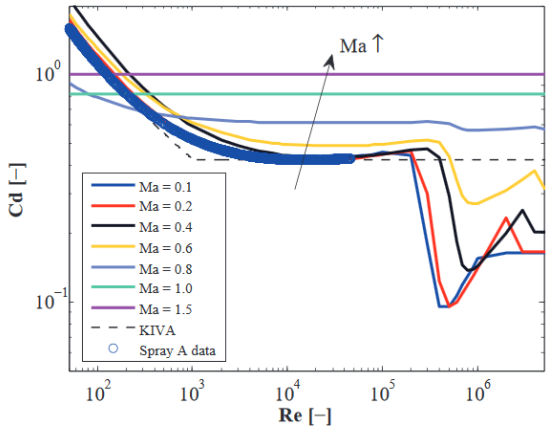

Figure 9. Mach-dependent sphere drag formulation.

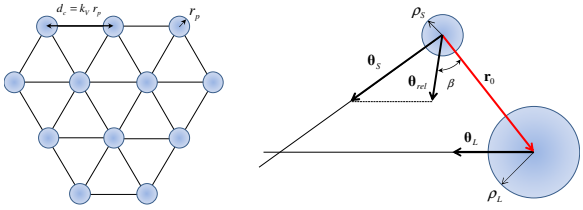

Figure 10. (left) regular tetrahedral drop-in-parcel displacement (2d example), (right) binary collision impact parameter definition.

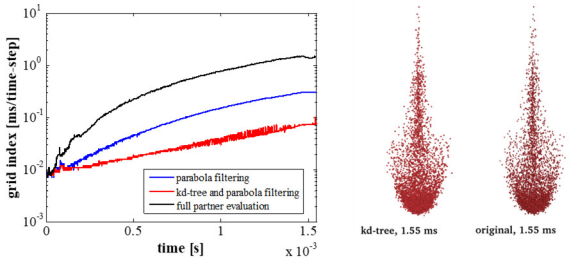

Figure 11. Collision eligibility method comparison.

This procedure allows for a fast parcel-parcel collision probability estimated, which, combined with a kd-tree partitioning of the parcel cloud space, allows for up to 2 orders of magnitude speedup in comparison with a conventional full partner evaluation procedure (Figure 11).

Chemistry

Chemistry solver. Combustion is handled via a well-mixed reactor approach: explicit source terms for internal energy and species mass fractions are computed for each cell as the result of a zero-dimensional adiabatic, constant-volume, well-mixed reactor calculation. The `SpeedCHEM` package [6-8] is employed for the time integration: using an arbitrary large reaction mechanism, and given initial conditions, mass and energy conservation equations for a reactive gaseous mixture are solved:

$$\begin{cases} \dfrac{dY_i}{dt} = \dfrac{W_i}{\rho}\sum_{k=1}^{n_r}\left(\nu''_{k,i} - \nu'_{k,i}\right)q_k, \\ \dfrac{dT}{dt} = -\dfrac{1}{\bar{c}_v}\sum_{i=1}^{n_s}\left(\dfrac{U_i}{W_i}\dfrac{dY_i}{dt}\right). \end{cases}$$

A sparse Analytical Jacobian approach [6] is employed to speed-up the calculation at the reactor level: using this technology, the code exhibits almost linear CPU time scaling with reaction mechanism size, as reported in Figure 12, which corresponds to a speed up of up to three orders of magnitude in comparison with the widely-adopted dense integration approach. SpeedCHEM has been the first package employing this technology to be openly available as a standalone library for non-commercial purposes.

Fast exponential functions. The benefits of a sparse Jacobian are not too large for CFD-sized reaction mechanisms of 50-100 species most relevant for engine simulations. In these cases as from Figure 13, the overall Jacobian sparsity may be not larger than 50-70%, and relevant CPU time is spent evaluating the mechanism's kinetic functions. For all kinetics and gas thermodynamics functions, FRESCO employs a fast exponential/logarithm evaluation method coupled with a high-order polynomial tabulation and interpolation approach [9].

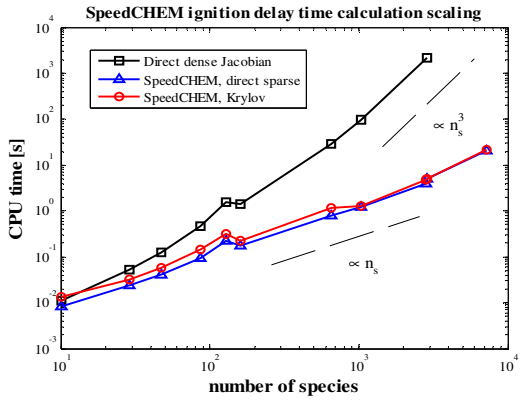

Figure 12. SpeedCHEM IDT calculation CPU time performance vs. reaction mechanism size, compared with the standard dense solver approach.

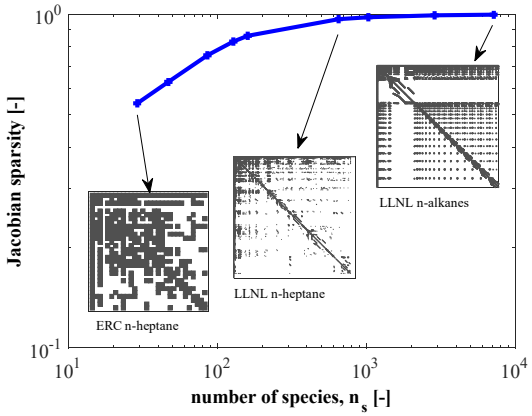

Figure 13. Mechanism sparsity patterns vs. number of species.

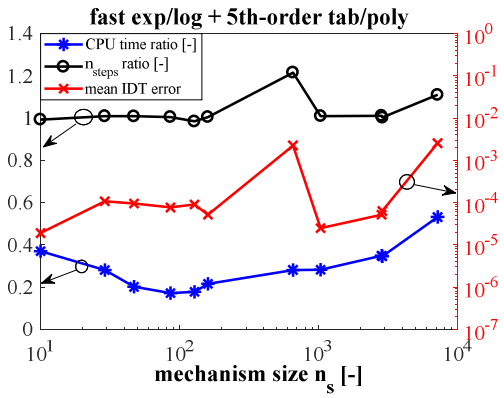

Figure 14. Performance and accuracy of fast exp/log computation and storage-retrieval with piecewise polynomial reconstruction. Performance expressed as ratios w/ corresponding quantities in the standard setup (intrinsics exp/log, no tab/interp).

Using this approach, time for the evaluation of complex exponential-based kinetics functions was reduced by up to two orders of magnitude, and overall CPU time for chemical kinetics integrations was reduced by up to -82.3% for the mid-sized ERC multiChem mechanism (Figure 14). The methodology also allows fast evaluation of thermodynamic functions from the Equation of State as the polynomial formulation allows both function value and its derivatives to be evaluated at the same time.

Dynamic Adaptive Chemistry via PCA-based kd-tree partitioning. Significant computational speed-up computing chemistry source terms is achieved by simplifying the problem at the domain level: cells with similar thermodynamic state are grouped into homogeneous reactors, so the number of actual chemical kinetics integrations is reduced. We developed a new 'Dynamic Adaptive Chemistry' (DAC) method which solves two major challenges: 1) it avoids tabulation storage needs by using an on-the-fly procedure (pressure dependency during the engine stroke limits data re-usability). 2) fuel behavior: user-defined species trackers or higher-level quantities like the equivalence ratio are avoided, which limit DAC benefits in multiple and multi-component fuel cases.

The DAC procedure implemented in FRESCO has three stages, as in Figure 15:

1) reduce the model size, using an appropriate and fast clustering algorithm; compute the thermodynamic states corresponding to each cluster center;
2) solve chemical kinetics IVP at each cluster center state;
3) remap solution back to the full model.

To accurately identify homogeneous cell clusters, FRESCO employs the full chemistry states space, with size n_s+1, so no simplification of the chemical model is needed. A new variant of the *k*-means was designed to make clustering of such large-dimensionality datasets possible, accurate and fast. The algorithm employs recursive kd-tree structures (Figure 16) both to generate an accurate initial partition of the dataset, and to accelerate the *k*-means iterations using nearest-neighbor constraining across the iterates [10]. Reliable temperature (ε_T<10K) and species mass fraction

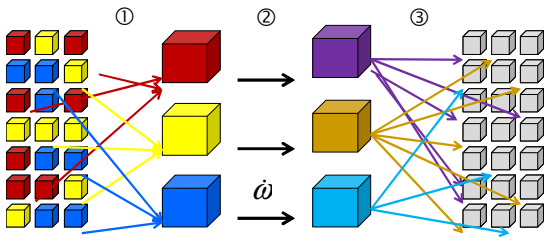

Figure 15. Schematic reproducing the DAC procedure.

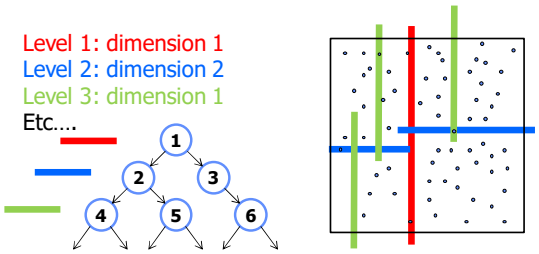

Figure 16: recursive kd-tree structure.

(ε_Y<1e-4) bins were demonstrated to provide CPU time reductions for chemistry of up to two orders of magnitude regardless of reaction mechanism size.

Figure 17 reports a comparison between the full chemistry approach and the fast kd-tree based high-dimensional clustering method for chemistry in a 3D conventional diesel combustion simulation in the Sandia 1.9L light-duty engine with a 93k cells sector mesh using the ERC-PRF mechanism (ns=47, nr=142) [11]. CPU times for chemistry were 7.03h (clustering) vs. 24.54h (full chemistry).

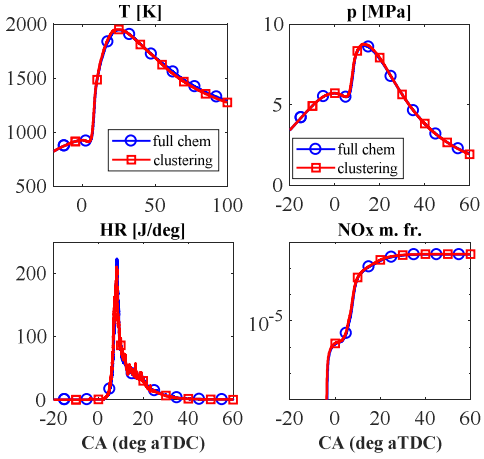

Figure 17. Comparison between full-chemistry and clustering solutions for a 3d sector CDC case.

Conclusions

We introduced FRESCO, a parallel finite-volume simulation platform for multidimensional engine modeling written in modern Fortran. The platform employs the object oriented

approach to hide computational machinery and numerics within field operations, to simplify the engine modeler work. State-of-the-art spray and chemistry models are implemented, to approach grid independency with limited computational demands. Good parallel scaling was demonstrated up to 256 processors (~3k cells/cpu) for a full-engine model simulations. The code is successfully employed for diesel engine combustion research with full-cycle simulations in realistic geometries.

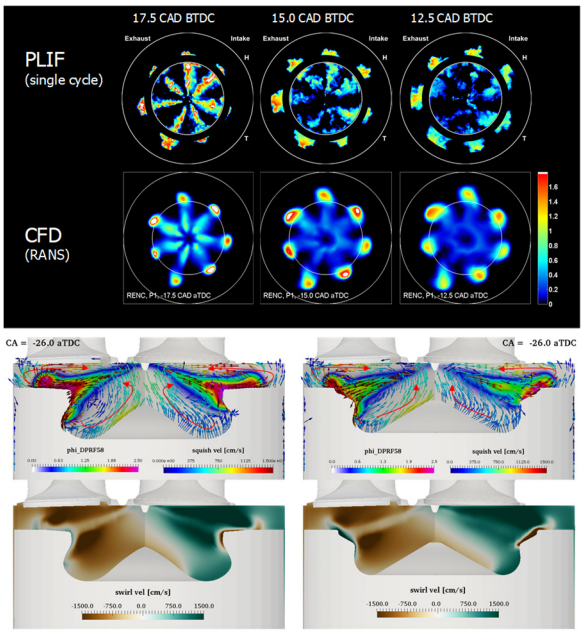

Figure 18. FRESCO application to diesel engine modeling.

References

1. Torres D.J., Trujillo M.F., “KIVA-4: an unstructured ALE code for compressible gas flow with sprays”, Journal of Computational Physics 219(2), 943-975, 2006.
2. Karypis G., Kumar V., “A fast and high quality multilevel scheme for partitioning irregular graphs”, SIAM Journal on Scientific Computing 20(1), 359-392, 1998.
3. Perini F., Reitz R.D., “A universal mesh optimization and rezoning method for Arbitrary Lagrangian-Eulerian Simulations”, Journal of Computational Physics, under review, 2017.
4. Nocedal J., Wright S.J., “Numerical Optimization”, Springer, 2006.
5. Beale J.C., Reitz R.D., “Modeling spray atomization with the Kelvin-Helmholtz/Rayleigh-Taylor hybrid model”, Atomization and sprays, 1999.
6. Perini F., Galligani E., Reitz R.D., "An analytical Jacobian approach to sparse reaction kinetics for computationally efficient combustion modelling with large reaction mechanisms", Energy&Fuels 26 (8), 4804-4822, 2012.
7. Perini F., Galligani E., Reitz R.D., "A study of direct and Krylov iterative solver techniques to approach linear scaling of the integration of Chemical Kinetics with detailed combustion mechanims", Combustion and Flame 161(5), 1180-1195, 2014.
8. Perini F., Das Adhikary B., Lim J.H., Su X., Ra Y., Wang H., Reitz R.D., "Improved Chemical Kinetics Numerics for the Efficient Simulation of Advanced Combustion Strategies" , SAE Int. J. Engines 7(1):2014, doi:10.4271/2014-01-1113.
9. Perini F., Reitz R.D., “Fast approximations of exponential and logarithm functions combined with efficient storage/retrieval for combustion kinetics calculations”, Combustion and Flame, under review, 2017.
10. Perini F., Reitz R.D., “A nearest-neighbor constrained, kd-tree accelerated k-means algorithm for fast high-dimensional model reduction, with application to engine combustion”, in preparation.
11. Ra Y., Reitz R.D., “A reduced chemical kinetic model for IC engine combustion simulations with primary reference fuels”, Combustion and Flame, 2008.