Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

SHAPE Project Creo Dynamics:

Scale Resolving CFD and CAA processes for Ground Vehicles based on Open Source

Torbjörn Larsson[a*], Johan Hammar[a] , Jing Gong[b], Michaela Barth[b], Lilit Axner[b]

[a]*Creo Dynamics AB, Westmansgatan 37A, SE-582 16 Linköping, Sweden*

[b]*PDC Center for High Performance Computing, KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden*

Abstract

Creo Dynamics [1] is a Swedish engineering company with core competence in fluid dynamics, acoustics and structural dynamics. Creo Dynamics has broad experience from participating in national and international research programmes focussing on the development of new emerging technologies for the automotive or aerospace industries. In the company, experienced engineers develop and deliver simulation tools and procedures – often based on open source software [2] – which are tailored towards specific needs and applications; this process always involves maintaining a conscious balance between turn-around time and accuracy.

Creo Dynamics has recently undertaken a PRACE SHAPE project together with application experts from the KTH Royal Institute of Technology. During the course of the project we developed templates and "recipes" for a range of tasks that are involved in the automotive industry – such as automated handling of computer-aided design (CAD), parallel meshing, solving and post-processing. These were all tailored towards a particular automotive application, namely the aerodynamics of a heavy-duty semi-trailer. We also focused on parallel implementations and executions for large-scale simulations. In addition to monitoring the efficiency of the processes (for meshing and solving run-time performance and scalability) and identifying critical bottlenecks, we gave significant attention to pinpointing performance deficits in the processes so as to give guidance for a further fine-tuning of the overall methodology.

1. Background and Introduction

The transport sector contributes to about 25% of the total CO_2 emissions in the EU and is the only sector where the trend is still increasing [3]. Hence, it is of paramount importance to increase the efficiency of freight transport in order to stop increasing (and preferably start to decrease) the amount of CO_2 that is emitted. With the strong likelihood of future stringent legislation requiring far more environmentally friendly technology, it is crucial that we develop more efficient virtual testing techniques.

Efficient numerical methods for time dependent flow simulations are becoming increasingly important for highly accurate predictions of aerodynamic drag, aero-acoustic sources, and vehicle dynamics and handling. Accurate and detailed predictions of the aerodynamic flow, and eventually the acoustics propagation, around road vehicles and aircraft are challenging - they put exceptionally high demands on the simulation methods that are used with respect to both numerical and physical fidelity. As a result, the computational resources that are required are

[a]* Corresponding author. *E-mail address*: torbjorn.larsson@creodynamics.com

often immense, thus making these types of analyses less viable in a high-paced production environment with short lead times and strict cost constraints. Despite this, Computational Fluid Dynamics (CFD) is a vital and necessary tool nowadays for the design of aerodynamically efficient products like cars, trucks, airplanes and turbomachines.

In the last two decades, the automotive industry has seen most of their dedicated in-house developed simulation codes being replaced by commodity commercial software. This trend can be exclusively attributed to the need to cut costs. The development effort required to keep in-house software (that is based on programming techniques from the late 1980s) up-to-date and at the cutting-edge is considerable. The outsourcing of this key technology has led to a less than ideal dependency on commercial Computational Fluid Dynamics (CFD) software vendors with software-development plans and corporate business strategies that are not necessarily perfectly in line with the needs and expectations of the automotive industry.

With emerging open source technologies on the horizon, the CFD engineering landscape is expected to change once again. The maturity of open source CFD software is starting to have an impact, even in industry. This is not only because of the attractive and obvious advantages of software being license-free, and thus deployable on a massive scale. The importance of software being fully accessible, and the inherent development potential that comes with the concept of open source, has undoubtedly been recognized. The leverage effect from an ever-growing open source community should not be underestimated either. Numerous examples have shown that open source software can be an ideal platform for developing state-of-the-art codes, in terms of flow physics, numerical algorithms and software performances.

The envisaged CFD methodology, as outlined in this paper, is expected to deliver simulation accuracy in line with the current industry best practices but at a considerably lower cost.

The first task in the project was primarily devoted to the development of templates and "recipes" for automated CAD handling, parallel meshing, solving and post-processing that were tailored towards the chosen application (that is, modelling a semi-trailer).

In the second task, we studied the parallel implementation of code and execution on large parallel clusters. Here the main focus was on the monitoring of process efficiency (meshing and solving run-time performance and scalability), plus the identification of critical bottlenecks and performance deficits in the processes to give guidance for fine-tuning the overall methodology.

The project was steered by engineers from Creo Dynamics and executed in collaboration with researchers and application experts from the KTH Royal Institute of Technology. Creo Dynamics, a small engineering company with core competence in fluid dynamics, acoustics and structural dynamics, has broad experience from participating in national and international research programmes focussing on the development of new emerging technologies for the automotive or aerospace industries.

2. Benchmark Description

The aerodynamic drag on a heavy-duty truck-trailer equipage is significant and, with new stringent legislations on fuel efficiency and CO_2 reductions coming into effect in the pipeline, substantial development to improve the aerodynamic efficiency of such ground vehicles is required. Furthermore, the trend towards electrification and hybrid technologies for vehicles means that now, more than ever, there is a need for the aerodynamic drag to be decreased in order to increase the range of electric vehicles (EVs). With EVs expected to dominate the road car market in the near future, there is also renewed interest in topics relating to airborne noise.

Experimental aerodynamic testing of full-size trucks is challenging, particularly as there are only very few facilities in the world where it is possible to do such tests. Highly accurate predictions of the aerodynamic (and acoustic) properties of these large vehicles via physical testing are immensely expensive, and consequently often not justifiable in a repetitive manner, especially in the fast-paced development that is paving a route to sustainable transportation. Instead, further investments in dedicated virtual testing techniques are required with a firm commitment to large-scale High Performance Computing (HPC).

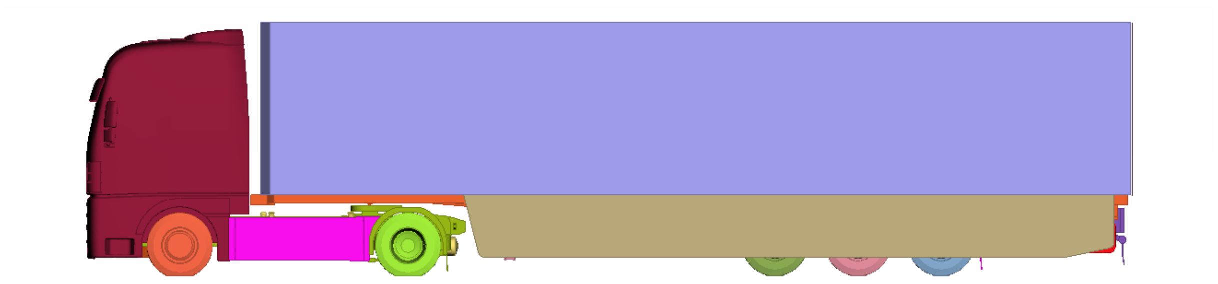

Figure 1 Truck-Trailer CFD model, side view

In the current project we use a generic, but representative, CAD model of a heavy-duty semi-trailer as shown in Figure 1 above. The model includes a fairly detailed chassis and underside as displayed in Figure 2.

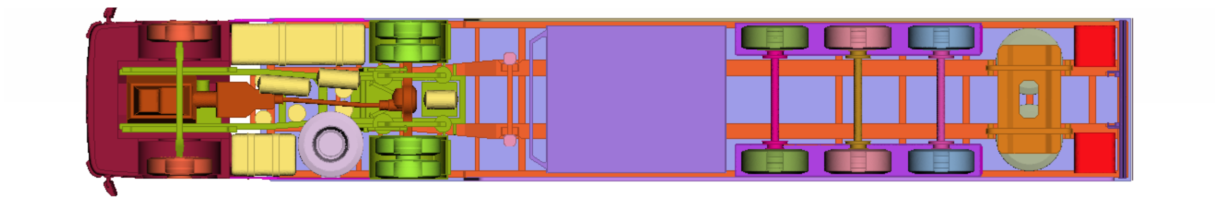

Figure 2 Truck-Trailer CFD model, bottom view

We are aiming for high fidelity scale-resolving simulations to enable us to make accurate predictions, not only of the airflow distribution, but also of the aeroacoustic noise generation and propagation. For this reason all the steps in the simulation processes need to be made efficient for parallel computation and also scalable to many thousands of computational cores.

Herein, we set up and analyse an automated process that, using prepared CAD surfaces as input, generates a high resolution CFD mesh fulfilling predefined quality metrics. The process also performs a steady state RANS (Reynolds-averaged Navier-Stokes) simulation, and extracts results and post-processing data. All these are steps implemented in parallel on distributed compute nodes without the need for any intermediate input/output (I/O) or data transfers.

3. Open Source CFD Workflow

Parallel performance of all steps of the simulation process, as well as efficient I/O and data extraction, are becoming vital as the size of models and the complexity of simulations increases. Building a fully parallelized simulation process that removes serial bottlenecks, time consuming I/O operations, and labour intensive manual work is a quest that is being actively pursued in the automotive industry due to the significant benefits such a process will bring.

Here, in this project, the basic idea is to derive a (semi) automated and fully scripted simulation process for a chosen application (truck-trailer) starting from prepared CAD data. The automation, along with the possibility to perform the mesh generation in parallel on distributed compute nodes where the same nodes are also used for subsequent CFD simulations, are central features of the work. These features mean that all the steps in the process can be executed in a homogeneous Linux-based computer environment thus avoiding serial bottlenecks, time-consuming I/O, and additional data transfers.

The main steps of this process, described in Figure 3 below, are based entirely on the *OpenFoam* toolbox.

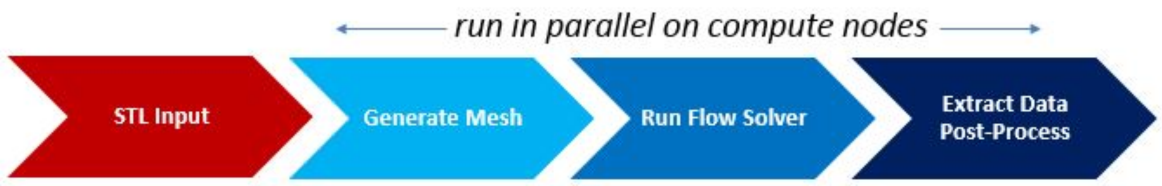

Figure 3 OpenFoam Workflow

4. The OpenFoam Utilities and Solvers

4.1 SnappyHexMesh

The pre-processing phase (that is, the Generate Mesh phase in Figure 3) is based on the *OpenFoam* meshing utility *snappyHexMesh,* which relies on a geometry description in stereolithography (STL) file format. Before executing the *snappyHexMesh* (in parallel) though, a couple of serial steps are performed to create and decompose an initial background hexahedral mesh and to extract geometrical feature edges. These operations only require a few seconds of computational time running on a single core, and consequently will not be described here.

The mesh generation is defined and controlled from the *snappyHexMesh* dictionary, *snappyHexMeshDict,* that contains numerous parameters for mesh control.

The five main sections of the dictionary are:

- **geometry**
  - Definition of geometrical entities
- **castellatedMeshControls**
  - Definition of features, surface and volume refinements
- **snapControls**
  - Definition of parameters for surface snapping
- **addLayersControls**
  - Definition of parameters for boundary layer meshing
- **meshQualityControls**
  - Definition of mesh quality metrics

The meshing process is divided into three main steps. The first step is the generation of a castellated mesh where all criteria relating to surface and volume resolution should be met. This “lego-type” mesh does not yet conform to the input STL surfaces, and will be saved in a folder known as *time 1*.

Once the castellated mesh has been completed, a snapping stage follows. Here the mesh points are snapped to the surfaces in an iterative fashion to create a fully conformal mesh according to some pre-defined quality metrics. The snapped mesh is then saved in a folder known as *time 2*.

The final mesh generation step in the *snappyHexMesh* process is the addition of boundary layer cells on selected parts of the geometry. This is arguably the most difficult, and least robust, step in the entire meshing process – this often requires plenty of model preparation and tuning before an acceptable mesh quality can be reached. The final layered mesh is then saved in a folder called *time 3*.

In this study, all three steps in the mesh generation process have been monitored in terms of memory usage and parallel run-time performance – these are discussed in detail in Section 5.1.

Creating high quality meshes on complex configurations with *snappyHexMesh* is far from trivial and finding suitable “meshing recipes” often requires a sound understanding of the various steps in the mesh generation algorithms, as well as rigorous knowledge of the model configuration and the predominant flow physics involved. Often a good deal of iterative testing and parameter tuning may be needed before a mesh of acceptable quality can be generated.

During meshing, the mesh quality is constantly being monitored and *snappyHexMesh* will, in an iterative fashion, try to improve the mesh quality until all the relevant criteria have been met.

In the *meshQualityControls* section of the *snappyHexMeshDict,* we define relatively strict criteria for the final mesh quality, with face skewness and face orthogonality being two very important quality metrics, see Figure 4 for the definition of these metrics.

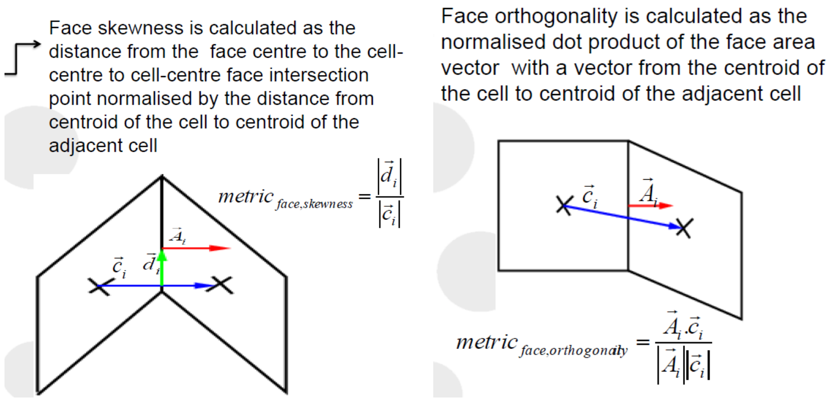
Face skewness is calculated as the distance from the face centre to the cell-centre to cell-centre face intersection point normalised by the distance from centroid of the cell to centroid of the adjacent cell
$metric_{face,skewness} = \frac{|\vec{d}_i|}{|\vec{c}_i|}$
$\vec{A}_i$
$\vec{c}_i$
$\vec{d}_i$
Face orthogonality is calculated as the normalised dot product of the face area vector with a vector from the centroid of the cell to centroid of the adjacent cell
$\vec{c}_i$
$\vec{A}_i$
$metric_{face,orthogonality} = \frac{\vec{A}_i.\vec{c}_i}{|\vec{A}_i||\vec{c}_i|}$

Figure 4 Skewness and orthogonality quality metrics

Since *snappyHexMesh* can run in parallel on distributed cluster nodes by using dynamic load balancing, it is well suited to become an integrated part of a fully scripted CFD process running in batch on a large parallel cluster.

However, parallel efficiency and scalability while running on multiple cores has been reported to be poor and the code suffers from high memory consumption as well. One possibility for reducing the memory consumption is to use a distributed STL input instead of the (default) approach where the STL data gets copied across to every compute node. This may decrease the significant overhead in RAM allocation that eventually causes performance issues. With the utility *surfaceRedistributePar* the geometry can be decomposed and distributed across the processors, thereby reducing redundancy and hence memory consumption. These two ways of providing STL input for *snappyHexMesh* are schematically illustrated in Figure 5.

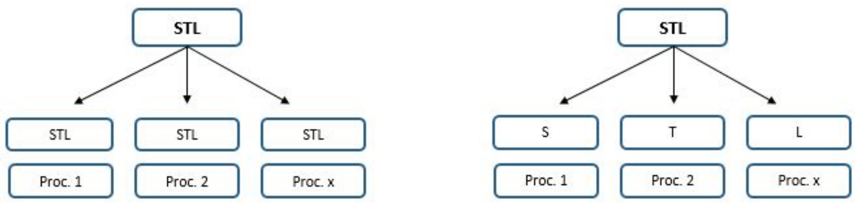

Figure 5 Standard (left) vs. Distributed (right) STL input

In this project both methods are evaluated and compared in terms of run-time and memory allocation.

An example where the STL data has been distributed over 8 cores is shown in Figure 6, where the colours represent the different core allocations.

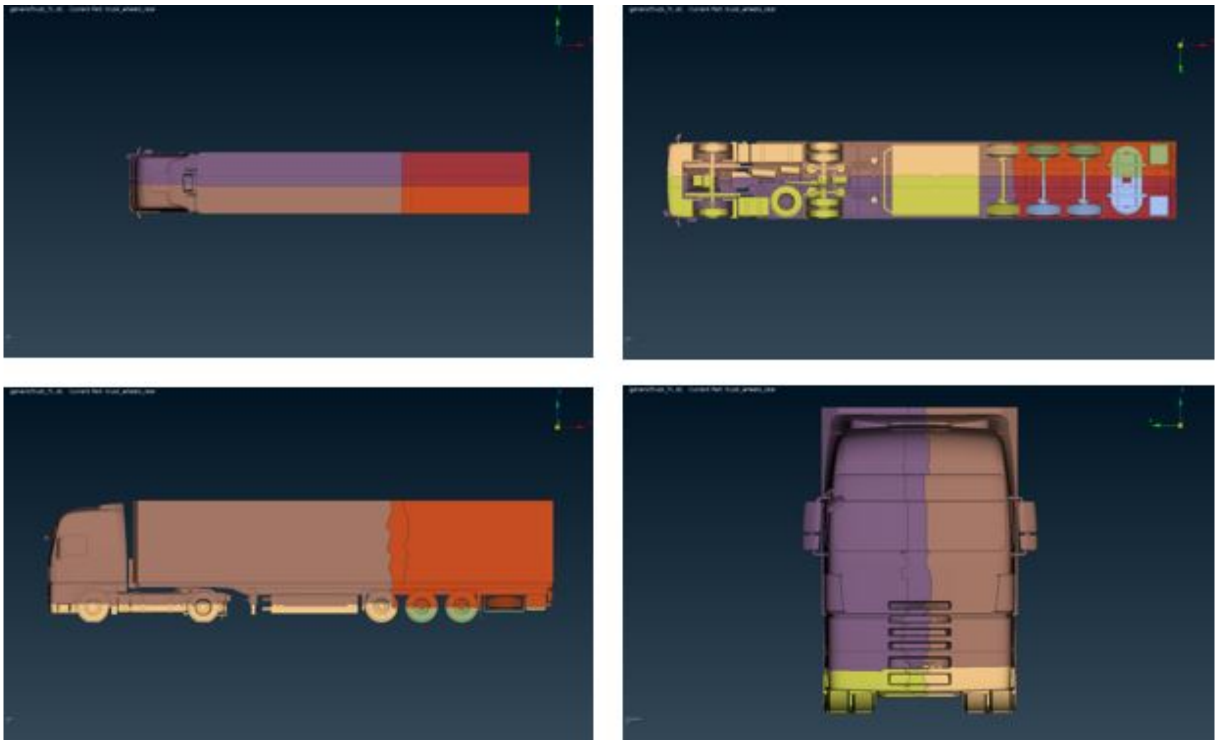
Figure 6 Distributed STL input (8 cores)

An illustration of a representative mesh is displayed in Figure 7 where a vertical cut through (part of) the computational domain is visible. Note the particularly dense mesh underneath and behind the vehicle.

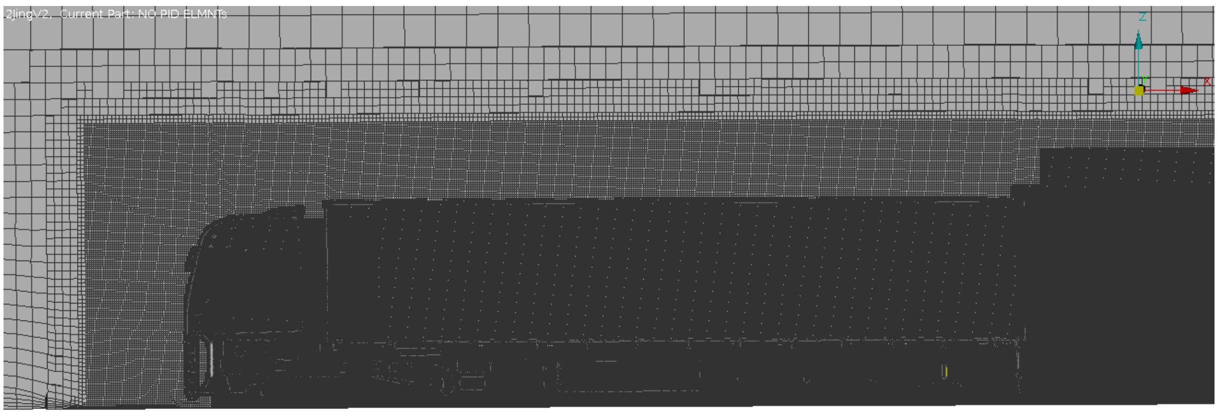

Figure 7 Vertical cut through the time 3 mesh

4.2 The OpenFoam Solver

The Navier-Stokes equations for the incompressible flows are written as

$$\nabla \cdot \mathbf{v} = 0$$

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla p + \nu \Delta \mathbf{v}$$

with suitable initial and boundary conditions, and where $\mathbf{v}$ is the velocity vector, p is the pressure and ν is the kinematic viscosity. Since the solutions for the coupled equations are not straightforward, the viscous and pressure sub-steps require solving a Poisson-equation subject to various boundary conditions.

Within the project, the *simpleFoam* solver is employed to address the large simulations. The Semi-implicit methods for Pressure-Linked Equations (SIMPLE) algorithm is implemented in the solver *simpleFoam* and couples the Navier-Stokes equations in an iterative procedure following [4].

- Set the boundary conditions.
- Solve the discretized momentum equation to compute the intermediate velocity field.
- Compute the mass fluxes at the cell faces.
- Solve the pressure equation and apply under-relaxation.
- Correct the mass fluxes at the cell faces.
- Correct the velocities on the basis of the new pressure field.
- Update the boundary conditions.
- Repeat till convergence.

5. Results and Discussions

The simulations were performed on the Cray XC40 system Beskow [5] at PDC, KTH Royal Institute of Technology, Sweden, and on MareNostrum [6] at BSC, Spain. Beskow is based on Intel Haswell processors and the Cray Aries interconnect technology. It consists of 1676 compute nodes, each of which consists of 32 Intel Xeon E5-2698v3 cores. MareNostrum has 36 racks dedicated to calculations. These racks have a total of 48,448 Intel SandyBridge cores with a frequency of 2.6 GHz and 94.625 TB of total memory.

5.1 snappyHexMesh

Before proceeding with the large scale distributed parallel meshing, the parallel scalability of *snappyHexMesh* was first evaluated on a couple of thin compute nodes at PDC where each node has 24 cores and 512 GB RAM. Different releases of *snappyHexMesh* (2.3.x, 2.4, 3.0+, 1606+) were tested. Although there were some performance differences observed between the various releases, unfortunately all the releases were suffering from poor parallel scalability. For these tests, an intermediate sized mesh of 160 million cells was constructed to make the mesh generation possible on a single core.

Figure 8 below shows the run-time performance and scalability obtained for this intermediate test case.

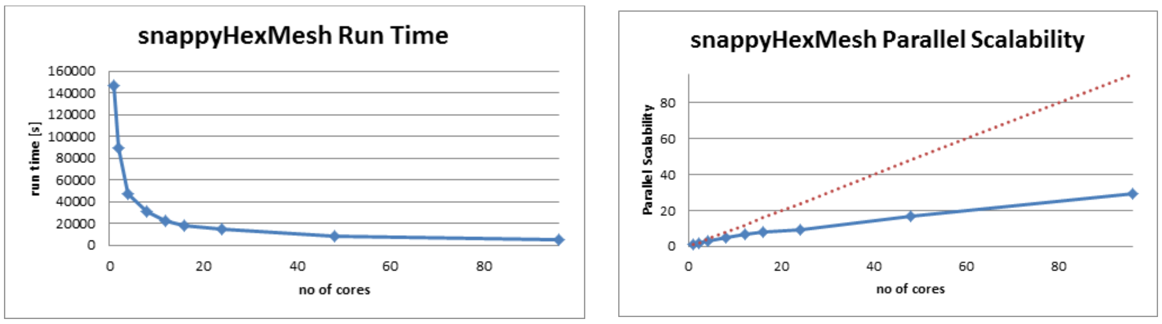

Figure 8 Run-time and scalability test of snappyHexMesh, intermediate case using OpenFoam v3.0+

Next, larger meshes of around 350 million cells were generated in parallel on a sequence of 512, 1024, 2048, 4096 and 8192 cores, respectively. The run-time performance and scalability of the entire meshing process, as well as the timings for each meshing step, are displayed in Figure 9. These graphs clearly illustrate that the castellated meshing step (*time 1*) represents a significant portion of the total meshing time (>50%) and that the parallel scalability of this step is particularly poor.

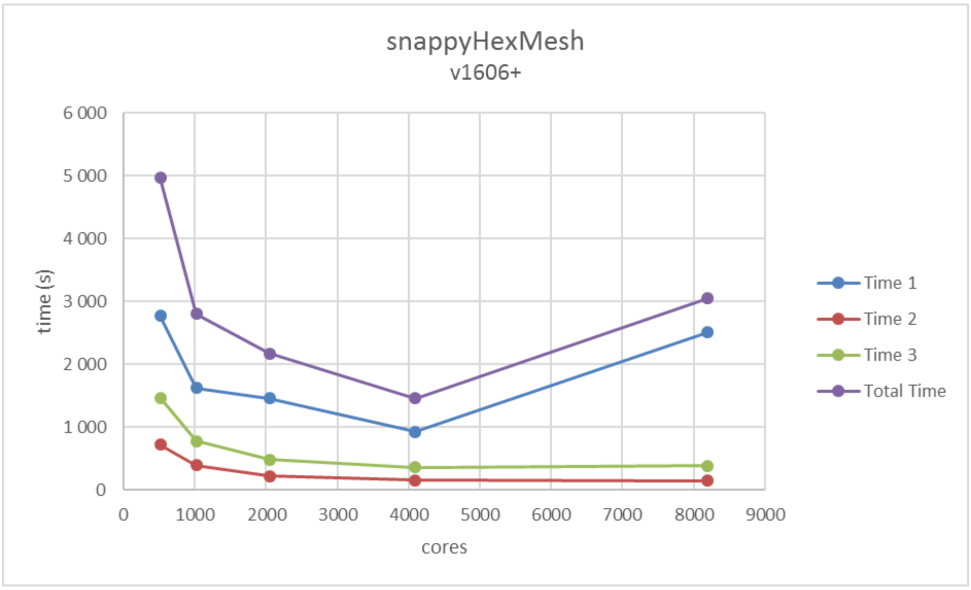

Figure 9 Run-time and scalability test of snappyHexMesh, large case

snappyHexMesh also suffers from relatively high memory consumption. This may lead to performance implications if the computational nodes have limited RAM. Often separate (fat) nodes or another shared memory compute architecture must be brought in for the mesh generation for large and complex cases. This not only creates additional bottlenecks and performance deficits in the overall simulation process but it also increases the cost and complexity of the necessary compute infrastructure.

Using distributed STL input can reduce the memory consumption of *snappyHexMesh* and may provide opportunities to generate larger meshes by using less overall memory. In the more recent versions of *snappyHexMesh* this functionality has been improved. Herein we present results obtained while using version 1606+.

Using a distributed STL input we save close to 300 GB in total accumulated memory while meshing in parallel on 1024 cores (see Figure 10). This represents about a 30% reduction in the overall memory consumption.

However, this saving in RAM comes with a significant penalty in mesh generation time as seen in Figure 11. Performance degradation in all steps in the mesh generation process is observed (time 1, 2 and 3).

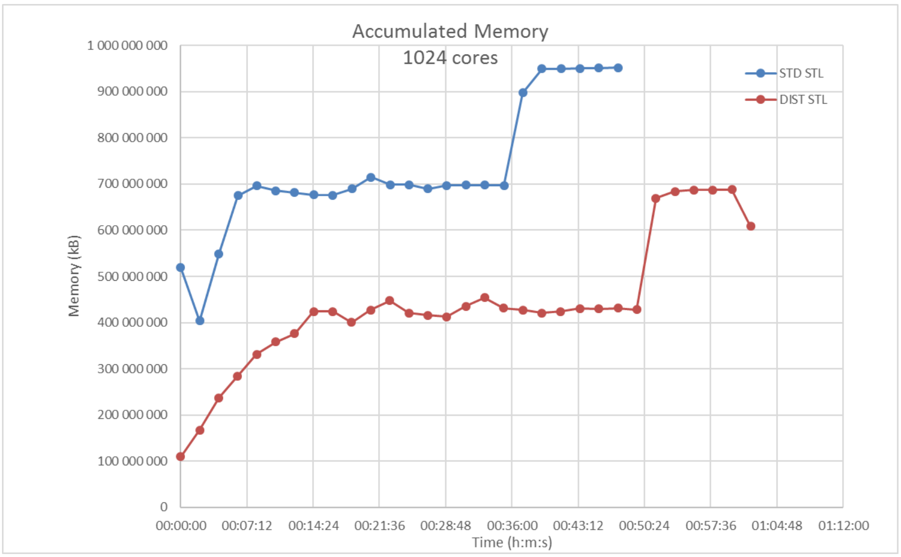

Figure 10 Accumulated memory of snappyHexMesh running on 1024 cores, large case

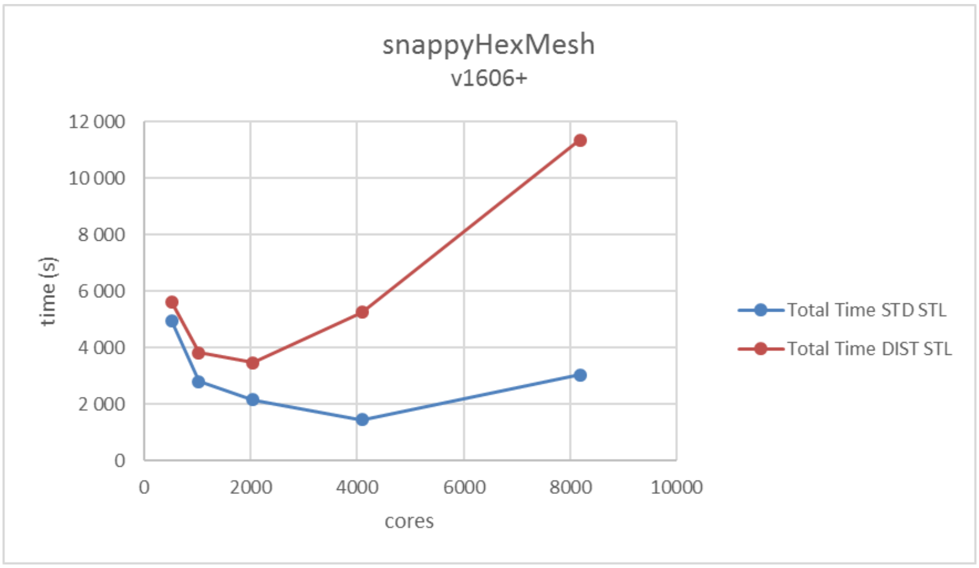

Figure 11 Run time performance of snappyHexMesh STD vs. DIST STL, large case

5.2 The scalability of simpleFoam

Assessments of the total run-times for the *simpleFoam* solver using the two mesh generation strategies introduced in Section 4.1, i.e. based on the standard and distributed STL input, are presented below. Figures 12 and 13 show the flow solver scalability for a case containing 110 million cells on MareNostrum and Beskow, respectively. For each run we average the timings over 500 time steps. The decomposition method *Scotch* [7] is used for the standard STL input whereas the *hierarchical* decomposition method is used for the distributed STL input.

The execution time per step reduces from 13.874 to 1.158 seconds when running the case with the standard STL on 2048 cores, see Figure 12. An almost linear speed-up can be observed from 128 to1024 cores on MareNostrum for the standard STL input. A parallel efficiency of 74.4% is achieved using 2048 cores compared with 128 cores for the standard STL input. Here we use the definition of the parallel efficiency as

$$\eta(\%) = \frac{T_{min}}{T_{max}} \cdot \frac{N_{min}}{N_{max}} \cdot 100$$

where is T_{min} the execution time per step for the minimum number of cores N_{min}, while T_{max} is the execution time per step using the maximum number of cores N_{max}.

Due to memory limitations, the minimum number of cores that can be used for this case on MareNostrum is 128. The performance and scalability for the Distributed STL input is a bit worse than those for the Standard STL, see Figure 12. A partial reason for this deficit is that the more efficient partition tool *Scotch* is employed for the Standard STL. Currently only simple partition methods, such as *hierarchical,* are implemented for the Distributed STL input.

On Beskow the execution time per step reduces from 17.057 to 0.643 seconds when run the case of standard STL input on 4096 cores, see Figure 13. We can obtain almost linear speed-up from 128 to 2048 cores for the Standard STL. The parallel efficiency η is 82.9% with 4096 cores compared with 128 cores. Using the Disturbed STL, the performance is worse, but this allows us to run on as little as just two nodes with 64 cores. Beyond 3072 cores the performance degrades significantly. Due to the limit of memory on Beskow (32G RAM per node), the minimum requirement for the Standard STL is four nodes (128 cores).

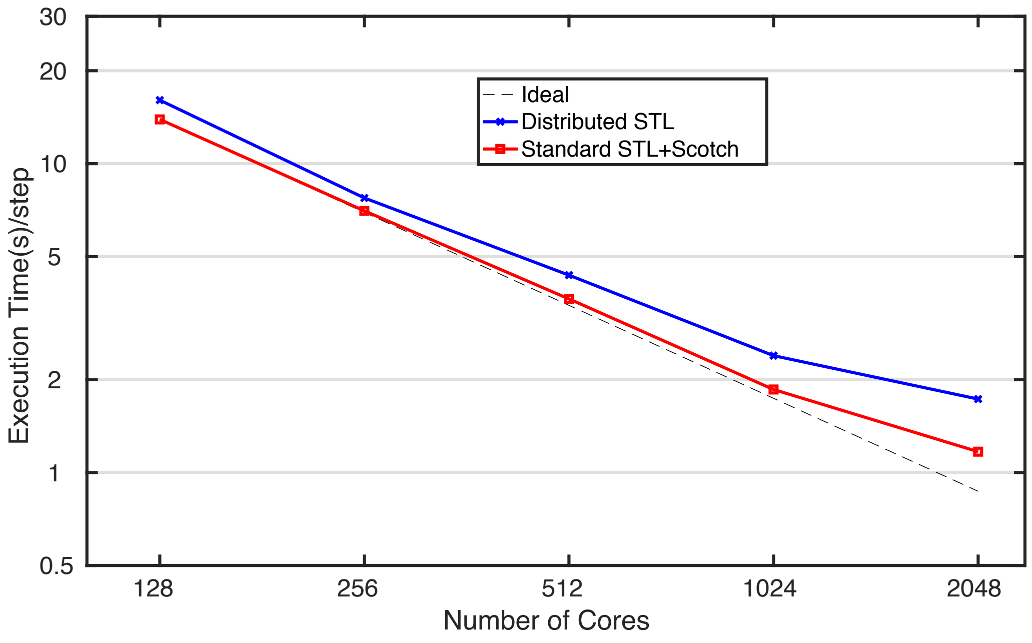

Figure 12 The performance results on MareNostrum

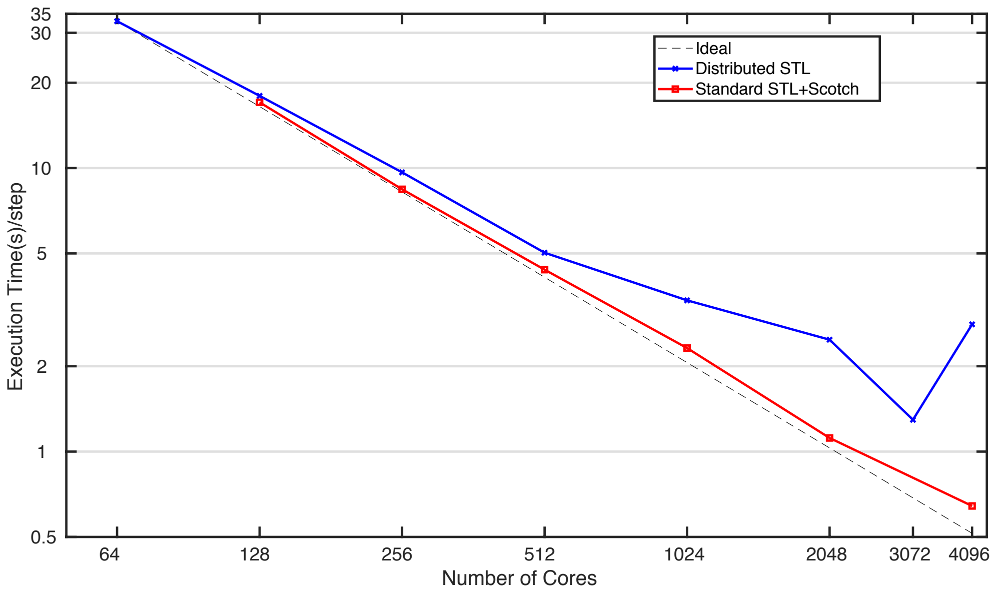

Figure13 The performance results on Beskow

5.3 Results

We have presented a complete CFD simulation process that runs in parallel on large Linux clusters. The entire workflow, from prepared CAD input to the output of results, is executed in batches without the need for any intermediate I/0, data transfer or human interventions in between. All steps in this process rely exclusively on open source software.

Although the scalability of *snappyHexMesh* is rather mediocre, we demonstrated that fairly large and complex CFD meshes could be successfully generated on distributed compute clusters in a limited period of time. Mesh generation of a detailed semi-trailer configuration comprising 350 million cells was completed in less than 25 minutes using 4096 compute cores. On the same number of cores, the *simpleFoam* solver was shown to perform well. Hence, for this size of problem, the presented process performs well.

However, while scaling beyond a few thousand computational cores and with model sizes above a few hundred million cells, a variety of problems and performance deficits arise. In particular, the mesh generation phase (*snappyHexMesh*) revealed weaknesses and several software issues have been reported.

Below are a few snapshots from a steady state Reynolds Averaged Navier Stokes (RANS) simulation.

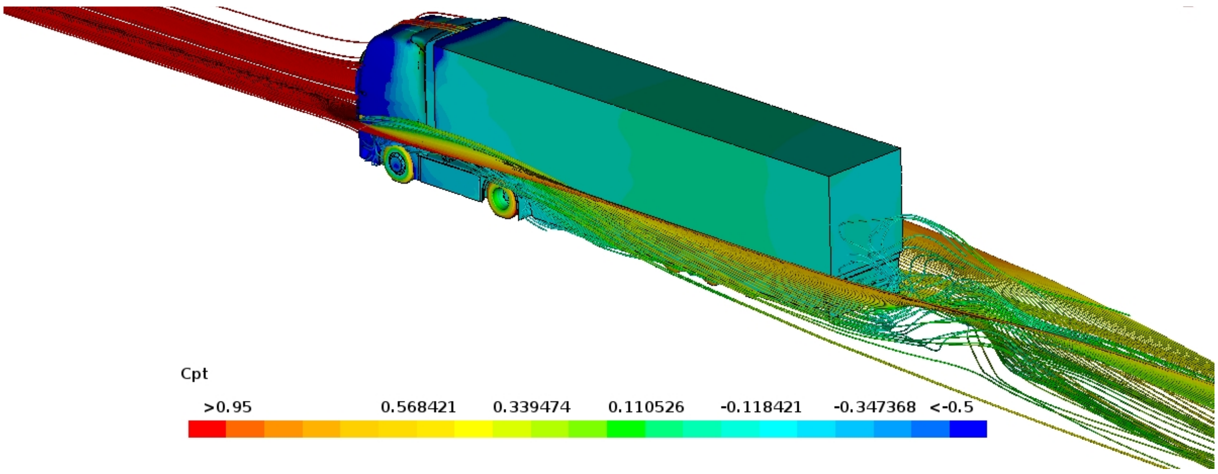

Figure 14 Flow path lines rendered by total pressure

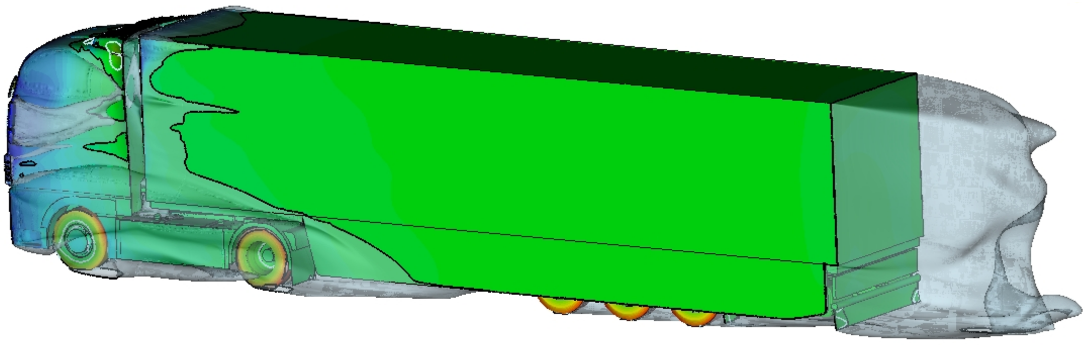

Figure 15 Iso-contours of total pressure

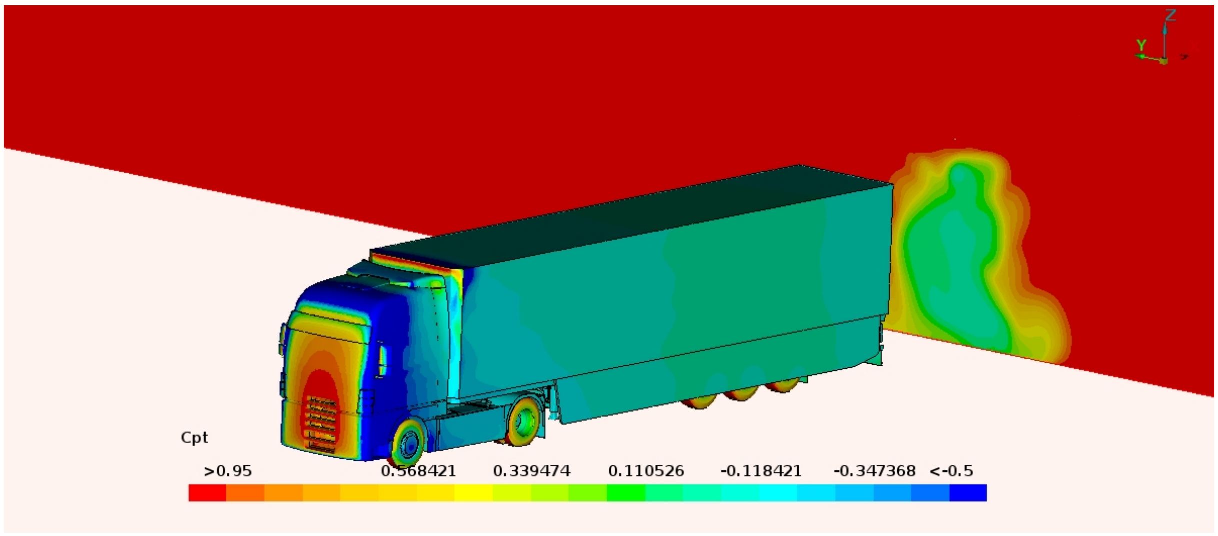

Figure 16 Vehicle surfaces rendered by static pressure and plane rendered by total pressure

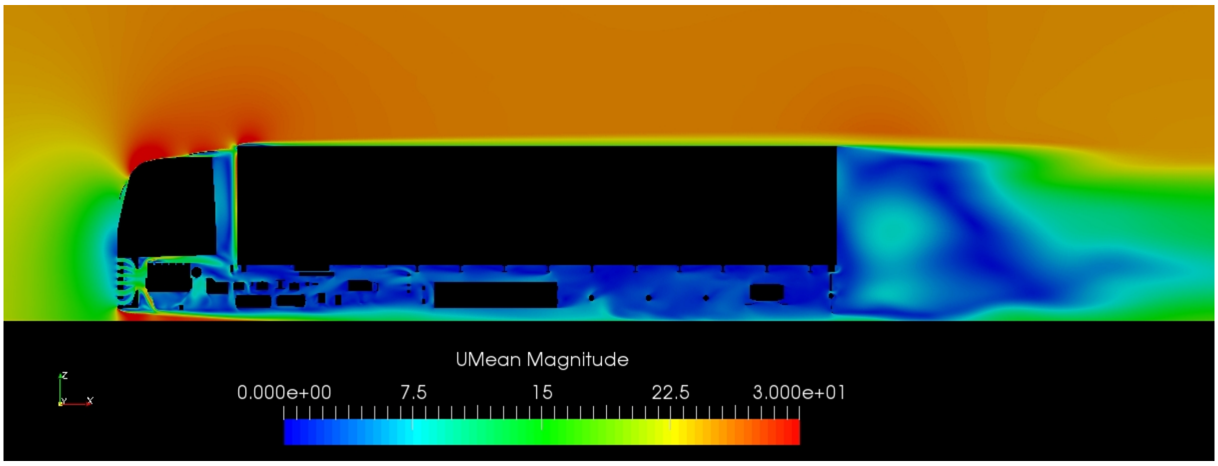

Figure 17 Vehicle centreline plane rendered by mean velocity magnitude

6. Conclusion

During the project a (semi) automated CFD simulation process tailored towards aerodynamics predictions for heavy duty semi-trailers was developed. A variety of benchmarks related to parallel meshing and flow solving using *OpenFoam* were performed. The memory usage and parallel scalability were monitored revealing performance deficits and weaknesses, particularly related to parallel mesh generation on many cores. For efficient simulations of large problems requiring many thousands of computational cores there are still critical bottlenecks and deficiencies in current *OpenFoam* distributions that need to be addressed.

Even if the Open Source alternatives do not as yet deliver the same performance as commercial CFD software, we conclude that time and effort that needs to be invested in Open Source software compares favorably to the expected sum of expenses for commercial CFD licenses to produce the same results (for a medium scale problem) and is therefore an option that is well worth continuing to investigate. Open Source software also has the advantage that, as more development is done within the automotive research community and hence more tools for automated processes become widely available free of charge, it has the potential to surpass commercial software in terms of performance to cost ratio.

Research programmes such as PRACE SHAPE, giving smaller companies and research teams access to large-scale compute infrastructures, are instrumental in the development and validation of numerical tools and methods capable of delivering cost-effective simulations based on Open Source software.

Creo Dynamics has gathered valuable information and experiences during this project that will help in building more efficient and competitive simulation processes tailored towards high fidelity analysis of coupled aerodynamics and acoustic problems in automotive applications.

References

[1] Creo Dynamics AB, http://www.creodynamics.com

[2] OpenFoam, www.openfoam.com

[3] https://ec.europa.eu/clima/policies/transport/vehicles_en#tab-0-0

[4] simpleFoam, https://openfoamwiki.net/index.php/OpenFOAM_guide/The_SIMPLE_algorithm_in_OpenFOAM

[5] Beskow, https://www.pdc.kth.se/resources/computers/beskow

[6] MareNostrum, https://www.bsc.es/innovation-and-services/supercomputers-and-facilities/marenostrum

[7] Scotch, http://www.labri.fr/perso/pelegrin/scotch/

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EU's Horizon 2020 research and innovation programme (2014-2020) under grant agreement 653838. The work was done using the PRACE research infrastructure Tier-0 resource MareNostrum at BSC, Spain, and the Tier-1 resource Beskow at KTH-PDC, Sweden. Support from the PRACE SHAPE programme for the technical work is also gratefully acknowledged.