H2020 FETHPC-1-2014

ExaFLOW

Enabling Exascale Fluid Dynamics Simulations
Project Number 671571

D3.4 – Evaluation from an Industry Perspective with an Industrial Use Case

WP3: Validation & Case Studies

Document Information

Deliverable Number	D3.4
Deliverable Name	Evaluation from an Industry Perspective with an industrial use case
Due Date	30/09/2018 (PM 36)
Deliverable Lead	asc-s
Authors	Johannes Demer (asc-s) Sebastian Wagner (asc-s) Alexander Frederic Walser (asc-s)
Responsible Author	Johannes Demer (asc-s) e-mail: Johannes.Demer@asc-s.de
Keywords	simulations, use cases, evaluation, industry perspective
WP	WP3
Nature	R
Dissemination Level	PU
Final Version Date	30/09/2018
Reviewed by	Erwin Laure, KTH Niclas Jansson, KTH Philipp Schlatter, KTH
MGT Board Approval	30/09/2018

Document History

Partner	Date	Comments	Version
asc-s	24/09/2018	Frist draft	0.1
asc-s	28/09/2018	Updated based on reviewer comments	0.2
KTH	30/09/2018	Final version	1.0

Executive Summary

This deliverable showcases a reflection of the improvements made in the ExaFLOW project under an industry perspective regarding necessity, scalability, usability and adaptability to a common standard development process in the industry.

We have summarized the automotive industry needs of the future regarding simulation processes and discuss the developments from ExaFLOW. Finally, we built up a commercial external aerodynamics use case of a sportscar and show the potential from Nektar++ regarding exascaling.

Contents

1 Introducing and discussion of the ExaFLOW developments

With the pressure of the global megatrend "digitalization", industry tries to adapt it to their development processes. In the future we will have digital twins which make it quite easy to implement design changes, visualize the results and even develop new products. This trend gets even pushed by the availability of newer and faster computation systems and effects also the world of Computational Fluid Dynamics (CFD). Simulations in the past included a lot of model simplification which led to low results accuracy, because of the limited hardware and software performance. From the hardware side it is in the future possible to run high order simulation with very fine mesh structures and accordingly high numerical resolutions. But also from the software side new methods and workflows have to be develop to make this possible. We from the ExaFLOW project believe that our developments have a significant influence on the simulation area in the future and will help to have more efficient and accurate simulation as well as to be able to simulate new physical phenomenons. The potential in the product development process for the numerical simulations is enormous.
If we look a little bit more specific at the current relationship between stationer and transient simulations used in the project development, the result in most cases is only a view processes built up as a transient simulation. This relates mostly on the missing numerical methods or on the complicate model preparation process. A big advantage brings here our implemented automatic mesh refinement (AMR) process. Now it is possible to simulate geometry changes in one simulation over the time and the user can look specifically into local flow features. We belief the workflow from our flagship run, the incompressible flow over an airfoil, can be easily adopted to other simulation cases in industry. An example from the automotive sector could be the simulation of the spinning wheels during a flow simulation to build up exactly the induced vorticities, which has the most influence on the air drag coefficient from a vehicle.
Let us assume the available CPUs numbers in the industry will be increased by a factor of ten over the next three years, what is quite realistic. More and more high accurate simulations will be then running with smaller element size and smaller time steps. To keep the same realtime of the simulation or even reduce it, it is necessary to optimize the parallelization processes. The performance optimizations applied in the ExaFLOW project are an important step towards this. Running simulations with a high number of CPUs brings new challenges to the preconditioning system. For example, for a simulation running on 1000 CPUs the volume mesh with its physical conditions is divided in 1000 identical groups. Each group is calculated on a single core and needs to exchange their border information with their neighbor, what results in high communication traffic between each group. In the future, with excascale calculation this will have an even greater impact on the simulation speed. In our mixed CG-HDG formulation we changed the discretization process of the spectral element methods to minimize the communication costs between each CPU.
Looking at the hardware of an HPC cluster, the system failure probability is also increasing with the number of CPUs, which will mean that when a running simulation is crashing due to hardware failure every calculated result gets lost - which relates to a big problem regarding economic efficiency and costs. Our fault

tolerance system implemented in Nektar++ is able to recover the simulation data from a memory checkpoint and make the simulation process especially for high CPU number calculation much more robust.
Complex simulations like a complete vehicle with a cooling model, are often divided into different modelling zones. To run this kind of simulations efficiently we developed error control methods for heterogeneous modelling.
Keeping aware of new trends regarding alternative hardware architectures, especially GPU hardware, which is getting cheaper and more widely used over the past years in the industry and thereby more attractive to the simulation world, we showed during our investigations the possibility to run Nek5000 on GPUs with good performance results.
Finally, with respect to energy efficiency we could reduce the energy consumption by 20%, what will help for an industry use to achieve the CO^2 goals from the development process.

2 Introduction of the automotive use case

In a billion-dollar market like the automotive sector different factors have an impact in engineering a new automobile. This section will describe the main challenges of a numerical development process and shows potential in improving it. The topics are, short time to result, economic efficiency, implementing new calculation methods, usability, accuracy and validation.
The simulation of external aerodynamics surrounding a vehicle is essential for the development of an effective automotive design for the car body. The Reynold Averaged Navier-Stokes (RANS) approach is a widely used method for this purpose. However, the RANS solution predicts only the mean flow field which rises to difficulties to analyze the time-dependent phenomena such as vortex shedding and flow separation. A promising solution to overcome this limitation is a Detached Eddy Simulation (DES) approach which utilizes both the RANS turbulent model for the boundary layer and the Large Eddy Simulation (LES) in separated regions to capture the fully three-dimensional turbulent movement [1]. Direct Numerical Simulation (DNS) provides even more accurate solutions with resolving all scales with high number of mesh cells and requires therefore much higher computational efforts as compared to the DES calculation. The computation efficiency is a key parameter for effective use of such massive computation methods.
As a part of ExaFLOW project, this technical report is devoted to describe the state-of the art modelling of a commercial software (Fluent) and the developments implemented in the open source code (Nektar++ [2]) with the automotive use case defined in [1]. The computations have been performed on the supercomputer Hazel Hen (position 8 of TOP500, 11/2015), a Cray XC40-system in HLRS [3].
The simulation focuses mainly on the rear wake of a sporty vehicle, therefore three submodels are taken into account. By means of the submodel, the computational performances of Fluent (DES) and Nektar++ (Continuous Galerkin method) have been compared with different number of processors. Based on the results, the relationship between the computational time and expenses is

presented, which comes up with the optimal range of the number of cores for each computation method.

2.1 Use of the ExaFLOW developments

For building up the automotive use case and make it comparable with a commercial workflow from the industry, the ExaFLOW developments were considered and used. The calculations were made with the latest software release from Nektar++ version 4.4.1, where the improvements, mentioned in the deliverables before, were already implemented. In our case on the commercial side Ansys Fluent version 18.01 was taken. In the following sections both simulation processes were introduced and compared.

3 Numerical setup

The numerical setup describes an overview about building a simulation in Nektar++ and Fluent and shows the difference between both software tools.

3.1 Geometry

Three Models are defined to explore the turbulent characteristic at the upper region of the vehicle [1]. Figure 1, 2 and 3 illustrates the computational domain and boundary conditions for the models. The size of the domain for model a is 7m, 1.3m, and 0.15m for the x, y and z direction, respectively. Model B is mirrored at the left side of model A and has a domain size of 7m x 2.6m x 0.15m. In Model C the upper front of the car body is also considered, the size domain is 15.3m x 5.2m x 1.8m.

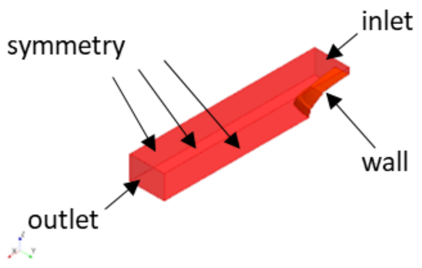


Figure 1: Model a (MA)

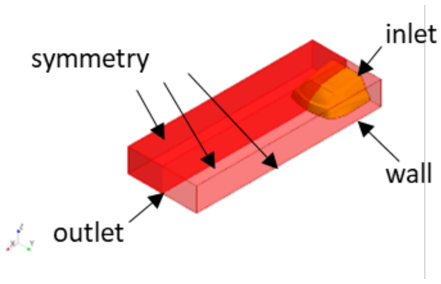


Figure 2: Model a and model b (MA and MB)

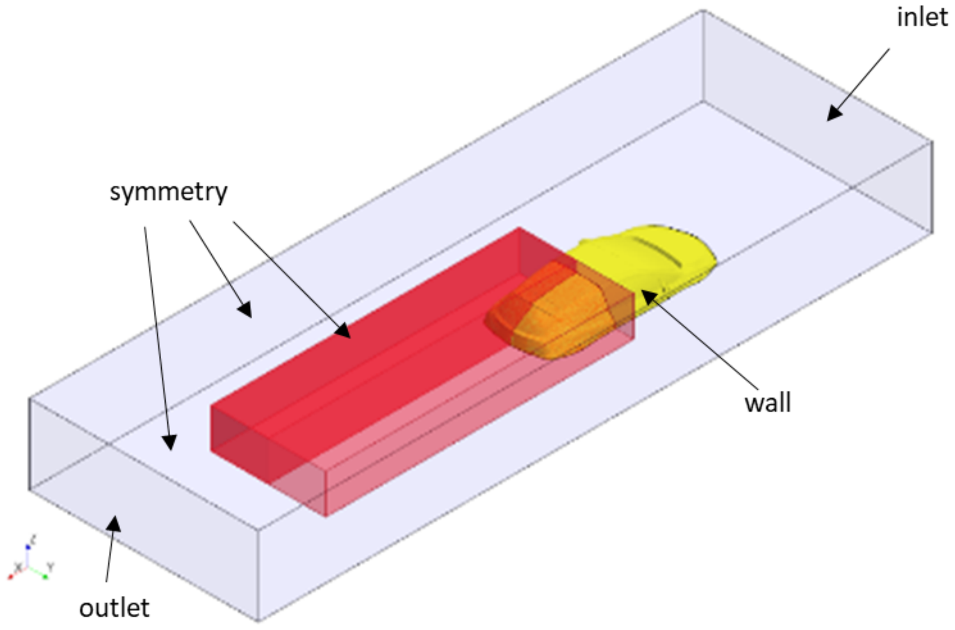


Figure 3: Model a, model b and model c (MA, MB & MC)

3.2 Mesh generation

According to the different computational method, the mesh for Nektar++ (finite-element method) is different from the mesh for DES (finite-volume method). In the presented use case, Fluent works with a structured hexahedral mesh and local refinement boxes for a finer mesh in important areas. The mesh for Nektar++ is unstructured and depending of the 2D surface sizes quality. Therefore, a lot of effort are pushed into the 2D geometry conditioning. Changing a finer CFD mesh can be done without generating a new finer mesh, as it is usual. In Nektar++, a simply increasing of the expansion factor for each cell and polynomial basis function is enough [2].

3.2.1 Finite volume mesh in fluent

For the generation of the computational mesh, the computational domain is discretized by volumetric cells based on the 2D surface mesh. For model computation, the DES employs the identical mesh used for the RANS simulation by ANSYS Fluent. The typical length of the elements for two-dimensional surface mesh is between 1~6mm. 7 prism layers are specified with growing cell size on the exterior surface of the vehicle to take into account the sharp velocity gradient in the boundary layer. The non-equilibrium wall function is utilized for the first cell near the wall. The 3D volumetric mesh is composed of a non-conformal hexcore mesh (hexaeder and tetrahedral) and can be observed in Figure 4. The total number of cells is ~21 mio, ~42 mio and ~41 mio for MA, MB, MC.

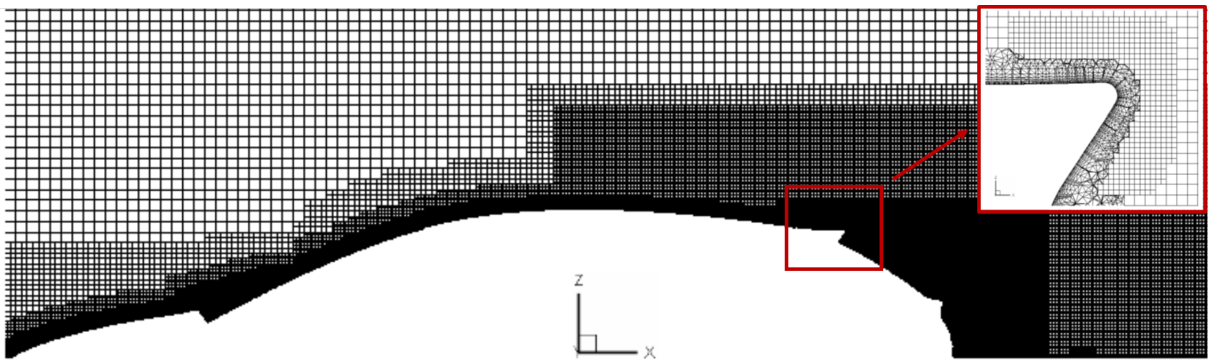


Figure 4: Fluent mesh, plane cut for MC in x-z direction at y = -0.13m; detail view at the spoiler

3.2.2 High oder mesh generation in Nektar++

Due to the various numerical method in Fluent and Nektar++, at the moment Nektar ++ is cannot run with the same mesh used in Fluent. Figure 5 describes the pre-processing procedure for Nektar++. ANSA pre-processor is used for the mesh generation. Once the mesh is created by ANSA pre-processor, the mesh is converted to xml format for Nektar++. In the data converting procedure, additional software such as Starccm+ and Tecplot 360 V2018 is utilized.

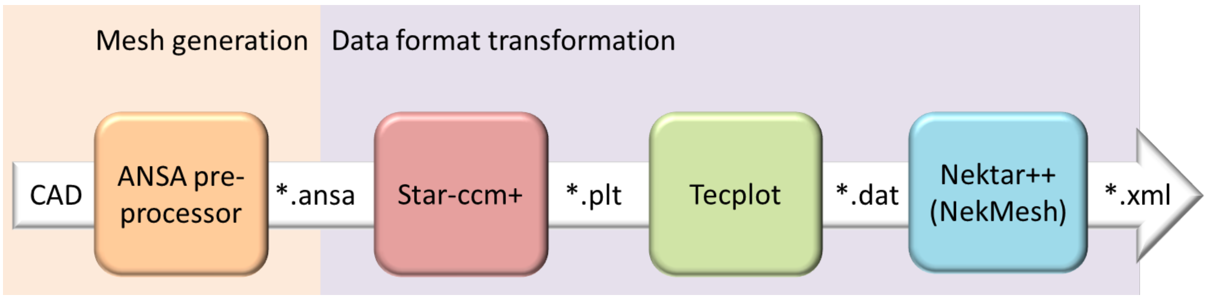


Figure 5: Mesh transformation and generation process for Nektar++

The triangle 2D surface mesh is generated in Ansa v18.1 with a cell size between 4mm and 40mm. For the volume mesh, first one prism layer at the car surface with a size of 3.6mm is generated, secondly the tetrahedral mesh with the growth rate, max shell size and FLUENT skewness are set to 1.1, 40 and 0.6. Figure 6 shows a detailed view of the high order mesh at y = -0.13m.

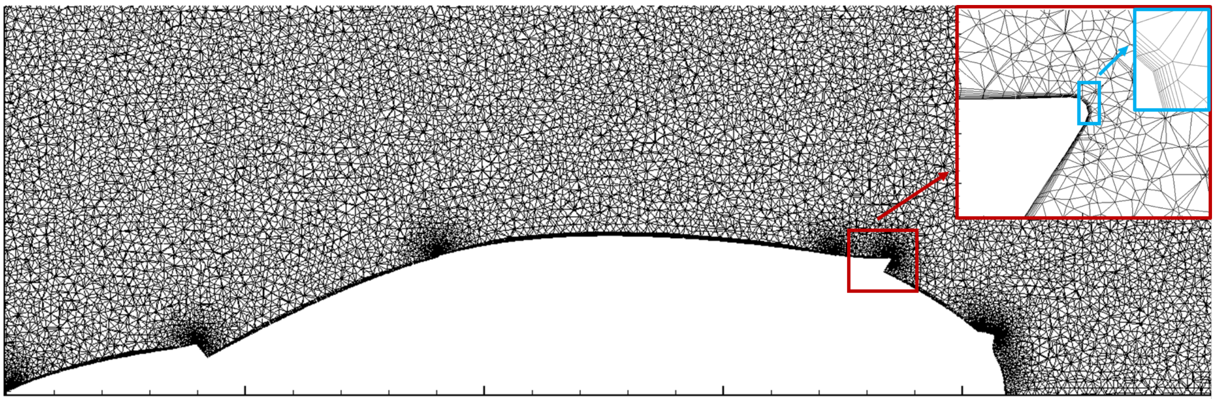


Figure 6: Nektar++ mesh, plane cut for MC in x-z direction for y = -0.13m, detail view at the spoiler

3.2.3 Number of elements

According to the different calculation methods, mentioned in section 3.2, the number of mesh elements are clearly smaller in Nektar++ than in Fluent. But the resolution is even higher because of the polynomial order of four for each model. The different numbers for each model and calculation method are visualized in table 1.

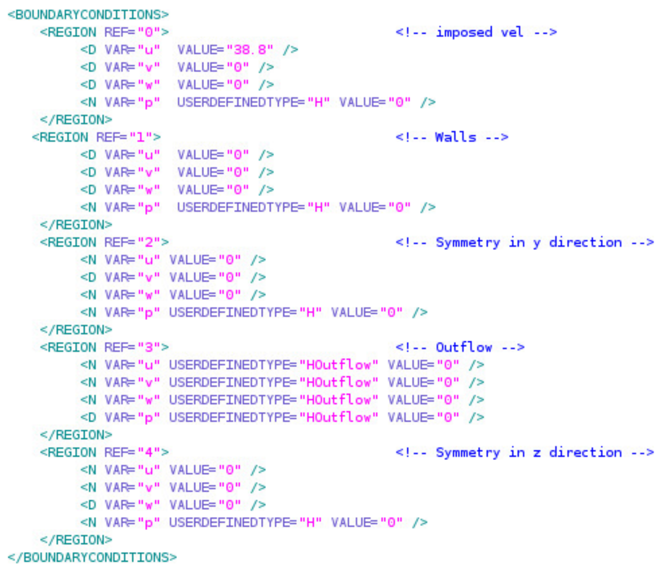
	model A	model B	model C
Fluent	21,4 mio	42,8 mio	40,5 mio
HOM Nektar++	2,7 mio	9 mio	25 mio

Table 1: Comparison of mesh elements between Fluent and Nektar++

3.3 Physical setup

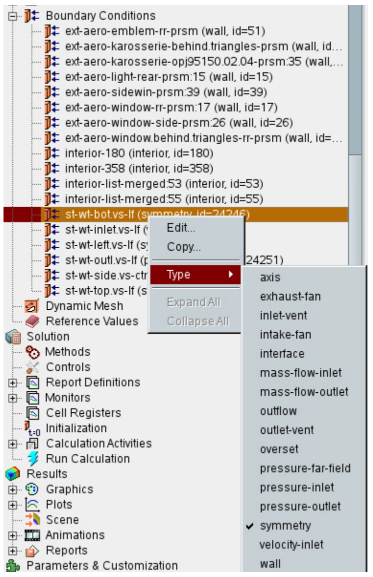
Each model (MA, MB and MC) is simulated like a wind tunnel case. Therefore, the front is defined as velocity inlet and the back as pressure outlet. To reduce the wall effects from the wind tunnel, on all side's symmetry conditions are used. In Fluent the symmetry conditions are already implemented and can be choosen by clicking. In Nektar++ they must be defined, which is shown in figure 7. For the velocity in surface orthogonal direction a Dirichlet condition and in surface direction von Neumann conditions are defined, all values are set to zero. A von Neumann condition is also set for the pressure field.

Nektar++



```
<BOUNDARYCONDITIONS>
    <REGION REF="0">                                  <!-- imposed vel -->
         <D VAR="u"  VALUE="38.8" />
         <D VAR="v"  VALUE="0" />
         <D VAR="w"  VALUE="0" />
         <N VAR="p"  USERDEFINEDTYPE="H" VALUE="0" />
    </REGION>
   <REGION REF="1">                                   <!-- Walls -->
         <D VAR="u"  VALUE="0" />
         <D VAR="v"  VALUE="0" />
         <D VAR="w"  VALUE="0" />
         <N VAR="p"  USERDEFINEDTYPE="H" VALUE="0" />
    </REGION>
    <REGION REF="2">                                  <!-- Symmetry in y direction -->
         <N VAR="u" VALUE="0" />
         <D VAR="v" VALUE="0" />
         <N VAR="w" VALUE="0" />
         <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
    </REGION>
    <REGION REF="3">                                  <!-- Outflow -->
         <N VAR="u" USERDEFINEDTYPE="HOutflow" VALUE="0" />
         <N VAR="v" USERDEFINEDTYPE="HOutflow" VALUE="0" />
         <N VAR="w" USERDEFINEDTYPE="HOutflow" VALUE="0" />
         <D VAR="p" USERDEFINEDTYPE="HOutflow" VALUE="0" />
    </REGION>
    <REGION REF="4">                                  <!-- Symmetry in z direction -->
         <N VAR="u" VALUE="0" />
         <N VAR="v" VALUE="0" />
         <D VAR="w" VALUE="0" />
         <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
    </REGION>
</BOUNDARYCONDITIONS>
```

Fluent

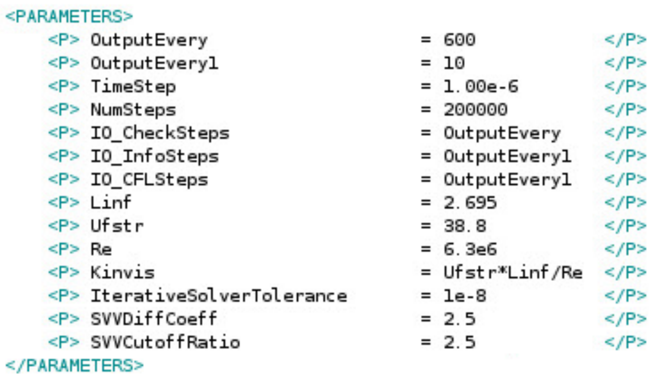


```
Boundary Conditions
    ext-aero-emblem-rr-prsm (wall, id=51)
    ext-aero-karosserie-behind.triangles-prsm (wall, id...
    ext-aero-karosserie-opj95150.02.04-prsm:35 (wall,...
    ext-aero-light-rear-prsm:15 (wall, id=15)
    ext-aero-sidewin-prsm:39 (wall, id=39)
    ext-aero-window-rr-prsm:17 (wall, id=17)
    ext-aero-window-side-prsm:26 (wall, id=26)
    ext-aero-window.behind.triangles-rr-prsm (wall, id=...
    interior-180 (interior, id=180)
    interior-358 (interior, id=358)
    interior-list-merged:53 (interior, id=53)
    interior-list-merged:55 (interior, id=55)
    st-wt-bot.vs-lf (symmetry, id=24246)
    st-wt-inlet.vs-lf (
    st-wt-left.vs-lf (s
    st-wt-outl.vs-lf (p                24251)
    st-wt-side.vs-ctr
    st-wt-top.vs-lf (s
  Dynamic Mesh
  Reference Values
Solution
  Methods
  Controls
  Report Definitions
  Monitors
  Cell Registers
  Initialization
  Calculation Activities
  Run Calculation
Results
  Graphics
  Plots
  Scene
  Animations
  Reports
Parameters & Customization
```

Figure 7: Comparison of the boundary implementation between Nektar++ and Fluent

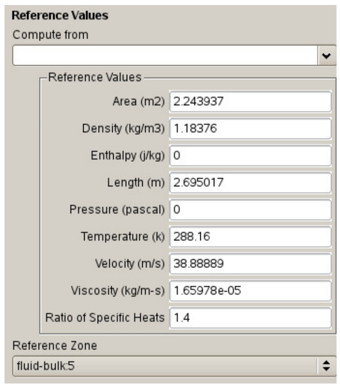
To simulate a constant drive with 140km/h, the constant velocity is set at the inlet in [m/s]. The characteristic length of the car is 2.695m and a high Reynoldsnumber of Re = 6.3×10^6 is used. The stability parameters in Nektar++, like SVVDiffCoeff and SVVCUtoffRatio are slightly increased regarding the constant value to 2.5. In Fluent the simulation setup is working with constant values. The simulation parameters are visualized in figure 8.

Nektar++



```
<PARAMETERS>
    <P> OutputEvery                    = 600            </P>
    <P> OutputEvery1                   = 10             </P>
    <P> TimeStep                       = 1.00e-6        </P>
    <P> NumSteps                       = 200000         </P>
    <P> IO_CheckSteps                  = OutputEvery    </P>
    <P> IO_InfoSteps                   = OutputEvery1   </P>
    <P> IO_CFLSteps                    = OutputEvery1   </P>
    <P> Linf                           = 2.695          </P>
    <P> Ufstr                          = 38.8           </P>
    <P> Re                             = 6.3e6          </P>
    <P> Kinvis                         = Ufstr*Linf/Re  </P>
    <P> IterativeSolverTolerance       = 1e-8           </P>
    <P> SVVDiffCoeff                   = 2.5            </P>
    <P> SVVCutoffRatio                 = 2.5            </P>
</PARAMETERS>
```

Fluent



Reference Values

Compute from

Reference Values

Area (m2)	2.243937
Density (kg/m3)	1.18376
Enthalpy (j/kg)	0
Length (m)	2.695017
Pressure (pascal)	0
Temperature (k)	288.16
Velocity (m/s)	38.88889
Viscosity (kg/m-s)	1.65978e-05
Ratio of Specific Heats	1.4

Reference Zone

fluid-bulk5

Figure 8: Comparison, simulations parameters between Nektar++ and Fluent

3.4 Preprocessing

During the preprocessing work everything essential for monitoring and evaluating the simulation over the calculation and afterwards must be implemented. Therefore, several points, planes, vorticity and functions were defined which are shown in figure 9, 10, 11, 12 and 13.

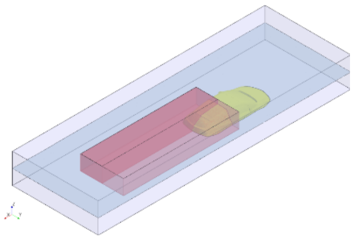


Figure 9: Plane z = 1,5 m

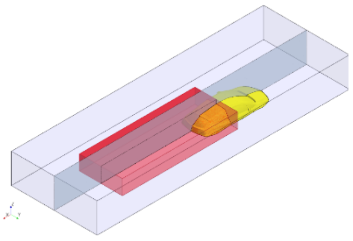


Figure 10: Plane y = -0,13 m

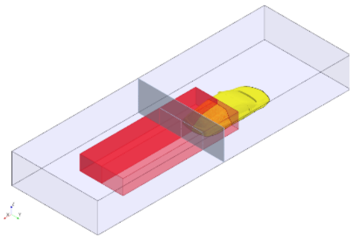


Figure 11: Plane x = 7,18 m

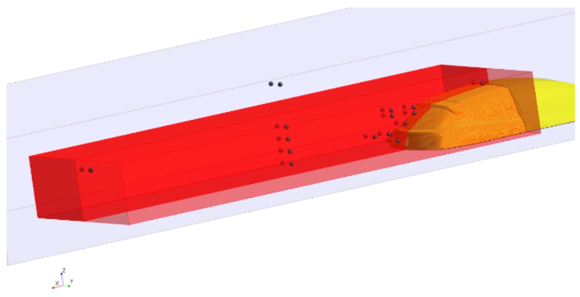


Figure 11: Position of points, 3d view

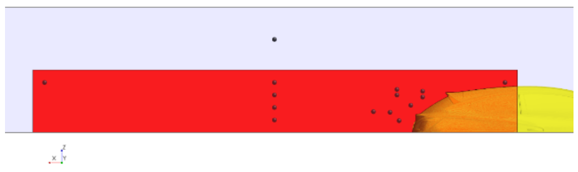
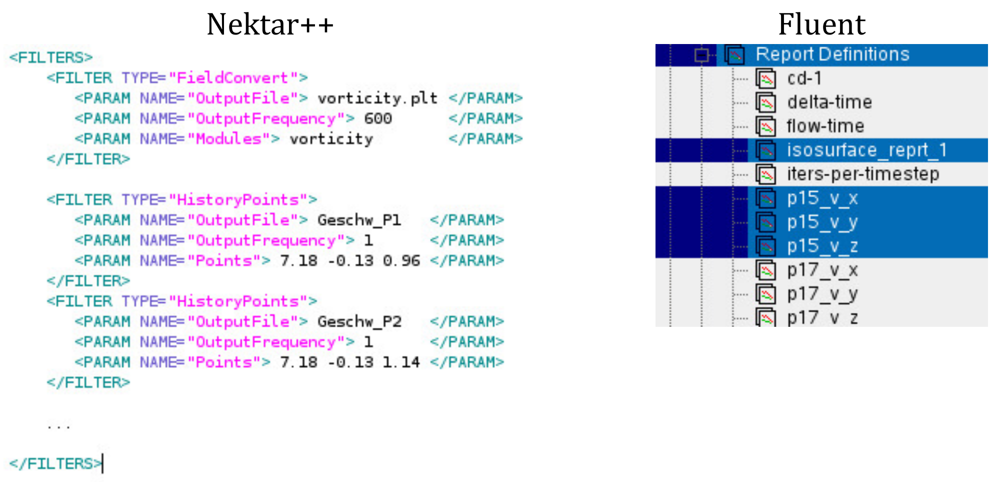


Figure 12: Position of points, side view

First of all, in Fluent the points, planes, and vorticity had to be defined, afterwards report definitions can be defined and automatically exported during the simulation process. Additionally, they could be visualized in the graphic window very easily. The process in Nektar++ is a little bit different. An intermediate step must be performed to make the Nektar++ results comparable with Fluent. Therefore, a complete solution file which includes also the calculated vorticity is exported frequently every Δt = 6x10^{-6}s meantime. Which means for MA in Nektar++ one output file every 600 iterations and for Fluent every 6 iterations.

The points and the solution file are implemented as a filter. In the figure 13, below the different filters and reports are visualized [9].



```
<FILTERS>
    <FILTER TYPE="FieldConvert">
       <PARAM NAME="OutputFile"> vorticity.plt </PARAM>
       <PARAM NAME="OutputFrequency"> 600      </PARAM>
       <PARAM NAME="Modules"> vorticity        </PARAM>
    </FILTER>

    <FILTER TYPE="HistoryPoints">
       <PARAM NAME="OutputFile"> Geschw_P1    </PARAM>
       <PARAM NAME="OutputFrequency"> 1       </PARAM>
       <PARAM NAME="Points"> 7.18 -0.13 0.96 </PARAM>
    </FILTER>
    <FILTER TYPE="HistoryPoints">
       <PARAM NAME="OutputFile"> Geschw_P2    </PARAM>
       <PARAM NAME="OutputFrequency"> 1       </PARAM>
       <PARAM NAME="Points"> 7.18 -0.13 1.14 </PARAM>
    </FILTER>

    ...

</FILTERS>
```

```
Report Definitions
    cd-1
    delta-time
    flow-time
    isosurface_reprt_1
    iters-per-timestep
    p15_v_x
    p15_v_y
    p15_v_z
    p17_v_x
    p17_v_y
    p17_v_z
```

Figure 13: Comparison, implementation process postprocessing between Nektar++ and Fluent

3.5 Simulation

The transient simulation process is divided into three phases, two initialization phases and one calculation phase, specified in figure 14. The initial phase is seen here as stabilization process for the flow and the calculation phase is used for the investigations.

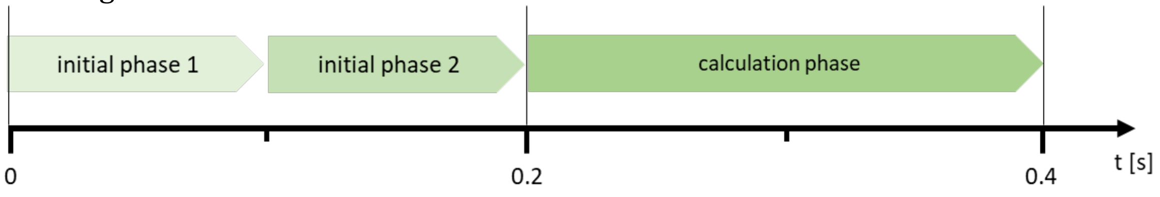


Figure 14: Schema of the simulation process

3.5.1 Initialization processes

In Nektar++ the first initial phase is started with a quite small timestep size of t = 1x10^{-7}s to get the turbulent flow stable. After a view 1000 iteration it could be increased to t = 2x10^{-6}s for MA, t = 4x10^{-6}s for MB and t = 2.5x10^{-6}s for MC. By catching a usable timestep size the second initial phase started. Here the simulation is running at least until the flow in x direction was able to cross the wind tunnel model (MA and MB) once. Due to economic reason, for MC the same initialize setup was chosen. With a simulation Real time of 0.2s the initial process is finished, and we assumed to have a quasi-stationary flow condition. In figure 15 and 16, the velocity magnitude of the front part of the y plane for Nektar++ and Fluent is visualized. It shows some differences at the back of the vehicle, the backwater area by Nektar++ is clearly more distinctive.

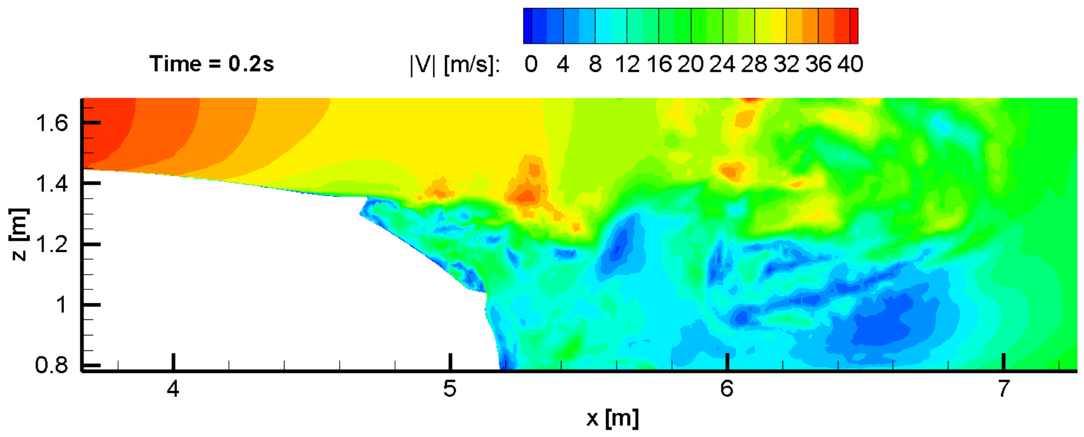


Figure 15: MA velocity magnitude in Nektar++ at the front of plane y

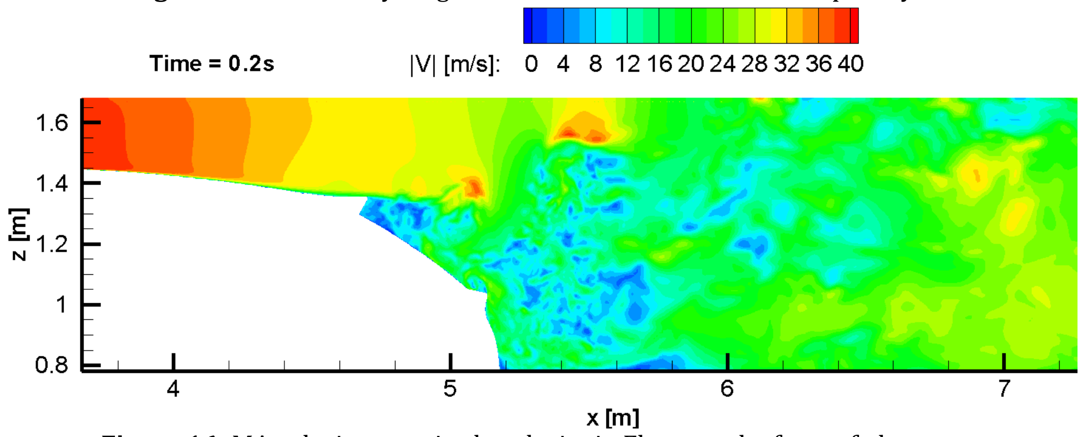


Figure 16: MA velocity magnitude velocity in Fluent at the front of plane y

3.5.2 Calculation phase

With a quasi-stationary starting point at a Simulation-Realtime of t = 0.2s the calculation phase initiates. The duration is defined with Δt = 0.2s and the end point of the calc phase with t = 0.4s. Regarding to the different cell sizes from MA, MB and MC and keeping in mind the profitability of the calculations in Nektar++, the amount of iterations and the number used CUPs varies. In figure 17 is the calculation setup for all models visualized.

	Nektar++ MA	Nektar++ MB	Nektar++ MC	Fluent
time step size [s]	2e-6	4e-6	2.5e-6	1e-4
start time [s]	0.2	0.2	0.2	0.2
end time [s]	0.4	0.4	0.4	0.4
time step iterations	100000	50000	80000	2000
inner iterations				20
number of CPUs	1920	3840	7680	480

Figure 17: Simulation setup, calc phase numerical investigation

In figures 18, 19, 20 and 21 below, the velocity magnitude and vorticity for MA at a Realtime of t = 0.4s from the front of plane y is visualized. Both simulations show a similar diffuse flow behavior at the top of the car body and also a similar turbulence structure at the back of the vehicle. The enstrophy is slightly a little bit more distinctive in Fluent. Reason to that might be the lower momentum in Nektar++ boundary layer which leads to less energetic enstrophy.

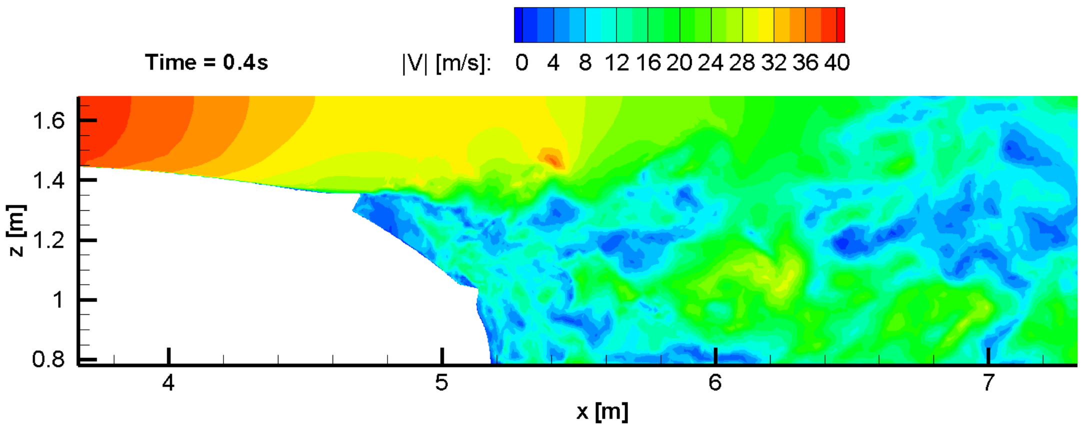


Figure 18: MA, velocity magnitude in Nektar++ at the front of the y plane

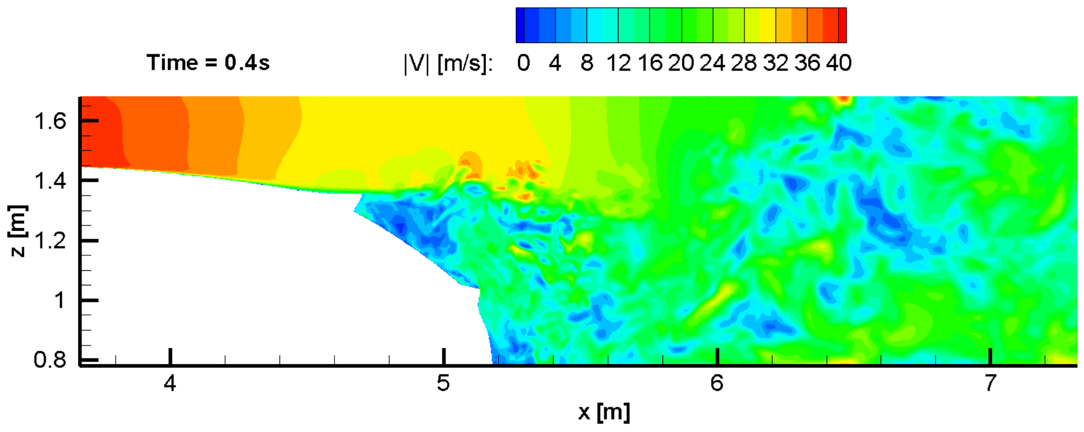


Figure 19: MA, velocity magnitude in Fluent at the front of the y plane

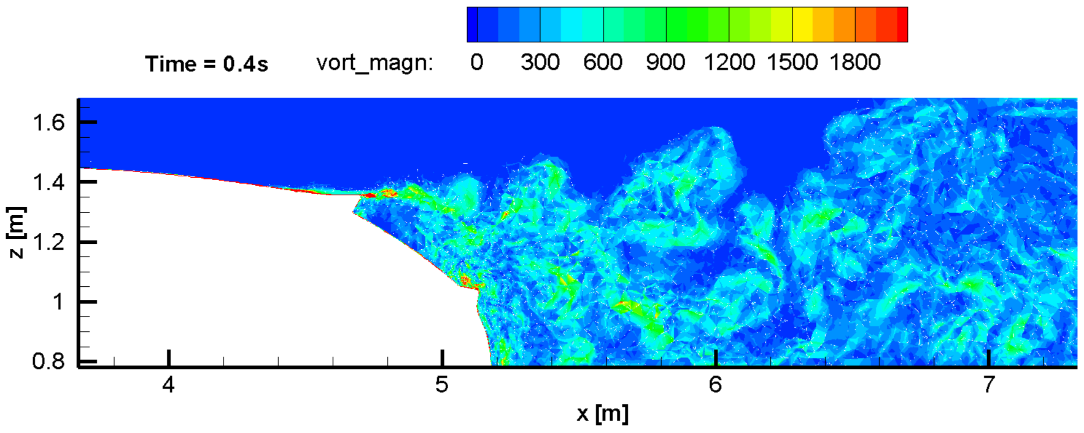


Figure 20: MA enstrophy in Nektar++ at the front of plane y

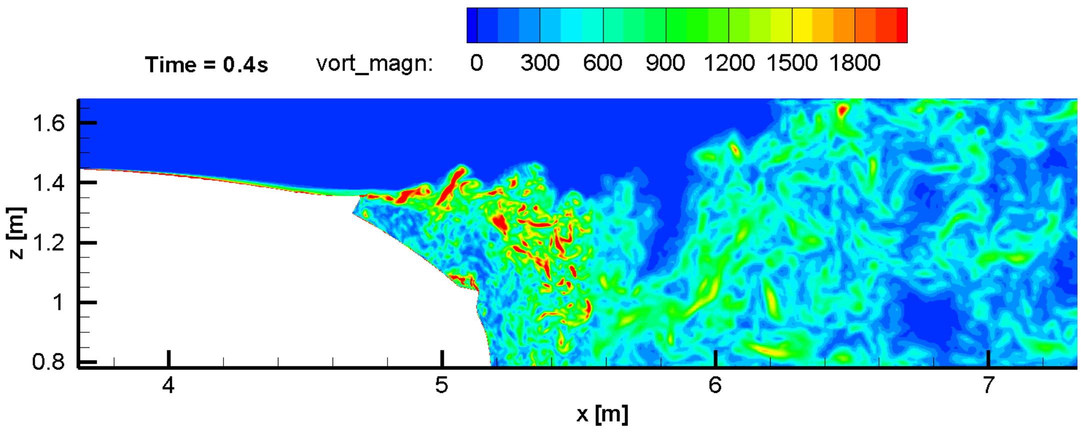


Figure 21: MA enstrophy in Fluent at the front of the y plane

3.6 Postprocessing

Since there is no user interface in Nektar++, the visualization and postprocessing was performed external in Tecplot 360 V2018. Therefore, the solution files from Nektar++ and Fluent were exported during the simulation already in the Tecplot format *.plt. The physical investigation was focusing on the velocity components and their behavior during the simulation time. The workflow was automated by scripting the input and output process, just as the plane extracting and the averaging during simulation time.

3.7 Comment

For saving some calculation time, at the beginning it was planned to use a steady RANS solution as an boundary condition for the transient calculation with Nektar++ and Fluent. Since we were not able to run the FieldConvert function "interppointdatatofld" from Nektar++ which should interpolate given values provided in a text *.pts - file to a usable inlet *.fld - file for Nektar++, we decided to use the described simulation process from chapter 2.5. In a further standard industry use won't be an initial simulation with a different software tool. The time and finance cost for setting up that kind of simulation is disproportionate compared to the simulation cost from our developed initialization process. At least Nektar++ can show here his potential under real industry conditions.

4 Investigations

The investigation is focused on the usability of Nektar++, the numerical accuracy between Fluent and Nektar++, the economics efficiency and on strong and weak scaling regarding exascale.

4.1 Hdf5 format

The number of files in simulations are continuously growing and slowing down the data management process. In Nektar++ the standard format was employed to reduce the number of files per simulation output.

The improvement becomes clear by comparing the number of files after the calculation has finished successfully. Table 2 show the total number of output files, partitioning included, from calculations with 3840, 5760 and 7680 cores. With the xml-format (n_{cores}+1) files are generated- with the Hdf5-format, only three files in total are generated for each MPI Rank (Table 2).

NO. OF FILES

MPI RANKS	3840	5760	7680
XML-FORMAT	7682	11522	15362
HDF5	3	3	3

Table 2: Comparison of the number of files between xml and hdf5 – format

4.2 Performance analysis HOM Nektar++ vs. Fluent

4.2.1 Partitioning

One of the bottlenecks within the calculation process with Nektar++, is the partitioning of the simulation domains. Up to now, this process is not parallelizable which means it can only run on a single core. Hence, the more partitions are needed, the process duration will increase depending on the core number used for the calculation. For the following scalability test, the partitioning time is not included in the Realtime, because the focus is there on weak and strong scaling regarding code performance. For a cost and time to result consideration it must be supplemented.

In Figure 22 the time courses for model A (MA), model B (MB) and model C (MC) are presented. Due to the long calculation times (t>24h), for MC, the partitioning with 480 -1920 cores is not considered in this report.

The curves of MA and MB have nearly linear courses. By increasing the element number, a factor of 2.5 (MC/MB) the partitioning time shows an exponential course. For 3840 and 5760 cores, the time increases by a factor of 2.3, for 7680 cores the factor grows up to 4.3.

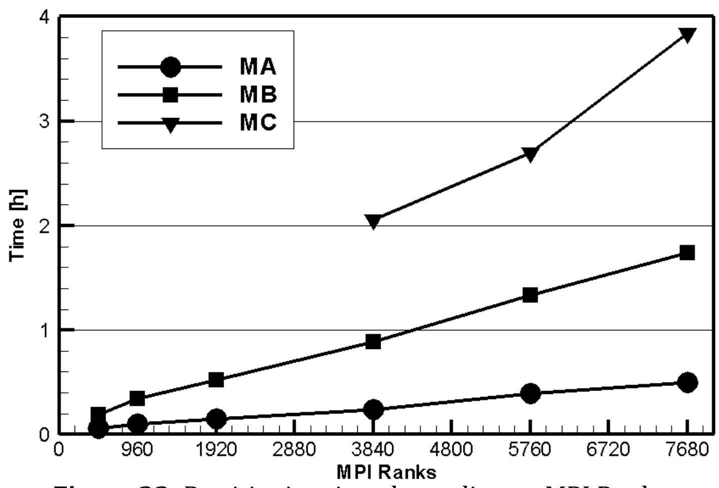


Figure 22: Partitioning time depending on MPI Ranks

4.2.2 Input/Output (I/O) process

The I/O process includes the data loading and writing processes, such as reading the mesh data, the Session, initial and boundary conditions) as well as the final state which is written out at the end of the calculation. Figure 22 show I/O times for MA, which represents the time difference between Realtime and Solvertime. Fluent show a nearly constant I/O time of approximately 0.14s (480 cores) - 0.16s (7680cores). The I/O time of Nektar++ increases at low MPI Ranks (960 cores). Afterwards the curve decreases drastically for about 82% to 0.36h (1920 cores) until its minimum at 5760 cores. The turning point between both simulations is at 3840 MPI Ranks, with higher MPI ranks Nektar++ is getting more economically than Fluent.

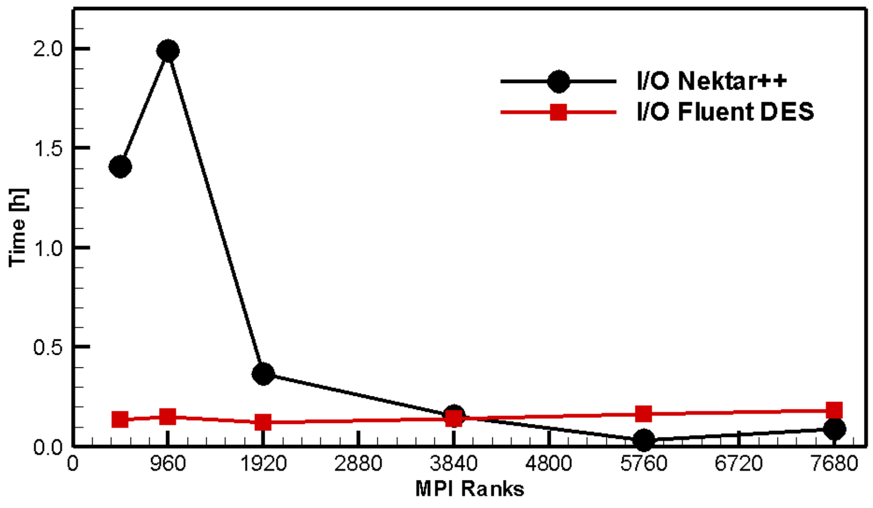


Figure 23: I/O time depending on the core number

4.2.3 Scalability test

In this section, the results of the scalability tests, are presented. This task was performed to show the code performance of Nektar++ in contrast to commercial software. As already mentioned, the difference between the simulation codes are the numerical methods. Nektar++ is based on the FEM, Fluent (DES) uses the FVM. In the FEM code, the polynomial expansion factors were set to 3 and 2 for velocity and pressure. Table 1 summarises the test conditions for the scalability tests. The calculations were started at a physical time of 0,2s and ended up at 0,22s. The different timestep sizes lead to an adaptation of their number (Table 3, line 5).

TESTCONDITIONS

PARAMETER	Nektar++	Fluent
INITIAL CONDITIONS	0.2s	0.2s
REYNOLDS NUMBER	$6.3x10^{-6}$	$6.3x10^{-6}$
TIME STEP SIZE	$2*10^{-6}$	$1*10^{-4}$
NO. OF TIME STEPS	10000	200
PHYSICAL TIME, ΔT	0.02s	0.02s

Table 3: Testconditions scalability test for Nektar++ and Fluent

4.2.3.1 Strong scaling performance between Nektar++ and Fluent

In figure 24 the Realtime and Solvertime graphs for both solvers are sketched. By the Solvertime only the calculation time per iteration is considered. The   On the x-axis the time is plotted, the y-axis represents the number of cores (MPI Ranks). The lowest core number was 480 cores, the maximum 7680 cores. Due to the fact that the partitioning process is not parallelizable, the Realtime includes only the Solvertime added to the I/O process.

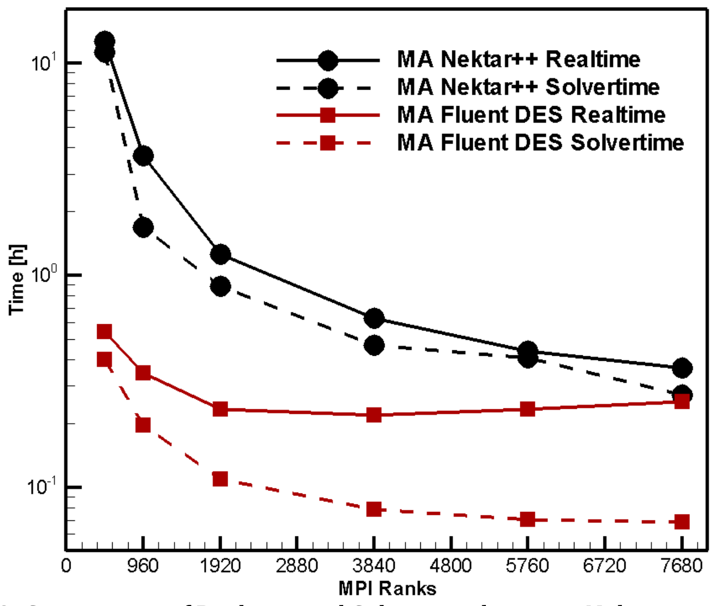


Figure 24: Comparison of Realtime and Solvertime between Nektar++ and Fluent, with a log scal on the time axis

Both graphs of Nektar++ show an exponential time fall. The Real time graph of Nektar++ reveals a continuously time fall a minimum at 7860 cores. Its Solvertime runs nearly parallel to the Realtime curve. After a steep gradient towards 1920 cores, the courses flatten which indicates a smaller time reduction as for lower MPI Ranks. In marked contrast to Nektar++, the Realtime of Fluent decreases until the usage of 1920 cores. Using more MPI Ranks won't lead to a time reduction, in contrary it increases. This fact shows that there is a limitation to increase the core number for this specific test scenario. Its Solvertime decreases exponential till 7680 cores which indicates on the one side the solver runs more efficiently and on the other side I/O process gets more complex with high MPI Ranks.

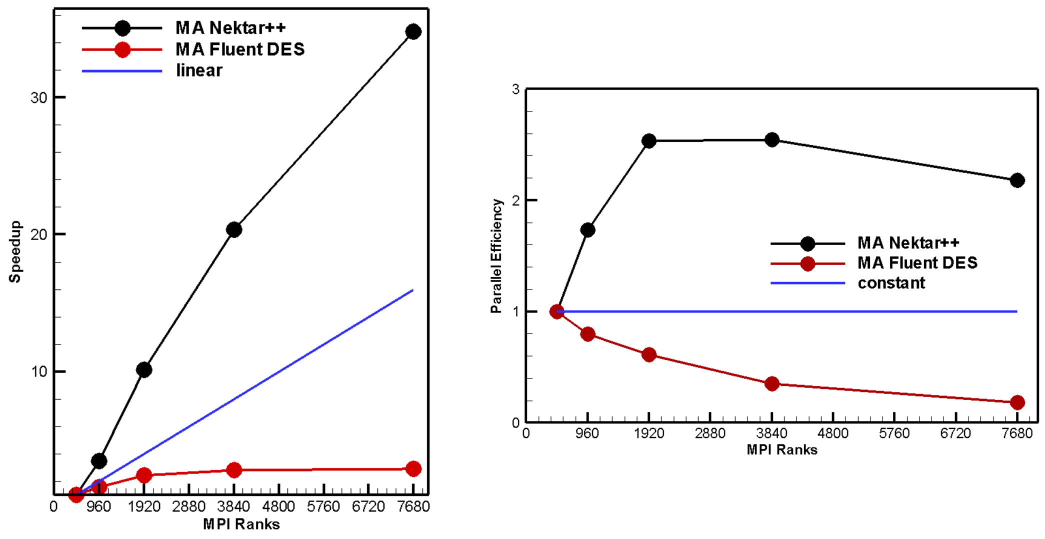


Figure 25: Speedup Ratio (a) and Parallel Efficiency (b) for MA

In figure 25, the Speedup Ratios for the scalability test with the linear graph are shown. The line graph clearly shows the super linear behavior of Nektar++. Its curve increases more rapidly than the linear one (Super linear behaviour [10]).

Fluent's curve starts with same Speedup ratio and increases till the maximum which is reached with the usage of 3840 cores (≈2,5). For 7680 cores, Nektar++s' graph keeps constant. The corresponding code efficiencies are presented in Figure 25b. Based on Figure 25a, for Nektar++, the best code efficiency is found by using 3840 cores. Between 3840 and 7680 cores the efficiency decreases keeping a value higher than 1. Fluents' curve decrease immediately when the core number rises up. This fact indicates the much parallelization capability of Nektar++.

4.2.3.2 Weak scaling performance of Nektar++ and Fluent

In this section the influence of the model size on the calculation time is investigated. The chart in figure 26 shows four lines with contrasting Realtime and Solvertime for MA with MB. With model B a higher calculation effort is achieved. Hence, a significant increase for both the Rea- land the Solver time was detected which is the reason only cores numbers bigger than 1920 cores are considered. The curves of model B show a continuously decreasing Realtime and Solvertime. A minimum Realtime is found for 5760.

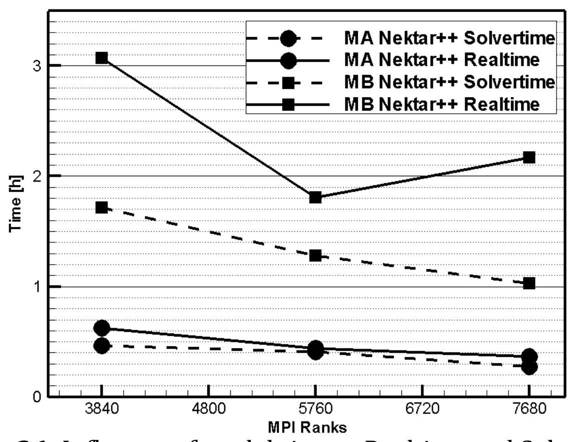


Figure 26: Influence of model size on Realtime and Solvertime

4.3 Numerical accuracy, comparison Nektar++ vs. Fluent

The previous section illustrated the reader scenarios how to reach the possible best scalability for the automotive use case. Besides the costs, also an accurate result-quality of submodels must be guaranteed. Hence, in this section, the physical accuracy compared to Fluent is shown.

As already mentioned, Fluent uses a DES method with a realizable k-e model. Nektar++ resolves the turbulent structures directly. All comparisons regarding the physical investigation were performed with model A. The considered time area starts after the initialise phase at t=0.2s with a duration of 0.2s till 0.4s. For the physical investigation immediate solutions were saved at every 0.006s (333 files). Figure 27 a-c show the submodels with the three planes which were considered in this investigation: x=7.18m, y=-0.13m and z=1.5m.

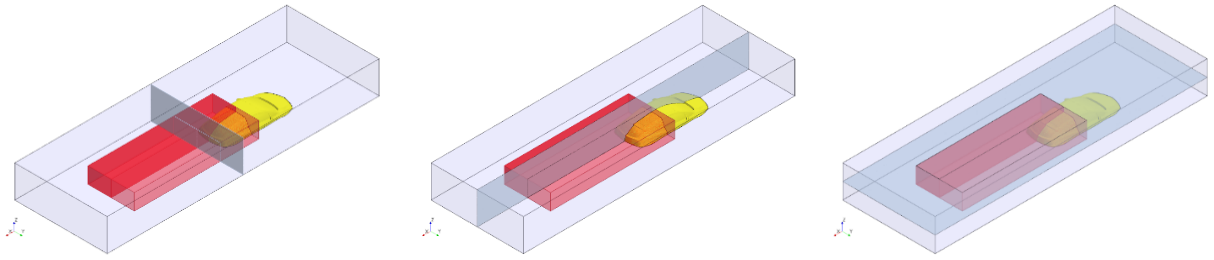


Figure 27: Planes to investigate the accuracy of Nektar++ compared to Fluent

4.3.1 Averaged velocities

Figure 28a shows the averaged velocities in x-direction (u) for Nektar++ at the position of x=7.18m (view in flow direction). In Figure 28b, the same results, calculated in Fluent are presented. Both pictures show the same tendency, the highest velocities are found in the right half of the planes and decrease to the left side.

On the right side of the plane the results look very similar because the influence of the car on the flow is low in this area. Differences can be found on the left side, behind the middle of the car where the strongest velocity gradients can be found.

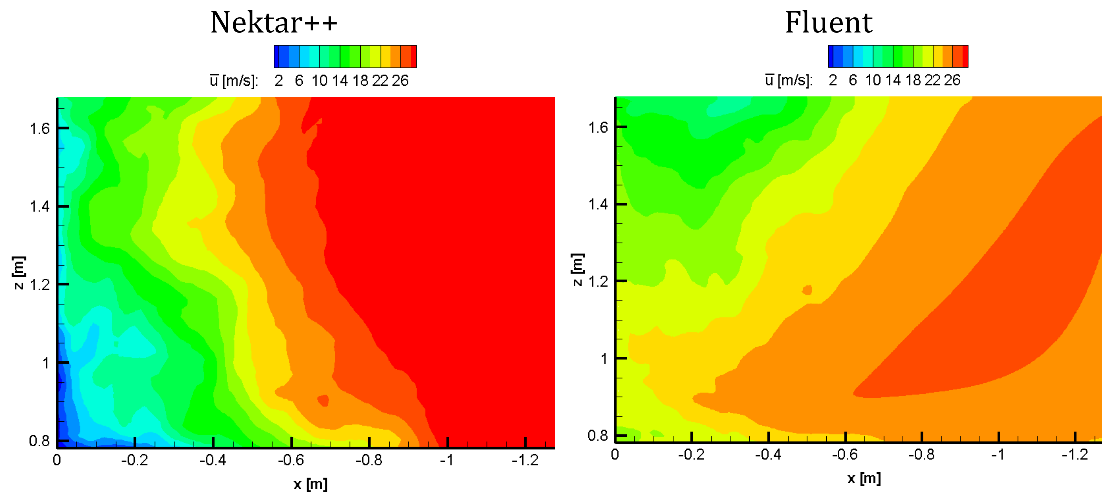


Figure 28: Averaged velocity $\bar{\mathbf{u}}$ in x-direction within the x-z plane at x=7.18m

Figure 29 shows the planes in x-z-direction at y=-0.13. Both snapshots show a nearly identical velocity course above the car. In the return flow area some discrepancies are shown. The reason to that might be the application of the in the FV- method, in Fluent.

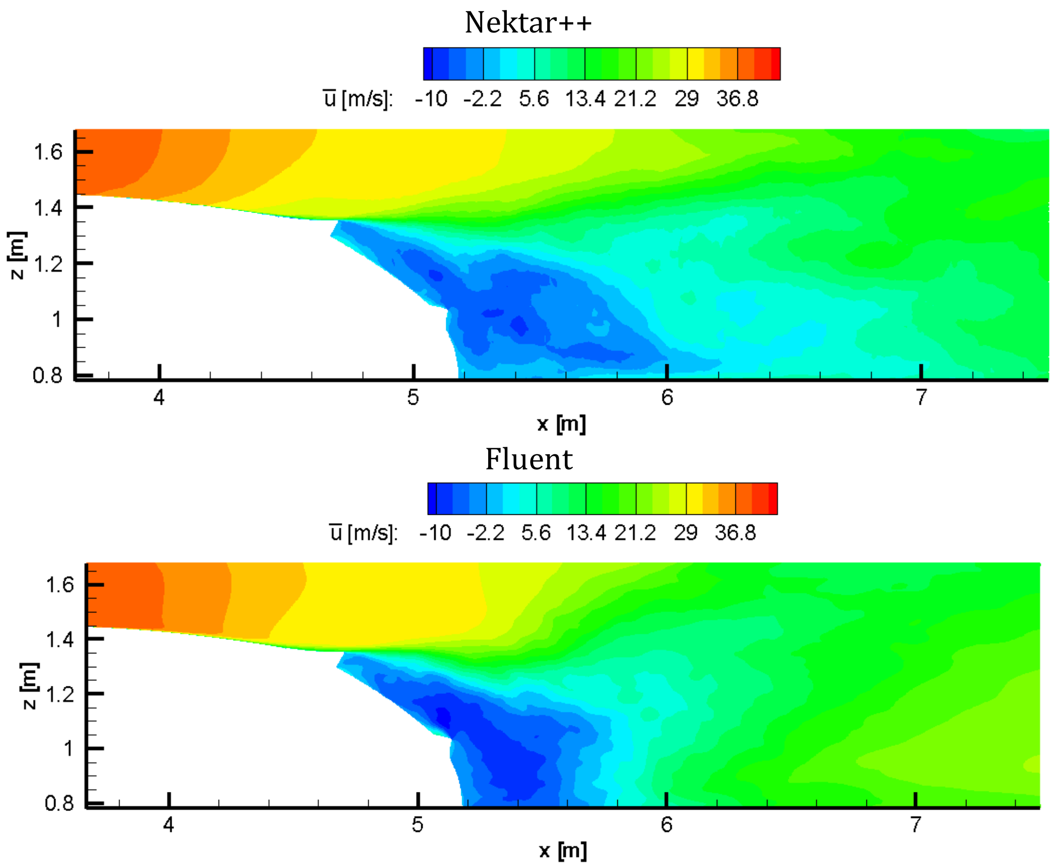


Figure 29: Averaged velocity $\bar{\mathbf{u}}$ in x-direction within the x-z plane at x=7.18m

The planes at z= 1.5m in x-y directions are presented in Figure 30. One sees that the averaged velocity looks nearly similar in both snapshots. Due to the car surface placed at the inlet the highest velocities are found in this area. Somewhat more to the right, the averaged velocity decreases due to the bigger cross-section area.

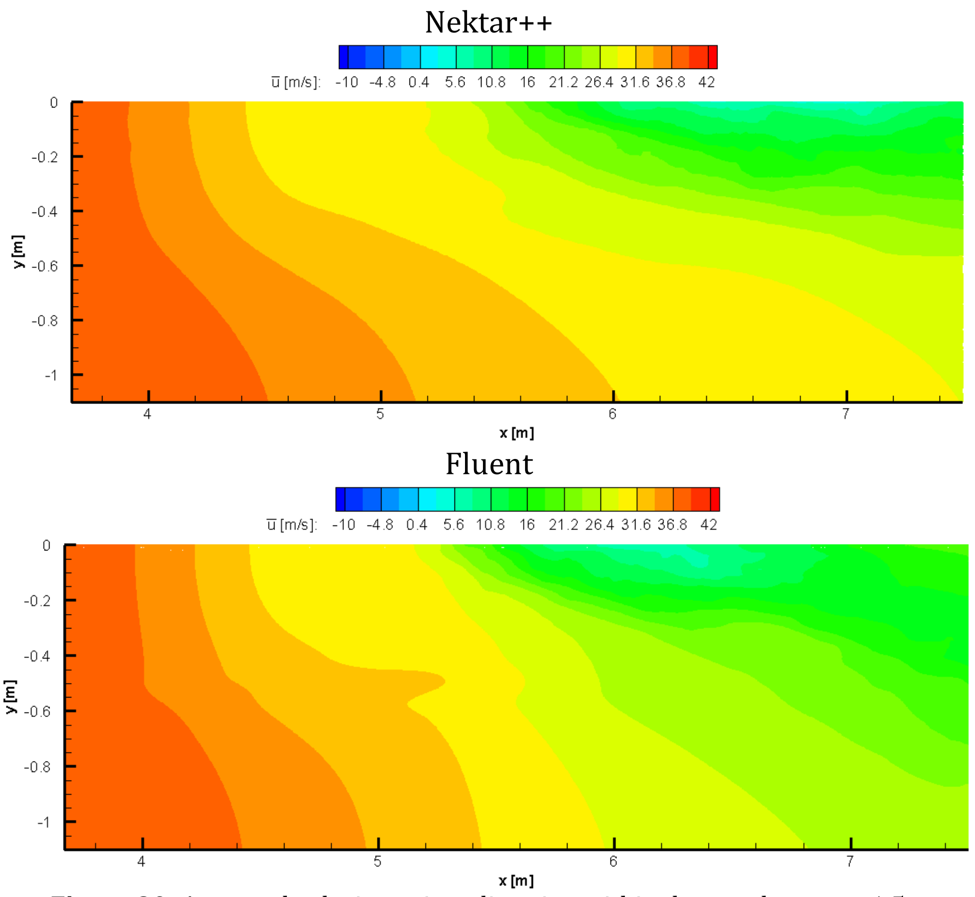


Figure 30: Averaged velocity $\bar{\mathbf{u}}$ in x-direction within the x-y plane at z=1.5m

4.3.2 History - velocity data

In order to compare exact velocities over time, some history points were defined (Section 2.4.). The three velocity components in x-, y- and z-components (u, v, w) were analyzed. The following points were defined for the velocity comparison in model A:

- Point 3 (7.18|-0.13|1.32)
- Point 18 (5.04|-0.13|1.29)

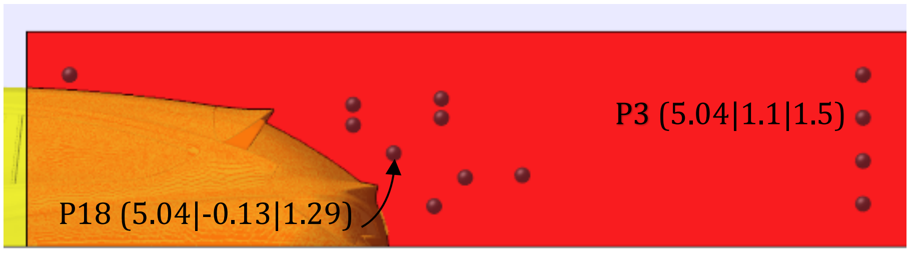


Figure 31: Chosen history points for investigating the temporal velocity course

Figure 31 show the places where Point 3 and Point 18 are located. Figure 32 and 33 present the several velocity components for the considered calculation phase. It gets clear that they have similar courses. Compared to Fluent, Nektar++ shows more sharp swings in its course (P 18 t= 0.25s). Nektar++ shows some big swing in its graph which are only weak pronounced in Fluent's velocity data. This fact might be due to lower diffusion of the numerics but has to investigated with high order simulations.

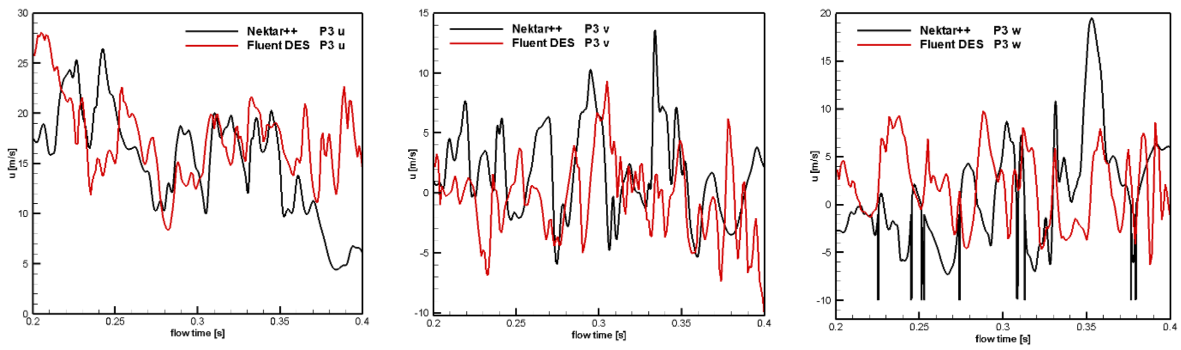


Figure 32: Point 3, history data of x-, y- and z-velocity components (u,v,w)

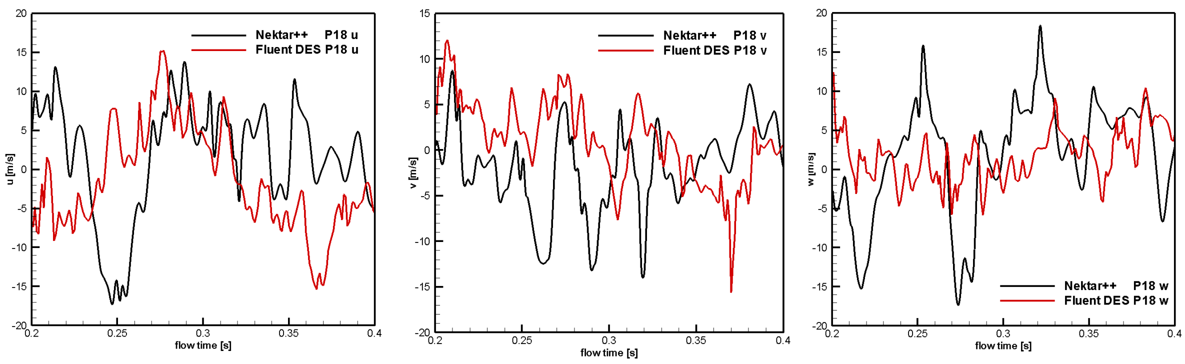


Figure 33: Point 18, history data of x-, y- and z-velocity components (u,v,w)

4.3.3 Costs

In this section, the costs of using a high-performance computing system are estimated. They are calculated for the computing System Hazel Hen at HLRS. The cost rate is 0.89€ per node. One node includes 24 cores and is only valid for users from universities or research facilities in Germany [3]. The user costs are calculated with Equation 1:

$$Calculation\ costs = 0.89€ \cdot t \cdot n_{nodes}$$

where t means the total Realtime and n_{nodes} represents the used number of nodes. All current calculations were run with 24 cores per node. In Figure 34 the user costs are shown, referring to the scalability results from section 24 for Nektar++ and Fluent (MA).

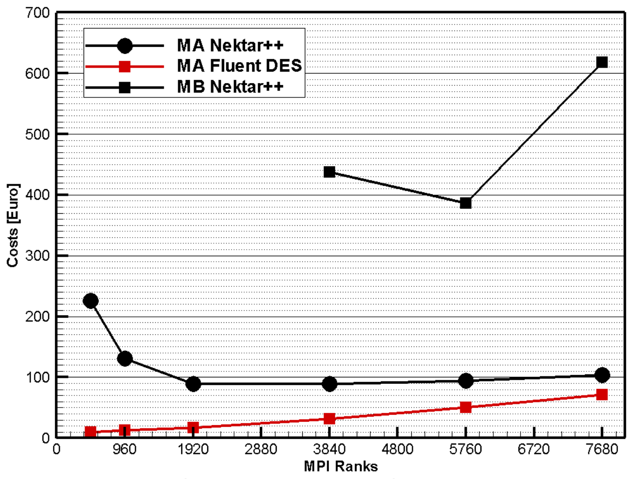


Figure 34: Comparison of calculation costs for using the high-performance computing system Hazel Hen at HLRS

Fluents calculation costs increase monotonic against the core number. In contrast, the application of Nektar++ results in a cost reduction until 3840 cores are used. When bigger core numbers are applied, the costs increase marginal. Finally, the results reveal that based on the much remarkable lower Real- times of Fluent at low core numbers (Figure 34, 480 cores) these calculations are much cheaper applied on this automotive use case. If we keep the cost tendency, which is related to the Realtime, between model A from Nektar++ and Fluent under consideration regarding exascale, the conclusion will be, that there is an efficiency turnover point where Nektar++ gets cheaper than Fluent.

5 Conclusion and Future Work

The comparison between Nektar++ and Fluent shows, yes, it is possible to rebuild an industry use case with a similar numerical accuracy in Nektar++. A high advantage is here related to developments during the ExaFLOW project. As mentioned in Section 3.2.2., the input and output process with HDF5-format shows quite a good performance by increasing the CPU numbers. The implemented standard HDF5-format reduces the number of files at least about ~66% per calculation, which makes it possible to run the numerical investigation. Strong scaling effects with a parallel efficiency better than a constant value of 1 confirm the claim of exascale being feasible.
Currently, in the industry we must be aware of three main factors, the total cost for running a simulation, the accuracy of the results and finally about the realtime of the simulation. Comparing Figure 24 and 34, the main investigation result regarding economic efficiency was that Nektar++ needed 7680 CPUs and Fluent 980 CPUs for a comparable realtime. Thus, a simulation with Fluent is eight times cheaper than with Nektar++. Keep also under consideration, that the mesh partitioning process in Nektar++ wasn't included in the realtime since the process was not parallelizable.
Truly not to be misunderstood, there is a lot of potential in the Nektar++ regarding exascaling and especially when it is possible to run aeroacoustics simulations. The timestep per iteration is there in the standard finite volume method much smaller, which means Nektar++ with the spectral element method could be more competitive in this application.
Finally, in a world of exascale machines, another focus point in the industry gets more and more important, the short time to result. An ideal development process will give a directly feedback to the user about how a parameter change will influence the solution. This includes from the calculation side an almost lifetime simulation and is only achievable by running the simulation with an extremely high number of cores. As already mentioned in Section 4.2.3.1 and Section 4.3.3, Nektar++ shows much better performance in parallel computing and should be in a future world of excascale machines the first choice.

In future work, a mesh and calculation study to optimize a given geometry can validate the numerical repeatability of the physical solution and turbulent flow.
Finding the developed workflow for the simulation process with Nektar++ was quite a big challenge. There can be done a lot of work to simplify the workflow of mesh converting and the postprocessing by standardization.
And finally, a parallelization of the partitioning from the simulation domains would make Nektar++ much more valuable in the field of excascale.

Bibliography

[1] C.J. Falconi, D. Lautenschlager, ExaFLOW use case: Numerical simulation of the rear wake of a sporty vehicle, ExaFLOW technical report, 2016, http://exaflow-project.eu/index.php/use-cases/automotive.
[2] C.D. Cantwell, et al., Nektar++: An open-source spectral/ element framework, Computer Physics Communications, 192 (2015) 205-219.
[3] Cray XC40 (Hazel Hen), http://www.hlrs.de/en/systems/cray-xc40-hazel-hen/.
[4] R. Nash, et al., Communication and I/O masking for increasing the performance of Nektar++, eCSE 02-13 technical report, 2016, https://www.archer.ac.uk/community/eCSE/.
[5] Using Cray Performance Measurement and Analysis Tools, S–2376–622, http://docs.cray.com/books/S-2376-622/S-2376-622.pdf.
[6] Preisblatt Alttarife Strom, http://stadtwerke-karlsruhe.de.
[7] Neufassung der HLRS Entgeltordnung (Stand 2016), https://www.hlrs.de/solutions-services/academic-users/legal-requirements/.
[8] C.J. Falconi, M. Woo, D. Lautenschlager, Performance Analysis of an ExaFLOW code with the automotive use case, ExaFLOW technical report, 2017.
[9] User Guide Nektar++ 4.4.1, http://www.nektar.info/downloads/file/user-guide-pdf-3/
[10] S. Yakovlev, D. Moxey, S. J. Sherwin and R. M. Kirby, https://davidmoxey.uk/assets/pubs/2015-hdg.pdf, *J. Sci. Comp.*, **67** (1), pp. 192–220, 2016