Tenth International Conference on Computational Fluid Dynamics (ICCFD10), Barcelona,Spain, July 9-13, 2018

ICCFD10-099

Using High Order Finite Difference Schemes on Nested Cartesian Grids for Large-Scale Separated Flows

Alejandro Figueroa and Rainald Löhner
Corresponding author: afiguer7@gmu.edu

Center For Computational Fluid Dynamics
George Mason University, Fairfax, VA, USA.

Abstract: Nested cartesian grid systems by design require interpolation of solution fields from coarser to finer grid systems. While several choices are available, preserving accuracy, stability and efficiency at the same time require careful design of the interpolation schemes. Given this context, practical interpolation methods for nested cartesian finite difference grids were developed and tested in several situations. These algorithms are based on post-processing, on each local grid, the raw (bi/trilinear) information passed to the halo points from coarser grids. In this way modularity is maximized while preserving locality. The results obtained indicate that the schemes improve the convergence rates and the overall accuracy of finite difference codes with varying grid sizes. In addition to basic testcases, some large-scale separated flows past complete car configurations are considered in order to demonstrate the capabilities developed.

Keywords: High order solvers - Interpolation - Parallelization - Cartesian Grids - Finite Difference Solvers

1 Introduction

Separated flows which require LES-type runs at large Reynolds-numbers have been at the forefront of CFD research for decades. Yet, to date, LES runs last at least 3 weeks even if the number of cores available is arbitrarily high. Simply put, in order to achieve the industrial goal of overnight ($4 \cdot 10^4$ secs) LES runs with $O(10^7)$ advective timesteps, one needs to run at 250 timesteps/sec, or 4 msec/timestep with $O(10^9)$ degrees of freedom/points. At present, we have not seen codes able to run faster than $O(0.050)$ sec/timestep when going beyond 10Kcores. A recent study conducted by the authors confirmed that if only communication is taken into consideration, the aim of 4 msec/timestep is achievable. Therefore, the focus at present is on how to minimize the CPU requirements per point per timestep. The large number of points (or degrees of freedom) required by these LES runs, the option to easily increase the approximation order and the low FLOP overhead associated with them makes solvers based on nested cartesian grid systems [1, 2, 3, 4, 5] ideal candidates for the flow solvers sought. On the other hand, nested cartesian grid systems by design require interpolation of solution fields from coarser to finer grid systems. While several choices are available, preserving accuracy, stability and efficiency at the same time require careful design of the interpolation schemes.

However, the transition between grids of different spatial resolution is still a source of difficulty. A study conducted by the authors for a 6th-order FD flow solver revealed that at the transition between grids the numerical diffusion and dispersion due to interpolation completely overwhelmed the physics. This prompted a search for better interpolation techniques and interface treatments for such problems.

A theme often discussed in the literature is the need to use conservative interpolation schemes (see, e.g. [6, 7, 8]). A common conclusion seen is that for smooth solutions the standard interpolation procedure accurately capture the physics of the problem. As expected, conservative interpolation are only required near discontinuities. A typical situation often seen are very slow shocks crossing a grid transition area.

If one considers the interpolation problem, the immediate inclination is to use standard classic high-order interpolation techniques. High-order interpolations for locally refined Cartesian grids have been reported repeatedly in the literature [9, 10, 11], where techniques such as Fourier basis, cubic least squares and limited monotone (WENO type) schemes were considered for designing the interpolations. In three dimensions, this often led to large stencil sizes (of the order of 20 to 30 coefficients) that needed to be synthesized for each fine grid point. Moreover, interpolation schemes were fixed to maintain a particular order of accuracy. Extending them to even higher orders often required revisiting the derivations and a re-implementation of new schemes.

Having a large stencil footprint often defeats the purpose of using 'fast and cheap' FD methods. This has led the search for simpler interpolation techniques, utilizing immediately available data and constructing fast schemes that reuse results of interpolations performed previously. The aim here is to assess whether improvements in accuracy with a modest increase in complexity are possible using such schemes.

In the sequel, we detail the solver methodology used, the interpolation schemes implemented, and the results obtained for model problems to show the benefit of using these schemes. Thereafter, we apply them to some large-scale separated flows around complete car configurations.

2 Grid Generation

One of the key attractions of cartesian grids is the very simple and fast grid generation. As each 'block' will be discretized further by at least O(10Kpts), the generation of the blocks (or skeleton) may be carried out on a laptop. Given a surface triangulation (e.g. an STL file for a complete car), the user specifies the desired cell sizes based on the distance to the surface, the surface curvature, surface corners or edges, and additional regions that should be refined based on user experience (e.g. wakes). Based on this information, the pre-processor (in this case FECAD) automatically generates the octree of the blocks, and subsequently merges them in order to reduce as much as possible the final number of blocks. Note that this depends on the number of cores available for the run. This block generation is fast enough to allow for interactive trial and error. It usually takes longer to draw the STL (which can reach into the 20Mtria range) and the blocks than regenerating a new mesh.

3 Finite Difference Navier-Stokes Solver (FDFLO)

3.1 Equations Solved

FDFLO solves the weakly compressible Navier-Stokes equations. The temperature is included as an option, as well as the Boussinesq approximation for natural convection. The system of PDEs is given by:

$$\frac{1}{c^2}p_{,t} + \rho\nabla\cdot\mathbf{v} = 0. \tag{1}$$

$$\rho\mathbf{v}_{,\mathbf{t}} + \rho\mathbf{v}\nabla\mathbf{v} + \nabla p = \nabla\mu\nabla\mathbf{v} + \rho\mathbf{g} + \beta\rho\mathbf{g}(T - T_0) + S_v. \tag{2}$$

$$\rho c_p T_{,t} + \rho c_p \mathbf{v}\nabla T = \nabla\lambda\nabla T + S_T. \tag{3}$$

where $\rho, \mathbf{v} = (u, v, w), p, T, c, \mu, c_p, \lambda, \beta, T_0$ denote, respectively, the density, velocity, pressure, temperature, (constant) speed of sound, viscosity, heat capacitance, conductivity, thermal expansion and reference temperature for the fluid, and $\mathbf{g}, S_v, S_T$ the gravity vector and source terms for velocities and temperature.

3.2 Numerics

The numerics implemented may be summarized as follows:

- Explicit timestepping via low-storage Runge-Kutta schemes;
- Conservative formulation for advection and divergence;
- Easy extensions to high-order stencils;

- Ordered access to memory;
- Minimum access to memory;
- Long 1-D loops (for optimal vector, OMP and GPU performance);
- Use of halo points to impose boundary conditions and enable easy extension to massively parallel machines.

In the next sections, we describe in detail the discretizations and boundary conditions employed.

3.3 Discretization in Space

The spatial discretization is carried out via Finite Differences on a cartesian grid with equal mesh size in all directions:

$$h_x = h_y = h_z = \Delta x. \tag{4}$$

All fluxes are written in conservative form as:

$$\mathbf{r}_i = \frac{1}{\Delta x}\left[\mathbf{f}^x_{i+1/2} - \mathbf{f}^x_{i-1/2} + \mathbf{f}^y_{i+1/2} - \mathbf{f}^y_{i-1/2} + \mathbf{f}^z_{i+1/2} - \mathbf{f}^z_{i-1/2}\right]. \tag{5}$$

Each flux is composed of the physical and the artificial dissipation flux, e.g.:

$$\mathbf{f}_{i+1/2} = \mathbf{f}^p_{i+1/2} - \mathbf{f}^d_{i+1/2}. \tag{6}$$

The advective physical fluxes are obtained from central difference operators of 2nd, 4th, 6th and 8th order:

$$\mathbf{f}^p_{i+1/2}\Big|^{II} = \frac{1}{2}\left(\mathbf{f}^p_{i+1} + \mathbf{f}^p_i\right). \tag{7}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{IV} = \frac{7}{12}\left(\mathbf{f}^p_{i+1} + \mathbf{f}^p_i\right) - \frac{1}{12}\left(\mathbf{f}^p_{i+2} + \mathbf{f}^p_{i-1}\right). \tag{8}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{VI} = \frac{37}{60}\left(\mathbf{f}^p_{i+1} + \mathbf{f}^p_i\right) - \frac{8}{60}\left(\mathbf{f}^p_{i+2} + \mathbf{f}^p_{i-1}\right) + \frac{1}{60}\left(\mathbf{f}^p_{i+3} + \mathbf{f}^p_{i-2}\right). \tag{9}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{VIII} = \frac{533}{840}\left(\mathbf{f}^p_{i+1} + \mathbf{f}^p_i\right) - \frac{139}{840}\left(\mathbf{f}^p_{i+2} + \mathbf{f}^p_{i-1}\right) + \frac{29}{840}\left(\mathbf{f}^p_{i+3} + \mathbf{f}^p_{i-2}\right) - \frac{3}{840}\left(\mathbf{f}^p_{i+4} + \mathbf{f}^p_{i-3}\right). \tag{10}$$

For the 2nd order (Laplacian) operators the physical fluxes of 2nd, 4th, 6th and 8th order are:

$$\mathbf{f}^p_{i+1/2}\Big|^{II} = \left(\mathbf{f}^p_{i+1} - \mathbf{f}^p_i\right). \tag{11}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{IV} = \frac{15}{12}\left(\mathbf{f}^p_{i+1} - \mathbf{f}^p_i\right) - \frac{1}{12}\left(\mathbf{f}^p_{i+2} - \mathbf{f}^p_{i-1}\right). \tag{12}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{VI} = \frac{245}{180}\left(\mathbf{f}^p_{i+1} - \mathbf{f}^p_i\right) - \frac{25}{180}\left(\mathbf{f}^p_{i+2} - \mathbf{f}^p_{i-1}\right) + \frac{2}{180}\left(\mathbf{f}^p_{i+3} - \mathbf{f}^p_{i-2}\right). \tag{13}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{VIII} = \frac{7175}{5040}\left(\mathbf{f}^p_{i+1} - \mathbf{f}^p_i\right) - \frac{889}{5040}\left(\mathbf{f}^p_{i+2} - \mathbf{f}^p_{i-1}\right) + \frac{119}{5040}\left(\mathbf{f}^p_{i+3} - \mathbf{f}^p_{i-2}\right) - \frac{9}{5040}\left(\mathbf{f}^p_{i+4} - \mathbf{f}^p_{i-3}\right). \tag{14}$$

These (unstable) approximations are stabilized by adding an appropriate artificial viscosity / damping [12, 13, 14, 2] of the form:

$$\mathbf{f}^d_{i+1/2}\Big|^{II} = c_d\lambda_{i+1/2}\left[(\mathbf{u}_{i+1}-\mathbf{u}_i)\right]. \tag{15}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{IV} = c_d\lambda_{i+1/2}\left[3\left(\mathbf{u}_{i+1}-\mathbf{u}_i\right)-\left(\mathbf{u}_{i+2}-\mathbf{u}_{i-1}\right)\right]. \tag{16}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{VI} = c_d\lambda_{i+1/2}\left[10\left(\mathbf{u}_{i+1}-\mathbf{u}_i\right)-5\left(\mathbf{u}_{i+2}-\mathbf{u}_{i-1}\right)+\left(\mathbf{u}_{i+3}-\mathbf{u}_{i-2}\right)\right]. \tag{17}$$

$$\mathbf{f}^p_{i+1/2}\Big|^{VIII} = c_d\lambda_{i+1/2}\left[35\left(\mathbf{u}_{i+1}-\mathbf{u}_i\right)-21\left(\mathbf{u}_{i+2}-\mathbf{u}_{i-1}\right)+7\left(\mathbf{u}_{i+3}-\mathbf{u}_{i-2}\right)-\left(\mathbf{u}_{i+4}-\mathbf{u}_{i-3}\right)\right]. \tag{18}$$

where λ is the maximum eigenvalue of the system

$$\lambda = |\mathbf{v}| + c. \tag{19}$$

and c_d the artificial viscosity / damping coefficient. Typical values are: $c_d^{II} = 0.2, c_d^{IV} = 0.10, c_d^{VI} = 0.02, c_d^{VIII} = 0.02$. For viscous cases, the artificial viscosity / damping coefficient of the momentum equations is reduced:

$$c_d^* = c_d \cdot r(u) \quad , \quad r(u) = max(0, min(1, Re_h - 1)) \quad , \quad Re_h = \frac{\rho u \Delta x}{\mu}. \tag{20}$$

Note that as the mesh is refined and the cell Reynolds-number Re_h falls below $Re_h = 1$, the artificial viscosity vanishes. The same type of advection and artificial viscosity is also used for the temperature equation, but limiting with the local Peclet-number.
Finally, for the **Boussinesq terms** the approximation taken is simply:

$$\mathbf{r}_i = \mathbf{g}\beta\left(T_B - T_i\right). \tag{21}$$

The same is done for the source-terms S_v, S_T.

In order to achieve long vector loops the formation of right-hand sides (RHSs) is carried out by forming a single array of point data. Given `nx, ny, nz`, and defining `nxny = nx * ny`, the points are traversed as `ip = nxny*(iz-1) + ny*(iy-1) + ix`. In order to minimize the use of registers, the RHSs are formed dimension by dimension.

4 Boundary Conditions

A variety of boundary conditions are required for practical computations. An easy way to implement these is via halo points. In order to be able to cast all operations in terms of large loops over all points, two (for 4th order stabilized fluxes), three (for 6th order stabilized fluxes) or four (for 8th order stabilized fluxes) halo points are added at the minimum and maximum extent of each dimension. This increases the total point count, but allows for maximal vector length. The boundary conditions are then imposed as follows:

- No-Slip (NavSto) Wall: Same $p, \mathbf{v}, T$ as wall;
- Symmetry/Euler Wall: same $p, \mathbf{v}\cdot\mathbf{t}, T$, opposite $\mathbf{v}\cdot\mathbf{n}$;
- Inflow to Field: $\mathbf{v}, T$ imposed, p free;
- Outflow of Field: p imposed, $\mathbf{v}, T$ free.

5 Immersed Body Option

A solver based on Cartesian Finite Differences may be very fast, but its use is very limited when considering geometrically complex objects. There are two options of treating these: either via immersed body methods, or via embedded surface techniques. We have implemented both.

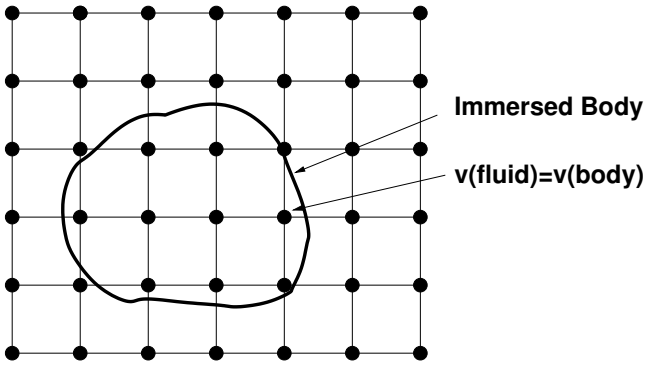

Figure 1: Immersed Body Approach.

The **immersed body methods** [15, 16, 17, 18, 19, 20, 21, 22, 23] (see Fig. 1) may be classified by the way they apply the boundary conditions into kinematic or kinetic [14].
The **kinematic** approaches simply set the velocity of the flow to the velocity of the body:

$$\mathbf{v}_{flow} = \mathbf{v}_{body}. \tag{22}$$

This 1st order scheme may be improved via interpolation or weighting functions, i.e. taking into account neighbour information [19].
The **kinetic** (i.e. force-based) approaches add a force to the momentum equations such that Eqn. 22 is fulfilled at the end of the timestep, or a penalty term that imposes Eqn. 22 weakly.
In either case, the immersed body options described require the following steps:

- Read in the immersed bodies as a mesh (elements, coordinates, velocities, temperature);
- Determine the cartesian points inside the bodies; this is done by performing a loop over the elements of the immersed bodies; for each of these the cartesian points inside the element are obtained and marked;
- Store the points marked in a separate boundary point array.

In order to be as general as possible, the immersed bodies are defined via tetrahedral meshes.

6 Embedded Surface Option

The second way to treat geometrically complex objects within a Cartesian Finite Difference code is via embedded surface techniques [24, 25, 26, 27, 28, 29, 30] (see Fig. 2).

The key idea is to 'extend' the stencil past the edges crossed by the embedded surfaces, mirror these points back into the proper computational domain, and then use mirroring or other boundary conditions to impose the presence of the embedded surfaces (see Fig. 3). This way of imposing the boundary conditions for embedded surfaces is theoretically 2nd order accurate. However, in many cases (particularly if the points are very close to the embedded surface, or in narrow passages) the available information for interpolation is not sufficient to guarantee this order.

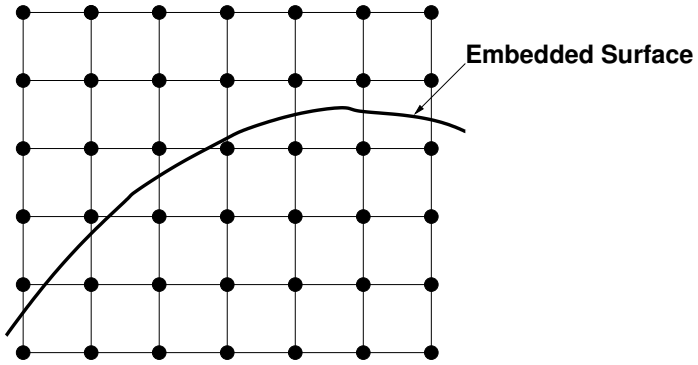

Figure 2: Embedded Surface Approach.

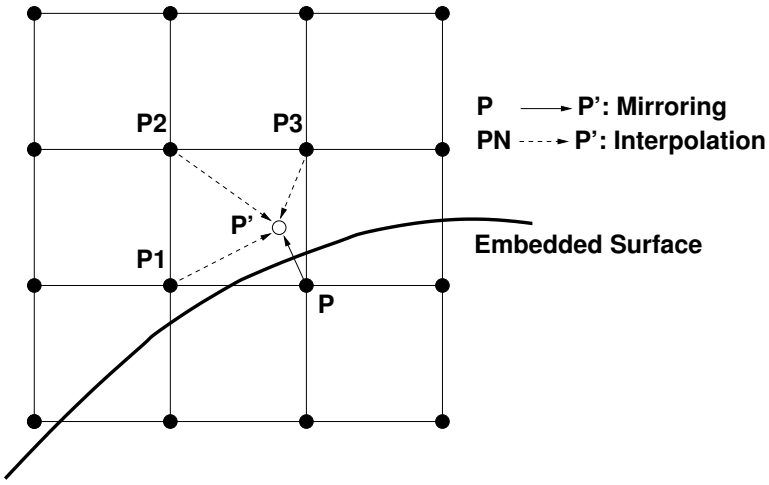

Figure 3: Embedded Surface Approach: Boundary Conditions.

The embedded surface option requires the following steps:

- Read in the embedded surfaces as a triangulation (elements, coordinates, velocities, temperature);
- Determine the cartesian edges crossed by the surface; this is done by performing a loop over the triangles of the embedded surfaces; for each of these the cartesian edges crossed are obtained and marked;
- Determine the interpolation conditions for the points 'on the other side' of the crossed edges.
- Store the points marked in a separate boundary point array.

There are several possible ways of incorporating the values of the unknowns required to imposed embedded surfaces. We have pursued a dual-loop approach: The first loop over the points is the usual one, i.e. it ignores the embedded surface boundary conditions. The second loop, over the points whose surrounding edges are crossed by embedded surfaces, subtracts the right hand of the 1st loop, and adds the right hand side replacing the unknowns 'on the other side' with the proper values. In this way, the extra burden of imposing embedded surface boundary conditions is minimized.

7 Spatial Discretization

After spatial discretization, the original PDEs given by Eqns. 1-3 form a coupled system of ordinary differential equations (ODEs) of the form:

$$\mathbf{u}_{,t} = \mathbf{r}(t, \mathbf{u}). \tag{23}$$

This system is solved using explicit, low-storage **Runge-Kutta** methods of the form:

$$\Delta\mathbf{u}^{n+i} = \alpha_i \, \Delta t \, \mathbf{r}(\mathbf{u}^n + \Delta\mathbf{u}^{n+i-1}) \quad , \quad i = 1, s \quad , \quad \Delta\mathbf{u}^0 = 0. \tag{24}$$

or via the usual 4-stage Runge-Kutta scheme [31]. We remark that for linear ODEs the choice

$$\alpha_i = \frac{1}{s+1-i} \quad , \quad i = 1, s. \tag{25}$$

in combination with Eqn. 24 leads to a scheme that is s-th order accurate in time. Like all explicit schemes, the allowable timestep is bounded by the condition:

$$\Delta t < CFL \cdot min\left(\frac{h}{|\mathbf{v}| + c} \, , \, \frac{\rho h^2}{\mu} \, , \, \frac{\rho c_p h^2}{\lambda}\right). \tag{26}$$

where the allowable CFL factor is proportional (i.e. increasing) with the number of stages s.
One complete timestep is given by the following steps:

- Apply BC / Transfer Info from Domain to Halo Points
- Get Allowable Timestep
- Set Timestep $\Delta t = 0$ for Halo Points
- Loop Over the Stages:
    - Set $\mathbf{r} = 0$
    - Compute $\mathbf{r}$
    - Obtain $\Delta\mathbf{u} = \alpha_i \Delta t \mathbf{r}(\mathbf{u})$
    - Apply Boundary Conditions
    - Update $\mathbf{u}$

8 Multiblock Option

A solver based on Cartesian Finite Differences may be very fast, and may be made applicable to complex geometries via immersed or embedded techniques. However, its use is still very limited when considering problems with varying spatial lengthscales. The best way of addressing this problem while keeping the speed advantages of the basic solver is via multiblocking [32, 26]. The key idea is to consider each cartesian grid or block independently, and to combine these by interpolating the unknowns of the halo points from the adjacent blocks.

9 Interpolation Between Cartesian Grids

The transition between grids of different spatial resolution can be a source of difficulties. High-order interpolation in local refined Cartesian grids have been reported in literature [9, 10, 11], where techniques such as Fourier basis, cubic least squares and limited monotone (WENO type) schemes were considered for designing the interpolations. In three dimensions, this often led to large stencil sizes of the order of 20 to 30 coefficients that need to be synthesized for each fine grid point. Having a large stencil footprint often defeats the purpose of using 'fast and cheap' FD methods. This has led to the consideration of simpler interpolation techniques, utilizing immediately available data and constructing fast schemes that reuse results of

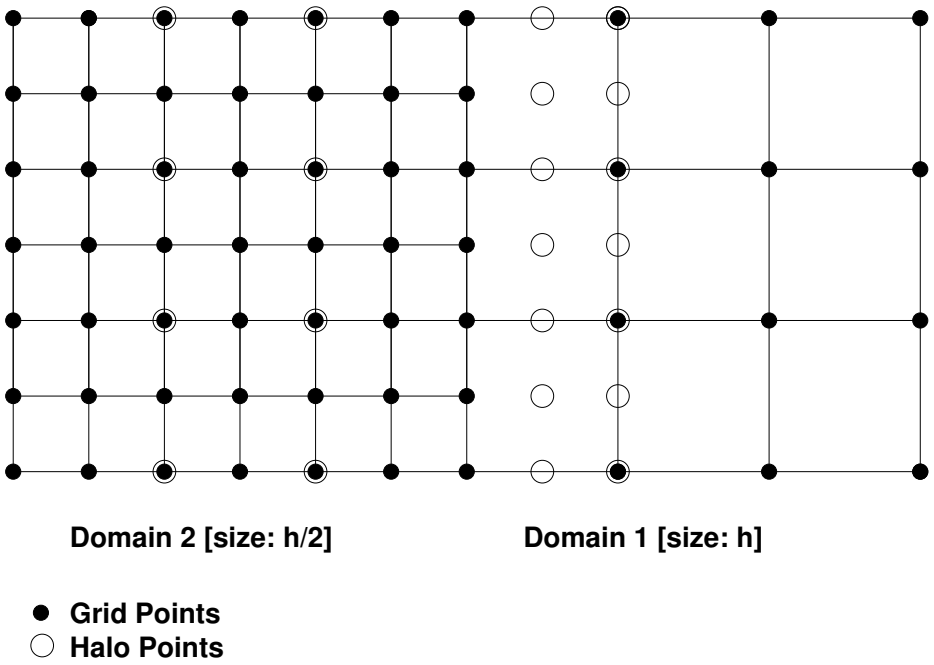

Figure 4: Multiblock Option.

interpolations performed previously [33]. At the beginning of each timestep, iteration, or Runge Kutta stage the information required for the halo points is obtained from the appropriate neighbouring grids. For the gridpoints that coincide this information is kept (implicitly assuming that the information given at gridpoints is the most accurate and should therefore not be changed). However, at intermediate points one is at liberty to interpolate with higher order schemes. This so-called 'post-processing interpolation' has shown excellent results in many tests carried out to date [33].

9.1 h/2h Interpolation

The situation commonly encountered is shown in Figures 5–6 for the 1-D and 2-D cases respectively. We denote the fine grids as grid h and coarse grid as grid 2h respectively. At the h/2h boundary, the Cartesian grids need to exchange information. The assumption is made that in order to maintain code modularity, halo points are used to transfer information between grids (and also for boundary conditions). In this way, the update and boundary condition stages of the different solvers are separated in a a clean, modular fashion. At the beginning of each timestep, iteration, or Runge-Kutta stage the information required for the halo points is obtained from the appropriate neighbouring grids. Furthermore, it is assumed that the information given at gridpoints is the most accurate and should therefore not be changed. This implies that for points that coincide (labeled D in Figure 6) a direct injection / transfer of information is desirable. On the other hand, for the points along edges or faces (labeled E, F in Figure 6), one is at liberty to apply interpolation schemes of different order. Note that this will only be required for grid h on a h/2h boundary, i.e. only for the finer grid.

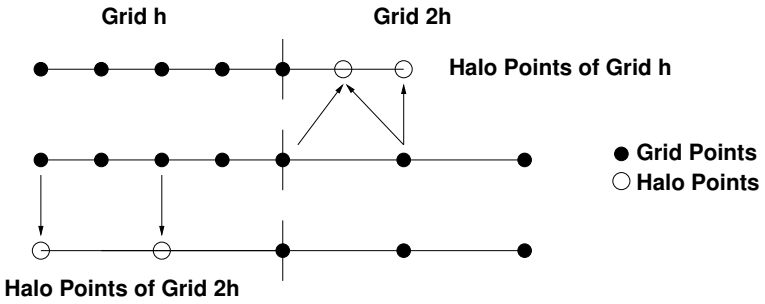

Figure 5: Interpolation Between FD Grids (1-D).

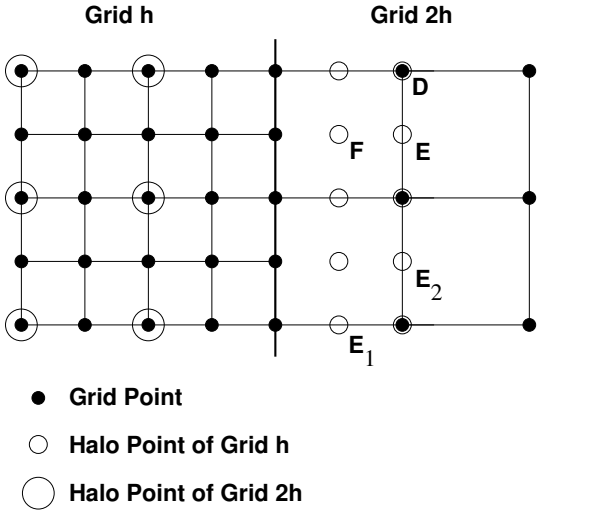

Figure 6: Interpolation Between FD Grids (2-D).

9.2 Post-Processing Interpolation

An interesting option when trying to maximize modularity is to transfer all direct injection points first from *2h* to *h*, and then obtain the missing information by post-processing this data on *grid h*. In 2-D, the cases that need to be considered are:

- E_1: Edge-points aligned with grid-lines from grid h
- E_2: Edge-points not aligned with grid-lines from grid h
- F: Face-points

The distinguishing factor for points of type E_1 is that information from the interior of *grid h* is readily available and can be used to improve the interpolation order. Figure 7 shows some of the possibilities, together with the interpolation weights. For the points of type E_2 usual high order Lagrangian interpolation schemes are employed. Figure 8 shows some of the possibilities, together with the interpolation weights. Points of type F may be interpolated either via a weighted average of the surrounding edge-points, or by treating them as points of type E_1 with the extra information required obtained previously for the points of type E_2.

For a 2D cell information is 'known' at 4 points, while 5 points need to be interpolated. However, in 3D information is 'known' at 8 points, while 19 points require interpolation. Based on this, the options for 3D cases are to use a simple bi/trilinear interpolation or develop different techniques. For example, for a E_2 point (the blue point in Figure 10), the information needed to obtain the required value is obtained as a combination of linear (red points) and high order (blue edges) interpolation.

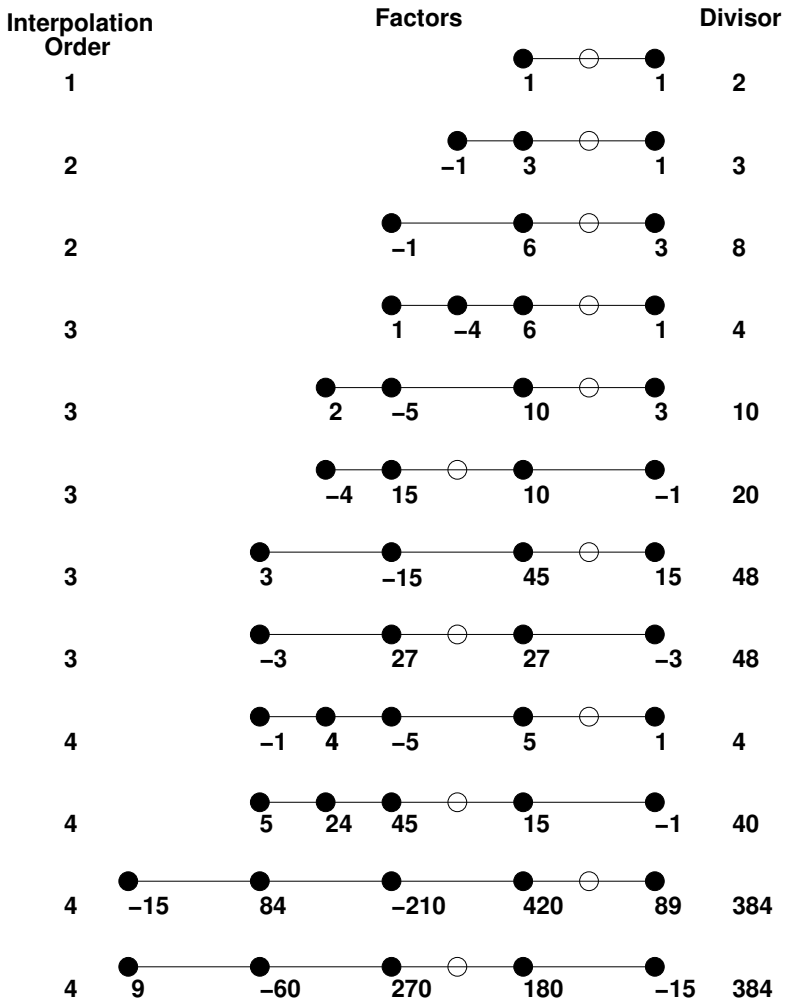

Figure 7: Interpolation Factors for Edges of Type E_1).

9.2.1 Interpolation Limiters

In order to avoid spurious oscillations or sharp changes in the high order interpolation due to Runge's phenomenon, several limiters were implemented and tried.

- Barth-Jespersen limiter: This is a simple limiter where the maximum and minimum (U_{max}, U_{min}) values from the closest points around the interpolated point are selected and then used to limit the high order solution, U_h.

$$U_i = max(U_{min}, min(U_{max}, U_h)). \tag{27}$$

- Radius of curvature limiter: Introduced by [34], this method constrains the high order solution with the argument that the interpolated value is accurate as long as the radius of curvature of an interface region is higher than 3 grid cells. It requires the calculation of a linear or low order interpolation, U_l.

$$U_i = \begin{cases} U_h & \text{if } |U_h - U_l| < \beta \Delta x \\ U_l & \text{otherwise.} \end{cases} \tag{28}$$

where β is heuristically chosen as $\beta = 1/20$.

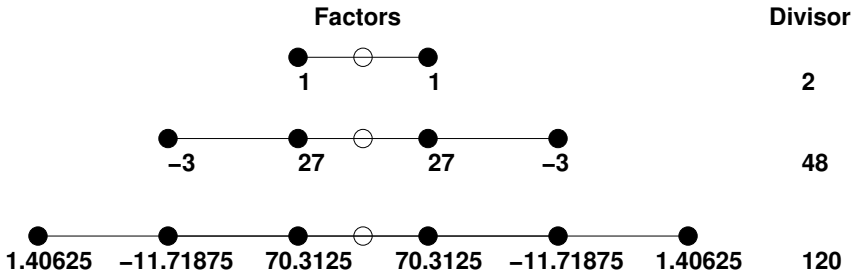

Figure 8: Interpolation Factors for Edges of Type E_2.

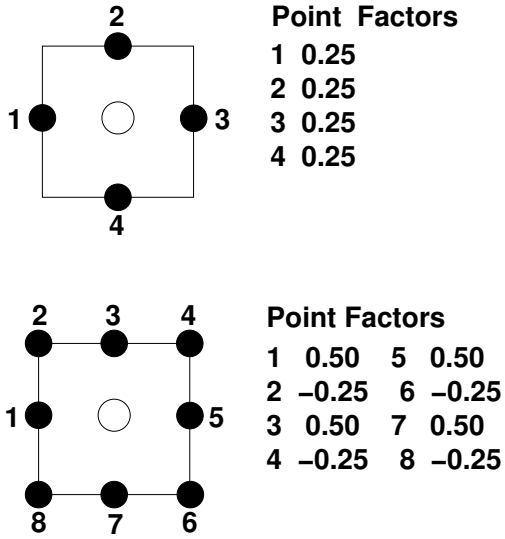

Figure 9: Interpolation Factors for Faces.

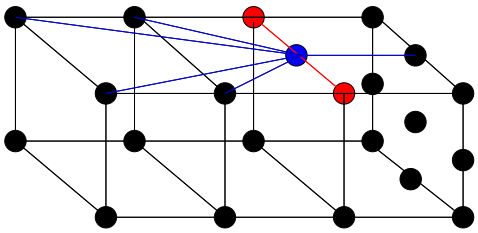

Figure 10: E_2 Interpolation in 3D.

- Modified radius of curvature limiter: Proposed by [35], this method attempts to overcome possible discontinuities that can appear with the previous technique. A modification is made to obtain a more regular interpolation

$$U_i = \begin{cases} U_l + \beta\Delta x & \text{if } |U_h - U_l| > \beta\Delta x \\ U_h & \text{otherwise.} \end{cases} \tag{29}$$

- QMSL limiter: With the aim of using a simple and efficient limiter algorithm, and based on the Flux Corrected Transport technique, Bermejo[36] proposed the following method. First define U_{max} and U_{min} and then set,

$$Q_+ = U_{max} - U_l \;\; ; \;\; Q_- = U_{min} - U_l \;\; ; \;\; P = U_h - U_l. \tag{30}$$

The value of the interpolated point is defined as

$$U_i = \begin{cases} C = min(1, \frac{Q_+}{P}) & \text{if } P > 0 \\ C = min(1, \frac{Q_-}{P}) & \text{if } P < 0 \\ C = 1 & \text{if } P = 0. \end{cases} \tag{31}$$

$$U_i = U_l + C(U_h - U_l) \tag{32}$$

10 Model Problems

The interpolation schemes developed were implemented and tested in FDFLO. In the present section the performance of these schemes is shown for some basic model problems.

10.1 2D Lamb Vortex

The so-called Lamb-vortex, centered at $x, y = 0$, is a classic testcase for vortical flows. The unknowns are given by:

$$u = u_0 - \frac{\alpha}{2\pi} y e^{\phi(1-x^2-y^2)} \quad ; \quad v = \frac{\alpha}{2\pi} x e^{\phi(1-x^2-y^2)} \quad ; \quad p = -\left(\frac{\alpha}{2\pi}\right)^2 \frac{1}{4\pi} e^{2\phi(1-x^2-y^2)} \tag{33}$$

and $\mu = 0$ (i.e. no viscosity). For the particular case tested, the domain was given by $-5 \leq x, y \leq 5$, and $\alpha = 1$, $\phi = 0.5$, $c = 1$, $\rho = 1$, and the grids were of size $h = 0.125$ and $2h = 0.250$. The vortex was propagated for $T = 200$ time units. Given that the domain is doubly periodic, the vortex should reappear in the exact location as at time $T = 0$ after traversing the mesh twice.
Figure 11 shows the initial conditions for the *mesh h2h*, where the discretizations used are clearly visible. Figures 12–13 show the pressure and velocity obtained using an 8th order spatial discretization and a 5th order low-storage Runge-Kutta timestepping scheme. From left to right, the cases are: *mesh 2h*,*mesh h2h* (i.e. mesh h inside mesh 2h) with usual bilinear interpolation, *mesh h2h* with cubic interpolation, and *mesh h*. One can see that for the case with bilinear interpolation, running the case on the *2h mesh* yields better results than the *h2h mesh*, defeating the purpose of mesh refinement. This should be a cause of alarm if one considers complex flow problems where vortices and other flow structures will traverse grids with different mesh sizes. On the other hand, the cubic interpolation does yield results on the *h2h mesh* that are demonstrably better than those on the *2h mesh*, indicating the potential of the procedures developed.

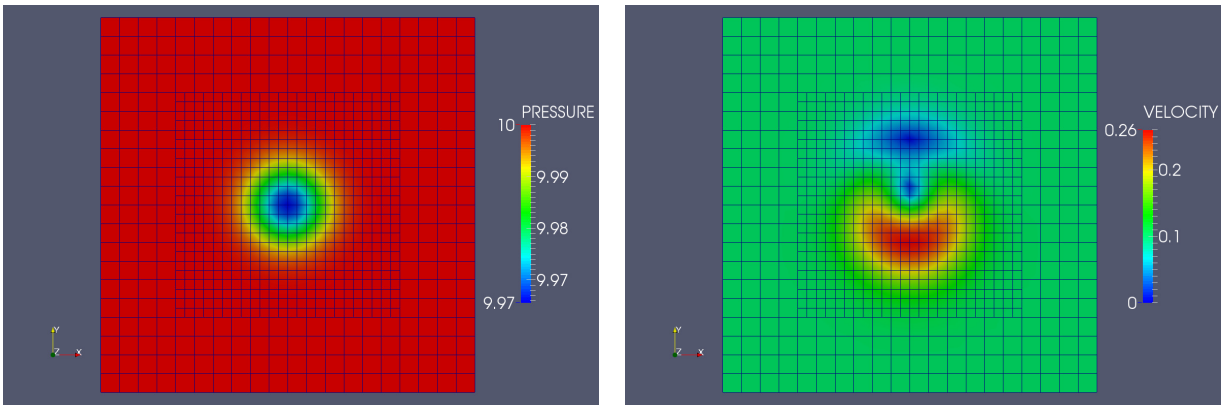

Figure 11: Lamb Vortex: Initial Conditions, *h2h Mesh*.

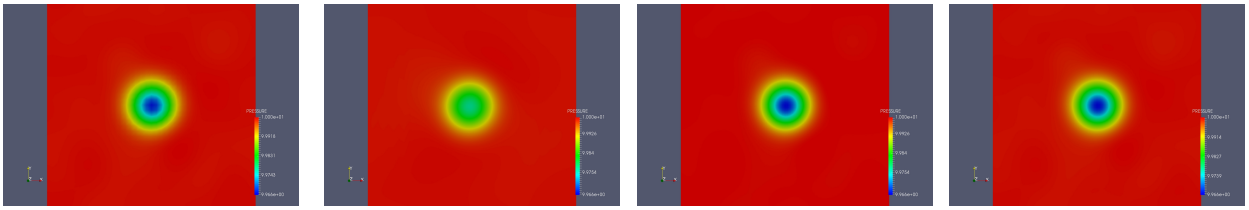

Figure 12: Lamb Vortex: Comparison of Pressures,
2h, h2h Bilinear, h2h Cubic, h.

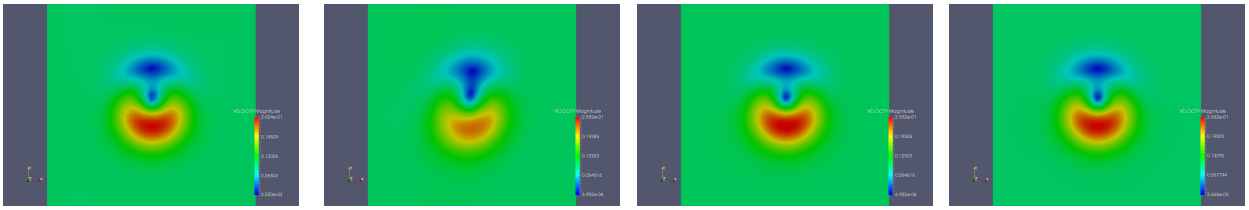

Figure 13: Lamb Vortex: Comparison of Velocity Magnitudes,
2h, h2h Bilinear, h2h Cubic, h.

10.2 2D Convergence Study with Stationary Lamb Vortex

In order to quantify the relative merit of the different interpolation schemes, a series of convergence studies were carried out. The same domain as before was used, but the boundary conditions were changed from periodic to gliding wall. The right half of the domain was of size h, the left of size $2h$. At the same time, uniform grids of size h and $2h$ were run for comparison. A stationary Lamb-Vortex was set as the initial condition. This is an exact steady solution, so the initial residual can be used to measure the convergence of the schemes. A typical configuration is shown in Figure 14.

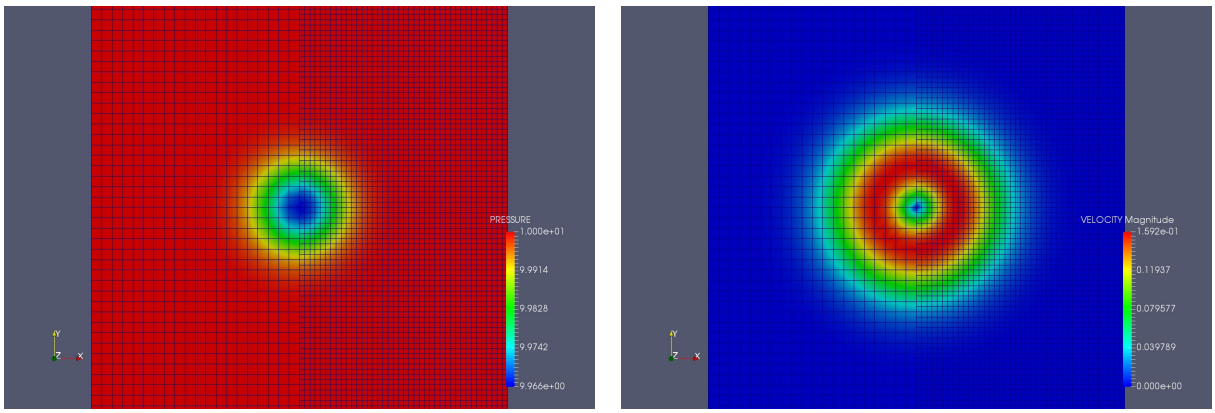

Figure 14: Stationary Lamb Vortex: Typical Initial Configuration.

Figures 15–18 show the convergence obtained for the 6th and 8th order solvers, together with different interpolation schemes. The notation is as follows: L2 denotes the $L2$ norm, LI the L_∞ norm, P the pressure, V the velocity, UNH and U2H the convergence on uniform grids of size h and $2h$, and M00, M33, M43 the cases of mixed $h, 2h$ grids with simple bi/trilinear interpolation, cubic interpolation and quartic interpolation. As expected, for the $L2$ norm the errors of the high order interpolation schemes fall between the values for uniform grids of size h and $2h$. This is not always the case for the L_∞ norm. Furthermore, one can see the serious negative effect on convergence of the bi/trilinear interpolation. The results show that the aim of interpolation schemes that are balanced and appropriate to the spatial discretization while being local and fast has been achieved.

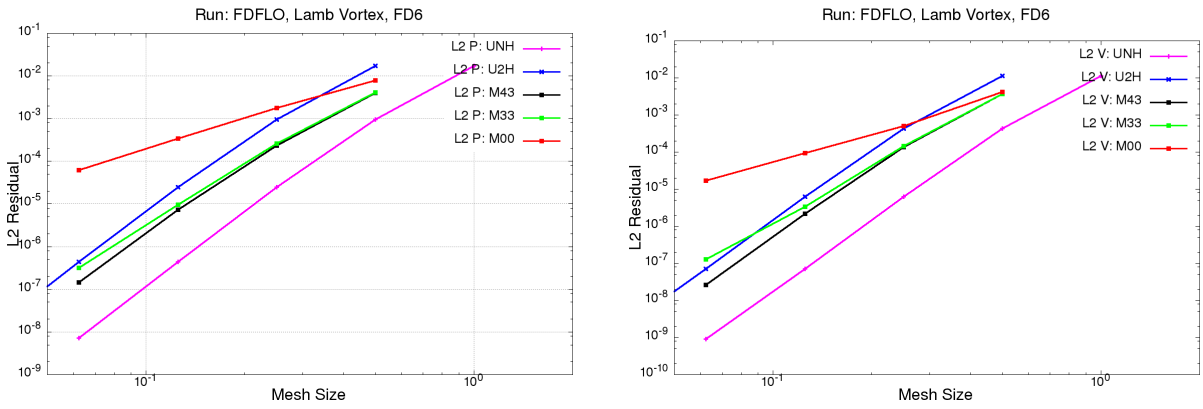

Figure 15: Stationary Lamb Vortex: $L2$ Convergence for 6th Order.

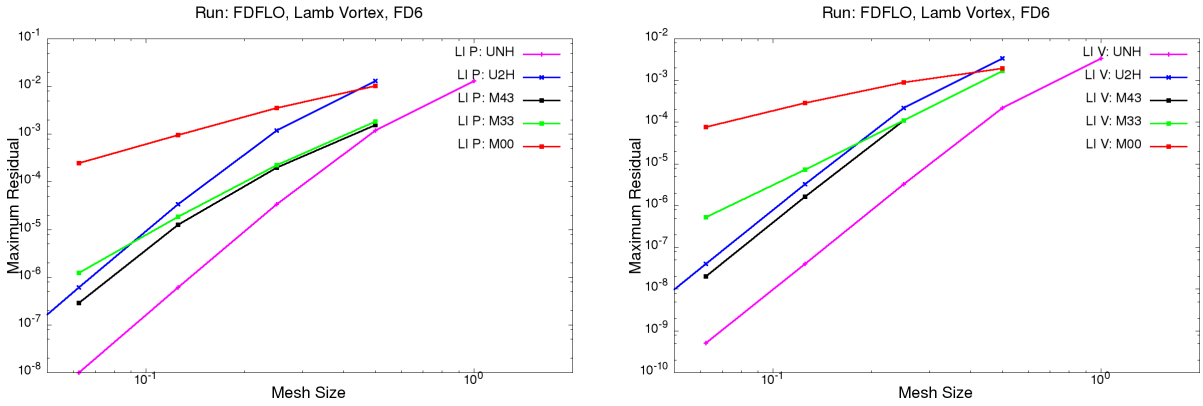

Figure 16: Stationary Lamb Vortex: L_∞ Convergence for 6th Order.

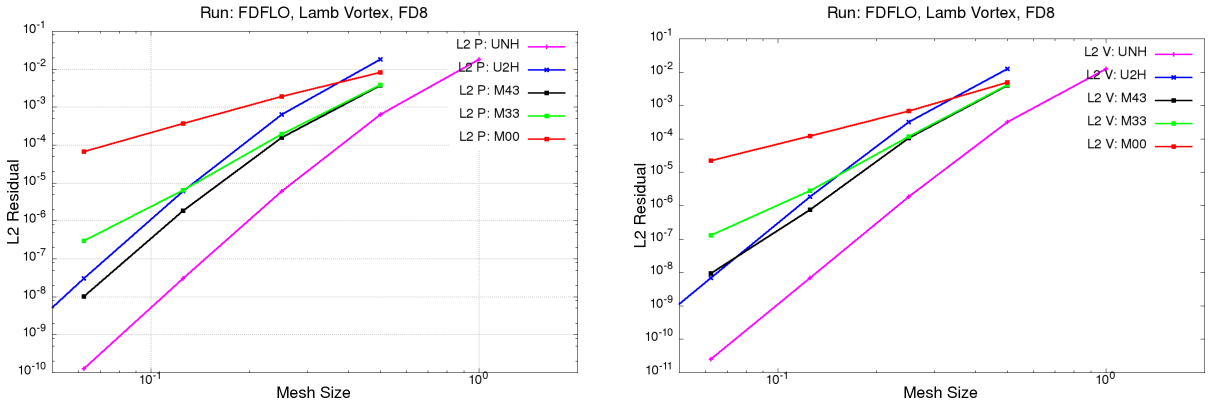

Figure 17: Stationary Lamb Vortex: $L2$ Convergence for 8th Order.

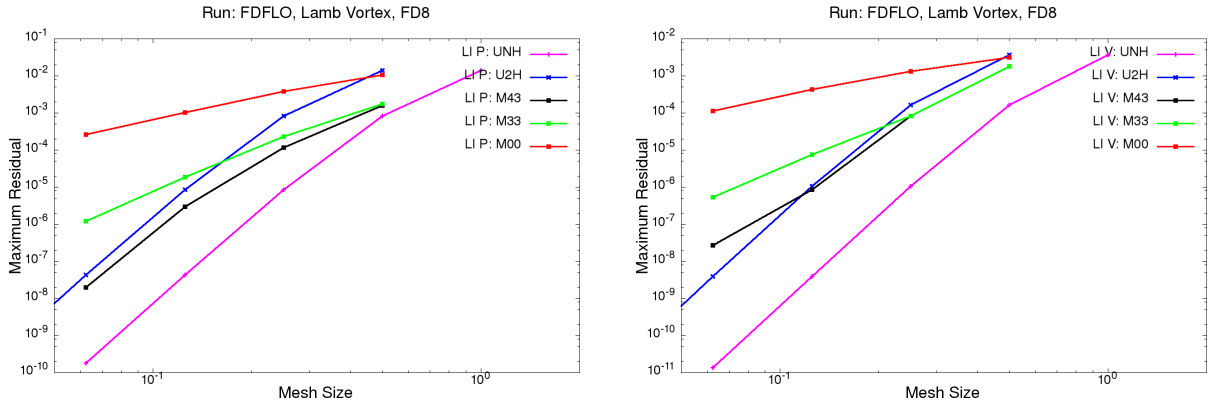

Figure 18: Stationary Lamb Vortex: L_∞ Convergence for 8th Order.

10.3 Immersed Cylinder

This classic testcase was added in order to show the effect of high-order interpolation for wake flows. The domain considered was $-4 \leq x \leq 10$, $-2 \leq y \leq 2$, with gliding wall boundary conditions at y_{min}, y_{max}, prescribed uniform inflow and prescribed pressure at outflow. As can be seen from Figure 19, the mesh consisted of 10 domains, with three levels of refinement: $\Delta x = 0.100, 0.050, 0.025$. The physical parameters were set as follows: $\rho = 1.0, \mathbf{v}_\infty = (1, 0, 0), \mu = 0.01, c = 5$, and the diameter of the cylinder was $d = 1.0$, yielding a Reynolds number of $Re = 100$. A 6-stage, low-storage RK scheme was used to integrate in time with a Courant-number of $C = 0.4$. The immersed boundary option was used with a spatial discretization of 6th order. The case was run with the usual, low-order bi/trilinear interpolation, the high-order interpolation with and without the QMSL limitor and with a homogeneous fine mesh of $h = 0.025$. The results obtained for the latter one at $T = 100$ are shown in Figure 20. A number of station time history points were placed in the flow and the results recorded. Figures 21–22 show the values for the pressure, x- and y-velocities for two stations. The most pronounced difference can be observed in the pressures and x-velocities where the differences compared with the fine mesh results are around 15% of the peak value. These differences are larger for the points far away from the body. Furthermore, even the y-velocities show larger variations in time for the high-order interpolation, indicating less dissipation. Note also that a slight change of frequency is incurred when changing interpolation order. Due to the smooth behaviour of the wake the interpolated+limiter results are very similar to the values obtained with the high order interpolation.

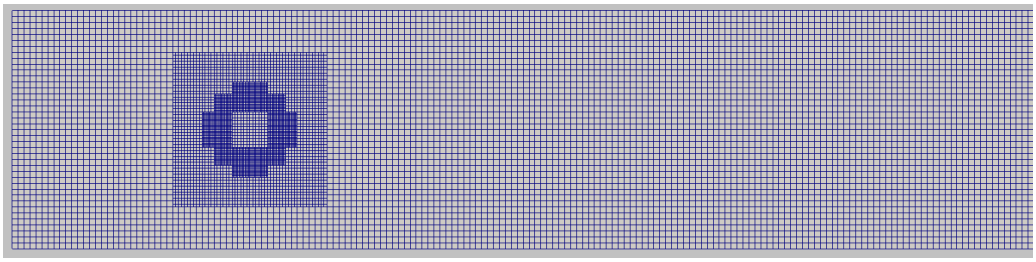

Figure 19: Cylinder: Grid System Used.

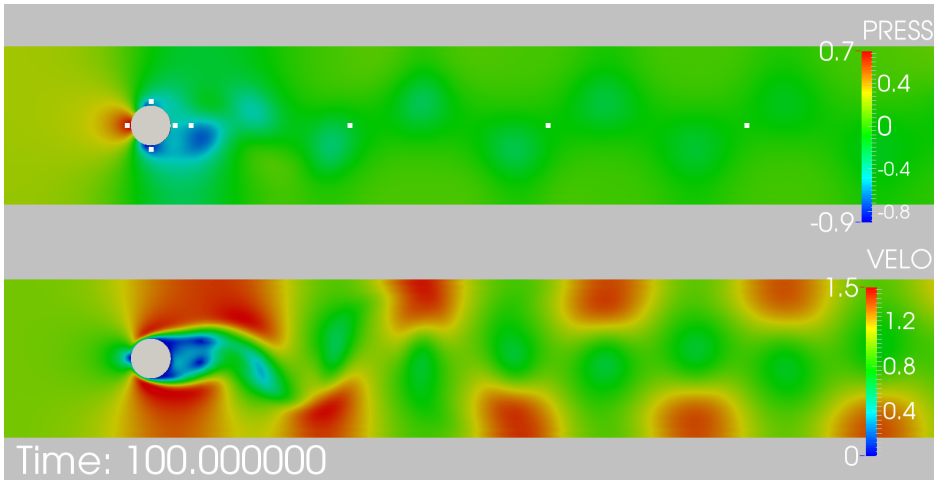

Figure 20: Cylinder: Results at Time $T = 100$.

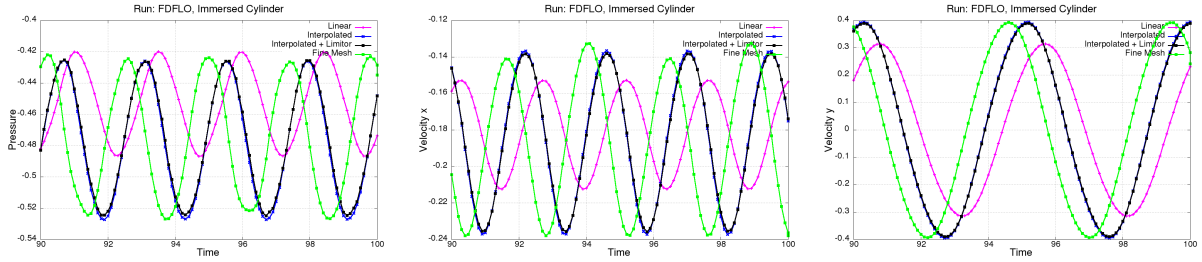

Figure 21: Cylinder: Station Time History for Station 5: $\mathbf{x} = (1, 0, 0)$.

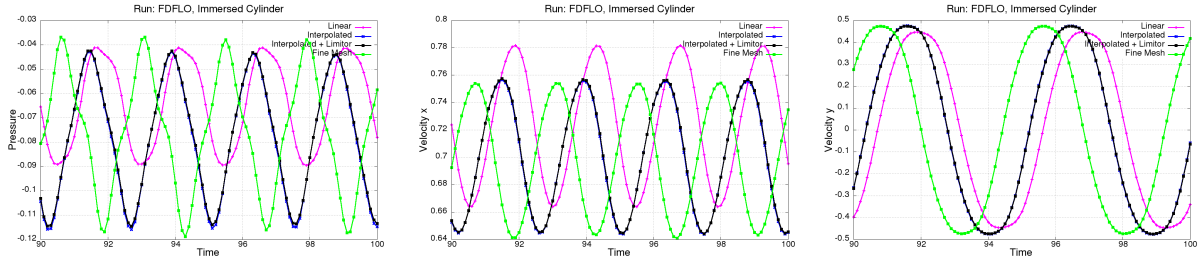

Figure 22: Cylinder: Station Time History for Station 7: $\mathbf{x} = (10, 0, 0)$.

11 Complete Car Configuration

As an example for a complete car configuration, we consider the flow past an Audi A4 as shown in Figures 23,25. The surface mesh provided, shown in Figure 23, consisted of nface=141,702 triangles and npoin=70,767 points, with several water-tight but intersecting pieces. It was run trough the FECAD preprocessor, which invoked the PRE-FDFLO grid generator, leading to ndomn=2,932 domains and a total point count of npoin=236.3 Mpts, of which nactp=210.5 Mpts were actually updated (some points are not updated are they are in nested grids or inside the car). The following physical and numerical setting were employed:

- Density: $\rho = 1.2\ kg/m^3$
- Velocity: $|\mathbf{v}| = 30\ m/sec$
- Speed of sound: $c = 300\ m/sec$
- Laminar viscosity: $\mu = 1.85 \cdot 10^{-5} kg/m/sec$
- Smagorinsky constant: $c_s = 0.325$
- Smagorinsky length: $h_s = 0.005\ m$
- Smallest cell/element size: $h_{min} = 0.003\ m$
- Largest cell/element size: $h_{max} = 0.470\ m$
- Spatial discretization: 4th order, central + AV
- Temporal integration: explicit 4th order (LS) RK

The run was performed on an older production machine based on Intel E5-2699v3 processors and 4x FDR InfiniBand, with 36 cores/node, using 128 domains and 16 cores/domain for a total of 2,048 cores, i.e. about 100 Kpts/core. The results obtained after 30,000 timesteps are shown in Figure 24. The timings obtained were 0.51 sec/timestep, i.e. 2.433e-9 sec/timestep/pt. While off by a factor of 100 from the stated goal of 4 msec/timestep, there is room to increase to core count and improve the network. We are presently performing such runs, and will report the timings and results at the meeting.

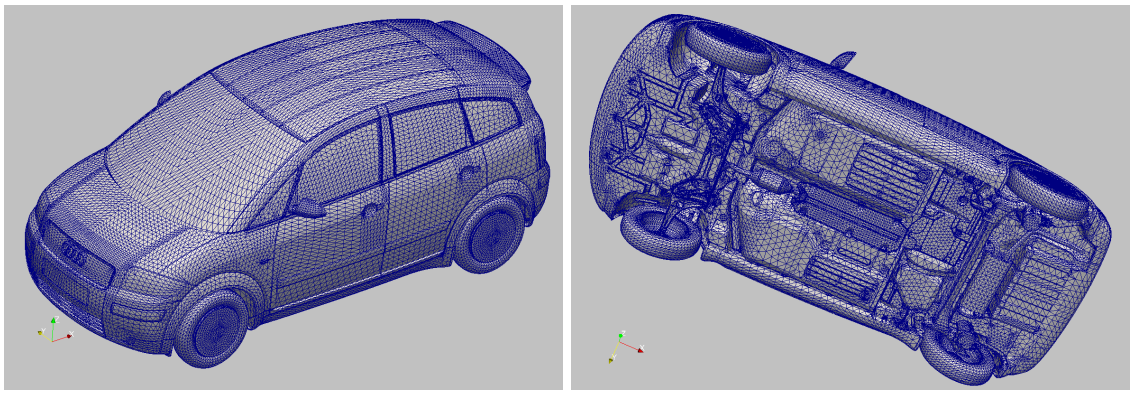

Figure 23: Audi A4: Surface Mesh Provided (STL)

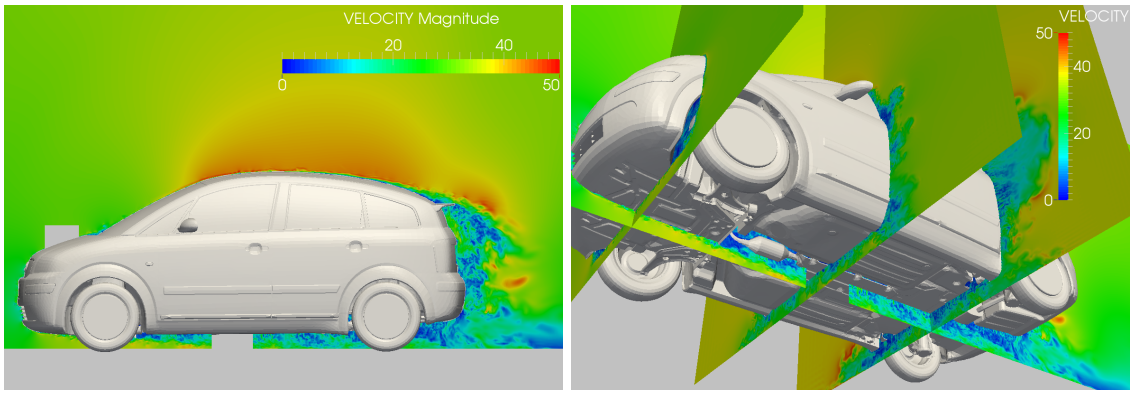

Figure 24: Audi A4: Velocities After 30,000 Timesteps

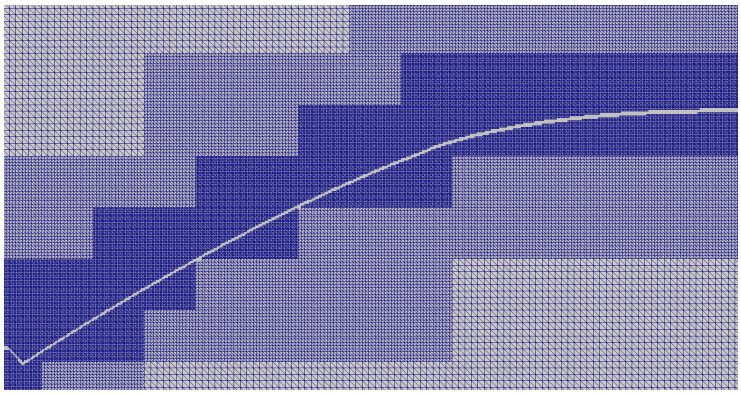

Figure 25: Audi A4: Closeup of Mesh in Symmetry Plane for Windshield Region

12 Conclusions

Simple interpolation schemes based on post-processing raw bi/trilinear halo-point transfer for nested cartesian grid systems have been developed. This allows to maximize modularity while preserving locality.
Results obtained for model problems indicate that the schemes improve the convergence rates and thus preserve the overall accuracy of finite difference codes with varying grid sizes.
The schemes were applied to complete car configurations, showing the expected overall performaned. It is expected that with further code optimization and hardware development that aim over overnight LES runs may be achievable in the near future.

References

[1] K. Nakahashi. *Building-Cube Method for Flow Problems with Broadband Characteristic Length*. Computational Fluid Dynamics, (S.W. Armfield, P. Morgan and K. Srinivas eds), Springer, Berlin, 2003.

[2] J. Sitaraman, A. Katz, B. Jayaraman, A.M. Wissink, and V. Sankaran. Evaluation of a multi-solver paradigm for cfd using overset unstructured and structured adaptive cartesian grids. *AIAA-2008-0660*, 2008.

[3] R. Löhner, A.T. Corrigan, K.R. Wichmann, and W. Wall. On the Achievable Speeds of Finite Difference Solvers on CPUs and GPUs. AIAA-2013-2852, 2013.

[4] R. Löhner, A.T. Corrigan, K.R. Wichmann, and W. Wall. Comparison of Lattice-Boltzmann and Finite Difference Solvers. AIAA-2014-1439, 2014.

[5] J. Sitaraman, V. Lakhminarayan, B. Roget, and A. Wissink. Progress in strand mesh generation and domain connectivity for dual-mesh cfd simulations. *AIAA-2017-0288*, 2017.

[6] E. Pärt-Enander and B. Sjögreen. Conservative and non-conservative interpolation between overlapping grids for finite volume solutions of hyperbolic problems. *Computer Fluids*, 23(3):551–574, 1994.

[7] D. L. Brown. An unsplit godunov method for systems of conservation laws on curvilinear grids. *Mathl. Comput. Modelling*, 20(10):29–48, 1994.

[8] G. Chesshire and W. D. Henshaw. A scheme for conservative interpolation on overlapping grids. *SIAM J. Sci. Comput.*, 15(4):819–845, 1994.

[9] J. Ray, C.A Kennedy, S. Lefantzi, and H.N Najm. Using high-order methods on adaptively refined block structured meshes – discretizations, interpolations and filters. *SAND2005-7981, Sandia National Laboratories, CA.*, 2005.

[10] P. McCorquodale and P. Collella. A high-order finite-volume method for conservation laws on locally refined grids. *Comm. in Applied Mathematics and Computational Science*, 6:1, 2011.

[11] J.A.F. Hittinger and J. W. Banks. Block-structured adaptive mesh refinement algorithms for vlasov simulation. *J. Comp. Phys.*, 241:118–140, 2013.

[12] C. Hirsch. *Numerical Computation of Internal and External Flow*. J. Wiley & Sons, 1991.

[13] Y. Kallinderis and A. Chen. An incompressible 3-d navier-stokes method with adaptive hybrid grids. *AIAA-96-0293*, 1996.

[14] R. Löhner. *Applied CFD Techniques, Second Edition*. J. Wiley & Sons, 2008.

[15] J. Mohd-Yusof. Combined immersed-boundary/b-spline methods for simulations of flow in complex geometries. *CTR Annual Research Briefs, NASA Ames Research Center/ Stanford Univ.*, pages 317–327, 1997.

[16] T. Ye, R. Mittal, H.S. Udaykumar, and W. Shyy. An accurate cartesian grid method for viscous incompressible flows with complex immersed boundaries. *J. Comp. Phys.*, 156:209–240, 1999.

[17] E.A. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof. Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. *J. Comp. Phys.*, 161:35–60, 2000.

[18] A. Gilmanov, F. Sotiropoulos, and E. Balaras. A general reconstruction algorithm for simulating flows with complex 3d immersed boundaries on cartesian grids. *J. Comp. Phys.*, 191, 2:660–669, 2003.

[19] E. Balaras. Modeling complex boundaries using an external force field on fixed cartesian grids in large-eddy simulations. *Comp. Fluids*, 33:375–404, 2004.

[20] A. Gilmanov and F. Sotiropoulos. A hybrid cartesian/immersed boundary method for simulating flows with 3-d, geometrically complex moving objects. *J. Comp. Phys.*, 207:457–492, 2005.

[21] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.

[22] J. Yang and E. Balaras. An embedded-boundary formulation for large-eddy simulation of turbulent flows interacting with moving boundaries. *J. Comp. Phys.*, 215:12–40, 2006.
[23] M. Storti, R.R. Paz, L. Dalcin, S. Costarelli, and S. Idelsohn. A fft preconditioning technique for the solution of incompressible flow on gpuâĂŹs. *Computers and Fluids*, 74:44–57, 2013.
[24] D.K. Clarke, H.A. Hassan, and M.D. Salas. Euler calculations for multielement airfoils using cartesian grids. *AIAA-85-0291*, 1985.
[25] J.E. Melton, M.J. Berger, and M.J. Aftosmis. 3-d applications of a cartesian grid euler method. *AIAA-93-0853-CP*, 1993.
[26] R.B. Pember, J.B. Bell, P. Colella, W.Y. Crutchfield, and M.L. Welcome. An adaptive cartesian grid method for unsteady compressible flow in irregular regions. *J. Comp. Phys.*, 120:278, 1995.
[27] M.J. Aftosmis, M.J. Berger, and G. Adomavicius. A parallel multilevel method for adaptively refined cartesian grids with embedded boundaries. *AIAA-00-0808*, 2000.
[28] A. Dadone and B. Grossman. An immersed boundary methodology for inviscid flows on cartesian grids. *AIAA-02-1059*, 2002.
[29] K. Nakahashi and L.-S. Kim. Building-cube method for large-scale, high-resolution flow computations. *AIAA-04-0434*, 2004.
[30] N. Peller, A. LeDuc, F. Tremblay, and M. Manhart. High-order stable interpolations for immersed boundary methods. *Int. J. Num. Meth. Fluids*, 252:1175–1193, 2006.
[31] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. J. Wiley, 2003.
[32] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comp. Phys.*, 53(3):484–512, 1984.
[33] A. Figueroa, R. Löhner, and J. Sitaraman. On Interpolation Schemes for Nested Cartesian Finite Difference Grids of Different Size. AIAA 2018-1561, 2018.
[34] S. Simakhina Y. Wang and M. Sussman. A hybrid level set-volume constraint method for incompressible two-phase flow. *J. Comp. Phys.*, 241:6438–6471, 2012.
[35] M. Olshanskii K. Terekhov, K. Nikitin and Y. Vassilevski. A semi-lagrangian method on dynamically adapted octree meshes. *Russ. J. Numer. Anal. Math. Modelling*, 30(6):363–380, 2015.
[36] R. Bermejo and A. Staniforth. The conversion of semi-lagrangian schemes to quasimonotone schemes. *Monthly Weather Review*, 120:2622–2632, 1992.