Documentation of the alternative FDS pressure solver SCARC

Beta-Version

Dr. Susanne Kilian

hhpberlin, Ingenieure für Brandschutz GmbH, D-10245 Berlin

s.kilian@hhpberlin.de

Contents

1 Motivation

The solution of elliptic partial differential equations (PDEs) is an integral part of the numerical solution of physical processes. Research into the development of numerical solvers for elliptic problems has a very long history and has produced a number of highly powerful approaches which, however, were originally tailored for purely serial applications of mostly small or moderate size. In view of the immense magnitude and complexity of today's CFD-problems, the efficient design of suitable parallel algorithms which fully exploit the current (super)computing power of modern high performance architectures seems inevitable and is a particularly great challenge.

But especially for elliptic problems, it is extremely difficult to develop algorithms which are numerically efficient and highly scalable at the same time: For this type of PDEs the solution at inner parts of the computational domain is strongly determined by all of its boundary values, that is, information must travel through the entire grid at least once. At the same time, these problems are characterized by an extremely high propagation speed for information such that even strictly localised effects immediately impact the overall solution in the whole domain. Thus, achieving fast convergence along with a high level of computational accuracy (*numerical efficiency*) is closely linked to the speed at which information can be exchanged across the entire domain. In contrast to that, the efficient parallelization of a numerical algorithm is mainly determined by the amount of interprocessor-communication needed to coordinate the single processors to ultimately produce a global solution. Achieving a good scalability to large processor counts (*parallel efficiency*) requires as much data locality as possible in order to minimize the communication overhead between the single processors. So, a proper compromise has to be found between those two very contradictory requirements.

A very natural approach for the parallel solution of PDEs based on finite difference discretization like in FDS is the use of *domain decomposition* techniques: The computational domain is subdivided into smaller subdomains which are assigned to the different processors of a parallel computer and simultaneously perform a series of local computations which are coordinated by synchronized data exchanges among each other. This parallel procedure may reduce the required runtime for a given constellation and enlarge the class of computable problems comprehensively.

Special care must be taken to ensure that this artificial subdivision does not impair the inherent global character of the underlying problem which especially holds true for elliptic problems. In particular, it must be guaranteed that the parallelization process preserves the convergence order and approximation quality of well-proven serial algorithms as best as possible. However, the efficiency of many serial elliptic solvers mainly relies on highly recursive data dependencies strongly connecting all parts of the domain which implies an extremely low level of intrinsic parallelism. In order to achieve a better parallel efficiency they must be split off and adapted to the new architectural features which typically requires extensive structural modifications to the numerical core and is often associated with more or less considerable losses of numerical efficiency (worse approximation quality, dependencies on the number of subdomains or the refinement parameters). The effects of this situation on the FDS pressure solution are described in more detail below and different solution techniques are presented.

2 The Pressure Poisson equation in FDS

The pressure equation in FDS, an elliptic partial differential equation of second order which is commonly referred to as *Poisson equation*, reads as

$$\nabla^2 H = -\frac{\partial(\nabla\cdot\mathbf{u})}{\partial t} - \nabla\cdot\mathbf{F}\,. \tag{1}$$

Here, $\mathbf{u}$ denotes the velocity field and $\mathbf{F}$ the force term which includes the thermodynamic contributions from the previous time step such as radiation, combustion, pyrolysis, etc. The pressure term $H \equiv |\mathbf{u}|^2/2 + \tilde{p}/\rho$ incorporates the density ρ as well as the perturbation pressure $\tilde{p}$ by which the fluid motion is driven. The numerical solution of Equation (1) requires the specification of appropriate boundary conditions. Detailed information about its mathematical derivation and the possible types of boundary conditions can be found in the FDS Technical Reference Guide [**?**].

The numerical time-stepping scheme in FDS, an *explicit second-order predictor/corrector scheme*, requires the solution of Equation (1) at least twice per time step which in total requires a considerable part of the overall computing time. Because of its strong interaction with the calculation of all other thermodynamic quantities, the solution of the pressure equation is an essential step in the entire solution process and must be treated as efficiently and accurately as possible.

Due to the explicit character of the time-stepping scheme, FDS basically possesses a very high potential for parallelization. While performing a new time step, each sub-mesh must only access values from the previous time step which have already been computed before. Typically only local data exchanges between neighboring sub-meshes are required which can be accessed with high computational efficiency on modern parallel architectures.

2.1 Discretization by finite differences

The first step towards a numerical solution of Equation (1) is to superimpose the computational domain Ω with a corresponding grid in whose cells the respective values of the analytical solution are determined only approximately. Thereby, based on default subdivision into structured rectangular cells within FDS, the spatial derivative of H in Equation (1) is replaced by a *central difference quotient* which is evaluated in the centers of the individual grid cells. In case of equidistant grid size h, Figure (1) sketches the underlying well-known *7-point stencil* in 3D which approximates the second derivative cell by cell, as well as its 2D counterpart. The final discretized version of the Poisson equation looks like

$$\begin{aligned}\frac{H_{i+1,jk}-2H_{ijk}+H_{i-1,jk}}{\delta x^2} + \frac{H_{i,j+1,k}-2H_{ijk}+H_{i,j-1,k}}{\delta y^2} + \frac{H_{ij,k+1}-2H_{ijk}+H_{ij,k-1}}{\delta z^2} \\ = -\frac{F_{x,ijk}-F_{x,i-1,jk}}{\delta x} - \frac{F_{y,ijk}-F_{y,i,j-1,k}}{\delta y} - \frac{F_{z,ijk}-F_{z,ij,k-1}}{\delta z} - \frac{\delta}{\delta t}(\nabla\cdot\mathbf{u})_{ijk}\,.\end{aligned} \tag{2}$$

The *finite difference* approximation (2) of H turns out to be of second-order accuracy, i.e. the error between the analytical and numerical solution is proportional to the 2th power of the grid width. Note, that different discretizations for the time derivative of the velocity divergence are used in the predictor and corrector parts of the time stepping scheme, see again the FDS Technical Reference Guide[**?**] for more details.

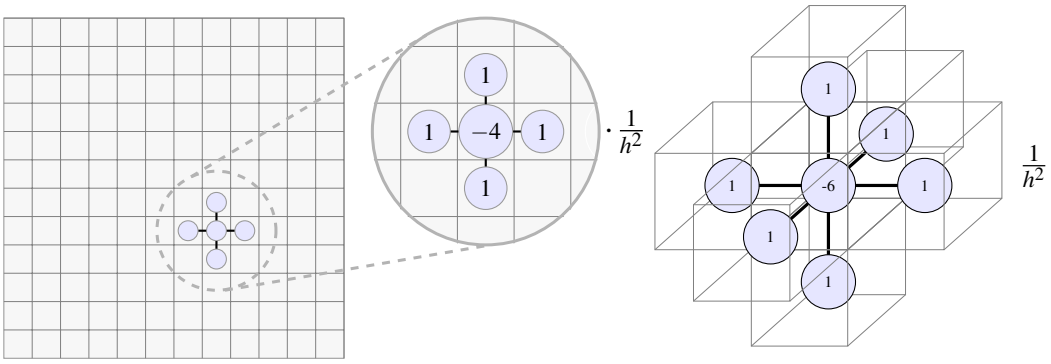

Figure 1: Central difference quotients: (Left) 5-point stencil in 2D. (Right) 7-point stencil in 3D.

Typically, this discretization process leads to a huge system of equations, which has to be solved efficiently by means of a suitable numerical solver. The growing complexity of contemporary applications thereby increasingly requires the use of high performance computers and corresponding discretizations, which are suitable for parallel application.

2.1.1 Global versus local discretization

In order to describe the underlying discretization processes and the basic operation principles of the various Poisson solvers available in FDS, Figure 2 introduces a small pipe-shaped geometry in 2D. This simple demonstration case is used repeatedly throughout the entire documentation and is also the basis for later test calculations.

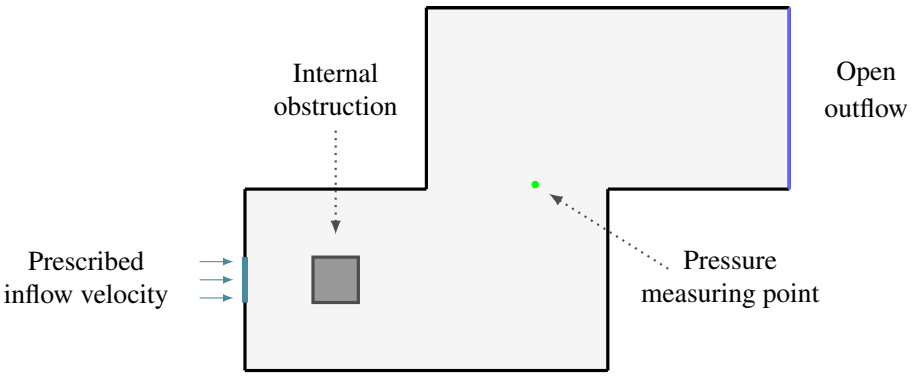

Figure 2: Pipe-shaped example geometry in 2D with a small internal obstruction, a predefined inflow from the left, a pressure measuring point for the pressure and an open outflow on the right.

If the whole pipe geometry is discretized in one piece, one global system of equations is obtained,

$$Ax = b\,, \tag{3}$$

whereby n denotes the number of total grid cells, $x, b \in \mathbb{R}^n$ the corresponding solution and right hand side vectors and $A \in \mathbb{R}^{n\times n}$ the overall Poisson matrix.

However, since FDS is based on partitions of the computational domain into M rectilinear sub-meshes Ω_i with regular local grids each, every sub-mesh holds its own system of equations

$$A_i x_i = b_i\,, \qquad i = 1,\ldots,M, \tag{4}$$

with corresponding local numbers of grid cells n_i, local solution and right hand side vectors $x_i, b_i \in \mathbb{R}^{n_i}$ and locally defined Poisson matrices $A_i \in \mathbb{R}^{n_i \times n_i}$. A more formal definition of these quantities will be given in Section (4.1.3). The resulting global and local discretization processes are illustrated in Figure 3.

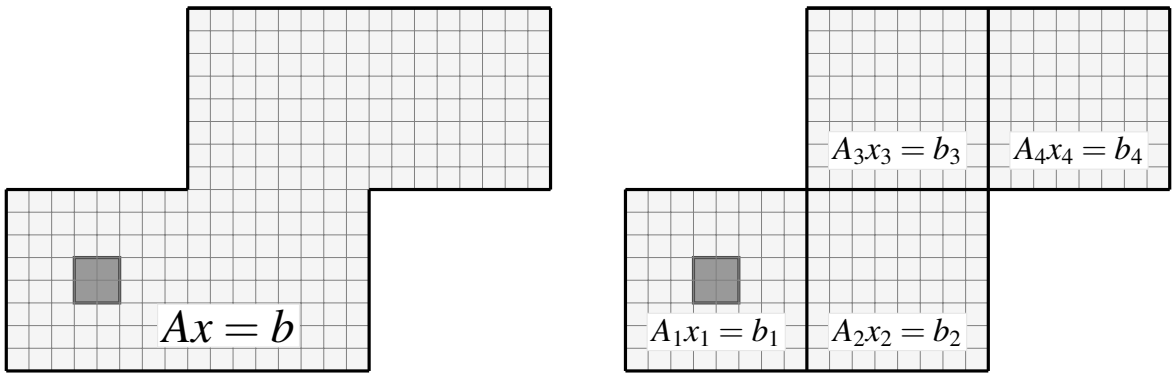

Figure 3: Different discretization types for the Poisson problem on the 2D-pipe with 4 meshes: (Left) Global discretization with one globally defined system of equations $Ax = b$. (Right) Local discretization with four locally defined systems of equations $A_i x_i = b_i$.

Note that a global discretization can also be achieved in a multi-mesh context, i.e. in a *data-parallel* sense. For this purpose, each mesh stores the respective parts of the solution and right-side vector as well as the global matrix, informally spoken "$A_i \equiv A\big|_{\Omega_i}$". By means of suitable data exchanges between adjacent meshes the individual matrix-vector operations are then performed in exactly the same way as would also be the case with an actual global execution.

With regard to parallelization the use of local discretizations seems to be the more natural approach. Thus, the crucial question is, if there are appropriate solutions strategies for the local discretization type such that the obtained collection of mesh-wise solutions is comparably accurate as the overall solution for the corresponding global discretization, i.e. $\sum_{i=1,\ldots,M} x_i \sim x$. This question will be addressed with respect to the various Poisson solvers below.

2.1.2 Structured versus unstructured discretization

For the discretization of the single meshes two different strategies can be applied in FDS. They essentially differ in how they treat cells internal to solid objects within the domain and their neighboring gas-phase cells. Both types have their advantages and disadvantages and significantly determine which solution strategy can be applied at all. To illustrate the differences between the two, Figure 4 illustrates the respective matrix stencils and grid numberings. Therein, both plots are related to the lower left mesh of the 2D pipe geometry containing the internal obstruction as can be seen in the right plot of 3 again for the case of local discretization.

Structured discretization: This type explicitly includes both, gas-phase and solid cells, and is the default type in FDS. Cells internal to solid obstructions are masked as blocked cells. But the same matrix stencil is applied all over the domain, regardless whether the underlying cell is a gas-phase or a solid cell. Without any exception all grid cells are incorporated into the resulting discretization matrix which therefore takes a very regular shape. As a major advantage this strategy allows the use of highly tuned solvers developed specifically for regular grids which can be performed with enormous computational efficiency. But, there is no way to describe the correct boundary conditions along internal objects: While the no-flux boundary condition is exact at external boundaries, it is not possible to prescribe the required homogeneous no-flux condition at internal boundaries directly. Erroneously, the velocity field may contain non-zero contributions towards internal solids associated with a corresponding loss of accuracy which represents the greatest disadvantage of this strategy. By using appropriate corrective actions, this effect can be remedied as is also done in FDS and will be described below.

Unstructured discretization: This type only incorporates the gas-phase cells while it omits all solid cells from the system of equations. This strategy is the only way to completely get rid of the undesirable penetration errors. On gas-phase cells which are directly adjacent to the surface of an obstruction the homogeneous no-flux Neumann condition is explicitly specified and included into the matrix. Thus, the greatest advantage of this approach is that it offers more accuracy because the correct boundary information along internal obstructions can now be included into the system. But, in contrast to the structured case this strategy requires the usage of individual matrix stencils for the different grid cells depending on their location with respect to obstructions. While the resulting unstructured matrix has less entries than the corresponding structured one, it no longer has a regular shape and makes higher demands on the robustness of the solver, which may comprehensively limit the achievable efficiency and is the main disadvantage of this approach.

Furthermore, Figure 5 opposes the sparsity patterns of the resulting Poisson matrices to each other. As can be seen clearly, both discretization types lead to sparse matrices which basically have only very less non-zero elements. Based on the underlying stencils the non-zeros are restricted to only a few diagonal bands (5 in 2D, 7 in 3D), which is extremely less compared to the number of possible entries. For mesh 1 in the local discretization of the 2D-pipe, consisting of n_1 cells in total, this amounts to only about $5n_1$ entries compared to the possible number of entries n_1^2. Note, that these are the values which have to be stored on the computing system and therefore significantly determine the memory requirements for the solver in use.

However, it also becomes apparent that the structured discretization results in a highly regular matrix with a constantly repeating pattern, while the unstructured discretization shows irregularities associated with the different treatment of the internal obstruction as indicated with the red circle in the right plot. For a simple obstruction like the one considered here, the differences are not big, but the situation is completely different for more realistic applications with complex geometric details. In any case, the mentioned optimized solvers for regular grids can no longer be applied in the unstructured case and more generally applicable solvers must be used.

Note, that the unstructured discretization even leads to less matrix entries since the obstruction wasn't included in the discretization process. But this supposed advantage typically does not carry weight due to the more complex matrix structure and the increased demands on a respective solver.

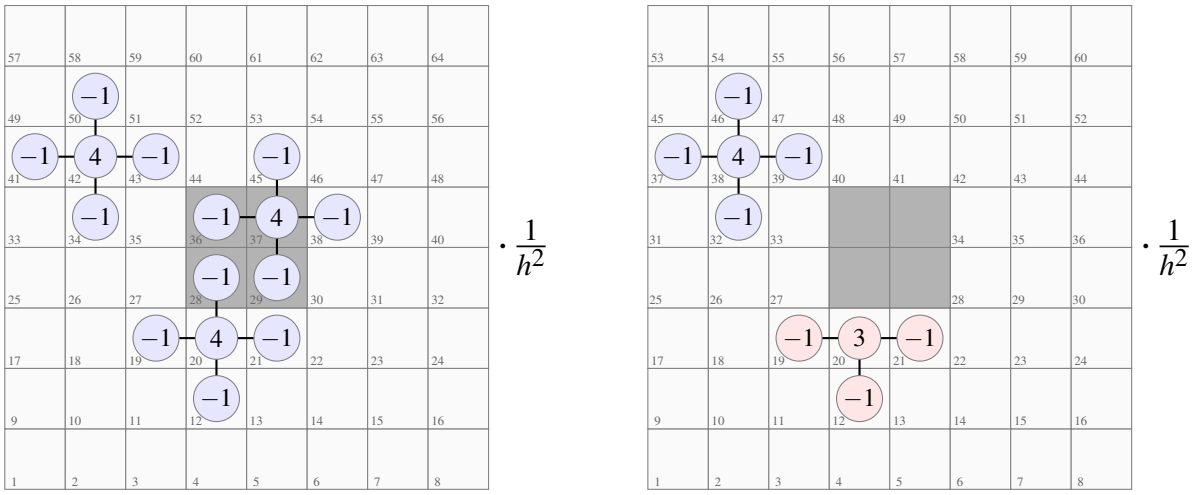

Figure 4: Matrix stencils and grid numberings for structured and unstructured discretizations:
(Left) The structured discretization uses the same matrix stencil everywhere, including cells internal to the solid obstruction. Based on a lexicographic numbering the connectivity graph of the matrix is defined a-priori and and the resulting matrix takes a highly regular shape.
(Right) The unstructured discretization uses individual matrix stencils, excluding cells internal to the solid obstruction. There is no consecutive numbering anymore. Instead, an additional connectivity graph must be stored which specifies the neighbor relations between the single grid cells. For gas-phase cells adjacent to the obstruction the homogeneous boundary conditions are individually incorporated into the matrix which no longer has a regular shape.

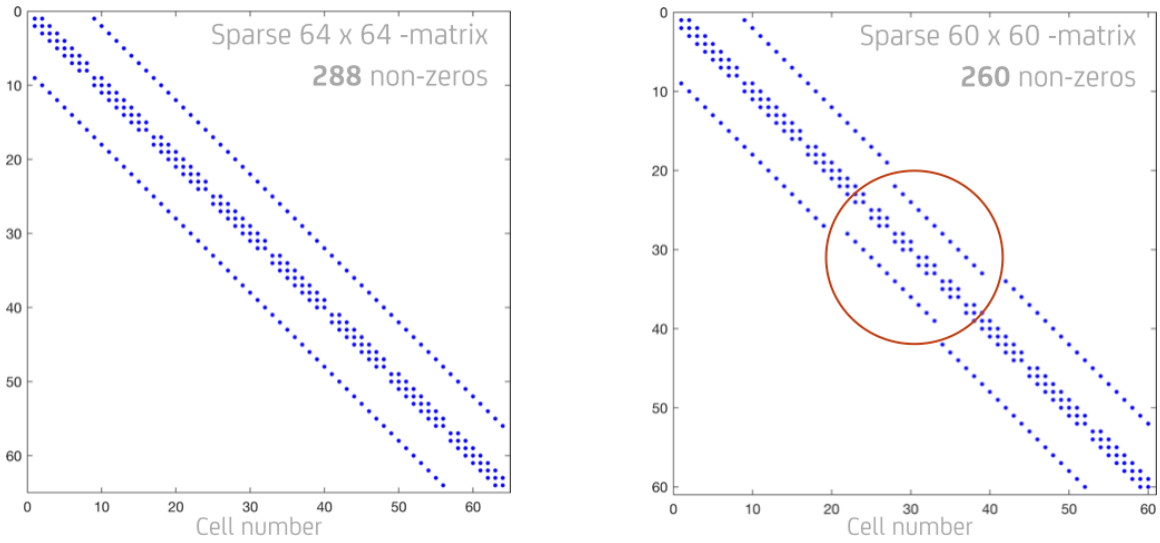

Figure 5: Sparsity patterns of the Poisson matrices for structured and unstructured discretizations:
(Left) The structured discretization shows a constantly repeating pattern leading to a highly structured matrix with 288 non-zero entries in total.
(Right) The unstructured discretization shows irregularities related to the internal obstruction leading to an irregularly structured matrix with 260 non-zero elements in total.

2.2 Treatment of boundary conditions

The appropriate treatment of the boundary conditions is of major importance for the accuracy of the numerical pressure solution. There are two basic classes of boundary conditions which must be specified at mesh faces matching with the exterior borders of the computational domain and will be explained subsequently.

The incorporation of the boundary conditions into Equation (3) or (4) leads to corresponding changes in the matrices and right hand side vectors of the single systems. In order to exemplarily illustrate these mechanisms for the different types of boundary conditions, the 2D-analogue of Equation (2) is considered for simplicity,

$$\frac{H_{i+1,k} - 2H_{ik} + H_{i-1,k}}{\delta x^2} + \frac{H_{i,k+1} - 2H_{ik} + H_{i,k-1}}{\delta z^2} = R_{ik}\,, \tag{5}$$

where R_{ik} represents an abbreviation of the corresponding right hand side in 2D. Note, that for 2D-applications, the number of cells in y-direction is fixed to 1 in FDS. Therefore, only the i and k indices are varied in Equation (5).

Dirichlet boundary conditions: This type of boundary conditions is applied to open boundaries, where the fluid motion into or out of the domain is driven by the pressure gradient. In this case a value for H itself is specified at corresponding grid cells.

To illustrate the computation of the related matrix entries, an example of the FDS Technical Reference Guide [**?**] is used. There, open boundary conditions along the left domain boundary $x = x_{min}$ are considered based on values $H_{\frac{1}{2},k}$ for all indices k along the z-direction. Using these values, the corresponding ghost cell values $H_{0,k}$ along $x = x_{min}$ can be defined by linear extrapolation

$$H_{0,k} = 2H_{\frac{1}{2},k} - H_{1,k}\,.$$

Replacing the ghost values for all matrix lines with index $i = 1$ in Equation (5), the updated matrix entries for all different values of k look like

$$\frac{H_{2,k} - 3H_{1,k}}{\delta x^2} + \frac{H_{1,k+1} - 2H_{1,k} + H_{1,k-1}}{\delta z^2} = R_{1,k} - \frac{2}{\delta x^2} H_{\frac{1}{2},k}\,.$$

Neumann boundary conditions: This type of boundary conditions is applied to internal solid obstructions as well as external faces which are entirely determined by a forced flow or a solid. In this case a value for the normal gradient $\partial H/\partial n$ must be specified via a corresponding *forced-flow* condition

$$\frac{\partial H}{\partial n} = -F_n - \frac{\partial u_n}{\partial t}. \tag{6}$$

F_n denotes the normal component of $\mathbf{F}$ at vents or solid walls and $\partial u_n/\partial t$ a prescribed rate of change in the normal component of velocity at a forced vent. In case of non reacting solid material, a *homogeneous* Neumann condition is present, i.e. the right hand side of Equation (6) is zero.

As an example, a forced flow condition is considered for the lower domain boundary $z = z_{min}$,

$$\frac{H_{i,1} - H_{i,0}}{\delta z} = B_{i,0}\,,$$

according to known values $B_{i,0}$ for all indices i. Using the relation $H_{i,0} = H_{i,1} - \delta z B_{i,0}$ for all matrix lines of Equation (5) with $k = 1$ leads to

$$\frac{H_{i+1,1} - 2H_{i,1} + H_{i-1,1}}{\delta x^2} + \frac{H_{i,2} - H_{i,1}}{\delta z^2} = R_{i,1} + \frac{1}{\delta z} B_{i,0}\,.$$

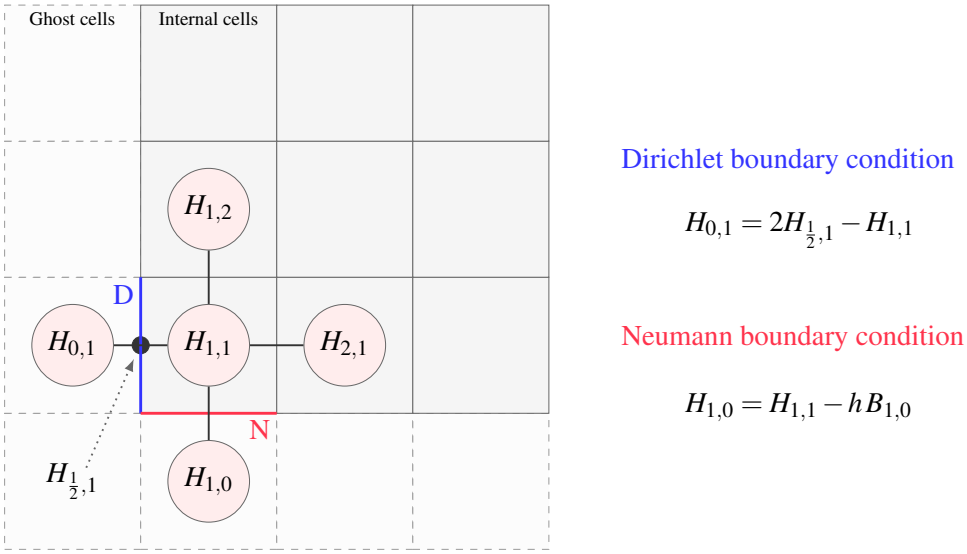

Figure 6: Specification of Dirichlet and Neumann boundary conditions for predefined values for $H_{\frac{1}{2},1}$ and $B_{1,0}$ at the lower left corner of a 2D domain; the ghost cell values $H_{0,1}$ and $H_{1,0}$ in Equation (5) are replaced correspondingly.

Suppose that a Dirichlet-boundary meets a Neumann-boundary, as illustrated in Figure (6) for the lower left corner of a 2D-domain and that the grid resolutions in x and z-direction are the same, i.e. $h = \delta x = \delta z$, then the matrix line for the grid cell $(1,1)$ finally looks like

$$\frac{1}{h^2}(H_{0,1} - 4H_{1,1} + H_{1,2}) = R_{1,1} - 2\frac{1}{h^2} H_{\frac{1}{2},1} + \frac{1}{h} B_{1,0}\,.$$

The resulting matrix A is irreducibly diagonally dominant which guarantees the solvability by many iterative methods. If there is a mix of open and solid surfaces along an external face, the specification of the Dirichlet condition is given precedence to the Neumann condition (i.e. it is more important to specify H itself than its gradient). In case of multi-mesh applications, it may be necessary to also specify Dirichlet boundary along internal mesh boundaries. However, this depends on the underlying solution technique and will be explained in more detail below.

2.3 Parallelization of Poisson solvers

Commonly used approaches for the solution of the Poisson equation (1) are either based on direct or iterative solution strategies which will be described in more detail in the next sections. Their efficient parallelization must bridge the gap between the following two very conflicting requirements:

- **Efficient convergence behavior** ('Globality' preferred)
  Especially for globally coupled systems such as elliptic equations fast convergence is essentially based on how well an algorithm is able to reproduce global dependencies. Powerful elliptic solvers mainly rest on highly recursive algorithmic structures which strongly couple the overall data of the whole domain. Thus, they have very little inherent parallelism.

- **Good parallelization properties** ('Locality' preferred)
  In contrast to that, methods with a high degree of parallelism often possess a strongly local character. In the ideal case they only need local communication between neighboring subdomains which can nowadays be performed with high computational efficiency on modern parallel platforms.

For single-mesh applications the strongly recursive design of traditional elliptic solvers very well reflects the fast and globally acting character of the pressure. In this case, all data required during the computations are easily accessible in the storage of the only processor in use which allows a high computational efficiency.

For multi-mesh applications the first obvious idea consists in the development of a 1-to-1 parallel analog by splitting up the serial structures according to the new decomposition. But in this case, all required recursive data would be distributed across the entire computing system and would have to be exchanged very frequently which is extremely inefficient from a computational point of view. In order to increase the degree of parallelism the only remedy is to modify existing techniques or to develop completely new strategies, which are better adapted to modern parallel architectures. However, this process is necessarily associated with a loosening of the global coupling. New local information can no longer be distributed globally just when it is needed, but only with a time delay or, in order to save computational overhead, only in reduced or coarser form. Typically, this leads to dependencies on the number of subdomains or the grid resolution associated with losses of numerical efficiency and possible troubles in the correct mapping of actual physical processes.

All in all, these observations can be summarised as follows: 'Well parallelizable algorithms for elliptic problems are often numerically inefficient, numerically efficient algorithms are hardly parallelizable.' The task in designing a suitable solver is therefore to find a suitable compromise. Usually there is no general rule that applies in all cases. Rather, attention must be paid to the specific characteristics of the underlying problem. The optimal strategy for a problem with only a few sub-meshes can be completely different from that for a massively parallel application. For this reason it is of great importance to understand the advantages and disadvantages of different solution strategies and to make the best possible use of this knowledge.

3 Direct Poisson solvers in FDS

At a first glance, the ongoing improvements in the current computer technology motivate the use of direct methods such as the Gaussian elimination method and its variants for symmetric, positive definite matrices for the solution of the elliptic equations. Another extremely powerful class of direct methods which is successfully used in many different branches of science is based on spectral solvers. Both classes will be explained in more detail below.

Direct solvers compute the solution of a system of equations within a possibly very complex step without intermediate approximations. They may be performed with enormous speed and are often used for the demonstration of potential computer power, see e.g. the LINPACK-tests by [**?**]. They have proven to be very robust even in the non-symmetric and ill-conditioned case. Besides, they are nearly independent of the degree of grid distortion (except for rounding errors) and well suited for the application on unstructured grids. In contrast to iterative methods, they are completely independent of whether a good starting solution exists or not, but they also do not benefit if this is already the case. They achieve a high level of computational accuracy, but they do not benefit if this is not required at all.

However, as already mentioned, the development of corresponding parallel analogues is highly complicated. Most often, reasonably efficient strategies follow purely algebraic considerations and are not conforming with geometrically motivated domain decompositions strategies. Nevertheless, for moderately sized problems, especially if only one or a few sub-meshes are used, direct solvers are incomparably fast and should generally be preferred over iterative ones. Typically, there is a critical number of subdomains for which the parallelization overhead dominates the numerical efficiency associated with the original serial methodology which should be analyzed carefully.

3.1 Parallel FFT with Pressure Iteration

Spectral solvers like the *Fast Fourier Transformation* (FFT) exploit a very special property of the underlying Poisson problem, namely that sine and cosine functions are eigenvectors of the Laplace operator. They expand the solution as a Fourier series which can be quickly performed at rather low complexity. In practice, this approach has proven to be highly efficient and is used in many different fields of application. However, FFT methods are restricted to structured grids which may impede the use for complex geometries.

As described in detail in the FDS Technical Guide [**?**], the pressure equation (1) is obtained from the momentum equation by a sequence of substitutions and simplifications. The major advantage of this derivation is that it finally leads to a system of equations which has constant coefficients (i.e. is separable). Since FDS furthermore relies on the use of cubic meshes each with structured rectangular grids each, it is possible to use a highly optimized FFT solver from the CRAYFISHPAK [**?**]. In the single-mesh case only one globally acting FFT-method is performed which has proven to be extremely powerful and fast over the past years. In the multi-mesh case every submesh performs its own local FFT in order to compute an exact solution to the related local Poisson problem as illustrated in Fig. 7. The global solution is then composed from the local ones.

A drawback associated with this approach is that the mathematical solvability of the local problems requires the definition of appropriate boundary conditions all over the local boundary, i.e. along the new artificial boundaries between the single meshes where the true boundary conditions

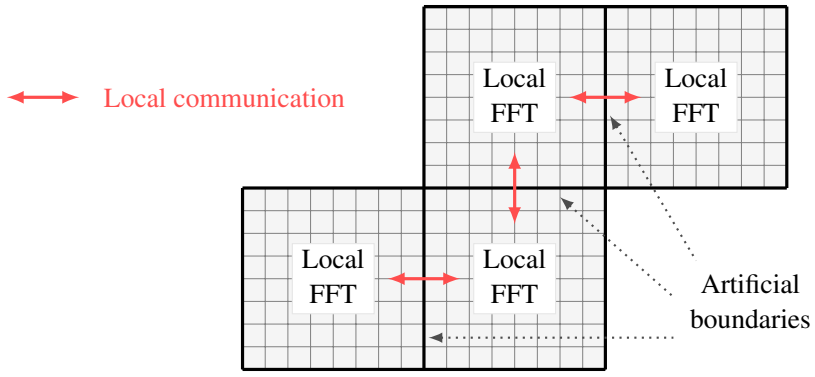

Figure 7: Mesh-wise FFT-methods for the 2D pipe geometry: First each mesh performs its own local FFT method to compute the solution of the respective local Poisson problem. Then, the global solution results from the composition of the local solutions.

are not known at all. Instead, they are only approximately defined as Dirichlet boundary values, consisting of the mean values of neighboring cells in adjacent meshes taken from the last time step, which may lead to losses of accuracy. On the other hand, this strictly local oriented approach possesses an extremely high parallel efficiency because the exchange of the internal boundary values only requires next-neighbor communications which can be performed with great computational efficiency on today's parallel architectures.

As already described, the most fundamental characteristic of the pressure equation (1) is a very fast propagation speed for information. Only a single time step may suffice to spread new pressure information over the whole computational domain, i.e. local effects or perturbations have immediate impact on the overall solution.

Due to the purely local character of the mesh-wise FFT-solver, however, new information can only be transferred mesh-by-mesh, successively using the data exchanges between adjacent meshes as illustrated in Fig. 8. This necessarily involves a time delay compared to the real physical propagation speed and brings dependencies on the number of subdomains into play. The higher the number of subdomains and the associated artificial fragmentation of the global connectivity is, the stronger this effect may be with corresponding negative impacts on the accuracy and stability of the whole method. Nevertheless, for small numbers of subdomains this effect has proven to be very weak and the efficiency of the local FFT-methodology mainly dominates.

There is yet another important issue regarding the treatment of internal obstructions that should be considered: Since the use of the FFT-solver is basically limited to structured meshes, it is not possible to specify internal boundary conditions and the velocity field may penetrate into internal solids as already explained in Section (2). In order to compensate this undesired effect an additional iterative correction strategy based on a *Direct Forcing Immersed Boundary Method* [**?**] is used in FDS: In every time step the mesh-wise FFT algorithm is not applied only once but multiple times until a specified *velocity tolerance* for the remaining velocity error along internal obstructions has been reached. This iterative corrective process will be denoted as *pressure iteration* below.

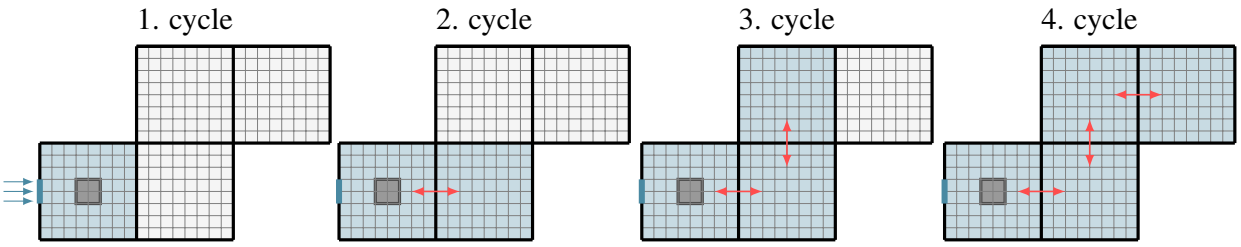

Figure 8: Delayed information transfer of the mesh-wise FFT solver for the 2D pipe geometry: New information can only be passed across the whole domain by successively using the next-neighbor communications between adjacent meshes.

Furthermore, although the pressure term H is continuous at mesh interfaces, the finite-difference of its gradient is not such that the normal components of velocity also may be different at common mesh interfaces. Thus, the above pressure iteration is also used to drive the velocity normals to match up to the specified velocity tolerance along common mesh interfaces.

For both, the internal obstructions and the mesh interfaces, the required velocity tolerance and the maximum allowed number of correction iterations can be specified by the quantities `VELOCITY_TOLERANCE` and `MAX_PRESSURE_ITERATIONS` in the `&PRES`-namelist. By default, a velocity tolerance of "characteristic mesh size" divided by 10 and a maximum of 10 iterations is used. Fig. 9 illustrates this procedure graphically. More detailed information about the underlying procedure can be obtained in the FDS Technical Reference Guide [**?**].

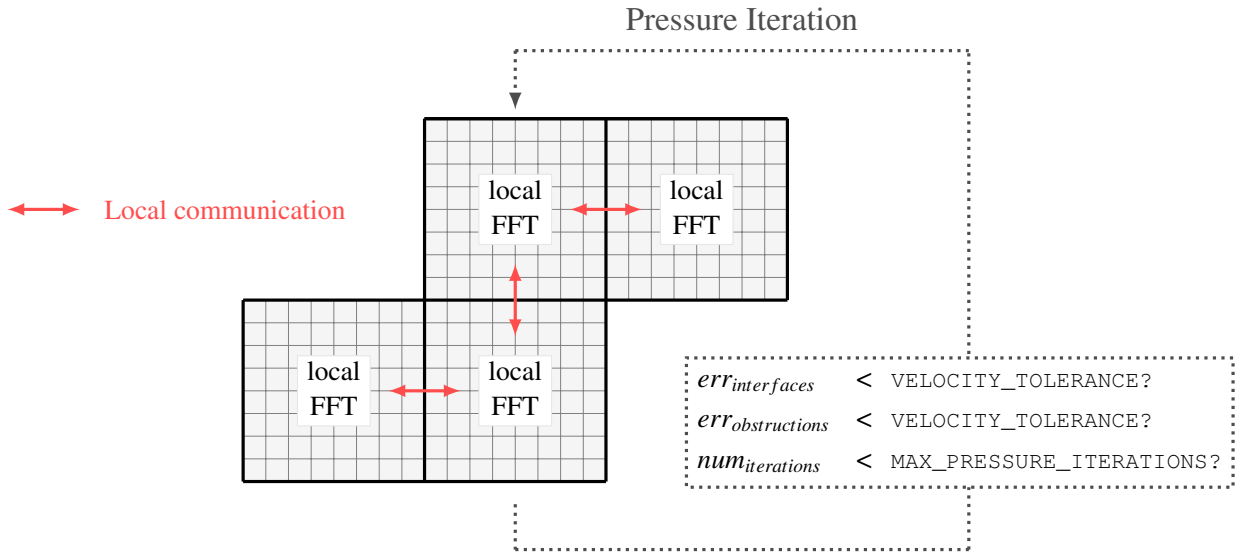

Figure 9: Repeated use of the FFT pressure solver for a 2D pipe geometry: In the scope of a surrounding pressure iteration the mesh-wise FFT's are performed multiple times until the error related to the normal components of velocity along internal obstructions and mesh interfaces has been driven below a specified velocity tolerance.

Certainly, the increased number of mesh-wise FFT solutions leads to a higher computational effort. Nevertheless, for a multitude of cases, especially for relatively small mesh numbers or in steady-state like situations, only less corrective iterations are needed and convergence is achieved very quickly. Increasing the number of subdomains may worsen the convergence and accuracy behaviour associated with a noticeable increase of computational costs.

Especially, for extended geometries with a big number of meshes (i.e. tunnels) and/or transient boundary conditions this purely local strategy may experience difficulties to reproduce the rapid propagation of information for elliptic problems fast enough and convergence of the pressure iteration may be slow. In particular, these situations make the development of alternative solution concepts appear necessary and have driven their development forward.

3.2 Parallel *LU*-factorization

Within the framework of the Gaussian elimination algorithm, many direct algorithms rely on the 'lower-upper'-factorization of the system matrix, $LU = PAQ$, with suitable permutation matrices P and Q and the triangular matrices L and U, see Fig. 10. The solution of Equation (3) can then be obtained using a forward substitution step, $Ly = P^T b$, followed by a backward substitution step, $U(Q^T x) = y$. If A is symmetric, a *Cholesky factorization* $PAP^T = LL^T$ can be applied.

The whole process can be subdivided into three phases: (i) a *reordering phase* where the matrix is analyzed to produce an ordering which allows a more efficient factorization, (ii) a *factorization phase* where the *LU*-factorization is actually computed and stored, (iii) a *solution phase* where the forward and backward substitution is performed. Typically, the factorization phase (ii) requires the most computing time while the solution phase (iii) is an order of magnitude faster.

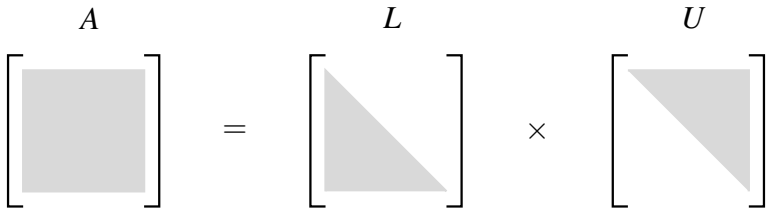

Figure 10: Decomposition of A into a lower triangular matrix L and an upper triangular matrix U

While *LU*-decomposition was originally developed for serial applications, there are optimized parallel variants today which calculate a distributed LU decomposition based on a global discretization with corresponding global Poisson matrix. In FDS a solver named UGLMAT is used, which is built on the optimized parallel `Cluster_Sparse_Solver` of the Intel® MKL library.

Due to its domain-spanning character the factorization phase (ii) is very communication-intensive which is reflected in long calculation times for initialization. However, as long as there are no geometric changes in the computational domain during the simulation (e.g. by the "opening" or "closing" of obstructions), this only has to be done once. In every FDS time iteration the solution of the respective Poisson equation is simply based on a forward and backward substitution step on every single mesh with respect to the stored factorization which can be done much faster.

Figure 11 illustrates the LU-decomposition for the 2D-pipe example. The dashed lines between the single meshes indicate that the whole method is based on the global Poisson matrix and the mutually required entries are communicated between the single meshes.

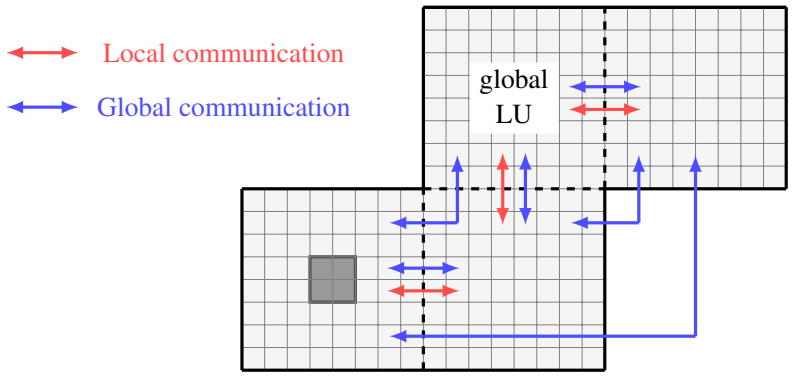

Figure 11: Decomposition of A into a lower triangular matrix L and an upper triangular matrix U

A main disadvantage of this approach is that less benefit can be drawn from a very convenient property of the Poisson matrix A, namely its intrinsic *sparsity*: Even though A has only very few non-zero entries compared to the total number of possible entries n^2, the LU-factorization process leads to *fill-in*, i.e. it produces non-zero entries in L and U where A was zero before. This relation is illustrated in Fig. 12. For a simple 3D-cube geometry which is refined into 16^3 cells it shows the sparsity pattern of the Poisson matrix A and the lower triangular matrix L of its LU-decomposition.

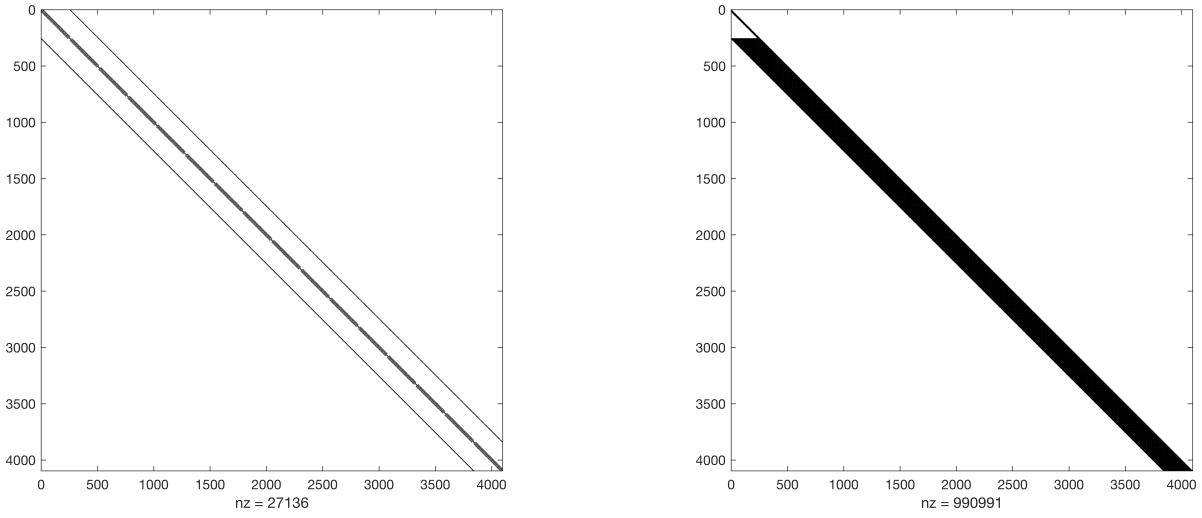

Figure 12: Sparsity patterns in case of the LU-decomposition for a refined 3D-cube with 16^3 cells. (Left) Poisson matrix A with about 27 thousand non-zeros; (Right) Lower triangular matrix L with nearly 1 million non-zeros;

The computing and storing of both triangular matrices can become prohibitively expensive, especially in case of huge systems of equations with many millions of unknowns. This is true even if only the lower triangular matrix has to be stored for symmetric problems.

The example of the 3D-cube is taken up again in order to show the magnitudes of memory requirements to be expected here. In case that the cube is refined from 16^3 up to 128^3 cells, Fig. 13 compares the number of non-zero entries of L with those of A. Apparently the ratio dramatically deteriorates as the grid is refined. For the finest grid resolution of 128^3 cells the matrix L has about 2.18 billions of non-zeros whereas A only has about 8.34 millions of non-zeros, corresponding to a ratio of about 238. Note, that for A only its lower triangular and diagonal part must be stored due to its symmetry.

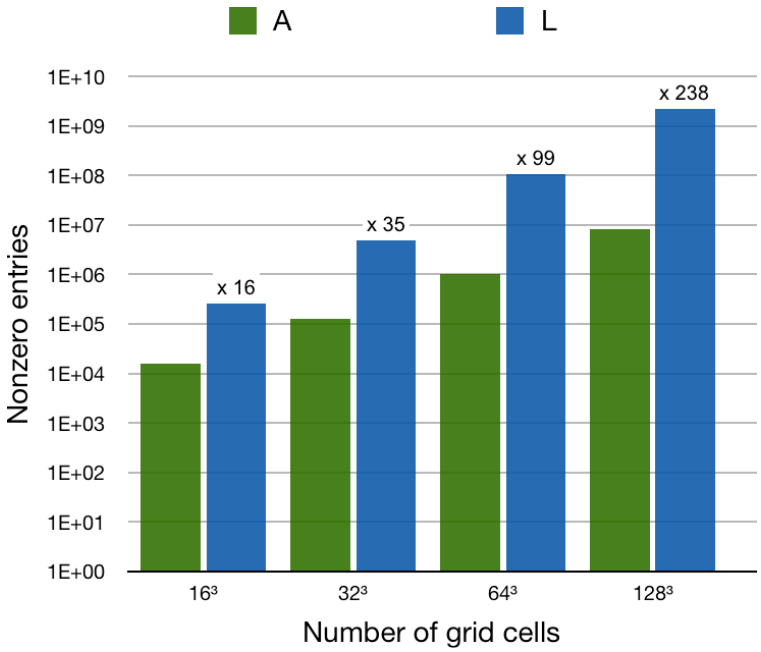

Figure 13: Number of non-zero entries in the Poisson matrix A (green) and the lower triangular matrix L (blue) of its LU-decomposition for different refinements of a 3D-cube from 16^3 up to 128^3 cells. The resulting ratio of L to A is indicated at top of the blue bars.

For realistically large CFD-problems the resulting additional storage requirements can exceed the capacity of the available computing system in the worst case. Thus, when considering the computational efficiency of LU-methods for a given application the resulting fill-in is a decisive criterion.

4 Iterative Poisson solvers in FDS

In contrast to direct solvers, iterative solvers perform multiple computational cycles producing a sequence of iterations which gradually improve an initial estimate of the solution until a predefined stopping criterion has been reached. Usually, they are easier to implement than direct ones, because they can be reduced to a series of core components such as matrix-vector multiplications, linear-combinations of vectors, scalar-products, etc. for which highly optimized program packages can be used, e.g. BLAS [**?**]. The computational complexity associated with each single cycle is comprehensively less compared to direct methods. Thus, the decisive question is how many cycles are needed for convergence.

Because iterative methods do not produce any fill-in, they preserve the sparsity structure of the system matrix and are much less demanding with respect to storage than direct methods. However, iterative methods may depend on special properties of the underlying problem such as symmetry or positive-definiteness. Convergence can be very slow for ill-conditioned problems such that many iterations must be performed to reach the specified tolerance. Furthermore, they often require the optimal choice of certain algorithmic parameters which are highly problem-dependent and mostly difficult to predict a-priori.

4.1 The Basic Iteration

Based on Equation (3) the following simple relation applies $x = A^{-1}b = x + A^{-1}(b - Ax)$. If the inverse A^{-1} of the original system matrix A is replaced by a suitable approximation matrix B with $B \approx A^{-1}$ the so-called *preconditioned Richardson method*, or shorter, *basic iteration* for the solution of Equation (3) can be derived

$$x^k = x^{k-1} + B\,(b - Ax^{k-1})\,. \tag{7}$$

$x^k, x^{k-1} \in \mathbb{R}^n$ are iteration vectors and $B \in \mathbb{R}^{n\times n}$ is a *preconditioning matrix* which can explicitly depend on an additional relaxation parameter $\omega \in \mathbb{R}$, that is, $B = B(\omega)$.

Equation (7) is the typical core component of many iterative methods for the solution of Equation (3). Starting with a initial guess x^0, it is used to successively minimize the *residual* or *defect* $d^{k-1} := b - Ax^{k-1}$ which gives it also the name *defect correction method*. Measured in a suitable norm the defect indicates how good the current iterate x^k already fulfills equation $Ax = b$. For the error $x - x^k$ in the k-th iteration step, there holds

$$x - x^k = (I - BA)^k (x - x^0)$$

with the *error propagation operator* $F = (I - BA)$. The sequence of iterations x^k converges to the solution of $Ax = b$ if and only if its *spectral radius*, the maximum of the absolute values of the eigenvalue, is smaller than 1.

The convergence rate of iterative methods usually depends on the grid resolution, but can be comprehensively improved by appropriate selection of the preconditioning matrix B. The goal behind is to transform the original system $Ax = b$ into an equivalent system $BAx = Bb$ whose solution requires significantly less iterations than that of the original system to satisfy a predefined termination criterion. Or in other words, the transformed system $BAx = Bb$ should have a much

better eigenvalue distribution than $Ax = b$ such that its condition number is substantially smaller and convergence can be reached much faster with less computational costs.

To achieve this goal, B unfortunately has to meet two very contradictory conditions: On the one hand, B should be a good approximative inverse of A, i.e. $B \approx A^{-1}$, such that BA is close to the identity matrix (with corresponding small condition number), but this means that B is still very complex and expensive to use. On the other hand, the application of B should be much cheaper than the application of A^{-1}, i.e. $B \sim I$, but that does not lead to an improvement at all. The more special properties of the original system A can be incorporated into B, the better the convergence typically is (i.e. the norm of the error propagation operator tends towards zero), but the higher the computational costs are, too. Again, a careful compromise has to be found.

4.1.1 Preconditioning based on additive matrix splittings

For the definition of different preconditioners an additive splitting of A into its diagonal part D, its lower triangular part L and its upper triangular part U can be used, i.e. $A = L + D + U$. Note, that for symmetric problems as the one considered here it holds $U = L$ which saves the additional storage of the upper matrix U.

Now, the simplest preconditioning is defined by using only the diagonal matrix, $B_{JAC} = D^{-1}$ (*Jacobi preconditioner*), which has the advantage that it can easily be inverted. But as expected this does not lead to any significant improvement of convergence speed. Involving more information of A by also using its lower triangular part gives in a first step $B_{GS} = (L+D)^{-1}$ (*Gauss-Seidel preconditioner*) which is already much better but also more expensive to apply. Further improvements can be achieved by introducing an additional (optimal) relaxation parameter ω which leads to $B_{SOR} = (\omega L + D)^{-1}$ (*Successive Overrelaxation preconditioner*). For symmetric problems symmetrized versions of the preconditioners should be used accordingly. The preconditioner B_{SSOR} (*Symmetric Successive Overrelaxation preconditioner*) combines two SOR passes, namely a forward pass followed by a backward pass traversing the cells in reverse order, such that the resulting preconditioning matrix is similar to a symmetric matrix.

The dependence on the relaxation parameter ω can be very sensitive. An incorrectly chosen ω may quickly lead to divergence. Unfortunately, its optimal value often depends on the refinement parameters and is a-priori difficult to determine.

Furthermore, the upper variants are mainly based on a very local way of proceeding: The iteration value in one single grid cell is computed as more or less simple mean value of its directly adjacent cells. New information can only be passed cell-by-cell through the whole domain which makes them unsuitable for problems with fast global data transport. Thus, they are rather bad solvers and in their pure form not suitable for complex problems. However, they have a decisive advantage which will give them further significance in the context of multigrid methods as will be explained below.

4.1.2 Preconditioning based on multiplicative matrix splittings

A more elaborate way of preconditioning is based on different multiplicative splittings of A. It was already described in Section 3.2 that the matrix can be decomposed into the product of a lower and upper triangular matrix $A = LU$, see again Fig. 10. Note, that in this case L and U are different from those in the additive decompositions above. If this factorization is used for

preconditioning in case of a single-mesh application, $B_{LU} = LU$ (*LU-preconditioner*), the basic iteration will terminate within exactly one iteration because then $B = A$. If the system only needs to be solved once, nothing is gained from this procedure because the costs will remain the same as for the original system itself, or in other words, the system could have been solved by *LU*-decomposition from the beginning. But if the system has to be solved multiple times as is the case for FDS in the course of hundreds or thousands of time steps, it may be very advantageous to accept the costs for the factorization in an initial phase because in every subsequent time step the corresponding solution can be found by simple for- and backward substitution based on this decomposition which is computationally much cheaper. Note, that the multi-mesh case differs from this and will be treated separately again later.

However, the *LU*-decomposition is rather memory-intensive because it suffers from the undesired fill-in as already illustrated in Fig. 13 above. For realistic 3D-problems this effect may be very pronounced. To remedy this situation, slimmed-down versions for the triangular matrices, $\tilde{L}$ and $\tilde{U}$, can be used alternatively. They are based on omitting entries smaller than a given tolerance or using the same (or a similar) pattern of non-zero elements as the matrix A itself such that only the approximate relation $\tilde{L}\tilde{U} \approx A$ holds true. Thus, the resulting preconditioner, $B_{ILU} = \tilde{L}\tilde{U}$ (*ILU-preconditioner*), may be regarded as an incomplete, but computationally much cheaper variant of its complete counterpart. Unfortunately, *ILU*-preconditioning is not always stable. But it has proven to be rather efficient in many fields of application if the most basic information of A can be incorporated. Its application as preconditioner in FDS will be tested in medium term as well.

Finally, the mesh-wise preconditioning can also be performed by means of local FFT methods, provided that a structured discretization is used. As the later test calculations will show, among the variants considered here so far, this turns out to be the fastest type of mesh-wise preconditioning for structured applications of the basic iteration and its later generalizations.

4.1.3 Preconditioning based on domain decomposition

In the course of a domain decomposition method the computational domain Ω is subdivided into single subdomains Ω_i, i.e. $\Omega = \cup_{i=1,\ldots,M} \Omega_i$. Now, the individual operations of the basic iteration must be performed in a distributed manner. To this end, let K be the set of all cells in Ω and $K_i \subset K$ the subset of cells belonging to subdomain Ω_i, such that $K = \cup_{i=1,\ldots,M} K_i$. The number of cells in Ω and Ω_i are defined by $n := |K|$ and $n_i := |K_i|$, respectively.

On every subdomain Ω_i the local coefficient matrix $A_i \in \mathbb{R}^{n_i \times n_i}$ is then defined as a restriction of the global matrix $A \in \mathbb{R}^{n \times n}$ to the set K_i. Its formal algebraic definition requires a *prolongation matrix* $R_i^T \in \mathbb{R}^{n \times n_i}$ which positions a local vector $x_i \in \mathbb{R}^{n_i}$ from subdomain Ω_i to the corresponding positions of the global vector $x \in \mathbb{R}^n$ in the global domain Ω,

$$(R_i^T x_i)_j := \begin{cases} (x_i)_j, & \text{if } j \in K_i, \\ 0 \quad\;\; , & \text{if } j \in K - K_i. \end{cases}$$

Vice versa, the transposed *restriction matrix* $R_i \in R^{n_i \times n}$ restricts a global vector $x \in R^n$ to its local counterpart $x_i \in R^{n_i}$ by only chosing the corresponding entries from K_i. Thus, the local matrices are defined by

$$A_i := R_i A R_i^T, \qquad i = 1, \ldots, M, \tag{8}$$

and respresent $n_i \times n_i$-sized sub-blocks of A. Special properties of A (as e.g. diagonal dominance and positive definitness) are preserved. Note, that the matrices R_i und R_i^T are never explicitly built but are only defined by their action.

Now, most of the core components of the basic iteration (7) can simply be computed in a data-parallel way. Matrix-vector multiplications Ax, as needed for the defect computation, are computed in two steps: First, only the sub-mesh-related local matrix-vector products y_i are computed simultaneously. Secondly, these local contributions are combined to a global matrix-vector product y using the upper prolongation operators,

$$y = Ax = \widetilde{\sum}_{i=1,\ldots,M} R_i^T y_i = \widetilde{\sum}_{i=1,\ldots,M} R_i^T A_i x_i \,. \tag{9}$$

Here, the corresponding entries at inner boundary cells require a special treatment: As illustrated in Fig. 14, a matrix stencil that is positioned at an inner boundary cell has one or more 'legs' that reach out of the subdomain into its direct neighbors, but the related values of x which are needed for the sum in Equation (9) are initially missing there. To guarantee that the same final matrix-vector product is achieved as for a corresponding serial execution, these values must be exchanged via next-neighbor communication which is indicated by the marked sigma sign in Equation (9).

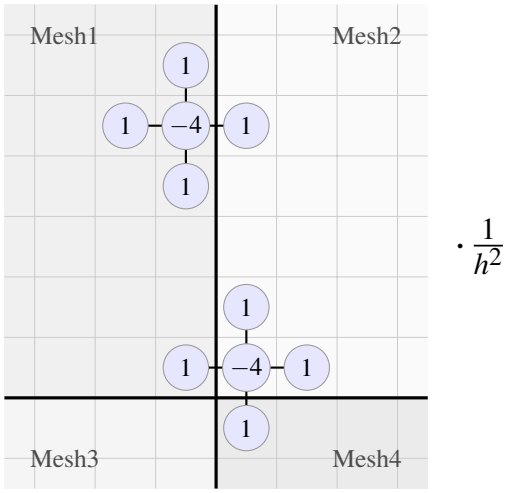

Figure 14: The computation of global matrix-vector products requires next-neighbor communications along mesh interfaces since in an internal boundary cell single legs of the corresponding matrix stencil may reach into a neighboring mesh.

Since the spatial discretization for the pressure is based on cell midpoints, linear combinations, $\alpha x + \beta y$, of vectors x and y do not require any communication at all. Every subdomain simply holds its associated part of the global vector, but along mesh interfaces no redundancies can occur. Thus, fundamental parts of the basic iteration can be done simultaneously and continue to operate globally in the same way as a hypothetic serial computation.

The situation is more complicated for the various preconditioners. With a view to the underlying domain decomposition, a natural strategy is to break up the preconditioning according to the domain decomposition. Now, each subdomain applies its own preconditioning based on approximations for the local matrix inverses A_i^{-1}.

For example, the block-wise analogue of the Jacobi preconditioner looks like

$$x^k = x^{k-1} + \sum_{i=1}^{M} R_i^T D_i^{-1} R_i \left(b - A x^{k-1}\right),$$

where each subdomain only uses the local diagonal matrix D_i for preconditioning. The solution of the local preconditioning problems only incorporates purely local computations which can be performed simultaneously with maximum possible exploitation of the local processor power. Their final global coupling only requires local next-neighbor communication. Thus, the resulting *block-preconditioning* yields significant improvements in the parallel efficiency.

However, as already explained, this strategy is associated with a breakup of the strongly recursive dependencies of optimal serial preconditioners such that numerical efficiency may strongly deteriorate in terms of poorer convergence rates compared to the serial analogues and also dependencies on the number of subdomains and/or other refinement parameters.

Besides, the efficiency of the block-preconditioning can be difficult to predict for realistic geometries with complex combinations of the single subdomains. More details about preconditioning based on domain decomposition techniques will be given in the subsequent Section (4.3).

4.2 Efficient Generalizations of the Basic Iteration

Even for powerful preconditioners the basic iteration (7) is rather inefficient and requires a comprehensible number of iterations to drive the solution within a predefined tolerance. This especially holds true for complex situations with complicated geometric details.

But the single representatives can be embedded into much stronger generalized schemes such as the global *preconditioned conjugate gradient method* (CG) or a *geometric multigrid method* (MG). Both schemes are closely related, because they are based on simple defect correction iterations, which approximate the error by using a sequence of smaller subproblems. The main difference between both is found in the choice of the underlying subspaces. A brief overview of both classes is given below.

4.2.1 Preconditioned Conjugate Gradient Method

CG-methods belong to the class of *Krylov subspace* methods which iteratively construct a k-dimensional vector space, $\text{span}\{r, Ar, ..., A^{k-1}r\}$, using A-orthogonalisation. They are designed to find the minimum of the quadratic form

$$F(x) = 1/2\, x^T A x - x^T b \to 0$$

which is equivalent to solving Equation (3). New search directions are built in such a way that they are orthogonal to all previous residuals and search directions. This process guarantees a steady improvement of the search directions and convergence to the minimum of the quadratic form after at most n steps.

Below a general algorithmic description of the CG-method for the solution of Equation (3) with preconditioning by B is given. Therein, the notation $(v, w) = v^T w$ denotes the standard scalar product of two vectors with the corresponding norm $\|v\| = \sqrt{(v, v)}$.

Conjugate gradient method for the solution of $Ax = b$ with preconditioning by B

Given an initial guess x^0, then perform the following steps:

Initialization:

r^0	$=$	$Ax^0 - b$	initial residual x^0
v^0	$=$	$B\, r^0$	preconditioning
d^0	$=$	$-v^0$	vector update

Iteration $k \geq 0$:

α_k	$=$	$(r^k, v^k)/(d^k, Ad^k)$	new search parameter
x^{k+1}	$=$	$x^k + \alpha_k\, d^k$	new solution
r^{k+1}	$=$	$r^k + \alpha_k Ad^k$	new residual
v^{k+1}	$=$	$B\, r^{k+1}$	preconditioning
β_k	$=$	$(r^{k+1}, v^{k+1})/(r^k, v^k)$	new search parameter
d^{k+1}	$=$	$-v^{k+1} + \beta_k d^k$	new search direction
$\|r^{k+1}/r^0\|$	$<$	ε	convergence check

Standard methods are restricted to symmetric positive-definite problems, but there also exist variants for the non-symmetric case, the so-called *BICG-methods* see e.g. [**?**] or [**?**]. They only need less storage space for several auxiliary vectors and are largely based on matrix-vector multiplications and global scalar-products. Using the 2D pipe geometry of Figure 2, the CG method is also graphically represented in Figure 15.

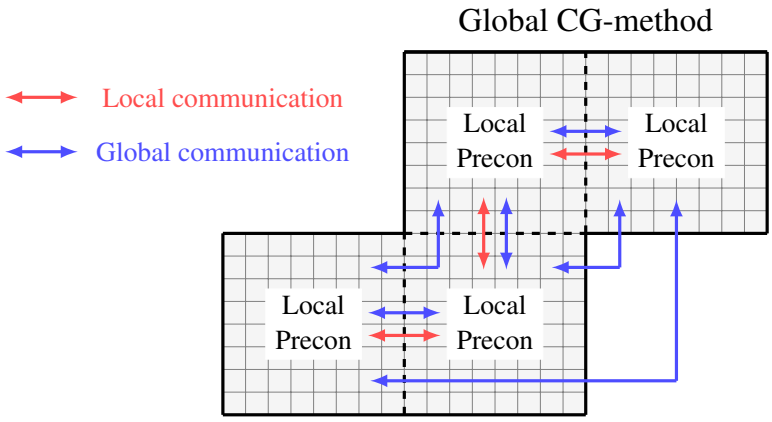

Figure 15: Data-parallel global CG-method with mesh-wise preconditioning for the 2D pipe geometry. Local communication is needed for matrix-vector products, global communication for scalar products and defect norms.

In the multi-mesh case, most operations can be performed in a purely data-parallel way: All matrix vector products, linear combinations, norms and scalar products give the same results as for a hypothetic single-mesh execution, but are only calculated in a distributed manner which is indicated by the only dashed boundaries between the single sub-meshes. For matrix-vector products this requires the (computationally cheap) local data exchanges between directly adjacent meshes as already described. For global scalar products and global norms, respectively, the local contributions must be exchanged between all meshes to finally get a global sum. Certainly, this leads to a worsening of the parallel efficiency because the processors have to synchronize and possibly wait for each other which has a negative effect especially in geometric complex cases where the meshes cannot be evenly distributed among the processors. On the other hand, these global operations also contribute to much stronger global coupling.

Typically, the convergence rate of CG-methods depends on the discretization parameters, but can be considerably improved by choosing a suitable preconditioning matrix B. For its construction, domain decomposition techniques can be used effectively, see [**?**]. There exists an upper bound for the required number of iterations which is proportional to the square root of the *condition number* $\kappa(BA) = \lambda_{max}(BA)/\lambda_{min}(BA)$,

$$\|e_k\|_A \leq 2\Big(\frac{\sqrt{\kappa(BA)}-1}{\sqrt{\kappa(BA)}+1}\Big)^k \|e_0\|_A$$

Here, λ_{min} and λ_{max} denote the smallest/biggest eigenvalue of BA, $e_k = x_k - x$ the error between the iterative solution x_k and the exact solution x, and $\|x\|_A := (Ax,x)^{1/2}$ the corresponding A-norm.

The acceleration of convergence by reducing the condition number of the preconditioned system $\kappa(BA)$ compared to that of the original system $\kappa(A)$ should at least compensate for the extra cost of computing the product Br^{k+1} in the upper algorithm, but in the best case, of course, be much larger. Geometrically, the idea behind the preconditioning in CG-methods can be understood as a transformation of the quadratic form into a more spherical shape such that the eigenvalues are closer together. While the simple diagonal Jacobi-preconditioning matrix D only scales along the coordinate axes, the perfect preconditioner $B = A$ performs a scaling along the eigenvector axes leading to the optimal possible effect $\lambda_{max} = \lambda_{min}$ with a resulting condition number of one.

The ultimate goal is to construct an optimal preconditioner with a convergence rate independent of the fine grid resolution, the size of the sub-meshes (or their number respectively) and also of possibly present anisotropies. In practise, this goal is usually hard to achieve and strongly depends on the individual situation. Nevertheless, there are many possibilities, especially on the basis of domain decomposition techniques, to achieve a more or less pronounced improvements.

4.2.2 Geometric Multigrid Methods

Fast convergence rates independent of the grid size with moderate computational complexity may be reached with MG-methods. The term 'multigrid' should not be confused with the term 'subdomain'. It does not refer to a sequence of different sub-grids arising from a domain decomposition but rather a hierarchy of grids with different resolutions for one and the same mesh. MG-methods act on a hierarchy of grids with different resolutions and are able to achieve fast convergence rates independent of the grid size with moderate computational complexity.

The basic idea behind MG-methods is to improve the convergence speed of the basic iteration by correcting the defects on successively coarser grids. This process explicitly uses an important property of the single representatives of the basic iteration, namely the so called *smoothing property*: As can be shown by a Fourier analysis, the high-frequency error components are damped out very quickly, often in only a few iterations, whereas the low-frequency error components are very persistent towards additional iterations. This special property is based on the fact that the new iteration value in a single grid point is computed as more or less simple mean value of the surrounding grid values. If this set of nodes is already smoothed out, hardly any further improvement can be achieved. Thus, new information is only propagated very slowly through the whole domain. Informally, the whole multigrid procedure looks like this:

- **Pre-Smoothing:** Starting from a given initial solution on the finest grid level, several steps of a simple basic iteration with a suitable matrix B are performed. After only a few iterations the mentioned smoothing property usually leads to a considerable reduction of the high-frequent error components of the defect while the low-frequent components are nearly unchanged and still may be very large. This suggests to restrict this *smoothed* defect on the next coarser grid (e.g. with the double grid size) by using a suitable grid transfer operator based on interpolation, where it can be approximated at much lower costs.

- **Coarse grid correction:** The alternating interplay of smoothing and restriction may be continued until the coarsest grid level has been reached where the remaining coarse grid problem can be solved exactly by some suitable method. At this stage the low-frequent components are resolved with the maximum possible global coupling, at least in a coarse sense.

- **Post-Smoothing:** The resulting coarse grid correction is then successively extended to the next finer level, whereby typically several steps of the basic iteration are performed at each level again until the finest level is finally reached.

For an algorithmic description of this procedure, let there be given a hierarchical sequence of L meshes for the domain Ω with corresponding mesh parameters $0 < h^{(L)} < \ldots < h^{(0)}$, where the finest grid is associated with index 'L' up to index '0' for the coarse grid. On every mesh level $l = 0, \ldots, L$, the corresponding Poisson matrix $A^{(l)}$, preconditioning matrix $B^{(l)}$, solution vector $x^{(l)}$ and right hand side vector $b^{(l)}$ are used. Further, let ${R^{(l)}}^T$ be a prolongation operator which interpolates a vector from level l to level $l+1$ and $R^{(l)}$ its corresponding counterpart which restricts a vector vice-versa.

In order to simplify the notation we will subsequently omit the iteration index k used in the basic iteration (7) and simply use the notation $x_i \leftarrow (\cdot)$ to express that the value on the right hand side is assigned to the left hand side within an iterative process. Thus, the basic iteration on each grid level l now simply reads as

$$x^{(l)} \leftarrow x^{(l)} + B^{(l)} \left(b^{(l)} - A^{(l)} x^{(l)} \right).$$

Based on these definitions the MG-method with smoothing by the preconditioning matrices $B^{(l)}$ reads like summarized below.

Multigrid method $x^{(l)} \leftarrow MG(l, A^{(l)}, b^{(l)}, x^{(l)}, b^{(l)})$ for $l \geq 1$

Given an initial solution, then compute an approximate solution of $A^{(l)}x^{(l)} = b^{(l)}$ by:

1. **Pre-smoothing:**

   Perform k_1 steps of a basic iteration scheme

   $$x^{(l)} \leftarrow x^{(l)} + B^{(l)}\left(b^{(l)} - A^{(l)}x^{(l)}\right).$$

2. **Coarse grid correction:**

   (a) Restrict the defect to level $l-1$

   $$b^{(l-1)} \leftarrow R^{(l-1)}(b^{(l)} - A^{(l)}x^{(l)}).$$

   (b) Solve the problem

   $$A^{(l-1)}x^{(l-1)} = b^{(l-1)}$$

   in case $l = 1$ exactly and in case $l > 1$ by p-fold recursive application of

   $$x^{(l-1)} \leftarrow MG(l-1, A^{(l-1)}, b^{(l-1)}, x^{(l-1)}, b^{(l-1)}),$$

   $p \geq 1$, with $x^{(l-1)} = 0$.

   (c) Correct the solution on level l by the prolonged coarse grid solution

   $$x^{(l)} \leftarrow x^{(l)} + {R^{(l-1)}}^T x^{(l-1)}.$$

3. **Post-smoothing:**

   Perform k_2 steps of a basic iteration scheme

   $$x^{(l)} \leftarrow x^{(l)} + B^{(l)}\left(b^{(l)} - A^{(l)}x^{(l)}\right).$$

Figure 16: Different types of multigrid cycles which prescribe the order in which and how often the single levels are run through.

Fig. 17 illustrates the whole procedure graphically for the already known 2D pipe example using simply 3 grid levels with different resolutions. Please note, that in practice typically more levels are used. Apart from the coarse grid level, a pre-defined number of basic iterations, e.g. with SSOR preconditioning, is performed on the different grid levels, which only requires local communication when calculating the global matrix vector products. The coarse grid problem is usually solved using a direct solver, which in turn requires global communication.

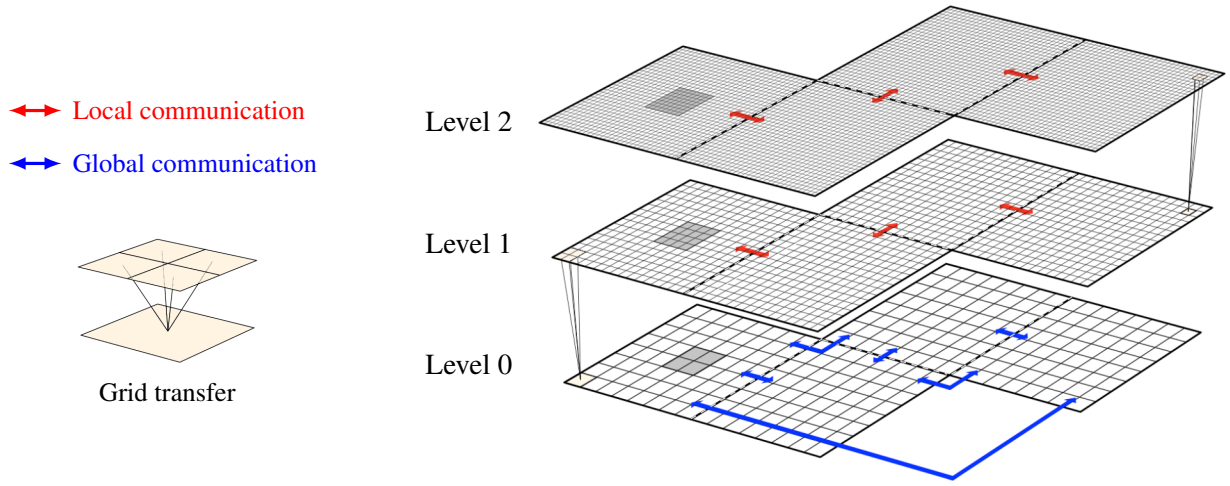

Figure 17: Geometric multigrid method with mesh-wise smoothing for the 2D pipe geometry. Local communication is needed for matrix-vector products, global communication for the solution of the coarse grid problem.

The number of smoothing steps may depend on the grid level. It is also possible to only apply the pre- or post-smoothing exclusively. The recursive change between the different grid levels can be performed in several ways, depending on how often and in which order the the individual levels are passed, see Fig. 16.

From a computational point of view the V-cycle is the most efficient type because it only needs one single coarse grid solution, but it also behaves the most sensitive to irregularities in the problem. The most robust, but also the most expensive, is the W cycle. As a good compromise between the higher computational efficiency of the V-cycle and the larger robustness of the W-cycle the F cycle has evolved. The cost of a complete cycle is only a modest multiple of the cost for a single pass on the finest grid.

All in all, each grid level is responsible for the reduction of a certain range of the error frequencies. The low-frequent components on a finer grid appear as high-frequent components on the next coarser grid. The efficiency of the complete method substantially depends on how good the ranges, which are smoothed on the single grid levels, are adjusted among each other. Besides, there is not only ONE single MG-method but a huge range of variations based on different combinations of its components (smoothers, grid transfer operators and coarse grid solvers) which should best be adapted to the underlying problem.

The upper MG-method can be used for both serial and parallel applications and the same considerations regarding the parallel executability of the single components hold true as for the CG-method before. The solution of the coarse grid problem takes a special position here: It is a very small problem which has to be solved globally such that the relation between computational and communication work is disproportionately bad. Its global computation implies a logarithmical growth of the communication costs if the number of subdomains is increased. This can be remedied by completely solving the coarse grid problem on a single processor, what in turn is associated with corresponding data transfers and waiting times on the other processors. Nevertheless, by using adequate domain decomposition strategies based on strong smoothers high numerical efficiencies can be achieved, such that usually only a few MG-iterations must be performed and the disadvantages mentioned do not matter that much.

4.2.3 Summary for generalizations

All in all it can be stated that both, CG- and MG-methods, are able to reasonably improve the convergence speed of the basic iteration returning the exact solution in a small up to a moderate number of iterations depending on the underlying problem.

Apart from using the aforementioned globally defined matrix-vector products, these methods incorporate even more globally acting features such as global scalar-products in case of CG or an additional coarse grid problem in case of MG which furthermore contribute to a stronger global coupling. Certainly, the acceleration of convergence speed achieved by these features must be set in relation to the increased computational overhead associated with them.

Due to their already mentioned restriction to different core components, iterative methods prove to be easier and more universally parallelizable than direct ones. In both methods the local solutions are not used as stand-alone solvers but embedded in an outer iteration where they only serve as corrections to the global solution. In this context, the potential of domain decomposition methods can largely be exploited.

4.3 Scalable Recursive Clustering - SCARC

With the aim to combine the advantages of the above Poisson solvers and to best possibly avoid their disadvantages the solution concept SCARC was developed. It can be understood as a combination of parallel Krylov and/or multigrid methods with techniques from multilevel domain decomposition to finally yield a robust, computationally and numerically efficient parallel solution for scalar elliptic equations.

In fact, SCARC is not just a single solver, but rather a whole package of solvers and various components that can chosen in the view of the present situation. Its design is focused on the following objectives: Most of the work required to solve the global problem should be done locally and coupled with as little communication effort as possible (high parallel efficiency). Convergence speed should be as high as possible and independent of the number of subdomains, the grid resolution and the complexity of the domain (high numerical efficiency). Further, the solver should be easily implementable and use only already existing standard methods. It should allow the application of optimized libraries and thus be able to make maximum use of the local processor power. Finally, it should guarantee the treatment of complicated geometries without lasting deterioration of the overall convergence rates.

These are certainly very ambitious objectives that are extremely difficult to achieve at the same time. The following sections are concerned with describing the underlying algorithmic core concepts, their most important generalizations and a discussion on the performance achieved so far with regard to the above requirements.

4.3.1 Core algorithm

SCARC is mainly defined as a symbiosis of global and local iterative techniques with the optional use of direct techniques locally. The domain decomposition does not serve as a stand-alone solver, but rather to define suitable sub-problems that are only used to correct the globally defined solution. In its original form SCARC only consists of the nested combination of an outer (global) basic iteration and M inner (local) basic iterations as displayed in Fig. 18.

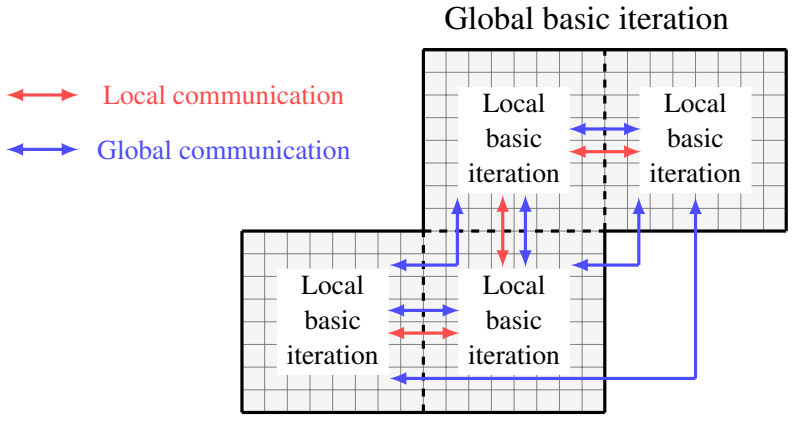

Figure 18: Core component of SCARC consisting of a nested combination of a global basic iteration with local mesh-wise basic iterations.

Basic SCARC algorithm with domain decompositon preconditioner B_S

Given an initial guess x^0, then solve the global system $Ax = b$ by the following steps:

(i) Perform a global basic iteration with a domain decomposition preconditioner B_S,

$$x^k = x^{k-1} + B_S\,(b - Ax^{k-1})\,. \tag{10}$$

(ii) In each cycle k of (i) simultaneously solve the related M local systems for the restricted defects $d_i^{k-1} := R_i\,(b - Ax^{k-1})$,

$$A_i y_i = d_i^{k-1}\,, \qquad i = 1,\ldots,M\,,$$

either by local direct solvers or by local basic iterations with preconditioners $C_i = part(A_i)$ for the local Poisson matrices A_i

$$y_i^m = y_i^{m-1} + C_i\,(d_i^{k-1} - A_i\,y_i^{m-1})\,, \qquad i = 1,\ldots,M\,. \tag{11}$$

The outer basic iteration relies on a globally acting domain decomposition preconditioner B_S for which various variants will be described below. The inner basic iterations are based on locally acting preconditioners C_i which can be chosen individually for the different subdomains Ω_i depending on the local situation. Alternatively, the local problems can also be solved directly.

As will be explained in more detail below, this algorithmic core can be generalized in various ways. A formal description of the restriction operator R_i which is used in part (ii) for the definition of the local defects can be found in Section (4.1.3). Furthermore, $C_i = part(A_i)$ means that only a partition of the local matrix A_i is used for the preconditioning as e.g. its diagonal part D_i which corresponds to a local Jacobi preconditioner. Again, different relaxation parameters ω_g and ω_l typically are used for the global and local iterations such that there actually holds $B_S = B_S(\omega_g)$ and $C_i = C_i(\omega_l)$ which will be omitted for the sake of simplicity below. A detailed description of the algorithmic concept and all related topics can be found in [**?**, **?**, **?**, **?**].

4.3.2 Multilevel Schwarz preconditioners

For the preconditioning matrix B_S numerous variants with different degree of complexity can be applied. All these variants have in common that they essentially rely on the solution of the local subdomain problems which are defined with respect to the original fine grid resolution. But they differ in the fact that, in addition to the fine grid level, an further coarse grid level can be used to ensure at least a basic transfer of global information. For the way in which the local subproblems are coupled with each other and possibly with a coarse grid problem, there are again several possibilities. In addition, the local preconditioners C_i can be chosen individually in best possible adjustment to the local conditions and the accuracy requirements for the local solutions can be varied.

To give a formal algorithmic description of the different variants, some notations related to the local fine grid preconditioners and the global coarse grid preconditioner are still introduced below. In each cycle k of the global basic iteration (10) the solution of the local basic iterations (11) makes use of the definitions below.

First, the global defect, $d = b - Ax^{k-1}$, is restricted to the single subdomains, $d_i = R_i d$, then the corresponding local Poisson problems are solved, $v_i = A_i^{-1} d_i$, finally the local results are prolonged back and summed up to a global solution, $v = \sum_{i=1}^{M} R_i^T v_i$. For each mesh the sequence of these steps can be summarized using the *local fine grid preconditioner* B_i defined as follows

$$B_i := R_i^T A_i^{-1} R_i\,, \qquad i = 1,\ldots,M\,.$$

Second, if an additional coarse grid problem is used, then also a coarse Poisson matrix A_C is required along with corresponding operators for prolongation, R_C^T, and restriction, R_C, which ensure the transfer between the fine and coarse level in a similar way. Based on the same sequence of steps as for the B_i above, the resulting *global coarse grid preconditioner* b_C is then defined as

$$b_C = R_C^T A_C^{-1} R_C\,.$$

Typically, the coarse grid consists of a finite difference discretization of the domain decomposition itself, i.e. $A_C \in \mathbf{R}^{M\times M}$, but it can also refer to a (still very) coarse refinement of it.

The favored variants from the above-mentioned multitude of possibilities are presented and discussed below.

(a) **Additive 1-level-Schwarz (additive in fine level, no coarse level)**

The simplest variant for B_S is based on the simultaneous solution of all local subdomain problems which is denoted as *additive* coupling. The *additive 1-level-Schwarz preconditioner* B_{AS1} is given by

$$B_{AS1} = \sum_{i=1}^{M} B_i\,.$$

Computationally, its application to a defect vector $B_{AS1}\,d$ is defined by

$$v \leftarrow \sum_{i=1}^{M} B_i\,d\,.$$

As the local calculations can be carried out completely in parallel this type of preconditioning corresponds to a mesh-wise Jacobi scheme on subdomain level.

(b) **Multiplicative 1-level-Schwarz (multiplicative in fine level, no coarse level)**

Alternatively, the subdomain problems can also be solved one after another which is denoted as *multiplicative* coupling. The *multiplicative 1-level-Schwarz preconditioner* B_{MS1} is formally given by

$$B_{MS1} = \left[I - \prod_{i=1}^{M}(I - B_i A)\right] A^{-1}\,.$$

Computationally, its application to a defect vector $B_{MS1}\,d$ is defined by

$$\begin{aligned} v &\leftarrow B_1\,d\,, \\ v &\leftarrow v + B_j\,(d - Av)\,, \quad j = 2\ldots M\,. \end{aligned}$$

Thus, new information obtained on mesh j is passed to mesh $j+1$ as soon as it becomes available which may contribute to a clear improvement of the overall convergence speed. This strategy basically corresponds to a Gauss-Seidel method on subdomain level, which, however, is immanently serial. So-called *red-black strategies* can be applied to increase the amount of parallelism, but this variant will be available only in the medium term. Thus, the coupling of the local sub-problems is always supposed to be of additive type for now.

(c) **Additive 2-level-Schwarz (additive in fine level, additive in coarse level)**

The two 1-level variants above are only based on the coupling of purely local solutions. But this strategy suffers from the same problem as the mesh-wise FFT: New information can only be distributed mesh-by-mesh throughout the entire domain, associated with a time delay which is proportional to the number of subdomains. By the additional use of a coarse grid problem, this dependency may be considerably mellowed and at least a rough assessment of the global information flow can be attained.

If the coarse grid problem is additively coupled to the (also additive) local fine grid problems, then the *additive 2-level-Schwarz preconditioner* is given by

$$B_{AS2} = B_C + B_{AS1}\,.$$

Computationally, its application to a defect vector $B_{AS2}\,d$ is defined by

$$v \leftarrow [B_C + B_{AS1}]\,d\,.$$

The solution of the coarse grid problem and the fine grid problems is completely independent of each other leading to the highest possible parallel efficiency which can be achieved for 2-level methods. But both levels only use the last defect information of iteration $k-1$ such that new information is spread only in the following iteration once all solutions involved have been processed.

(d) **Hybrid 2-level-Schwarz (additive in fine level, multiplicative in coarse level)**

Alternatively, the computations of only one grid level can be carried out first, while those of the other level are performed afterwards which again corresponds to a multiplicative coupling. If the (additive) solution of the fine grid problems is completely done before the solution of the coarse grid problem, the resulting *hybrid 2-level-Schwarz preconditioner* B_{HYB2} is given by

$$B_{HYB2} = B_C + B_{AS1} - B_C\,A\,B_{AS1}\,.$$

Computationally, its application to a defect vector $B_{HYB2}\,d$ is defined by

$$\begin{aligned} v &\leftarrow B_{AS1}\,d\,, \\ v &\leftarrow v + B_C\,(d - Av)\,. \end{aligned}$$

Now the latest available information from the fine grid problems can already be incorporated into a new defect calculation, which in turn is available for the next coarse grid problem. Despite the increased arithmetic and communication effort caused by the additional defect calculation, the associated gain in convergence speed is usually worth-while.

The relation of additive to multiplicative Schwarz methods is similar to that of Jacobi and Gauss-Seidel methods. It can be shown that the addition of a coarser grid level significantly reduces the dependence on the number of meshes.

4.3.3 Algorithmic and performance-oriented aspects

Treatment of mesh interfaces: Since SCARC is based on a global discretization, there is no need to impose artificial boundary conditions along inner mesh boundaries or to use an additional pressure iteration to correct the velocity field there because internal boundary cells are treated as usual internal cells with respect to the virtual global matrix which is only handled in a distributed way. With a view to FDS, this has the advantage that the final global solution has consistent values along mesh interfaces by design, i.e. the normals of the single velocity components automatically match up to machine precision.

Accuracy of local solutions: Numerical experience has shown that block-wise schemes like the ones described before often perform well as long as complex parts are locally hidden and the local problems are solved with sufficient precision. In this light the quality of the local subdomain problems plays a major role. If the iterative processes are continued until machine precision for the local defect norms has been reached, the global basic iteration should be able to compute the same final solution as a corresponding direct solver that would be applied for the whole global problem.

However, in order to save computational costs, the iterative processes are typically stopped earlier, e.g. if only one digit or a moderately small tolerance has been reached or a fixed number of iterations has been performed. According to the accuracy which is achieved locally at the end, the global basic iteration will in turn require more or less iterations to fulfil the globally desired stopping criterion.

More accurate local solutions are certainly associated with an increase in arithmetic effort, but can also reduce the number of globally required iterations and thus the overall communication effort. Since *data processing* (the local computations) is still cheaper than *data movement* (the communications), this is usually worthwhile. In this context, the local use of optimized libraries can make a significant contribution to an acceleration of the purely arithmetic components. To sum up, the interplay between global and local effort must be balanced very well in order to achieve the shortest possible computing time and the highest possible convergence speed of the whole method.

Interplay of global and local iterations: The preconditioner B_S is never built as a whole, but is rather available as a collection of local (and possibly global) components which are only implicitly represented by their action, i.e. solving the local sub-problems to a specified accuracy and exchanging shared data. The single sub-problems are in turn only thought as corrections to a global solution vector, however, their parallel solution is much cheaper than that of a single global problem. By a suitable interaction of the different preconditioning components, the inevitable losses in global coupling (and thus numerical efficiency) induced by the domain decomposition can be significantly decreased.

A major advantage of the underlying methodology is that the local preconditioners can be defined completely independently of each other. The additive 1-level-Schwarz preconditioner doesn't care how the local problems were solved, it just couples them together. If particularly complex conditions exist in a certain subdomain, an optimized local multigrid method with a powerful recursive smoother could be used there, while in another subdomain only one SSOR-cycle might be enough to achieve the same local accuracy. This certainly leads to the fact that the arithmetic load on the individual subdomains can be very different depending on the local levels of difficulty such that a proper balance between numerical and computational efficiency must always be kept in mind.

Computational and communication effort: The overall communication effort of the different variants strongly depends on the choice of the global preconditioner B_S. As far as there is no coarse grid problem involved, the parallel efficiency is rather high because the preconditioning only needs local communication for the matrix-vector products in the defect computations. Apart from the coarse grid problem, which will be discussed immediately below, global communication is only needed for the computation of global defect norms to check the termination criterion of the global basic iteration.

The coarse grid solution can be achieved by any optimized direct solver which is typically performed on a selected 'master' processor. This requires the communication of all related data between the subdomain processors to the master including synchronization overhead and possible waiting times (especially in case of unequal load distribution) which certainly leads to a considerable reduction of parallel efficiency. Alternatively, the coarse grid problem can be solved iteratively in a distributed fashion. However, this is associated with frequent exchanges of very small data associated with a correspondingly poor ratio of computational to communication overhead.

If the hybrid 2-level-Schwarz preconditioner is used, the need to compute an additional defect (including next-neighbor communication) certainly increases the arithmetic costs per cycle of the SCARC basic iteration. On the other hand, the 2-level variants are also associated with a considerable gain in numerical efficiency since information is used directly when it is available such that less global iterations are needed. Typically, the achieved acceleration of convergence speed makes up for the additional effort.

For decompositions with many subdomains, or in case that the coarse grid consists of a slightly refined version of the underlying domain decomposition, the coarse grid problem itself may be so large that it is no longer suitable for a direct solution. In this case the above 2-level concept can be applied recursively by solving the coarse grid problem again by a 2-level Schwarz method which leads to a full *multilevel Schwarz method*.

Adjusting algorithmic parameters: A typical feature of iterative methods is that various parameters must be optimally adjusted which turns out to be a disadvantage of this concept. In particular, this applies to the global and local relaxation parameters which have a large influence on the convergence speed of the whole method up to divergence in the worst case. Their optimal choice depends on the problem characteristics and may be difficult to determine a-priori. However, appropriate sensitivity studies have already been capable of identifying related ranges which generally provide reasonable results.

Possible enhancements for multi-core architectures: If, as is possible in FDS, several meshes are mapped to one processor, the associated local mesh calculations are performed one after the other there. But instead of additively coupling these calculations as usual, a multiplicative coupling could now be considered within the scope of the processor, which might result in a further increase of the numerical efficiency in case of a clever clustering of the meshes. Furthermore, for the related meshes one common (unstructured) Poisson matrix could be assembled on the processor instead of the usual collection of locally structured matrices with corresponding overall solution of the locally clustered system. This strategy is only rudimentarily programmed so far and will be available in the medium term.

4.3.4 Generalizations of the core algorithm

As already discussed in section (4.1) the convergence properties of the pure basic iteration are not satisfactory for complex situations. Thus, the efficiency of the global and local basic iterations can be significantly improved by incorporating Krylov and/or multigrid techniques on the global and/or the local layer as summarized in section (4.2).

Generalizations of the global basic iteration: In the default version of SCARC currently available in FDS the outer basic iteration is simply replaced by a data-parallel global CG-method. Another powerful generalization is to use a data-parallel global MG-method with mesh-wise smoothing instead. For the preconditioning/smoothing of both the global CG- and MG-methods simply some steps of the local basic iterations can be performed, e.g. based on the SSOR-preconditioner leading to either a global CG-method with mesh-wise SSOR preconditioning or a global MG-method with mesh-wise SSOR smoothing. As expected, the global CG-variants which only use a 1-level-Schwarz preconditioner have a much better parallel efficiency than the global MG-variants because only a fine grid level with high computational complexity is used. But it should not be forgotten that the numerical efficiency of the CG-method may deteriorate significantly when the number of subdomains is increased which again is based on the lack of global data transfer for 1-level-Schwarz preconditioners. The use of a suitable 2-level-Schwarz preconditioner is able to remedy the situation, but associated with increased computational costs again.

In contrast, the global MG-variants show by far the better convergence behavior. This is mainly based on the stronger global coupling which is achieved by the interlocking of levels with different resolutions and the addition of a coarse grid problem. Due to the restriction to local tensor product meshes, very sophisticated smoothers can be used which are capable of fully exploiting modern processor capabilities. On the other hand, the coarse grid problem represents the already described bottleneck in terms of parallel efficiency. Furthermore, with every change to the next coarser grid level, the amount of local computational work decreases. Thus, during the changes to increasingly coarser grid levels, the related matrix vector products are associated with a decreasing ratio of local arithmetic work to communication work. These losses of parallel efficiency have to be taken into account when evaluating the overall efficiency and must clearly be compensated by the convergence acceleration induced by the use of different levels and the coarse grid solution.

A very special variant of SCARC is the combination of a global MG-method with optimized local MG-methods of Schwarz type. Moreover, a further increase of efficiency may be achieved if the global MG-method is not used as standalone solver but in turn only as preconditioner inside a global CG-method, which provides an additional global coupling especially in case of big anisotropies on coarse grid level. Again, the gain of numerical efficiency induced by these sophisticated combinations of different solution techniques must be set in relation to the associated increased effort.

Generalizations of the local basic iterations: There are numerous possibilities to also replace the local basic iterations with more efficient strategies. As those computations are performed completely locally, an obvious idea is to use the already described power of serial direct methods there. Thus, a favorite approach is to apply local FFT-methods or LU-decompositions. Alternatively, the local basic iterations can be replaced by local MG-methods with specially optimized local smoothers which are best possibly adapted to the underlying problem.

Important representatives and conclusion: In the light of the above discussions, the acronym SCARC can be explained now. It stands for:

- *Scalable*, with respect to the number of global ('k') and local solution steps ('m'),
- *Recursive*, since it can be applied recursively to more than 2 levels with different refinements,
- *Clustering*, since neighboring meshes can be merged into larger clusters with special treatment.

All in all, SCARC represents a large class of techniques, which contains a wide spectrum of the known Krylov, multigrid and domain decomposition approaches for the solution of discretized PDE's. The most important representatives are:

1. set $k = 1$ (globally), solve exactly (locally)
   $\longrightarrow$ Parallel CG-method with Additive 1-level-Schwarz preconditioning
2. set $m = 1$ (locally) and $C_i = part(A_i)$
   $\longrightarrow$ Standard multigrid with mesh-wise smoothing
3. set $m > 1$ (locally) and $C_i = part(A_i)$ and solve approximately (locally) via MG
   $\longrightarrow$ full SCARC

The first two variants are used in the course of the subsequent test calculations, see the definitions in Section 2.1.2. The last variant is still under development, but will also be available shortly.

It becomes clear that SCARC is not a pure black-box solver and its application requires a careful assessment of the current circumstances. However, its major advantage is that it allows a very diverse combination of different components with regard to the selection of the global solvers (CG, MG or combinations of both), the local solvers (from simple Jacobi to highly optimized MG) and the local accuracy requirements (from gaining only one digit up to machine precision) such that any available information about the underlying problem can be exploited.

4.3.5 Discretization types and application notes

The SCARC functionality described in this documentation is not yet fully available in the latest precompiled FDS release. To access the most recent version of SCARC, it is necessary to download a clone of the current FDS repository and compile the FDS code yourself. It must be emphasized that SCARC is still in a beta development state and there are still certain restrictions on its application. These actually still correspond to those for the UGLMAT solver, namely that SCARC can only be applied for non-overlapping, non-stretched meshes at the same refinement level. While the use of an overlapping decomposition isn't necessary by design, the other two restrictions are only temporary in nature. Corresponding enhancements are basically possible and will be on the agenda in the immediate future.

In the recent past, many different cases from the verification and validation directory of FDS have been recalculated with SCARC. Thereby, it has been proven repeatedly that the results obtained are algorithmically correct and that high accuracy can be achieved with completely correct transitions of the velocity field along inner mesh boundaries.

However, particularly in case of longer lasting validation cases, it has also been observed that the previous runtimes are not yet optimal. This remaining weakness is mainly based on the fact that the focus of the development so far has been on the fundamental algorithmic correctness of the individual solution components. Nevertheless, there is still a great potential for runtime improvements and appropriate improvements will be done in the immediate future as well.

So far, there has been no discussion about what kind of discretization SCARC is working on. Basically, SCARC can be applied for structured and unstructured discretizations. Some application notes must be considered with respect to the globally and locally used components. Currently, the two following default versions for the structured and unstructured case are available:

Global structured discretization: The default variant for the structured case, simply abbreviated with SCARC, is illustrated in Figure 19. It can be called by setting `SOLVER='SCARC'` in the `&PRES` name list and is based on a global, data-parallel CG method. The local Poisson matrices A_i are assembled in a structured way, i.e. using the same matrix stencil everywhere and incorporating both gas-phase and solid cells. Based on this structured type, the mesh-wise preconditioning is done by optimized local FFT from the CRAYFISHPAK package.

Since there still may be velocity penetration errors towards internal obstructions, this variant is embedded into the already known pressure iteration as explained in Section 3.1. However, in contrast to the default FFT solver the velocity field is already correct along mesh interfaces and the pressure iteration only relates to internal obstructions.

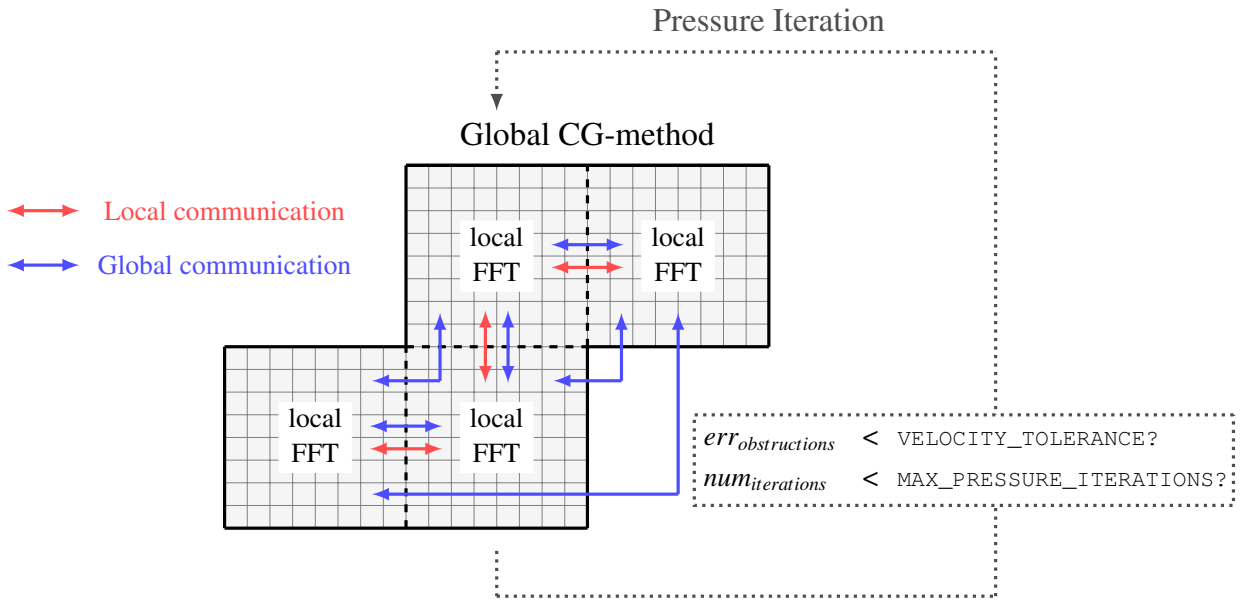

Figure 19: The default SCARC version is defined on a global structured discretization, uses a global data-parallel CG-method with mesh-wise preconditioning by local optimized FFT-methods and is embedded in a pressure iteration to correct velocity penetration at internal obstructions.

Global unstructured discretization: The default variant for the unstructured case, simply abbreviated with USCARC, is illustrated in Figure 20. It can be called by setting `SOLVER='USCARC'` in the `&PRES` name list and is also based on a global data-parallel CG method. The local Poisson matrices are assembled in an unstructured way, i.e. using individual matrix stencils and incorporating only gas-phase cells while setting no-flux boundary conditions along internal solids. Based on this unstructured type, however, the mesh-wise preconditioning can no longer be done by local FFT's, but local *LU*-decompositions are used instead based on the `PARDISO` solver of the Intel® Math Kernel Library. Apart from the already correct transitions along inner mesh boundaries, the treatment of inner obstructions is also correct and no pressure iteration is needed anymore.

As the following test calculations will show, the local *LU*-decompositions are not as performant as the local FFT methods. For test cases whose structured execution by SCARC only requires a small number of pressure iterations, it can therefore happen that the corresponding more accurate USCARC version, although it only requires one pressure iteration, is still slower. Strategies to improve this situation are currently being developed.

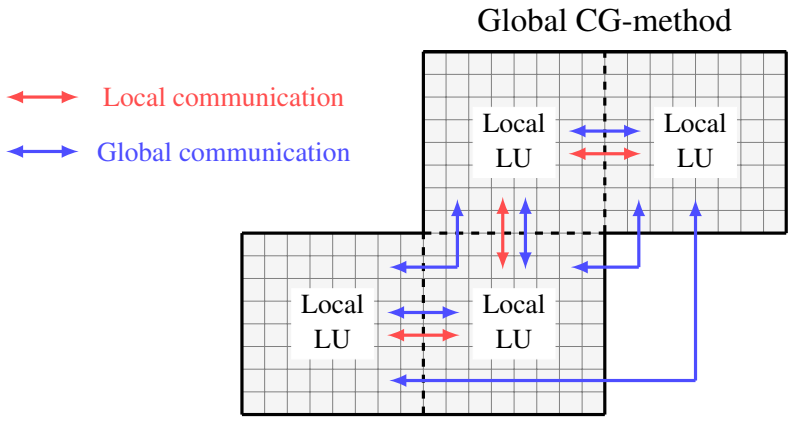

Figure 20: The default USCARC version is defined on a global unstructured discretization and consists of a global data-parallel CG-method with mesh-wise preconditioning by local optimized LU-methods.

In general, MG methods must ensure that the number of grid cells in the various spatial dimensions can be divided by 2 at least once which may be an undesired restriction in geometric flexibility. The use of so-called *algebraic multigrid methods*, which work on arbitrary numbers of cells, is already well advanced, but will only be available in the medium term. At the moment, the MG variant should only be used for structured discretizations (along with the already mentioned divisibility condition). In the unstructured case it may otherwise happen that during the coarsening process inner obstructions can no longer be resolved with the coarser grid width. An MG variant that basically uses structured grids on coarser levels, even if the fine grid level is unstructured, is in progress. Since the coarser levels only serve to correct the fine global problem, a combination of different discretization techniques is possible. The same considerations currently holds true for the usage of the 2-level-Schwarz preconditioners as they also require a divisibility of the grid size at least by 2.

4.3.6 Description of parameters

The following listing of SCARC related parameters in the &PRES namelist is still incomplete. At this stage, it is not yet clear whether most of the SCARC parameters should remain hidden and only some basic parameters should be provided for user-defined variation. Once this is determined, a detailed explanation of the remaining parameters will follow.

Table 1: For more information see Section (4.2.1).

PRES (SCARC Parameters)				
SCARC_ACCURACY	Character	Section 4.3		'ABSOLUTE'
SCARC_COARSE	Character	Section 4.2.2		'ITERATIVE'
SCARC_COARSE_ACCURACY	Real	Section 4.2.2		10^{-14}
SCARC_COARSE_ITERATIONS	Integer	Section 4.2.2		100
SCARC_COARSE_LEVEL	Integer	Section 4.2.2		1
SCARC_COARSE_OMEGA	Real	Section 4.2.2		0.8
SCARC_CSV	Character	Section 4.3		'NONE'
SCARC_DEBUG	Character	Section 4.3		'NONE'
SCARC_DISCRETIZATION	Character	Section 2		'STRUCTURED'
SCARC_KRYLOV	Character	Section 4.2.1		'CG'
SCARC_KRYLOV_ACCURACY	Character	Section 4.2.1		10^{-10}
SCARC_KRYLOV_INTERPOL	Character	Section 4.2.1		'CONSTANT'
SCARC_KRYLOV_ITERATIONS	Integer	Section 4.2.1		500
SCARC_METHOD	Character	Section 4.3		'NONE'
SCARC_MULTIGRID	Character	Section 4.2.2		'GEOMETRIC'
SCARC_MULTIGRID_ACCURACY	Real	Section 4.2.2		10^{-10}
SCARC_MULTIGRID_COARSENING	Character	Section 4.2.2		'FALGOUT'
SCARC_MULTIGRID_CYCLE	Character	Section 4.2.2		'V'
SCARC_MULTIGRID_INTERPOL	Character	Section 4.2.2		'CONSTANT'
SCARC_MULTIGRID_ITERATIONS	Integer	Section 4.2.2		100
SCARC_MULTIGRID_LEVEL	Integer	Section 4.3		-1
SCARC_PRECISION	Character	Section 4.2.1		'DOUBLE'
SCARC_PRECON	Character	Section 4.2.1		'NONE'
SCARC_PRECON_ACCURACY	Real	Section 4.2.1		10^{-10}
SCARC_PRECON_ITERATIONS	Integer	Section 4.2.1		100
SCARC_PRECON_OMEGA	Real	Section 4.2.2		0.8
SCARC_SMOOTH	Character	Section 4.2.2		'SSOR'
SCARC_SMOOTH_ACCURACY	Real	Section 4.2.2		10^{-8}
SCARC_SMOOTH_ITERATIONS	Integer	Section 4.2.2		5
SCARC_SMOOTH_OMEGA	Real	Section 4.2.2		0.8
SCARC_TWOLEVEL	Character	Section 4.3		'STRUCTURED'
SCARC_VERBOSE	Character	Section 4.3		'NONE'

5 SCARC verification tests

In the following, SCARC is subjected to a series of test calculations to demonstrate its suitability as an alternative pressure solver in FDS. To check whether SCARC fundamentally delivers correct results, the first step focuses on the analysis of its approximation quality with special view on the handling of multi-mesh decompositions and internal obstructions. For this purpose, a new SCARC specific test case `poisson2d` will be presented, which analyzes its various features and variants. Additionally, SCARC is applied to a number of existing verification test cases from FDS. In order to evaluate the results in a meaningful way, all SCARC computations are compared with the respective computations of the other available pressure solvers based on FFT or global *LU*-decomposition. Subsequently, both structured and unstructured discretizations are taken into account and the respective solvers including their short names are summarized below.

5.1 Summary of solvers applied

Structured discretizations: In case of a structured discretization, different variants of FFT and structured SCARC will be considered. Due to their structured nature, FFT and SCARC are not able to incorporate the right boundary conditions along internal obstructions as explained in Section (2.1.2). Thus, both are embedded in the corrective pressure iteration which applies them several times until the predefined velocity tolerance has been fulfilled, see again Figure 9 and Figure 19. In case of FFT the pressure iteration additionally serves to correct the velocity errors along mesh interfaces. In contrast, SCARC doesn't need any further correction there because the inter-mesh transitions of the flow field are consistent by design.

To analyze the effectiveness of the pressure iteration for a given test case, both FFT and SCARC take into account the default and a finer value for the velocity tolerance. Since FFT is basically the default pressure solver in FDS, it does not have to be specified separately. For SCARC the extra specification `SOLVER='SCARC'` must be used in the `&PRES` name list, see again Section (4.3.5).

- The variants with default velocity tolerance are denoted by FFT-default and SCARC-default. Apart from the basic solver selection, nothing more needs to be specified. The default value for `VELOCITY_TOLERANCE` is automatically set corresponding to the respective grid width and a default limit of 10 is used for `MAX_PRESSURE_ITERATIONS`.

- The variants with finer velocity tolerance are denoted by FFT-tight and SCARC-tight. The desired value of `VELOCITY_TOLERANCE` must additionally be specified in the `&PRES` namelist which is done individually for the different test cases. For both variants the increased setting of `MAX_PRESSURE_ITERATIONS=1000` is basically used.

Both SCARC-default and SCARC-tight are based on a data-parallel global CG-method using an 1-level-Schwarz-preconditioning by mesh-wise FFT-methods. In some of the subsequent cases two further generalized variants for SCARC will be used. The first variant, SCARC-twolevel, is also CG-based, but uses a 2-level-Schwarz preconditioning incorporating an additional coarse grid. The second one, SCARC-multigrid, uses a global MG-method instead of the global CG-method and is based on local SSOR-smoothers. Both will only be applied in their default variant, i.e. with the related default velocity tolerance.

Unstructured discretizations:

In case of an unstructured discretization, FFT can no longer be applied. Instead, the alternative pressure solver UGLMAT is used, which is based on the computation of a global *LU*-decomposition using the optimized parallel `Cluster_Sparse_Solver` of the Intel® MKL library as described in Section (3.2). Furthermore, USCARC as the unstructured counterpart of SCARC will be applied, see Figure 20 again. To call these solvers the additional settings `SOLVER='UGLMAT'` or `SOLVER='USCARC'` must be used in the `&PRES` namelist, respectively.

Both UGLMAT and USCARC are able to correctly treat mesh interfaces and internal obstructions without additional pressure iteration. Thus, no specification of a velocity tolerance is required and machine accuracy is achieved by design. Again, USCARC relies on a global CG-method by default. However, due to the unstructured nature of the single sub-grids, the preconditioning can no longer be done by local FFTs. Instead, local *LU*-decompositions are used by default which are based on the optimized serial `PARDISO` solver of the Intel® MKL Library.

As mentioned earlier, the local *LU* decompositions prove to be slower than the local FFTs. Thus, it may happen that USCARC (which exactly needs one pressure iteration) may be nevertheless slower than a corresponding SCARC (if it succeeds to rapidly reach its velocity tolerance in only a few pressure iterations). Optimized strategies to remedy this situation are in work.

Now, the crucial question is how well the different structured and unstructured variants are able to preserve the accuracy of the corresponding exact solutions and how good they can be scaled to larger numbers of sub-meshes. To provide a better overview, the solvers including their discretization types and their short names are summarized again in Table 2.

Discretization	Solver class	Name	Pressure Iteration	Velocity tolerance
Structured	FFT	FFT-tight	IB + MI	fine
		FFT-default	IB + MI	default
	SCARC	SCARC-tight	IB	fine
		SCARC-default	IB	default
		SCARC-twolevel	IB	default
		SCARC-multigrid	IB	default
Unstructured	SCARC	USCARC	-	-
	LU-decomp	UGLMAT	-	-

Table 2: Overview of the different pressure solvers for structured and unstructured discretizations which are used in the subsequent test computations. The structured variants are imbedded in a surrounding pressure iteration which is related to the internal boundaries (IB) and possibly to mesh interfaces (MI) as well. Two different velocity tolerances will be considered for all structured variants.

5.2 Basic Poisson test case

To algorithmically describe the different Poisson solvers available in FDS, a simple demonstration case was used throughout the entire documentation which will be denoted as `poisson2d` case subsequently. As illustrated again in Figure 21, it consists of a small angled pipe in 2D with a small inflow on the left and an open outflow on the whole right hand side. The length of the left and right face amounts to 40 cm each, the side length of the small internal obstruction to 10 cm. Apart from the descriptive purposes this case has mainly been designed to analyze the capabilities of the different SCARC variants to deal with internal obstructions and multi-mesh decompositions.

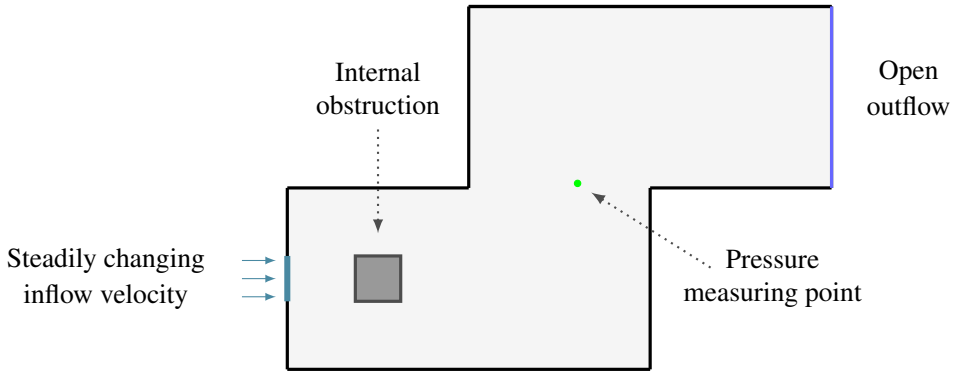

Figure 21: `poisson2d` test case consisting of a 2D-pipe geometry with a small obstruction and steadily changing inflow conditions.

As already explained in detail in Section (2), parallel elliptic solvers usually have trouble to map global information transfer correctly. Just this effect shall be examined here in terms of accuracy and convergence speed. To this end, air is blown into the domain whereby the inflow velocity is continuously varied within a range of 0 to 2 m/s such that the flow pattern in the entire domain frequently changes. As will be seen in the later plots this RAMP-based setting will cause a very characteristic course for the pressure trace which is measured in the green indicated device in the middle of the domain up to the final simulation time of 0.5 s. The need to continuously adapt to varying global situations poses a particular challenge to the pressure solver and is intentionally used to analyze the related scalability of the different variants with respect to an increasing number of meshes. To this end, two different variants for this case will be analyzed subsequently which differ in the number of used meshes and the underlying grid resolutions.

5.2.1 Subdivision into 4 meshes (`poisson2d_4mesh`)

In this case the pipe geometry from Figure 21 is subdivided into 4 meshes with a side length of 40 cm each. The single meshes are refined into 16^2 cells corresponding to a grid resolution of 2.5 cm. The resulting flow field at time $t = 0.31$ is illustrated in Figure 22. For the grid resolution considered here, the structured FFT-default and SCARC-default are based on a default velocity tolerance 0.0125 m/s, while FFT-tight and SCARC-tight apply the finer tolerance 0.00001 m/s. No velocity tolerance is needed for both unstructured variants UGLMAT and USCARC.

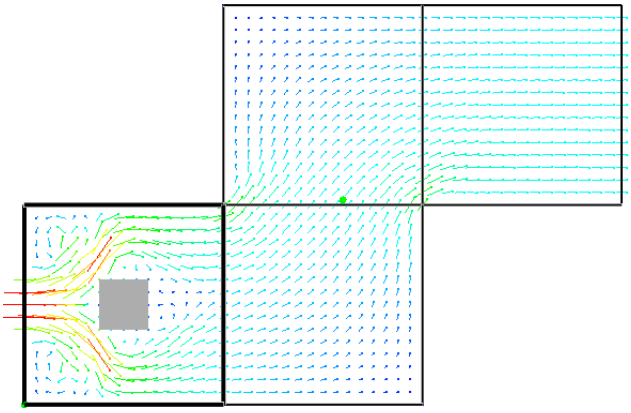

Figure 22: Flow field for a 4-mesh subdivision of the poisson2d test case at $t = 0.31$.

The pressure traces which were measured for these six different solvers in the indicated device are illustrated in Figure (23). Note, that the observed stair-like course correlates to the alternating strengths of the inflow velocity. As a globally operating direct method, UGLMAT provides an exact solution to the Poisson problem in each time step. Thus, its pressure trace is regarded as a reference solution for all other solvers.

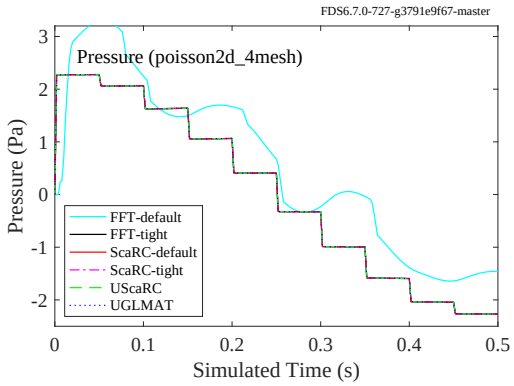

Figure 23: Pressure traces of the different pressure solvers for `poisson2d` with 4 meshes.

Obviously FFT-default (cyan) has troubles to map the pressure trace correctly while SCARC-default (red) already seems to match exactly. Both are structured solvers which are not able to set the correct boundary values along the inner obstruction by design. However, this basically doesn't seem to carry much weight here, as can be seen for SCARC-default for which the obstruction is the only possible error influence in this test case. With a view to the constantly changing global velocity field, the multi-mesh decomposition seems to cause much more troubles which is reflected in the serpentine line of FFT-default. The steady changes of the inflow conditions can only be spread across the entire domain by successive mesh-by-mesh communications. Thus, the new inflow information reaches the pressure device in the third mesh only with delay. Evidently, this cannot be sufficiently remedied by the default pressure iteration.

However, as observed for FFT-tight (black), the tighter pressure iteration works great and is able to completely resolve these troubles leading to a correct pressure trace if only a higher number of pressure iterations is performed. In contrast to this, there seems to be no reason at all to apply SCARC-tight in this case. Since all SCARC variants provide correct transitions at mesh interfaces by design, the overall error influences are significantly less pronounced here. Finally, USCARC (green dashed) operates independently of all these effects and produces the same correct pressure trace as UGLMAT (blue dotted).

So far, Figure (23) reflects the accuracy which is achieved by the different solvers in the approximation of the pressure trace, but it does not say anything about the numerical effort required for this. While for both unstructured solvers UGLMAT and USCARC only one pressure iteration is needed by construction, the structured variants require the execution of different numbers of pressure iterations until the respective velocity tolerance is met. In order to analyze this situation in more detail, the achieved accuracies for the velocity errors, as displayed in Figure 24, are set in relation to the number of pressure iterations required to reach them, as displayed in Figure 25.

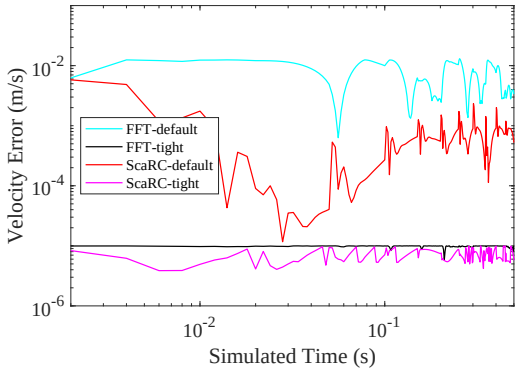

Figure 24: Velocity tolerances of the structured pressure solvers for `poisson2d` with 4 meshes.

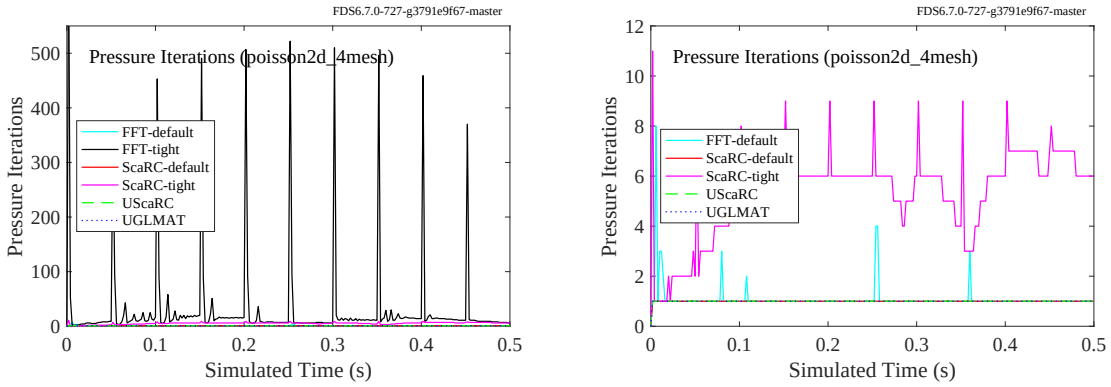

Figure 25: Pressure iterations of the different pressure solvers for `poisson2d` with 4 meshes. (Left) Comparison of all solvers. (Right) Zoomed view without FFT-tight.

Note, that Figure 24 only illustrates the velocity tolerances for the structured solvers because UGLMAT and USCARC achieve machine precision in the range of 10^{-16} by design which is omitted here to allow a more diversified view of the remaining solvers. Similarly, while the left plot in Figure 25 compares the pressure iterations for all six solvers, the right plot omits FFT-tight in order to give a more detailed overview of the others.

According to the size of the default velocity tolerance, FFT-default (cyan) leads to a coarse velocity error in the range of about 10^{-2} as can be seen in Figure 24. This error consists of two parts, namely the accuracy along the internal obstruction and the accuracy at the mesh interfaces. The related pressure iteration mostly converges within 1 cycle. Only with changing inflow conditions occasionally up to 4 cycles are needed as can be seen at the cyan line in the right plot of Figure 25. The maximum number of 8 cycles is only required at the beginning of the simulation until the flow field has built up for the first time, but never again after. Thus, convergence is fast, but it is also associated with the error of the pressure trace displayed in Figure 23.

Although its surrounding pressure iteration is based on the same default velocity tolerance, SCARC-default (red) already provides a basically smaller velocity error compared to FFT-default in the range of about 10^{-4} up to 10^{-3}, see Figure 24 again. This is because the only part that makes up this error relates to the inner obstruction, whereas nothing is added at mesh interfaces any more. Convergence can always be achieved in just 1 pressure iteration independently of the inflow changes and there is no major fluctuation at the beginning.

Figure 24 also proves that FFT-tight (black) and SCARC-tight (magenta) succeed to fulfill the required velocity tolerance of 10^{-5}. However, when looking to Figure 25 the biggest difference between both solvers becomes apparent: While FFT-tight needs relatively many pressure iterations (averagely 35) to fulfill the fine tolerance, SCARC-tight requires considerably less (averagely 6). Note, that the average calculation was restricted to the time interval $[0.05, 0.5]$ such that the higher fluctuations which only occur at the beginning were neglected in order not to falsify the whole picture. In particular, at every change of the inflow conditions, the latency of FFT-tight gets obvious in a short-term increase of the number of iterations (up to 522 in the worst case) which basically relies on the fragmentation induced by the subdivision.

In fact, the right plot in Figure 25 also reveals slight increases of SCARC-tight for every inflow change, because it still has to prevent the velocity penetration into the internal obstruction. However this is significantly less than for FFT-tight and a maximum of 11 iterations is never exceeded. This difference again reflects the fact that the subdivision is already captured correctly by SCARC and that the only disturbing influences are caused by the internal obstruction. As expected, UGLMAT (blue dotted) and USCARC (green dashed) produce machine precision accuracy and both need exactly one pressure iteration.

5.2.2 Subdivision into 16 meshes (`poisson2d_16mesh`)

To analyze the scalability towards higher mesh numbers the pipe geometry of Figure 21 is now subdivided into 16 meshes with a side length of 20 cm each. Furthermore, a finer grid resolution for the single meshes into 32^2 cells is used which corresponds to a grid resolution of 0.625 cm. The respective default velocity tolerance for the structured solvers FFT and SCARC amounts to 0.3125 cm, while the fine velocity tolerance of 0.00001 m/s is used again. The resulting flow field at time $t = 0.31$ is illustrated in Figure 22.

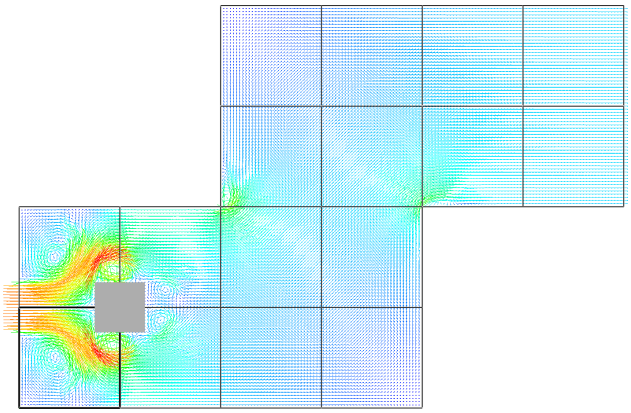

Figure 26: Flow field for `poisson2d` with 16 meshes at $t = 0.31$.

Subsequently, similar plots for the pressure traces, velocity errors and required number of pressure iterations will be presented as already done for the 4-mesh subdivision before. To this end, the left plot in Figure 27 shows the measured pressure traces over the whole simulation time $[0, 0.5]$. Obviously, the already known stair-like course of the pressure trace is still overlaid with small oscillations which are captured differently well by the various pressure solvers. To better illustrate their resolution qualities, the right plot of Figure 27 also gives a zoomed view to the time interval $[0.29, 0.302]$.

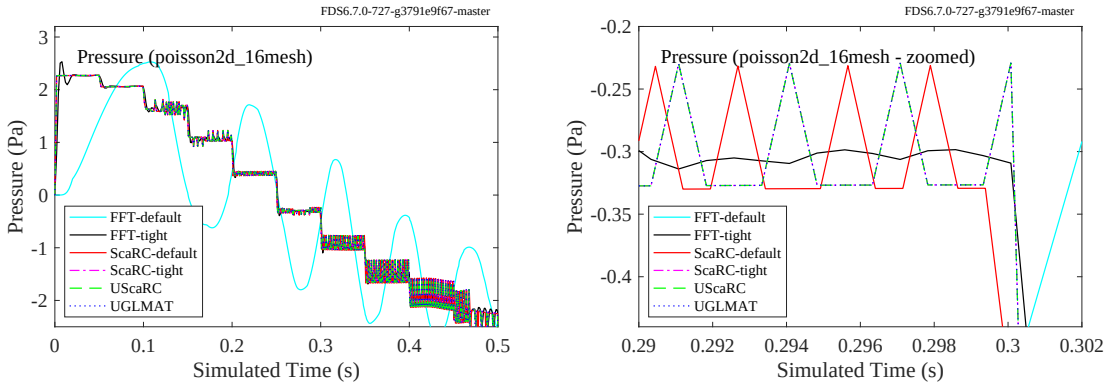

Figure 27: Pressure traces of the different pressure solvers for `poisson2d` with 16 meshes. (Left) Whole time interval $[0.0, 0.5]$. (Right) Zoomed view to time interval $[0.29, 0.302]$.

As expected, the higher number of meshes poses an even greater challenge for FFT-default (cyan) to correctly map the pressure trace as can be seen in the left plot of Figure 27. Thus, the use of a tighter velocity tolerance for the FFT solver seems to be indispensible. In fact, the zoomed view on the right reveals that FFT-tight (black) is close to the correct pressure course as indicated by UGLMAT, but still doesn't completely match the small oscillations. However, to reach this approximate consistency, it already requires a large number of pressure iterations, see the left plot of Figure 29. While on average 642 pressure iterations have to be performed, the maximum

available number of 1000 pressure iterations is often not sufficient to map the fine velocity criterion of 10^{-5} and occasionally only a lower accuracy can be achieved. This is also illustrated in Figure 28 which displays the achieved velocity errors for all structured solvers. Obviously, the black FFT-tight line has a hard time falling below the 10^{-5} line. When comparing the different plots therein, please note that the velocity errors for both SCARC variants only result from the penetration error towards the internal obstruction while the velocity errors for the FFT variants are a combination of both the error at the internal obstruction and at mesh interfaces.

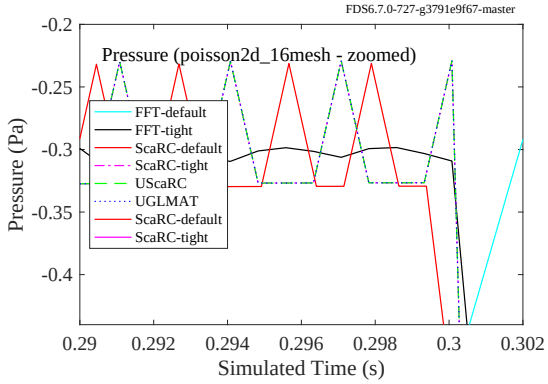

Figure 28: Achieved velocity tolerances for the structured solvers for `poisson2d` with 16 meshes.

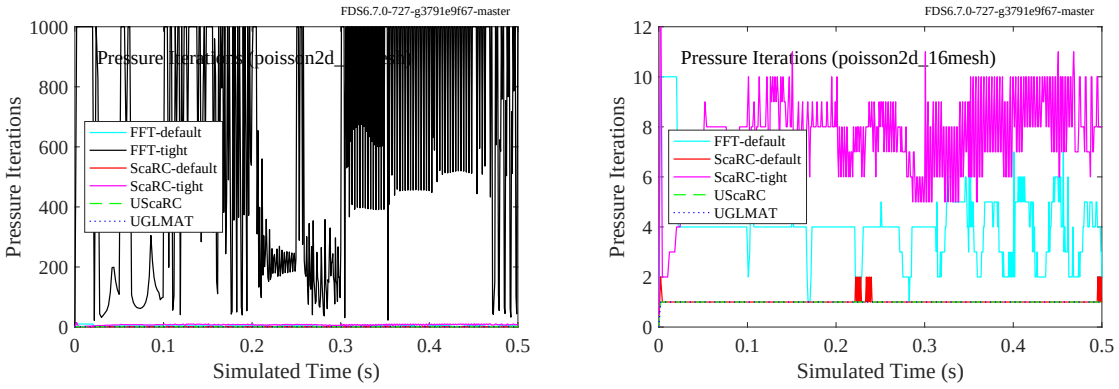

Figure 29: Number of required pressure iterations of the different pressure solvers for `poisson2d` with 16 meshes. (Left) Comparison of all six solvers. (Right) Zoomed view without FFT-tight.

Furthermore, the right plot in Figure 27 proves that USCARC (green dashed) and SCARC-tight (magenta) produce the same exact pressure trace as the reference solver UGLMAT. While UGLMAT and USCARC just require 1 pressure iteration by design, SCARC-tight needs only 7 pressure iterations on average and 11 in maximum to fulfil the tight velocity tolerance, see the related number of pressure iterations in the right plot of Figure 29 and the velocity tolerance in

Figure 28 which is completely below 10^{-5}. Since SCARC-tight already provides correct inter-mesh transitions by design, it not only requires significantly less pressure iterations than FFT-tight, but it also reliably achieves the desired velocity tolerance. Even SCARC-default (red) seems to be in the correct order of magnitude and only needs 1 pressure iteration in average (and only very rarely 2), however its pressure trace still shows the slight offset to the correct course indicating that the default tolerance isn't just fully sufficient to handle the internal obstruction.

In conclusion, in this case the domain decomposition seems to be far more difficult to handle than the internal obstruction. However, these observations do not represent a general rule. That the situation can be completely different and that the inner obstructions may have a greater influence than the mesh decomposition becomes apparent subsequently in the `duct_flow` case.

5.2.3 Application of generalized SCARC solvers

As explained in detail in Section (4.2) there are multiple possibilities to generalize the SCARC solver. The different SCARC variants, which were used for the multi-mesh `poisson2d` computations so far, were all based on a global data-parallel CG-method with different types of preconditioning. Two further variants, SCARC-twolevel and SCARC-multigrid, were introduced at the beginning of Section (5) which are now also applied to the `poisson2d` case in order to give an impression of their properties and performance. Both are based on structured discretizations. While also using a global CG-method, SCARC-twolevel relies on a 2-level-Schwarz-preconditioner whose additional coarse grid is based on one refinement of the domain decomposition itself. SCARC-multigrid uses a global MG-method instead with smoothing by local SSOR-methods, again with a coarse grid based on one refinement of the domain decomposition and additionally all intermediate grid levels in-between.

Since it was shown above that USCARC gives the same correct result as UGLMAT, the presentations in this subsection will only refer to different SCARC versions where USCARC is used as reference. For the 16-mesh decomposition of `poisson2d`, again with the fine grid resolution of 0.625 cm, Figure (30) compares the pressure traces for the unstructured USCARC (green) with the structured SCARC-default (red dashed), SCARC-twolevel (cyan dash-dotted), SCARC-multigrid (blue dotted) where the later two were applied for the default velocity tolerance, too.

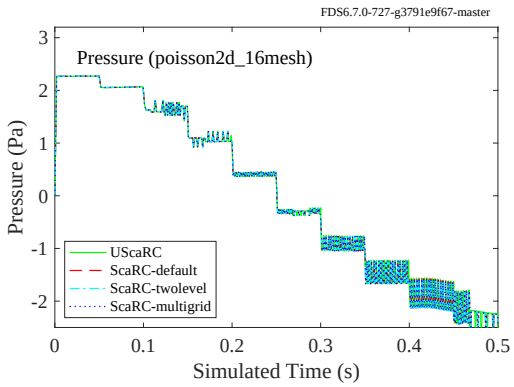

Figure 30: Pressure traces of different SCARC variants for `poisson2d` with 16 meshes.

A zoomed view to the interval $[0.29, 0.302]$ is also given in the left plot of Figure (31). Obviously, all default structured SCARC variants identically produce the same pressure trace which proves their basic correctness. But all show a slight offset compared to the correct pressure trace given by USCARC, which has already been observed in Figure 27. Again, this is based on their structured nature and the associated inaccuracy along the internal obstruction which obviously cannot be completely eliminated by the default pressure iteration. However, using the tight pressure iteration instead, the shift disappears for all structured variants as can be seen in the right plot of Figure (31). For all of them, this requires the same number of averagely 7 pressure iterations.

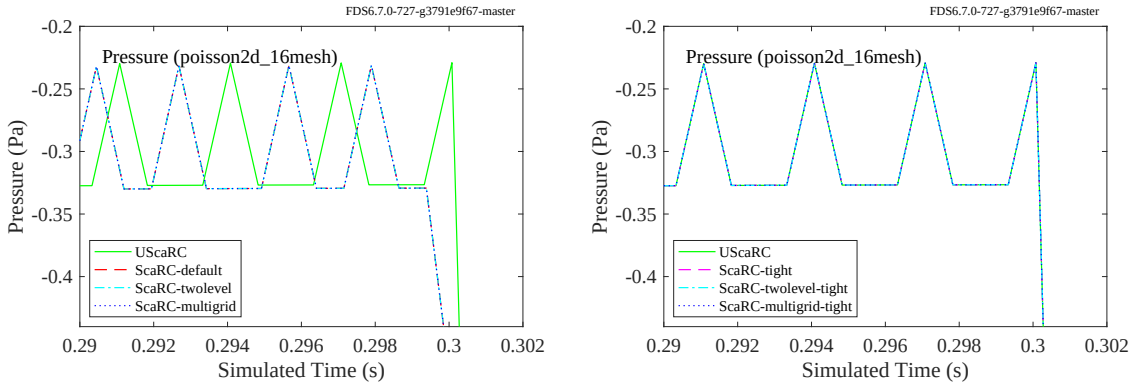

Figure 31: Zoomed view of time interval $[0.29, 0.302]$ for the pressure traces of different SCARC variants for `poisson2d` with 16 meshes. (Left) Default velocity tolerance. (Right) Tight velocity tolerance.

The focus of the subsequent investigations lies in the comparison of the achieved numerical efficiencies. The left plot in Figure (32) reveals that the convergence rates for the single SCARC variants are quite different. The pure CG-variants, USCARC and SCARC-default, only achieve relatively bad convergence rates at 0.81, needing about 150 SCARC iterations per Poisson solution which is quite a lot and still needs further optimization. Please note that if only a 4-mesh decomposition instead of the 16-mesh decomposition is used for the `poisson2d` geometry while maintaining the same fine-grid resolution of 0.625 cm, a convergence rate of 0.7 at about 90 SCARC iterations is achieved, which is still not good, but significantly better than for the 16-mesh case.

This observation reflects the fundamental design property of the 1-level-Schwarz variants that their convergence quality decreases with increasing number of sub-meshes. An improvement can only be achieved by adding global information mechanisms as can be clearly observed with the SCARC-twolevel variant (cyan). Here, the additional usage of a coarse grid level leads to a halving of the convergence rate to about 0.42, see the left plot of Figure (32). The right plot therein shows that this reduction is also associated with a comprehensive reduction of the number of SCARC iterations from about 150 to 40, which is quite much because it has to be performed twice per FDS time step. Certainly, the single iterations are more expensive due to the additional computations and communications associated with the coarse grid problem what has to be kept in mind when comparing the different approaches.

As expected the multigrid version shows by far the the smallest convergence rate in the range of 0.15 and requiring only about 15 SCARC iterations per Poisson solution. Here, the additional use

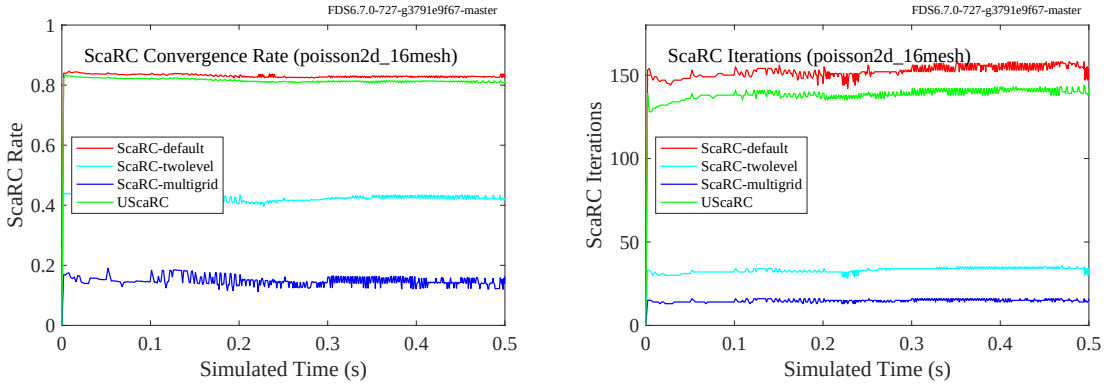

Figure 32: Comparison of different SCARC variants for the 16-mesh `poisson2d` case. (Left) SCARC convergence rates. (Right) Number of required SCARC iterations.

of a coarse grid level and all intermediate levels in-between has a very positive effect on the global coupling. In practice, this typically leads to convergence rates which are more or less independent of the local grid refinement level and the number of sub-meshes.

There are numerous possibilities to vary the input parameters for the different SCARC variants, which may have a significant influence to the convergence speed. This especially holds true for SCARC-multigrid. So far, its smoothing consisted of only 2 SSOR-steps per grid level. It has already been explained in Section (4.3.3) that the accuracy achieved in solving the local problems has a major impact on the quality and convergence speed of the global solution. Hence, if the number of smoothing steps is increased, the finally required global number of SCARC iterations can further be decreased. Figure 33 compares the resulting convergence rates and needed number of SCARC iterations for the use of 2, 3, 4 and 5 smoothing steps.

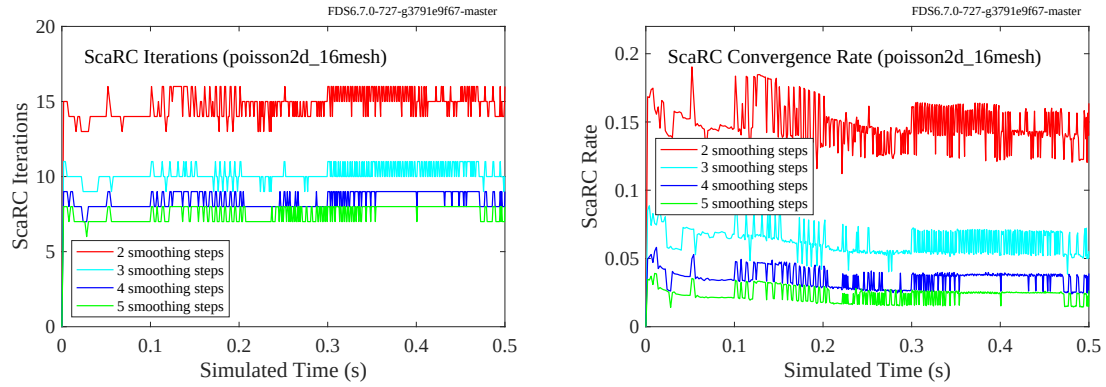

Figure 33: Convergence of SCARC-multigrid for different numbers of smoothing steps for `poisson2d` with 16 meshes. (Left) Required number of SCARC iterations. (Right) SCARC convergence rates.

Apparently, each additional smoothing step leads to a clear reduction of the achieved convergence rate and the number of required SCARC iterations. This amounts from 15 SCARC iterations

with a convergence rate of 0.15 in case of 2 smoothing steps up to only 7 SCARC iterations with a convergence rate of only 0.025 in case of 5 smoothing steps. Note, that during the smoothing global matrix-vector products (including local communication) are needed. Thus, the use of additional smoothing cycles leads to a corresponding increase of work. But at the same time, the number of coarse grid solutions (including global communication) is comprehensively reduced which in turn represents a major saving and usually makes up for the extra cost.

5.2.4 Preliminary conclusions

The cases just presented have shown that there is a wide range of SCARC variants that can be used to solve the Poisson equation. These differ significantly in the type of the underlying discretization, the preconditioning mechanisms used, and whether they work only at the fine grid level or at additional coarser grid levels. The biggest difference to the current FFT solver consists in the fact that SCARC has no more errors along inner mesh boundaries. Thus, for the structured SCARC variants often much less pressure iterations are needed than for the default FFT solver, because it only has to take care of the reduction of the penetration error towards inner obstructions. For the unstructured variant USCARC, no pressure iteration at all is needed anymore.

As a basis for all further developments, the focus of the previous analyses has been on proving the fundamental correctness of the various SCARC approaches. For this purpose the case `poisson2d` was developed and tested in the above as well as many other ways. Corresponding 3D studies related to geometries with one or more internal obstructions and with subdivisions into different numbers of meshes were also carried out. However, since it was not possible to gain any insights beyond those shown so far, no presentation of these cases has currently been made. Instead, the coming test series will concentrate on the official FDS verification and validation cases to examine the suitability of SCARC for more realistic cases.

The previous studies have shown that the addition of coarser grid levels can contribute to a significant improvement of numerical efficiency. However, the achieved convergence acceleration must be put in relation to the increased costs for each single iteration. A final assessment for the performance of the different variants can only be done on the base of additional measurements of the computational times. In order to arrive at meaningful estimates for the optimal parameters a-priori, different sensitivity studies are in work which should finally allow to identify an optimal set of parameters that delivers resilient and efficient results for a large number of general cases. Some of the most recent tests, which have already been carried out for various validation cases, suggest that there is still a need for further optimisation with regard to the required running times which is currently in progress.

5.3 Karman vortex street

5.3.1 Subdivision into 4 meshes (`dancing_eddies`)

The following computations refer to the `dancing_eddies` case from the Pressure_Solver verification directory. It concerns a 30 cm long, two-dimensional channel with a grid resolution of 0.1 cm which is subdivided into 4 meshes. Air is pushed into the channel at 0.5 m/s. A flat obstruction in the left third of the channel causes the formation of a Karman vortex street. Figure 34 displays a contour plot of the pressure at the final simulation time t=2 s.

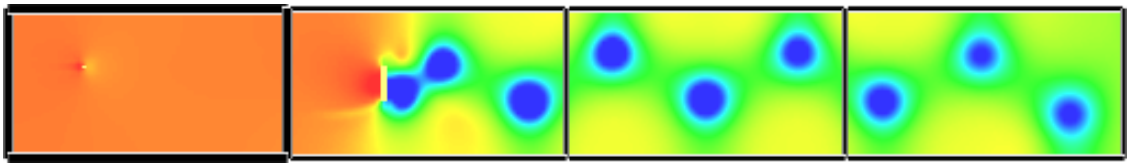

Figure 34: Contour plot of the pressure for USCARC after 2 s in case of a 4-mesh subdivision.

The original test case compares the structured solvers FFT-default (with the default velocity tolerance of 0.0005 m/s) and FFT-tight (with the finer tolerance of 0.00001 m/s) with the unstructured direct solver UGLMAT. These tests are now extended by the application of the structured solvers SCARC-default and SCARC-tight (both with the same velocity tolerance settings as their FFT counterparts) as well as the unstructured USCARC. As in the original case, the computed pressure traces are compared with the corresponding single mesh solution which is used as reference solution here, see Figure 35.

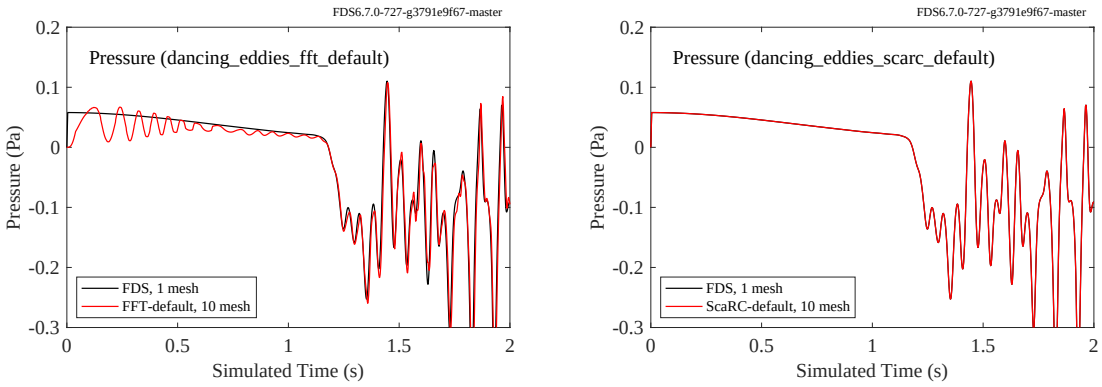

Figure 35: Pressure traces of the default structured pressure solvers for the 4-mesh `dancing_eddies` case compared to the single-mesh computation. (Left) FFT-default. (Right) SCARC-default.

As already observed for FFT-tight and UGLMAT in the original test case, SCARC-tight and USCARC show full agreement with the single mesh solution and are therefore not further illustrated graphically. In contrast to FFT-default, however, SCARC-default already shows a good match for the coarse default tolerance as can be seen in Figure 35.

Furthermore, Figure 36 displays a comparison of the required number of pressure iterations for the different solvers. The left plot shows a comparison for all solvers, the right plot omits FFT-tight to give a zoomed view to the situation for the remaining solvers.

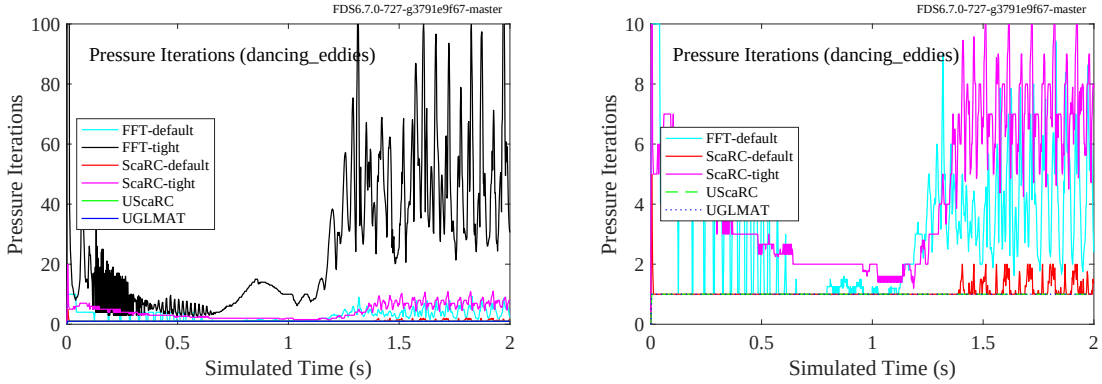

Figure 36: Comparison of the required number of pressure iterations. (Left) All solvers. (Right) Zoomed view to all solvers except of FFT-tight.

FFT-default (cyan) needs 9 pressure iterations in maximum, but associated with the differences for the volume flow displayed in Figure 35. FFT-tight (black) gives an accurate result, but requires averagely 24 iterations, occasionally even up to 120. In contrast, SCARC-default (red) only needs a maximum of 2 iterations, but on average only 1, while already reaching a high approximation accuracy as seen for the pressure trace in the right plot of Figure 35 above. Thus, the requirements of SCARC-tight (magneta) are hardly more, it needs at most 4 and on average also only 1 iteration to fulfil the finer tolerance. By construction, UGLMAT (blue) and USCARC (green) are finished within exactly 1 iteration. Please note again that the maximum and mean statistics were only performed after a certain initialization time of 0.1 s in order to exclude the large fluctuations at the beginning seen for the tight variants.

5.3.2 Subdivision into 10 meshes (`dancing_eddies_10mesh`)

To analyze the scalability of the various solvers, a subdivision into 10 instead of the previous 4 meshes is considered as well, see Figure 37. Now, the same analysis is performed again.

Figure 38 gives an overview of the approximation quality of the different pressure solvers. The lower-left plot therein shows that FFT-default (cyan) reaches the default tolerance in an average of 3 and a maximum of 9 iterations. However, as the top-left plot reveals, this isn't enough to capture the course of the pressure trace completely. The resulting oscillations are even more pronounced than for the 4-mesh case while the remaining solvers show full consistency again. In particular,

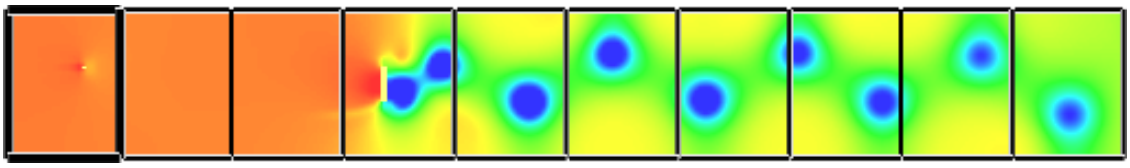

Figure 37: Contour plot of the pressure for USCARC after 2 s in case of a 10-mesh subdivision.

FFT-tight (black) matches perfectly, but it also needs comprehensively more pressure iterations to fulfil the finer tolerance (averagely 28 iterations, but occasionally up to 99).

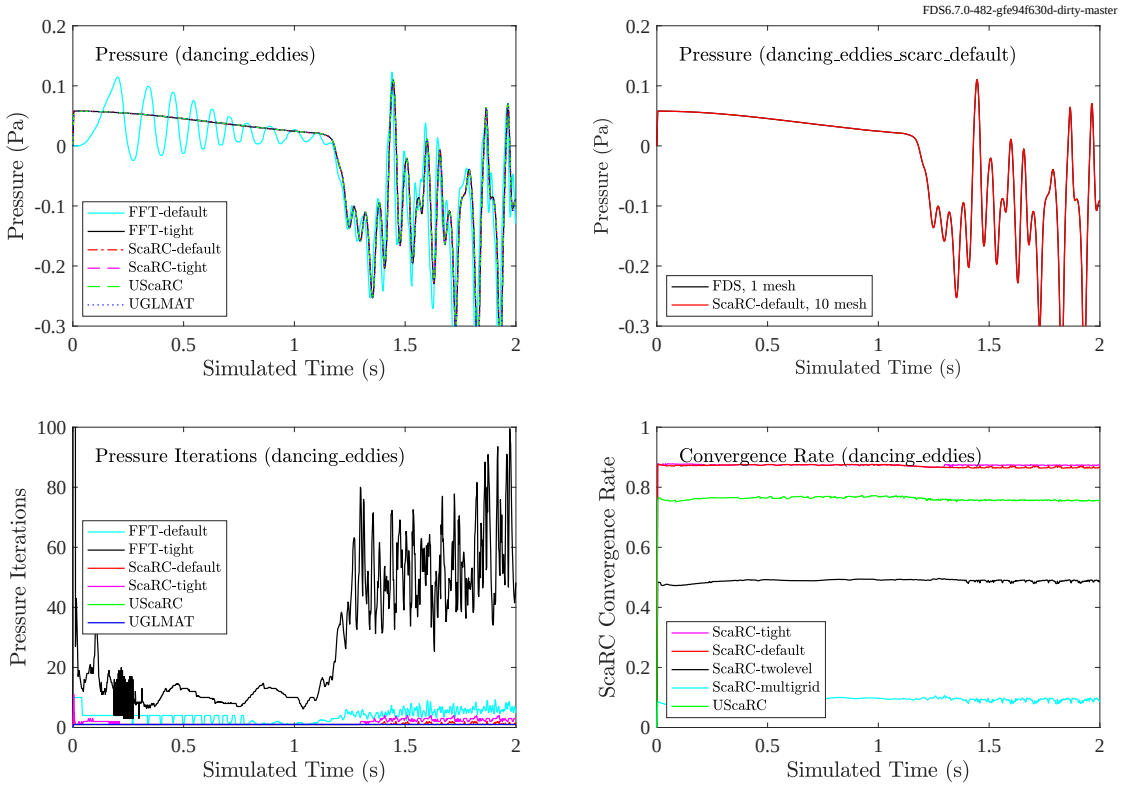

Figure 38: Results of different pressure solvers for a `dancing_eddies` test case subdivided into 10 meshes. (Top-left) Pressure traces of all solvers. (Top-right) Sole pressure trace of SCARC-default. (Bottom-left) Number of pressure iterations for all solvers. (Bottom-right) Convergence rates of different SCARC variants.

The top-right plot of Figure 38 proves that the 10-mesh SCARC-default already gives a sufficiently good approximation of the pressure trace. Furthermore, there is no big difference between SCARC-default and SCARC-tight, both need only 1 iteration on average and not more than 2 or 4 at the maximum. Again, USCARC and UGLMAT match perfectly in 1 iteration. Due to the increasingly transient nature of the flow, all solvers need more pressure iterations in the later course of the simulation, but it seems to stay within bounds for all.

Furthermore the bottom-right plot in Figure 38 displays the convergence rates of the different SCARC variants. Obviously, the additional use of a coarse grid level within SCARC-twolevel almost leads to a halving of the convergence rate to 0.49, while the other CG-based variants lie in the range of a slow 0.8 or even worse. Especially for longer pipe-shaped domains like the one considered here, it can be very advantageous to use a coarse grid in order to spread at least a coarse average of the global information over the complete length of the domain.

As expected, only the MG variant is out of the scope here in a positive sense and delivers a convergence rate of only 0.09 which, however, required some sensitive preliminary studies to adjust the optimal parameters. In this context, we refer again to the considerations on the evaluation of the overall performance at the end of Section 5.2.3.

A final look to the CPU-times in Figure 39 shows that, despite the sometimes higher number of pressure iterations, that at least for the 4-mesh case SCARC-tight gives a smaller CPU time than USCARC although its convergence rate is still worse. That's because in case of structured SCARC, the locally regular grids allow the use of local FFT methods for preconditioning, which can be performed with highest performance. In case of unstructured USCARC, local LU decompositions must be used instead. Although these are also taken from an optimized program package, the Intel® MKL library, they unfortunately prove to be slower than the local FFTs. Strategies to deal with these observations and to improve the situation are currently in work.

In the 4-mesh case UGLMAT performes best. But the situation changes for the 10-mesh case since the *LU*-decomposition also suffers from increased communication effort for larger mesh numbers. Here the different SCARC variants are in the same range as FFT-tight, while UGLMAT and SCARC-multigrid are in the midfield The SCARC-twolevel variant, despite its better convergence rate, needs the longest computing time and still requires computational improvement. Basically, for the generalized structured SCARC variants there is still a lot of room for further optimisation since they also can make use of local FFTs for smoothing which will be tested soon.

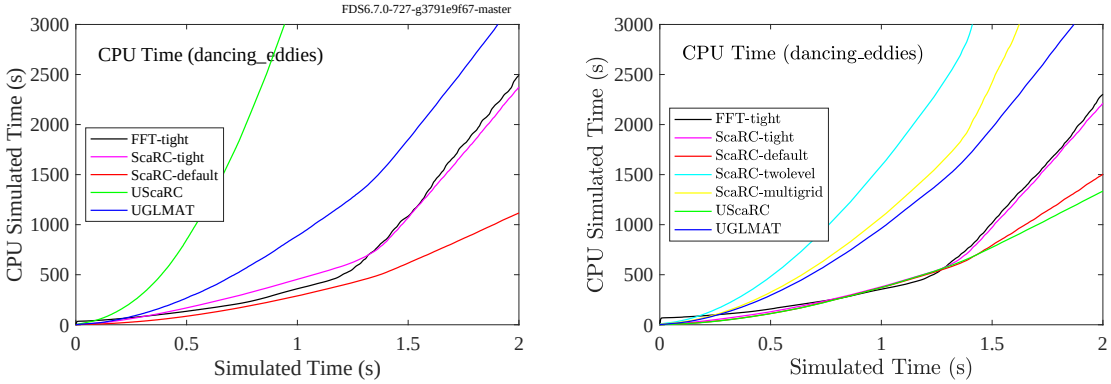

Figure 39: CPU times of different pressure solvers for the 4- and 10-mesh `dancing_eddies` test case

In summary, two important observations can be made: For this special case, the domain decomposition seems to place greater demands on the pressure solver than the inner obstruction. Thus, the worse convergence for FFT-default compared to SCARC-default (despite the same default tolerance) can be explained by the fact that the error influence observed in FFT-default is essentially

caused by the domain decomposition, but not by the inner obstruction. This doesn't hold true for ScaRC since all considered variants solve for the global pressure matrix and no longer produce velocity errors along the mesh interfaces. The remaining slight increase of pressure iterations is only related to the correction of the velocity errors along the internal obstruction which, in this special case, already works quite well for the default tolerance.

In view of the small number of internal obstructions considered here, it does not really seem to be necessary to perform a tighter pressure iteration for structured ScaRC, but the default velocity tolerance is already sufficient to reach a good approximation accuracy. Due to the mentioned runtime differences for the structured and unstructured preconditioners, there is no big advantage in taking the unstructured UScaRC in this case. The unstructured variant would only make sense if it achieved a substantially more accurate approximation along the inner obstruction (or the same in a comprehensibly shorter time), which is not the case here. But this must not be understood as a general rule. As the `duct_flow` case 5.4 will show, the situation can be completely different if the influence of the internal obstructions is much larger than that of the domain decomposition.

5.4 Flow through a duct (`duct_flow`)

The next section follows the `duct_flow` case from the Pressure_Solver verification directory which is defined on a cubic domain of side length 6.4 m. A subdivision into 8 meshes and a grid resolution of 0.2 m are used. With a speed of 1 m/s, air is pushed into a multi-angled duct of size 1 m^2 which meanders through the domain, as illustrated in Figure 40. Since there is no thermal expansion, the volume flow at the inlet and the outlet of the duct must be the same which provides a good basis test for the approximation quality of the different solvers. More information can be found in the FDS User's Guide[**?**].r

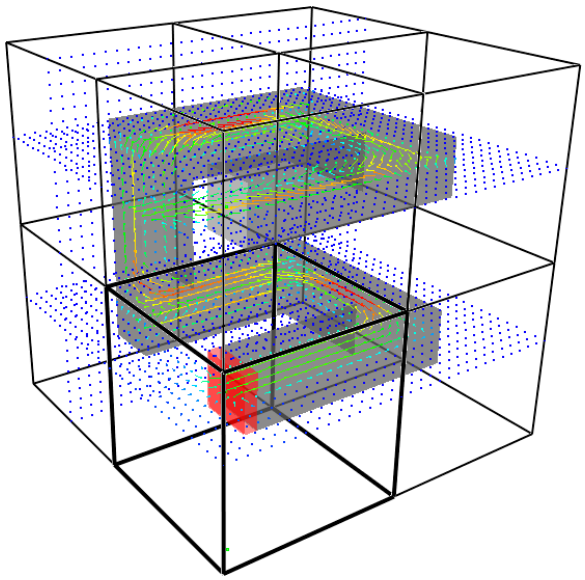

Figure 40: Flow field of the `duct_flow` case for a 8-mesh subdivision

By design, the flow is only limited to the inside of the duct and is forced to frequent changes of direction due to the numerous internal obstructions. This situation places particularly high demands on the structured pressure solvers to ensure a correct boundary approximation there. The subdivision of the domain into individual subdomains makes the situation even more difficult.

In order to classify both possible error influences, namely at inner obstructions and at mesh interfaces, the different pressure solvers introduced in Section 5 are applied and compared. Unfortunately, the default structured solvers, FFT-default and SCARC-default, are not capable of providing sufficient accuracy along the internal obstructions and will be omitted subsequently. For the tight structured variants, FFT-tight and SCARC-tight, a `VELOCITY_TOLERANCE` of 0.001 m/s is used along with `MAX_PRESSURE_ITERATIONS` set to 1000. Furthermore, the unstructured solvers, UGLMAT and USCARC, are applied as well.

According to the original study the left plot in Figure 41 compares the recorded volume flows at the in- and outflow while the right plot shows the number of required pressure iterations. For both structured solvers, FFT-tight and SCARC-tight, considerable differences between the in- and outflow get obvious. However, despite occasional larger outbreaks, SCARC-tight mostly requires less pressure iterations than FFT-tight. This is based on the fact that FFT-tight additionally has to correct the velocity field along inner boundaries, which is not the case for SCARC-tight.

Nevertheless, it seems that for this case the main cause of errors can be traced back to the variety of inner obstructions. All in all, both structured variants do not really deliver satisfactory results here. Instead, both unstructured solvers UGLMAT and USCARC provide exact matches for the outflow while needing exactly one pressure iteration.

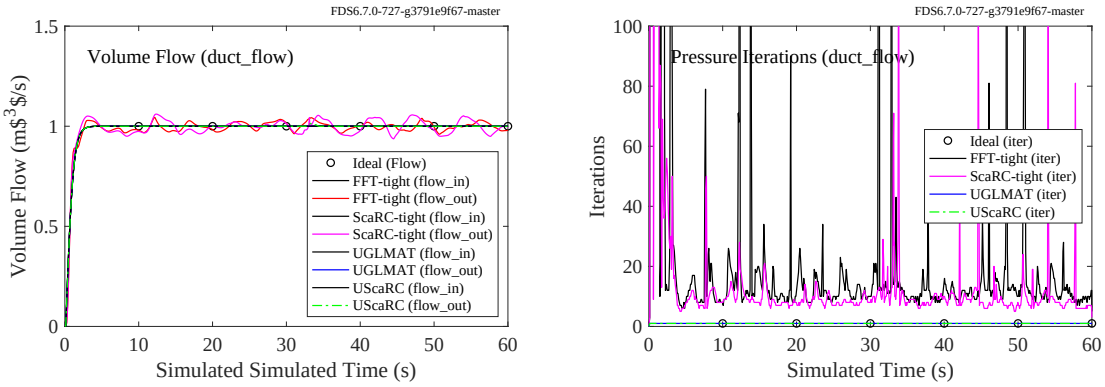

Figure 41: (Left) Volume flow into and out of a square duct. (Right) The number of pressure iterations as a function of time.

In contrast to the `dancing_eddies` case, in which the application of the structured solvers was even more time-efficient, the unstructured solvers can play out their full potential for irregular grids here and deliver better results by far. This becomes especially clear in Figure 42 which compares the velocity errors along the walls of the duct for the structured and the unstructured variants.

In the left plot a snapshot of FFT-tight at $t = 43.8$ s is shown. Despite the high number of pressure iterations there are obvious errors which can only be further reduced at disproportionately

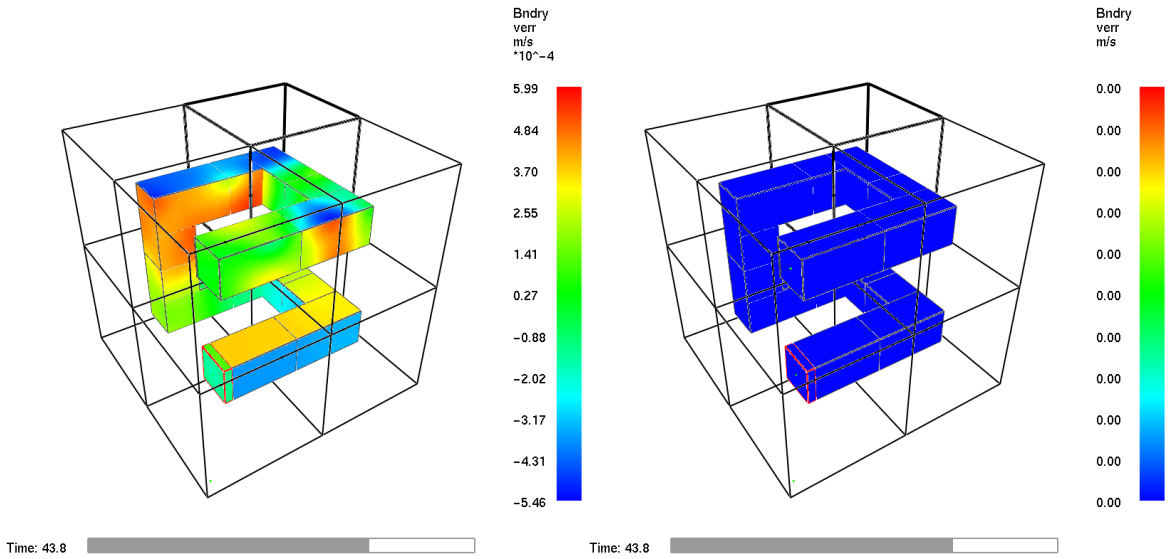

Figure 42: Velocity error along internal obstructions in `duct_flow` case. (Left) The error is in the range 10^{-3} for FFT-tight and SCARC-tight (Right) Machine precision is reached for UGLMAT and USCARC.

great expense. Although it requires slightly fewer pressure iterations, SCARC-tight shows a comparable picture as well. The large number of obstructions and the associated flow dynamics make it extremely difficult here for the structured solvers to achieve a good approximation with moderate effort. These troubles clearly dominate the additional negative effect of the domain decomposition in this case, which is why both structured solvers are equally affected. By contrast, the right plot shows the corresponding snapshot of USCARC which gives machine precision instead. The same zero-error picture can be obtained for UGLMAT as well.

Finally, the memory requirements of the UGLMAT solver are analyzed. This solver has to build the global LU-decomposition of the Poisson matrix A and store it in a distributed way over all meshes. Table 3 compares the number of non-zeros of A with that of the corresponding triangular matrix L for different grid refinements of the current case as measured within the solver.

Cells	$NNZ(A)$	$NNZ(L)$	$NNZ(L)/NNZ(A)$
32^3	125,685	4,513,040	36
64^3	1,027,008	82,630,016	81
128^3	8,300,736	1,693,459,406	204

Table 3: Growing memory need of UGLMAT in case of a grid refinement from 32^3 to 128^3 cells; The number of non-zero entries of the Poisson matrix $NNZ(A)$ is compared to the corresponding number $NNZ(L)$ of the triangular matrix L

For the calculations shown above a total grid resolution of 32^3 cells was considered. In this case the ratio of the non-zeros between L and A already amounts to 36. If the grid is further refined, however, it drastically raises up to a ratio of 204 in case of 128^3 cells. For this finer

resolution the application of the *LU*-decomposition requires additional memory for the storage of about 1.69 billions of double precision values. In view of realistically large problems with a significantly higher number of grid cells and submeshes, this 'memory-hunger' can become an invincible constraint. Thus, on a computing platform with a given amount of storage more finely resolved problems can be considered with FFT and SCARC since both are much less memory intensive.

At this point it must be mentioned that USCARC, as an unstructured solver, also uses local *LU*-decompositions, because the application of the local FFT is no longer possible. However, these are only built in the scope of the individual meshes and do not act across the entire domain. Thus the associated increase of memory requirements seems to be acceptable or does at least not grow with increasing number of meshes. But as already mentioned, for performance reasons, alternatives for the local *LU*-decompositions are being worked on anyway.

5.5 Periodic Boundaries

A number of test cases which are based on periodic boundaries can be found in the FDS Verification Guide [**?**]. Periodic boundary conditions are often used to evaluate the quality of a 'pure' solver without the disturbing influence of boundary effects. Alternatively, they are also used to reduce the size of a computational domain for a problem with recurring patterns.

For single-mesh cases true periodic boundary conditions can be easily integrated into the Crayfishpak-based FFT solver. Unfortunately, this is not possible in the multi-mesh case. Here, matrix entries belonging to a periodic boundary cell are interpreted as Dirichlet entries for which the solution value is explicitly specified. Missing values are exchanged between the related 'periodic neighbors' in the subdivision. While reasonable results can be achieved, this approach is not entirely correct and possibly leads to small errors in the solution. More information can be found in the corresponding section of the FDS User's Guide [**?**].

In contrast to that, SCARC is able to apply true periodic boundary conditions also in the multi-mesh case. Instead of the Dirichlet-based matrix stencil used for the mesh-wise FFT solver, a true periodic matrix stencil is incorporated into the Poisson matrix. Again, the values for the missing 'legs' are correspondingly exchanged. To prevent the system from becoming indefinite, a corresponding *condensed system* is used. More detailed algorithmic information on this topic will follow. To check the basic correctness of this procedure, some of the available periodic test cases in FDS were also performed with different SCARC variants und will be briefly summarized below.

5.5.1 Analytical Solution to Navier-Stokes Equation (`ns2d_4mesh, ns2d_16mesh`)

This series of test cases, from Verification/NS_Analytical_Solution, is concerned with the approximation of a given analytical solution for the 2D incompressible Navier-Stokes equations. The basic geometry is a square of side length $L = 2\pi$ with a uniform grid resolution of $\delta x = \delta z = L/N$ in each direction for which different resolutions $N = \{8, 16, 32, 64\}$ are analyzed. The resulting solutions are spatially periodic on the interval 2π. Furthermore, the inviscid case $\nu = 0$ and a viscous case $\nu = 0.1$ are considered. In the inviscid case the solution happens to be also temporally periodic on 2π. Detailed information can be obtained in the corresponding section of the FDS Verification Guide [**?**].

Several multi-mesh computations were performed with structured SCARC-default. According to the analysis in the FDS Verification Guide [**?**], Figure 43 shows the initial ($t = 0$) and final ($t = 2\pi$) state of the numerical solution for the grid resolution $N = 64$ and a 16-mesh subdivision of the domain. Due to the temporal periodicity, the analytical solution requires both to be completely identical. And in fact, 16-mesh SCARC-default works correctly and remains in conformity with the analytical conditions. This means in particular that the periodic boundary conditions are correctly integrated into the system of equations.

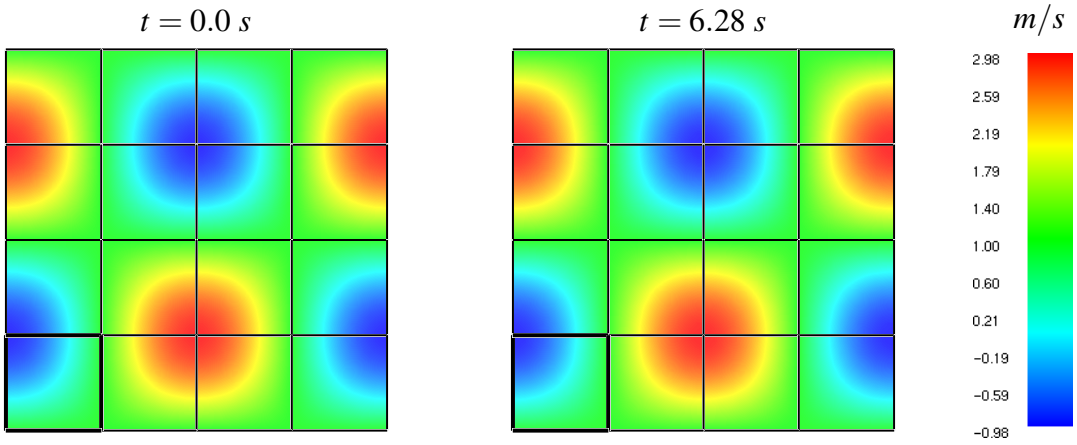

Figure 43: Identical initial and final state of the u-component of velocity for a 16-mesh SCARC computation of the `NS_Analytical_Solution` case, showing that SCARC is able to treat periodic boundary conditions for multi-mesh cases, too.

In the further course of the original test it was shown that in the single-mesh case the advective and viscous terms are approximated with second-order accuracy. In line with this analysis the related cases are analyzed with 4-mesh SCARC-default and the corresponding results are displayed in Figure 44. As can be seen, with increasing grid refinement SCARC clearly converges to the analytical solution. Moreover, as shown in Figure 45 for both the inviscid and the viscous case, it also preservers the second order convergence rate of the original single-mesh scheme.

5.5.2 Variable-Density Manufactures Solution (`shunn3_16mesh`)

The `shunn3` test case from Verification/Scalar_Analytical_Solution is intended to prove the second order accuracy of the FDS time-stepping scheme. It is based on a manufactured sinusoidal solution which diagonally translates across a square-shaped domain of side length 2. The solution is spatially periodic in this domain and has a time period of 1 s. Different grid resolutions of $N = \{32, 64, 128, 256, 512\}$ cells per direction were considered there. The settings for this case are very complex and can be read in more detail in the FDS Verification Guide [**?**].

As before, SCARC was applied for different subdivisions of this case into 4 and 16 meshes. Exemplary, the evolution of the density ρ, the mixture fraction Z and the u-velocity for the grid resolution of 256^2 were shown in Figure 46 for a corresponding 16-mesh SCARC computation.

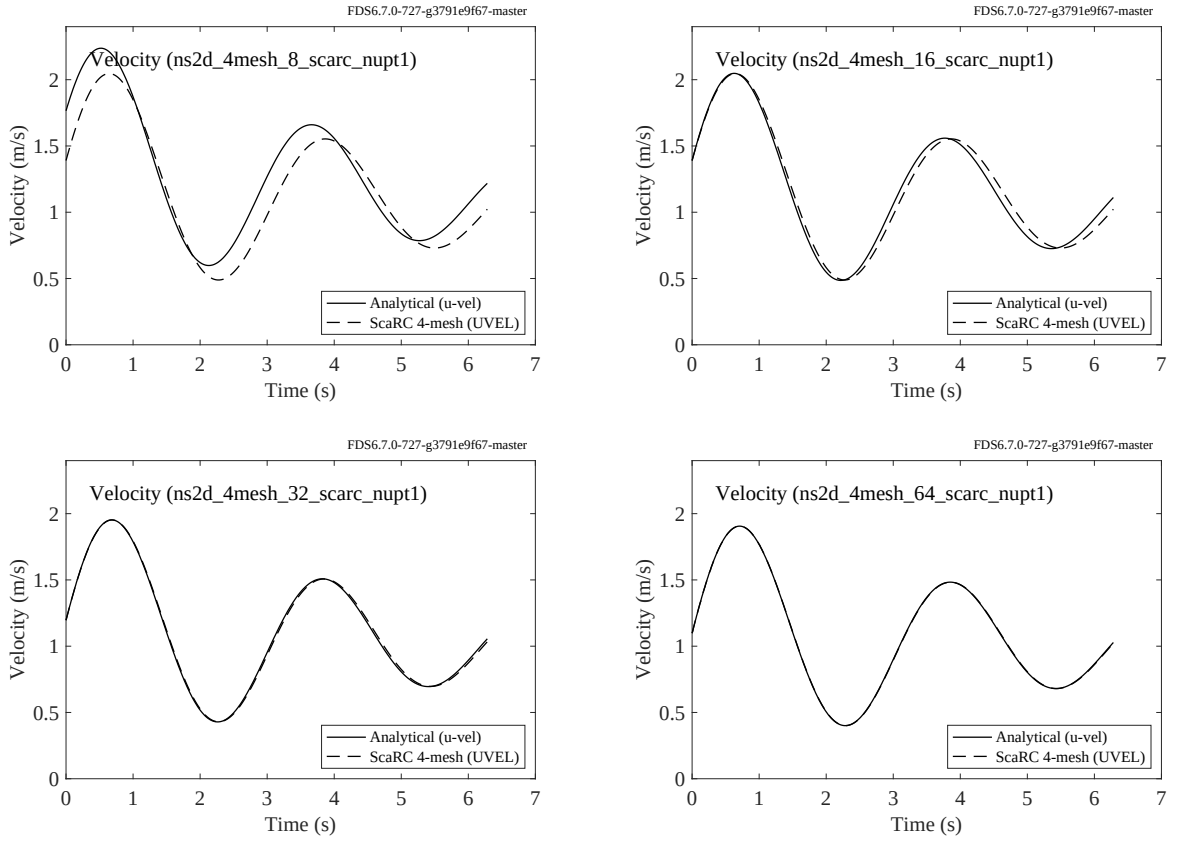

Figure 44: Time history of the u-component of velocity for the `NS_Analytical_Solution` case with $\delta x = L/N$ where $L = 2\pi$ m and $N = \{8, 16, 32, 64\}$ for a 4-mesh SCARC computation, showing that the second-order convergence for the single-mesh case is preserved by 4-mesh SCARC.

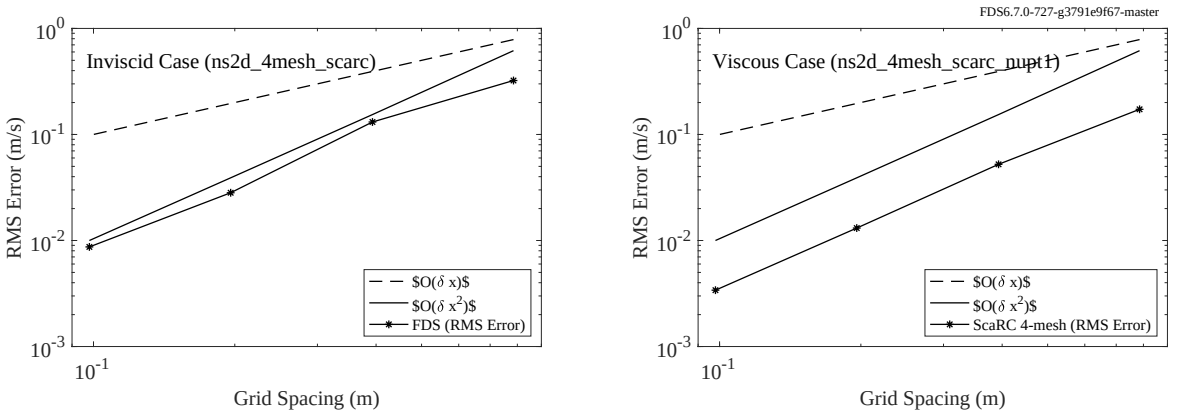

Figure 45: Convergence rate for the u-component of velocity for a 4-mesh SCARC computation with $\nu = 0$ (Left) and $\nu = 0.1$ (Right) for the `NS_Analytical_Solution` case. The dashed and solid lines serves as references for first- and second-order accuracy. The second order accuracy for the advective terms of the original single-mesh case is preserved by 4-mesh SCARC.

Obviously, the periodicity of the analytical solutions is completely preserved and there is no recognizable difference to the single mesh approximation, which in turn proves the correct handling of periodic boundary values for multi-mesh decompositions by ScaRC.

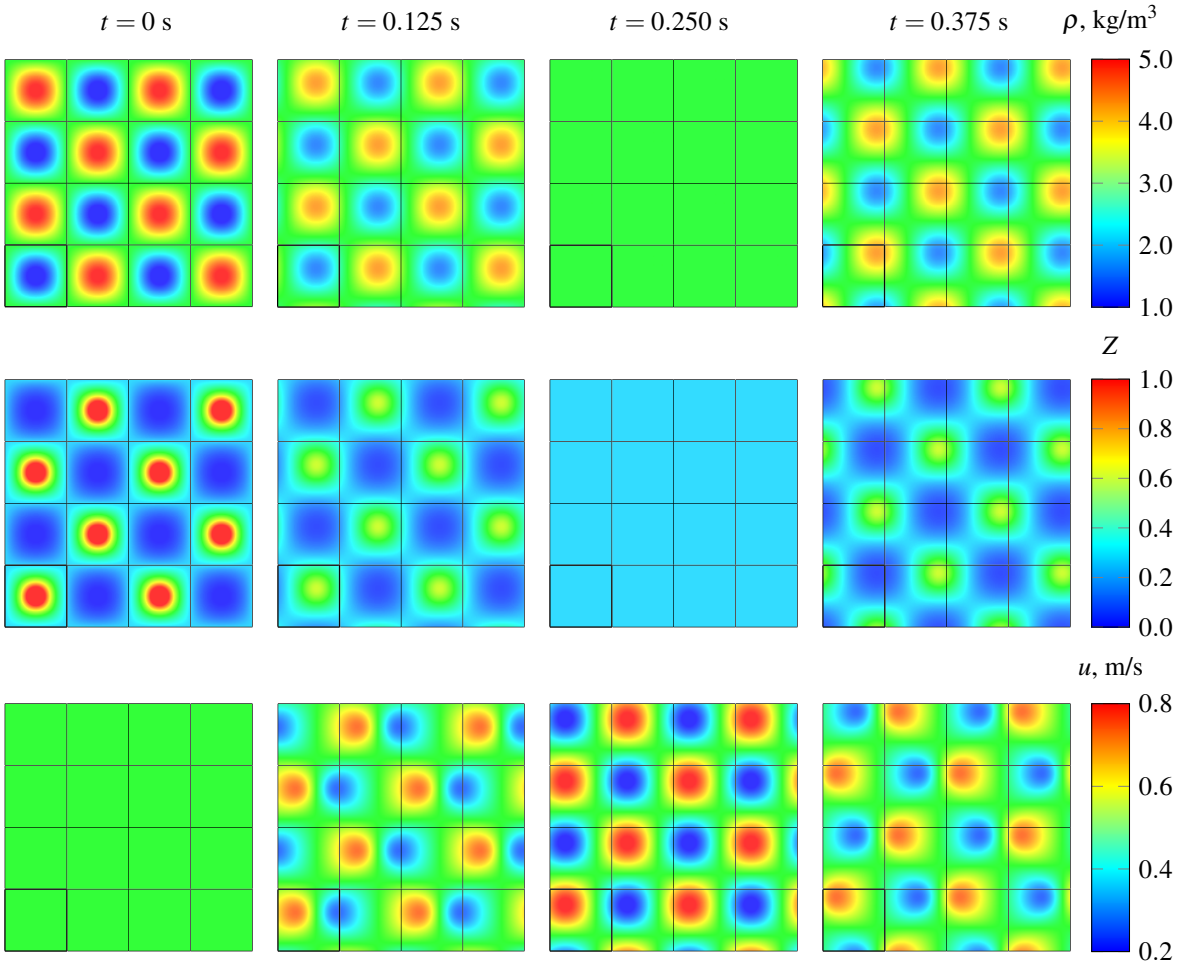

Figure 46: Evolution of the variable-density manufactured solution for a 16-mesh ScaRC computation for the `shunn3` case. From top to bottom, the images show density, ρ, mixture fraction, Z, and the u-velocity component from the 256^2 simulation at the specified times. The parallel computation by ScaRC clearly preserves the periodicity of the analytical solution.

5.5.3 Ribbed Square Duct Flow (`ribbed_channel_4mesh`)

In the previous two cases, periodic boundary conditions were applied to the entire boundary of the computational domain. The situation is different, however, if there is a mixture of Neumann and periodic boundary conditions, since here only the periodic boundary values have to be communicated between the corresponding partners, while those for the Neumann components are set according to the boundary specifications of the case.

SCARC is able to handle these components individually. For the Neumann parts of the boundary, corresponding Neumann matrix stencils are integrated into the system with according settings of the right hand side. For the periodic parts the corresponding periodic stencils are used. As long as there are no additional Dirichlet components on the boundary, the condensed system must be used again in order to avoid that the matrix becomes indefinite.

Figure 47 gives an example where periodic boundary conditions are used at `XMIN` and `XMAX` while Neumann boundary conditions are used elsewhere which is related to the Turbulence/ribbed_channel test case. More details can be found in the FDS Verification Guide [**?**]. Due to the large internal obstruction the unstructured USCARC solver based on a 4-mesh subdivision was used. Evidently, the original flow pattern is matched again.

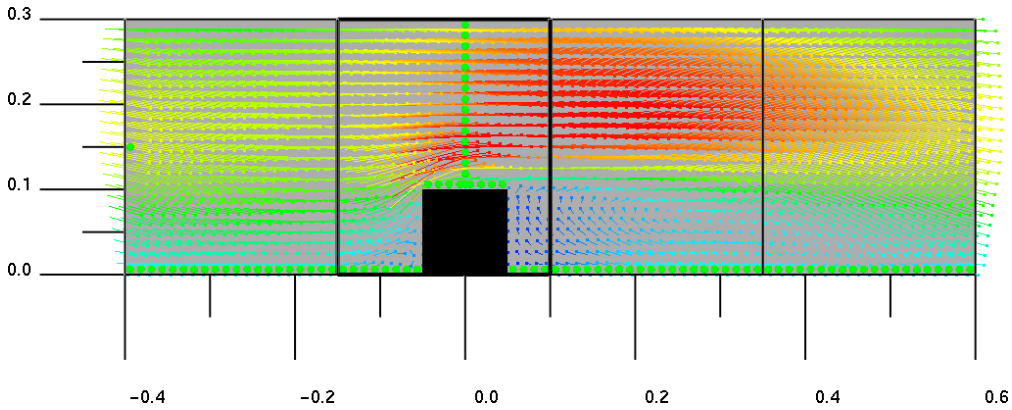

Figure 47: Mean velocity vectors colored by velocity magnitude (red is 2 m/s) for the ribbed square duct case with $h/\delta x = 8$ for a 4-mesh USCARC computation for the `ribbed_channel` case.

This case aimed to predict the mean recirculation patterns on the windward and leeward side of the rib. To this end line devices were used at the floor and at the center of the duct. The corresponding measurements for the 4-mesh USCARC computation are summarized in Figure 48. Note, that in comparison to the original test case the cases for the finest case $h/\delta x = 16$ are still missing due to their large complexity and will be handled later. In summary, these plots are in qualitative accordance with the original single-mesh values which again proves the basic correctness of the periodic boundary treatment and the parallel processing in SCARC.

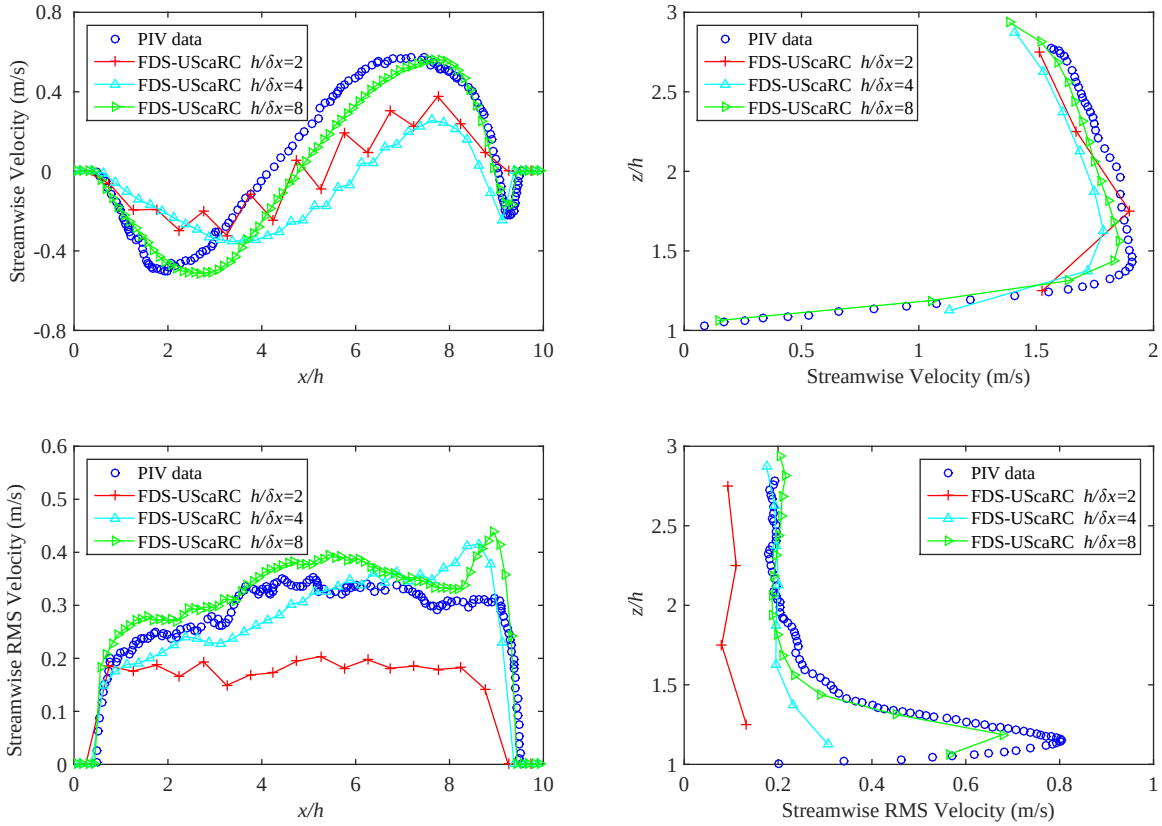

Figure 48: Mean and RMS velocity profiles for ribbed square duct flow in case of a 4-mesh USCARC computation for the `ribbed_channel` case.