The FDS pressure equation:
Intuitive understanding and solution strategies

Dr. Susanne Kilian
hhpberlin, Ingenieure für Brandschutz GmbH, D-10245 Berlin
s.kilian@hhpberlin.de

Abstract

The pressure equation plays an important role within the entire FDS solution process. Particularly with regard to an efficient parallelization, it represents a major challenge. The following presentation is intended to provide a more intuitive understanding of the inherent properties of this equation and the associated implications regarding its efficient and accurate parallel solution. Basic design principles of currently applied and potentially alternative solution strategies will be presented and compared regarding their advantages and disadvantages.

1 MATHEMATICAL BACKGROUND

Starting from the non-conservative formulation of the momentum equation, a series of transformations and simplifications leads to the pressure equation as used in FDS,

$$\nabla^2 H = -\left[\frac{\partial(\nabla\cdot\mathbf{u})}{\partial t} + \nabla\cdot\mathbf{F}\right]. \tag{1}$$

The pressure term $H \equiv |\mathbf{u}|^2/2 + \tilde{p}/\rho$ on the left hand side corresponds to the total pressure due to Bernoulli divided by the density ρ and is largely based on the perturbation pressure $\tilde{p}$ by which the fluid motion is driven. The first term on the right side implies the influence of all thermodynamic quantities on the velocity field over time whereas the second term subsumes the effects of gravity, particle drag, viscosity and perturbation pressure, too. For simplicity, the content within the bracket on the right side of the equation (1) is abbreviated to R, finally leading to the general form of the so-called *Poisson equation*

$$-\nabla^2 H = R. \tag{2}$$

From a mathematical point of view, the Poisson equation is an elliptic partial differential equation of second order. Its unique solvability also requires the definition of appropriate boundary conditions which consist either of the specification of known values of H itself (*Dirichlet conditions*) or its normal gradient (*Neumann conditions*) along the boundary of the computational domain.

Equations of Poisson type play a major role in many fields of science such as e.g. the calculation of gravitational potentials, temperature distributions, electrostatic fields and more. Thus, research into the development of corresponding numerical solvers follows a very long history and has produced a variety of powerful approaches, all of which have their advantages and disadvantages. The selection of a suitable representative from this collection must always be made in view of the characteristic features and special needs of the underlying problem.

Because of the strong interaction between the pressure term H, the velocity field $\mathbf{u}$ and the different components within the force term $\mathbf{F}$, the Poisson equation has a large impact on the entire solution process in FDS, too. Typical FDS simulations are often based on tens of thousands of time steps, where each time step consists of at least two solutions of the Poisson equation as sketched in Figure (1).

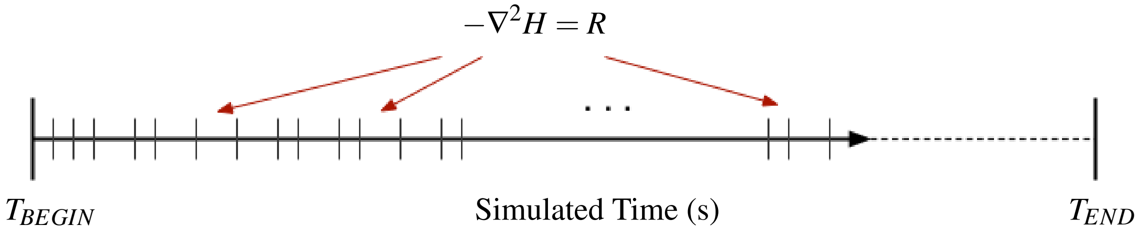


Figure 1: The Poisson equation has to be solved at least twice per FDS time step

It is therefore of great importance that the solution of this equation is both of high accuracy and high efficiency. Detailed information on the derivation of the pressure equation (1) and all the related quantities can be found in the FDS Technical Reference Guide [1].

1.1 Towards a more intuitive understanding of the Poisson equation

In order to gain a better understanding of the physical meaning of the pressure equation (1), some essential steps of its derivation are examined in more detail below. First, it is important to know that this derivation originates from a simplified version of the momentum equation

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla H = -\mathbf{F}. \tag{3}$$

This is a major consequence of the *Low Mach number assumption*, which is a fundamental design principle of FDS. It assumes a subdivision $p = \overline{p} + \tilde{p}$ of the full pressure p into a kind of background pressure $\overline{p}$, which only involves time and height and reflects the stratification of the atmosphere, and a perturbation pressure $\tilde{p}$ which is related to the various acting forces (e.g. air or heat being introduced into a room). For a typical fire simulation in a building which is open to the atmosphere, $\overline{p}$ is essentially constant. What actually drives the flow field is the perturbation pressure $\tilde{p}$ which may be different in every point of the geometry and is usually much smaller than the background pressure $\overline{p}$.

This special pressure decomposition allows to get rid of some very restrictive numerical requirements that would otherwise have to be met, i.e. the step width in the numerical time stepping scheme must only be constrained by the speed of flow (which is typically only some tens of meters per second) as opposed to the speed of sound (which is about 340 meters per second).

In contrast to the original momentum equation, which uses the gradient of the full pressure p, the simplified momentum equation (3) is only dealing with the perturbation pressure $\tilde{p}$. During the upper derivation process the perturbation pressure is again split into two further parts which are distributed to both, H and $\mathbf{F}$. The main reason for this additional splitting is that the linear algebraic system resulting from the discretization of the equation (2) does not change over time and will have constant coefficients, which allows the use of highly optimized solvers as will be shown in more detail below.

Finally, applying the divergence operator to the simplified momentum equation (3) leads to the second derivative term $\nabla^2 H$ as used in the Poisson equation (2). Summed up as follows, $\nabla^2 H$ corresponds to an application of the so-called *Laplace operator* Δ, or shortly, the *Laplacian*,

$$\Delta = \nabla^2 = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{4}$$

to the scalar-valued pressure field H. In other word, the expression ∇^2 represents the divergence of its gradient field which helps to gain a more intuitive understanding of the underlying physics.

Note, that any event that generates varying pressure values in different parts of the domain will lead to unbalanced forces that induce a flow. Figure (2) illustrates an example of a two-dimensional pressure field H where each point of the x-y-plane is assigned a corresponding value in z-direction (left plot). The application of the gradient operator to this scalar field, ∇H, results in a vector field where for every point (x, y) the associated vector shows in the direction of the steepest increase of H and is orthogonal to the *contour lines*, i.e. the lines of constant pressure. The direction of the resulting flow is opposite to this gradient, namely in the direction of lowest pressure (right plot). Note, that the gradient is zero at the bottom of the valley and the top of the hill.

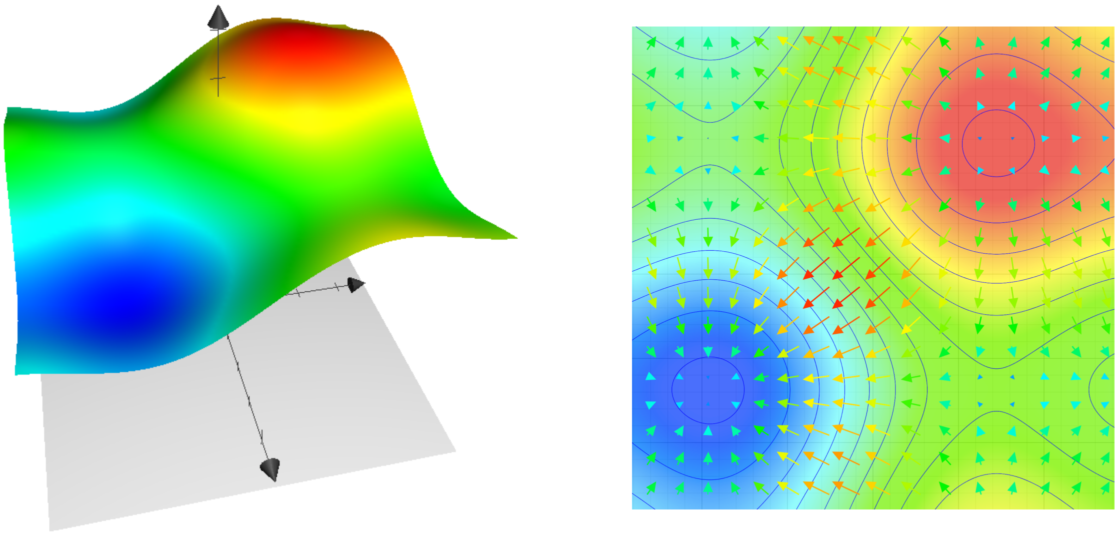


Figure 2: Example of a scalar-valued pressure field H in 2D (left) with corresponding contour lines and its negative gradient field $-\nabla H$ (right)

Applying the divergence operator to this vector field gives another scalar field which can be regarded as a measure of how much the vector field expands or contracts for each individual point. A **positive divergence** indicates a kind of source where the vector field tends to diverge, see the red-coloured part where all directions are pointing away from a high pressure zone. A **negative divergence** indicates a kind of sink where the vector field tends to converge, see the blue-colored part where all directions are pointing towards a low pressure zone. In the middle part there seems to be a balance between incoming and outgoing vectors, with roughly a **zero divergence**.

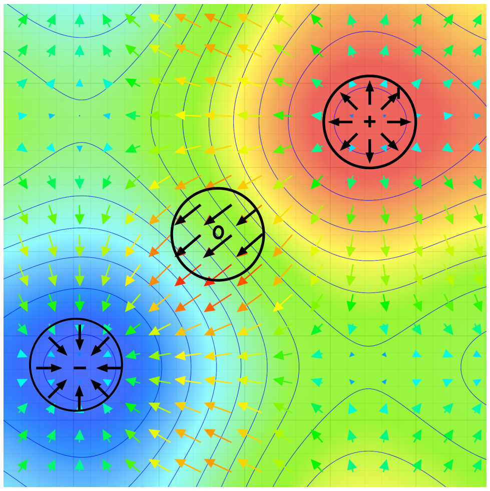


Figure 3: Regions with positive, negative and zero divergence within a velocity field

Mathematically speaking, the first derivative (the gradient) describes the change of H itself and the second derivative (the Laplacian) in turn indicates how this rate of change itself changes, or in other words, it defines an average rate of change of the function H.

In this sense, the Laplacian reflects the tendency of various physical quantities to reach an overall state of equilibrium (e.g. pressure or temperature). If there were no force terms representing any kind of sources or sinks, each point would be in perfect average with its neighbors, corresponding to a zero Laplacian. Functions whose Laplacian is zero are called *harmonic functions*. If instead e.g. air is blown into a room at a certain point, the whole room is pressurized, because from this point an expansion or rather a positive divergence will start. The same holds true if a fire is lit in a room, where the heat will cause a positive divergence at the location of the fire. The pressure now balances out very quickly or tries to neutralize itself through the domain. And for each point of the domain, the Laplacian defines how this averaging process is related to the acting forces.

1.2 *Finite Difference Discretization of the Poisson equation*

To transform the analytical equation (2) into an algebraic analogue that can be handled on a computer system, a finite-difference approximation of the Laplacian is used. For this purpose, the computational domain is covered with a rectilinear grid with resolution h, as illustrated in Figure (4). In fact, different grid widths can be used in FDS for all spatial directions, but for the sake of simplicity this is not considered here. For each individual grid cell (i,k) the corresponding value of H is now approximated by an average of its neighboring values and the corresponding contribution of the force term based on the well-known *5-point stencil* in 2D and the corresponding *7-point stencil* in 3D, which exactly reflect the averaging properties of the Laplacian described above. Applying this stencil to all n cells of the grid leads to a large linear system of equations

$$Ax = b \tag{5}$$

with a *Poisson matrix* A, representing the term $-\nabla^2 H$, a right hand side vector b, representing the discretized version of R and a solution vector x which finally represents the discretized solution H.

Whereas the sizes of b and x are equal to the number of grid cells n, the matrix A has 5- or 7-times as much non-zero entries, based on the number of H-items in the matrix stencil for the respective dimension. The above mentioned boundary conditions of Dirichlet and Neumann type require corresponding adjustments of the matrix stencils at cells adjacent to the external boundary of the domain which are directly included into the matrix A. Now, every predictor and corrector step of the FDS time stepping procedure requires at least one solution of system (5), meaning that per time step at least two solutions of it are needed. For more information see again the FDS Technical Reference Guide [1].

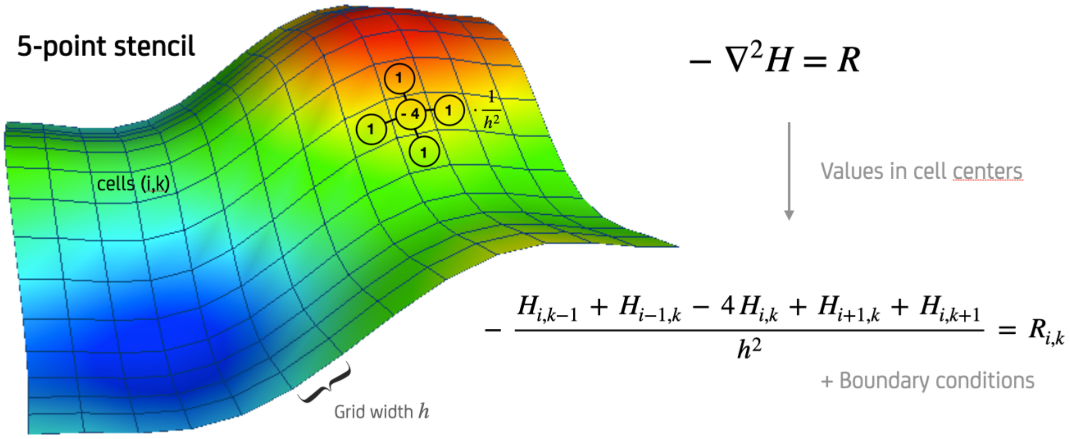


Figure 4: Discretization of the Poisson equation by the 5-point stencil in 2D

In accordance with its analytical analogue, the upper discretization process illustrates very clearly the intrinsic properties of the discrete Laplacian: Each point is defined as mean value of its immediate neighboring points. In this way, all points in the domain are strongly coupled to each other. A change of one point directly influences all other points via this successive chain of neighborhood relations. Whatever strategy is used to solve the discrete equation (5), it must account for this highly recursive relationship within the domain, otherwise the intrinsic physical properties may be disrupted.

2 DESIGN PRINCIPLES OF DIFFERENT PRESSURE SOLVERS

2.1 *Structured versus unstructured discretization*

FDS is fundamentally based on the subdivision of the computational domain into rectangular blocks which in turn are subdivided into rectangular grid cells. Objects like walls or other obstructions internal to these meshes are mapped as rectangular shapes conforming with the underlying grid.

Regarding the discretization of the Poisson equation (2), two alternative approaches can be distinguished which differ in the treatment of these internal obstructions:

- A **structured discretization** explicitly includes both gas phase and solid state cells and is the default type in FDS. The same matrix stencil is applied to the entire mesh. Without any exception, all grid cells are incorporated into the resulting discretization matrix, which therefore takes a very regular shape. A major advantage of this strategy is that highly tuned solvers can be used which have been specially developed for regular grid structures and can be performed with enormous computational efficiency. However, it is not possible to directly prescribe the correct boundary conditions along internal obstructions. Erroneously, the velocity field may contain non-zero contributions towards internal solids with a corresponding loss of accuracy, which is the main disadvantage of this strategy.
- An **unstructured discretization** only includes the gas cells, while all solid cells are omitted from the equation system. For gas cells directly adjacent to the surface of an obstruction, a *homogeneous no-flux* Neumann condition is explicitly specified and included in the matrix such that penetration errors along internal obstructions no longer occur. The main advantage of this strategy is therefore that it offers more geometric flexibility and achieves a higher degree of accuracy. In contrast to the structured case, however, it requires the use of individual matrix stencils for the different grid cells depending on their position related to obstructions such that the regular matrix shape gets lost. The solution of such an irregular system places significantly higher demands on the robustness of the solver which can comprehensively limit the achievable efficiency and is the main disadvantage of this methodology.

Using the example of a small 2D mesh with an internal obstruction, Figure (5) illustrates the differences between both strategies. In the left plot the same matrix stencil is used in and along internal solid cells as for the rest of the mesh, all contributing equally to matrix A. In the right plot, however, the solid cells are excluded from the matrix and corresponding modifications of the adjacent matrix stencils are made.

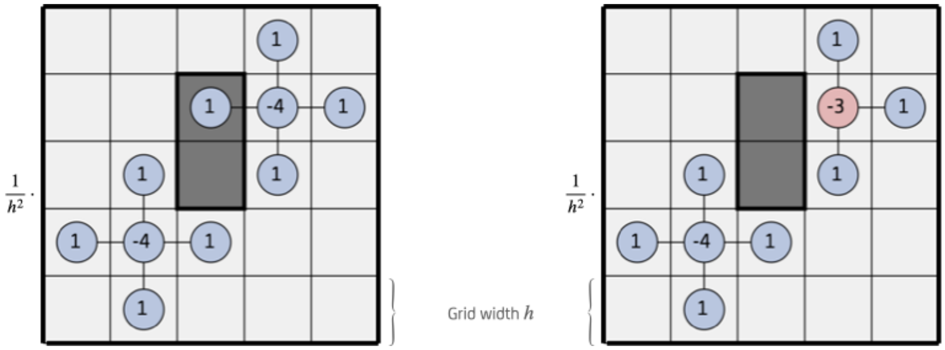


Figure 5: Matrix stencils in case of a structured discretization (left) and an unstructured discretization (right) for a 2D example geometry with internal obstruction

This procedure is reflected in the resulting Poisson matrices as shown in Figure (6). While the structured matrix has a completely regular and predictable structure which is only associated to the uniform cell numbering, the unstructured matrix obviously contains irregular parts with varying structures, see the grey colored parts in the right plot whose smaller size is related to the fact that the cells internal to the obstruction are not part of the matrix. This also results in the unstructured matrix being smaller than the structured one, since fewer grid cells are included to it.

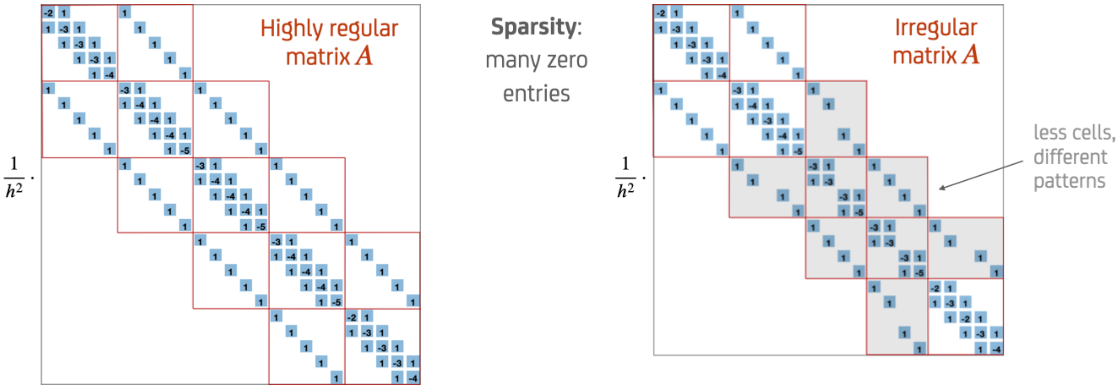

Highly regular matrix A
Sparsity: many zero entries
Irregular matrix A
less cells, different patterns
$\frac{1}{h^2}\cdot$
$\frac{1}{h^2}\cdot$

Figure 6: Poisson matrices resulting from a structured discretization (left) and an unstructured discretization (right) for the 2D example geometry with internal obstruction

However, despite the smaller size of the unstructured matrix, the solution of the structured system can usually be done much more efficiently, since highly optimized solvers can be used which are specially tuned to exploit the regularity of the grid. In FDS, both the structured and unstructured discretization type can be used in combination with suitable solution strategies each, as will be shown subsequently.

2.2 *Local versus global data exchanges*

If the computational domain can only be mapped with a single mesh, the related calculations are performed on a single processor and the subsequent considerations can be ignored. But typical FDS geometries are much too complex to be reasonably represented on a single square-shaped mesh, or they simply require far too many computing and memory resources to be calculated on a single processor.

To run them on a parallel computer, the domain is subdivided into several meshes, which are then assigned to the different processors such that the individual mesh calculations can be performed more or less in parallel. This, in turn, means that suitable parallel versions of the solution algorithms must be applied that can cope with this decomposition. To solve the overall problem and not just an accumulation of independent local problems, the processors must regularly exchange data at coordinated intervals. Two basic types of data exchange can be distinguished as illustrated in Figure (7):

- The meshes are only coupled by **local data exchanges**. In this case, data is only exchanged between directly adjacent mesh neighbors, which can usually be performed at very high speed on modern parallel computers. This procedure is often sufficient to ensure a basic coupling of the overall solution. However, it may not be possible to transfer global effects in time.

- The meshes are coupled by **local and global data exchanges**. In this case, data is exchanged both between directly neighboring meshes and between all meshes (no matter how far apart they are geometrically), resulting in a more immediate transfer of global information. But the global exchange causes significantly higher execution times, especially with large mesh numbers or an unbalanced distribution of cells across the meshes (so that with every global exchange the smaller meshes have to wait for the larger ones).

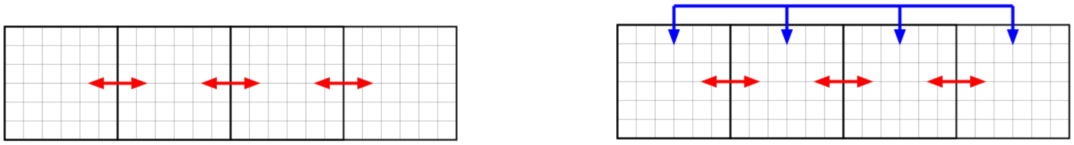


Figure 7: Purely local data exchange between directly adjacent meshes (left) or local data exchange with additional global data exchange between all meshes (right)

Now, what are the consequences of this distinction with regard to the solution of the pressure equation? Imagine that a new information enters a cell somewhere in the domain, e.g. because air is blown in or a combustion takes place. Certainly, the pressure is going to grow from this cell and it will immediately try to reach a state of equilibrium with its surrounding neighbors, according to the structure of the matrix stencil, as shown in Figure (8). These neighboring cells in turn will try to come into balance with their respective neighbors and so on and so forth until finally the boundary of the domain is reached.

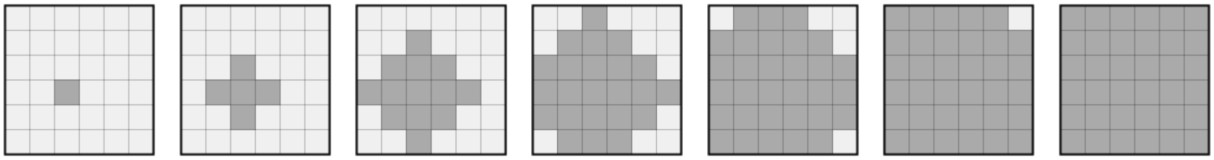
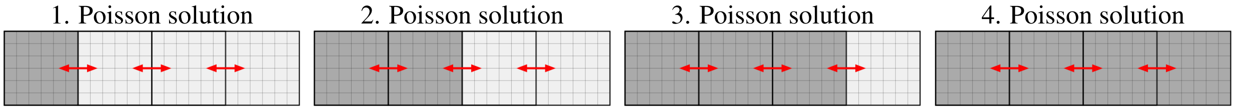


Figure 8: Highly recursive data coupling within one single Poisson solution

Important to know is that this whole averaging process takes place within ONE single Poisson solution! This means that every single cell in the entire domain immediately 'feels' the new information. If there is only a single mesh then everything is fine because each of the considered Poisson solvers, whether direct or iterative, manages to reproduce this highly recursive single-mesh coupling immediately.

But what happens if the domain is too large for one processor and has to be subdivided among several processors, as is often the case for e.g. long tunnels? In this case the new information is first trapped in the initial mesh and can only be forwarded by a corresponding data exchange. However, if only local data exchanges are performed, the new information can only be passed through the domain step by step after a whole sequence of local exchanges related to the local Poisson solutions, until it finally reaches the other end of the tunnel, as shown in Figure (9).



1. Poisson solution

2. Poisson solution

3. Poisson solution

4. Poisson solution

Figure 9: Multi-mesh case with delayed information transfer in case of only local data exchanges

Typical velocities of fire-induced flows only amount to tens of meters per second such that in reality it would probably take several seconds for the new pressure information to reach the other end of the tunnel. But remember, that as a consequence of the underlying Low Mach number assumption, the FDS pressure computation is based on the special Poisson equation (1) inherently assuming an infinitely fast propagation speed for information which will inevitably be fragmented by a multi-mesh decomposition.

If there is only a small number of meshes, experience shows that a sequence of local exchanges (as the standard FFT solver does) is sufficient to ensure appropriate coupling of the individual meshes and to generate a sufficiently accurate solution to the global problem. However, if the number of meshes increases, there can be a considerable delay in information transfer, which can eventually lead to numerical instabilities, as has sometimes been observed in the discussion group, especially for long tunnel geometries.

One way to get rid of this problem could be to distribute at least basic information globally across all meshes. And exactly this approach is followed by the alternative pressure solvers, which are described in more detail in the Section 3.

2.3 *Direct versus iterative solver*

There are two fundamentally different classes of Poisson solvers which distinguish widely in their underlying algorithmic approach and have various representatives each:

- **Direct solvers** compute the solution of the system of equations in only ONE computational cycle, which may be very complex but which finally leads to the exact solution up to machine precision. Usually they are based on highly recursive algorithms that strongly couple the total data of the whole domain.

- **Iterative solvers** perform multiple computational cycles producing a sequence of approximations which gradually improve an initial estimate of the solution until a specified tolerance has been reached. The computational complexity associated with each single cycle is comprehensively less compared to direct methods.

The difference between direct and iterative solvers is shown in Figure (10), where both classes are represented as a kind of black-box solver. As input the respective solver receives the Poisson matrix A and the right-hand side vector b. After either a single big computational cycle or several smaller ones, it outputs the solution vector x.

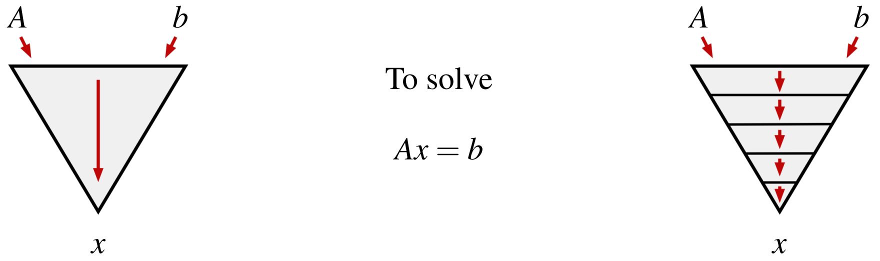


Figure 10: Solution of system $Ax = b$ with either a direct solver, using one large computational cycle (left), or an iterative solver, using many smaller computational cycles (right)

Many direct representatives belong to the frameworks of spectral solvers, like the *Fast Fourier Transformation*, or of the Gaussian elimination, like the *LU-decomposition*, as they are both available in FDS and will be presented in more detail in Section 3.

For single-mesh applications their highly recursive design principle excellently reflects the globally averaging character of the Laplacian operator as described in Section 1. In contrast, for multi-mesh applications the subdivision into single meshes causes a fragmentation of its physical connectivity which leads to dependencies on the number of meshes (due to the need of global data exchanges) and possible deteriorations of the resulting efficiency and/or accuracy.

Many iterative representatives belong to the classes of *Conjugate Gradient* oder *Multigrid* methods as they are implemented within the SCARC solver class (**Sca***lable* **R***ecursive* **C***lustering*), see [2, 3, 4], and are available as experimental solvers in FDS. They are often easier to implement than direct ones, because they can be reduced to a series of core components such as matrix-vector multiplications, linear-combinations and scalar-products of vectors. A great advantage of these methods is that they naturally allow the incorporation of domain decomposition methods and are therefore easier to parallelize than direct ones. But their speed of convergence may be very different depending on specific iteration parameters and the number of meshes, too. For a detailed overview of iterative methods see for example [5, 6]. Now, the decisive question is how many cycles are needed for convergence and how the finally required sum of small cycles computationally compares to the single complex cycle of a direct method.

3 SOLUTION STRATEGIES

Subsequently, possible solution strategies for the pressure equation (1) will be presented. This concerns both the solvers currently available in FDS and experimental alternatives currently being developed within the solver class SCARC. The focus here is not on a detailed mathematical description, but rather on illustrating the underlying algorithmic principles in a visual manner and increasing awareness of possible difficulties in connection with parallelization.

3.1 Fast Fourier Transformation

The current default pressure solver in FDS is based on the mesh-wise use of *Fast Fourier Transformation* (FFT) methods which belong to the class of direct solvers. To give an idea of how this method basically works, a deeper understanding of the terms *eigenvectors* and *eigenvalues* of a linear transformation $Ax = b$ as defined by the discrete Poisson equation in (5) is very useful. Such eigenvectors/-values are very special quantities: An eigenvector v doesn't change its direction when the transformation A is applied to it (except of perhaps precisely reversing its direction), but is only stretched by a scalar value, namely its respective eigenvalue λ, such that $Av = \lambda v$, as illustrated in Figure (11).

The real importance of these quantities is that they allow to reduce a given linear transformation to its bare essentials. If it is known what happens to each individual eigenvector under the transformation, then it is also known what happens to any other vector because every vector can be expressed as a linear combination of these eigenvectors. Thus, if all eigenvectors v_i of a given transformation matrix A are known, the solution to system $Ax = b$ can be represented as linear combination of the transformed eigenvectors Av_i. Or in other words, the actual solution which is very complicated on its own, can be expressed as combination of much simpler things which are perfectly understood.

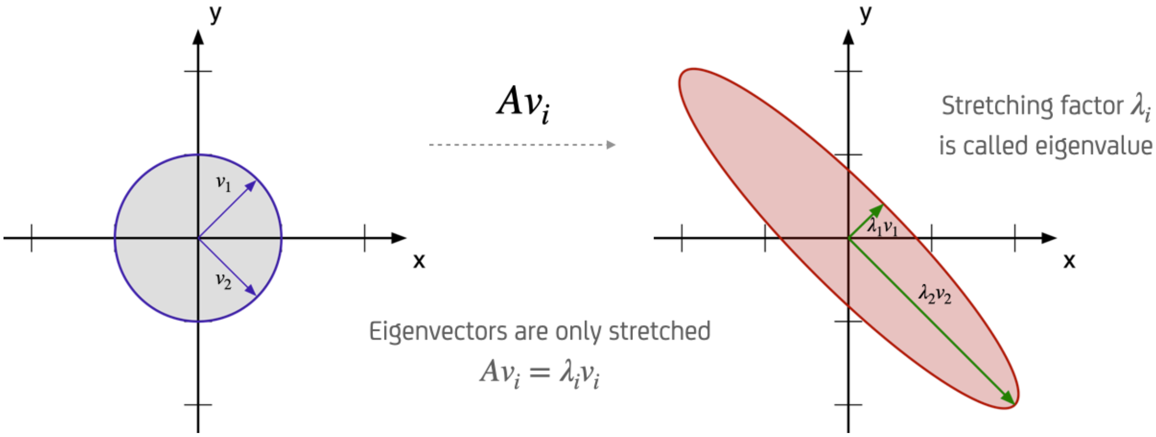


Figure 11: Eigenvectors and -values of a linear transformation

Usually the eigenvalues of a transformation are not known or it is extremely costly to determine them. This holds especially true for irregular, unstructured grids. For FDS, however, the situation is different. Remember that in the derivation of the pressure equation a special decomposition of the perturbation pressure $\tilde{p}$ into two parts was used, of which only one was included in the definition of the Poisson matrix A. If this splitting had not been done, then A would contain variable entries that would change at each time step in relation to the changing density values over time. In this case, a new Poisson matrix would have to be generated in each time step which would involve an enormous amount of work. But thanks to this special splitting the resulting Poisson matrix has constant entries that do not change during a simulation.

This allows a local FFT to be performed on each mesh to expand the solution as a linear combination according to these eigenvectors. The local FFT-solutions are based on the highly optimized solver package CRAYFISHPAK [7], which is extremely fast in terms of computing time. The local solutions are then coupled together by local data exchanges to finally provide an overall solution as illustrated in Figure (12). For a multitude of constellations this strategy has proven to be highly efficient and robust.

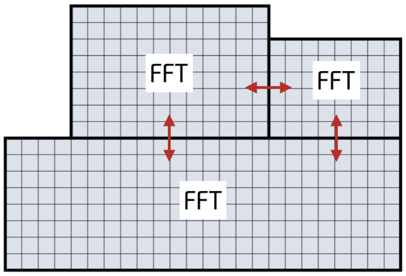


Figure 12: Execution of optimized local FFT methods on the structured meshes coupled by local data exchanges

To ensure the availability of the local eigenvectors, this FFT-based solver can only be used for structured meshes. This may lead to a kind of penetration of the velocity field into the internal obstructions, i.e. to non-zero normal velocity components towards them as described above.

In addition, although this purely locally oriented approach produces an overall Poisson solution that is continuous along inner mesh interfaces, it cannot guarantee that the normal derivatives of velocity will also match there. And finally, due to its purely local character this strategy may experience difficulties to reproduce the strong overall coupling of the pressure equation which may especially hold true in case of big geometries with many meshes and/or transient boundary conditions with frequently changing global information.

In order to compensate these potential errors an additional iterative Direct Forcing Immersed Boundary Method [8] is used in combination with the local FFT methods within FDS. In every time step this method requires the repeated mesh-wise FFT-solution of the local Poisson problems in order to achieve an iterative correction until a specified tolerance for the velocity errors along internal objects and at mesh interfaces has been reached. In the worst case this procedure may converge slowly along with a comprehensible increase of computing time. In this sense, the FFT solver in FDS is not a pure direct method, but a combination of local direct methods with a global iterative correction. For more details see again the FDS Technical Reference Guide [1].

3.2 LU-decomposition

Within the framework of the well-known Gaussian elimination algorithm, many direct algorithms are based on the *LU-factorization* of the system matrix. In this process, A is divided into the product of a lower left and an upper right triangular matrix, L and U, as sketched in Figure (13). Because of the symmetry of the underlying problem, it holds $U = L^T$ here, such that only one triangular matrix must be stored.

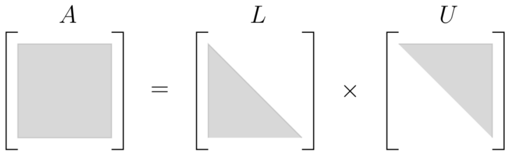


Figure 13: Decomposition of the Poisson matrix A into the product of a lower triangular matrix L and an upper triangular matrix U

Again, it proves to be very useful that matrix A has constant entries which do not change during the whole simulation such that the very complex factorization process must only be performed a single time during the initialization phase. Once the decomposition is available, the solution of the individual Poisson problems in the different time steps can simply be achieved by back substitution exploiting the triangular structures of the matrices L and U.

In FDS an optimized LU-solver being part of the Intel Math Kernel Library (MKL) is used to perform the respective decomposition and back substitution processes. A big advantage of this LU-strategy is that it also works on unstructured grids, such that the correct boundary conditions can be incorporated along internal obstructions. Depending on whether this solver is used on a structured or unstructured discretization, it is called GLMAT or UGLMAT. Because both versions are actually based on a mesh-spanning LU-decomposition of the global Poisson equation, the normal derivatives of the velocity field along mesh interfaces are correct. Due to its correct handling of internal obstructions, UGLMAT offers the exact solution to the global, unstructured Poisson problem up to machine precision.

But a main disadvantage of these LU-based solvers is that less benefit can be drawn from a very convenient property of the Poisson matrix, namely its intrinsic *sparsity*: Even though A has only very few non-zero entries, the factorization process leads to *fill-in*, i.e. produces many non-zero entries in L and U, where A was zero before. It therefore requires a large amount of additional memory and may be impossible in the worst case due to limited memory resources in case of large problems with a high number of grid cells.

Although this globally oriented strategy very well reflects the strongly recursive overall character of the Poisson equation, its parallel version also requires global data exchanges to generate a global solution. And even if this is done using the optimized Intel MKL library, these LU-based solvers are therefore usually slower than the locally oriented FFT.

3.3 Conjugate Gradient method

Conjugate gradient (CG) methods belong to the class of iterative solvers. They do not aim to solve the system $Ax = b$ itself, but take advantage of the fact that its solution x is also the minimum value of the respective quadratic form $Q(x) = 1/2x^T Ax - x^T b$. This is due to $Q'(x) = Ax - b$, such that x is a critical point of $Q(x)$.

As illustrated in Figure (14), CG methods are mainly based on the use of *gradient descent* techniques which iteratively define a sequence of vectors moving down in the direction of steepest descent (as defined by the negative gradient) to finally find the minimum of the quadratic form Q. By applying mutually conjugate directions these search directions are continually improved and the minimum is obtained in at most n steps. They are restricted to symmetric positive-definite problems and only need less storage for several auxiliary vectors.

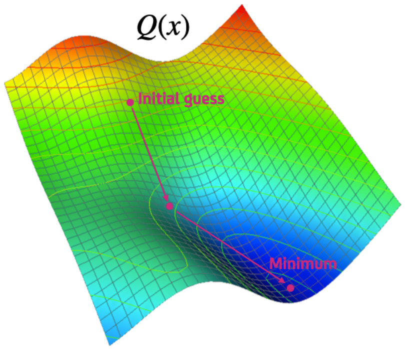


Figure 14: Minimization in the CG method

CG methods are largely based on matrix-vector multiplications whose parallel calculation only requires computationally cheap data exchanges between neighboring meshes. However, they also include global scalar products requiring global data exchanges which are computationally more expensive but also contribute to a stronger overall coupling.

A parallel CG solver is available as experimental FDS pressure solver within ScaRC. Its main advantage is that it can be applied for both structured and unstructured discretizations. In the latter case, the correct treatment of internal obstructions is thus possible. Besides it really solves the global mesh-spanning Poisson problem such that the transitions of the velocity field across mesh interfaces are correct, too.

This CG-based pressure solver has proven to be robust and accurate for many different test cases, but its main disadvantage is that it needs a comprehensible amount of iterations involving local and global data exchanges each. Thus, it is relatively slow in terms of computational speed and cannot compete with the FFT for standard problems that do not present significant problems of delayed global data transfer.

Nevertheless, it can be combined with an additional coarse grid solution in order to capture the global transfer of information faster, but now on the basis of a smaller (and thus computationally less expensive) number of grid cells. It has turned out that this combination may lead to a comprehensible reduction of the number of iterations which is currently being analyzed. In any case there is good reason to hope that further optimizations of the computing time are still possible.

3.4 Multigrid methods

Multigrid methods belong to the most powerful iterative solvers for the Poisson equation and are widely used in many fields of science. They do not only use the original fine grid, but instead a complete hierarchy of grids with increasingly coarser degrees of resolution. All levels work together in a well tuned choreography to finally produce a common global solution. The combination of approximate solutions on the different refinement levels with an exact solution on the coarsest level provides a much stronger global coupling which usually results in very good convergence rates as could be seen for many test cases. Two big classes of multigrid methods are distinguished which are both implemented as experimental pressure solvers within SCARC.

The *geometric multigrid methods* (GMG) produce the coarser grid levels by simply doubling the mesh size. This is relatively straightforward to implement in terms of the underlying data structures, but has the disadvantage that the number of cells in each spatial dimension must be divisible by 2 at least once, or better more often, in order to make this doubling possible once or several times. In addition, obstructions that can be perfectly displayed on the finest grid level, may be too small to be displayed on the coarser levels. Within SCARC this GMG variant is therefore only available for structured meshes.

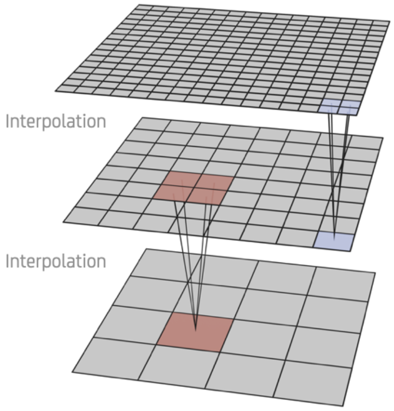


Figure 15: Geometric multigrid based on doubling the mesh size

In contrast to this, the coarsening process within the *algebraic multigrid methods* (AMG) is not geometrically based, but it only uses purely algebraic information which is included in the matrix A. There are many variants of this class that differ in the interpretation of this algebraic information. Within SCARC, a special variant was recently implemented which is based on so-called *smoothed aggregation*. This variant proceeds in such a way that individual cells from the finer grid are clustered together to form a so-called *aggregate*, which then represents a cell on the coarser grid.

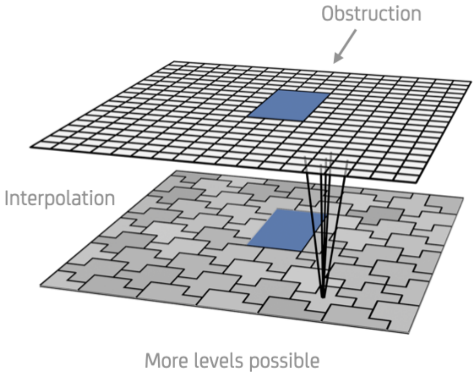


Figure 16: Algebraic multigrid based on smoothed aggregation

The shape of the aggregate is most closely oriented to the shape of the underlying matrix stencil in order to ensure a high density of information transfer in the interpolation between the grid levels. Another smoothing operation is performed to further improve the quality of the interpolation which ultimately gives the method its name.

A major advantage of the AMG approach is that obstructions of any size can be treated and correctly displayed at the coarser levels, so that unstructured discretizations are applicable. Furthermore, it works for both even and odd cell counts and is therefore much more flexible than the GMG approach.

Figure (17) shows the first coarsening level for a 3D geometry with a large internal obstruction. The fine grid levels consists of 27 cells per edge. Obviously the coarsening process is capable of correctly displaying the obstruction and internal boundary conditions can be applied there. In accordance with this, the shapes of the remaining aggregates are automatically adapted to the shape of the 7-point stencil in the best possible way.

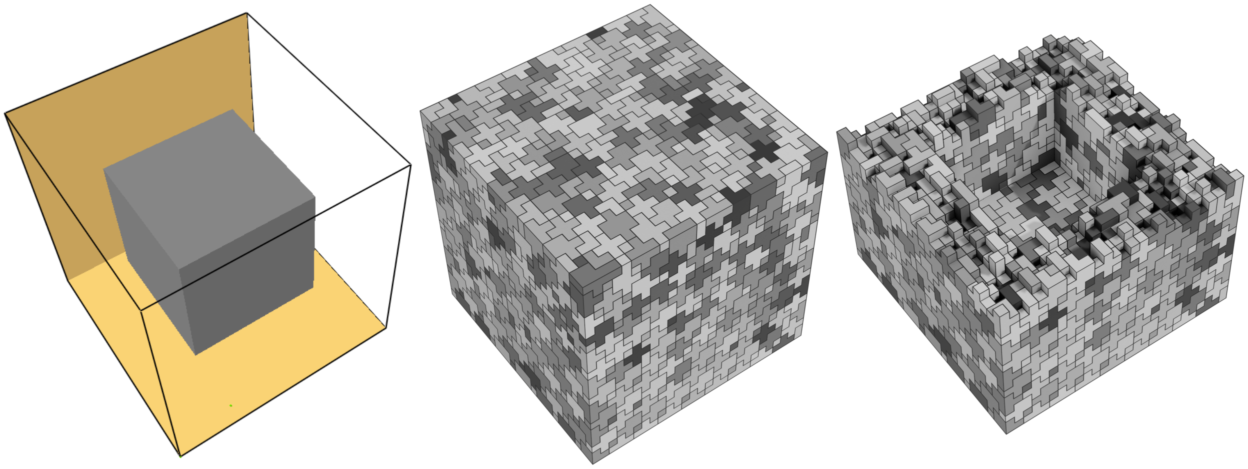


Figure 17: 3D geometry with internal obstruction (left) with corresponding first AMG-coarsened grid level as a whole (middle) and with a look inside (right)

With regard to its numerical convergence rate, the upper AMG variant is the most efficient representative within the whole ScaRC class so far, because only a very small number of iterations for the global Poisson solution is usually needed.

As already noted at the end of Section 3.3, a grid level coarsened with AMG techniques can also be used as a coarse grid within a 2-level CG method to achieve a stronger global coupling with a corresponding improvement of the convergence rate.

Both GMG and AMG manage to perfectly compute the correct normal components of the velocity field along the mesh interfaces because they solve the global Poisson problem. But surely, the gain in numerical efficiency (in the sense of significantly accelerated convergence rates) must be set in relation to the computational effort required for this (more global data exchanges and handling of more complex data structures). At present, no final evaluation of the computational efficiency of the multigrid solvers is available. However, it is currently being analysed on the basis of extensive test series in order to finally arrive at a comprehensive rating.

3.5 *McKenny-Greengard-Mayo method*

As explained above, CG and AMG can be used for both structured and unstructured discretizations. In both cases, as global Poisson solvers, they succeed to calculate the correct velocity field along the mesh interfaces. Nevertheless, it often turns out that faster convergence rates can be achieved with structured discretizations than with unstructured ones. On the other hand, using a structured discretization can lead to corresponding errors along internal obstructions. So if another way could be found to ensure the correct handling of internal obstructions, the use of a global structured solver would be highly desirable.

The *McKenney-Greengard-Mayo* method (MGM) follows such an approach. As sketched in Figure (18) it splits the solution of the Poisson equation into two subsequent passes. The first pass consists of a global Poisson solution defined on the structured discretization (i.e. ignoring the obstruction), which satisfies the correct external inhomogeneous boundary conditions. In a second pass a corrective global Poisson solution on the associated unstructured discretization is performed, which satisfies the correct homogeneous no-flux boundary conditions along internal obstructions, but has zero values at the external boundary. It is important that both passes accurately calculate the velocity field along the mesh interfaces. Adding them together gives the solution of the original global unstructured Poisson problem, which is now correct both at the internal obstructions and at the mesh interfaces.

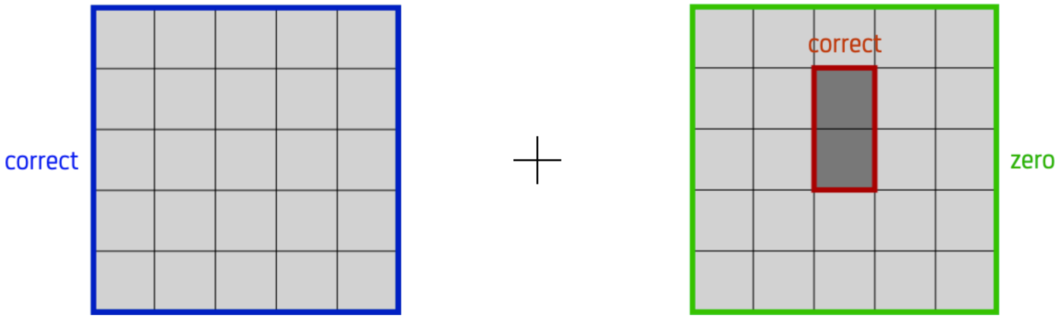
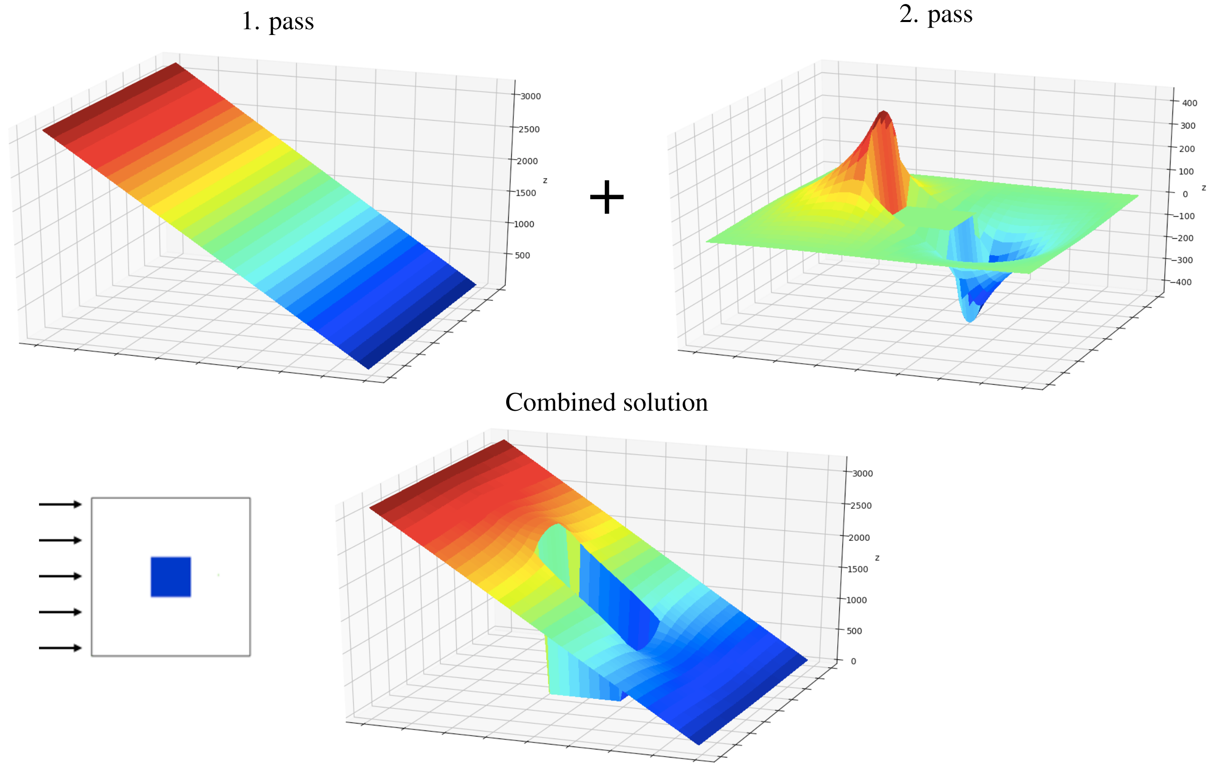


Figure 18: McKenney-Greengard-Mayo method for the global Poisson problem consisting of an inhomogeneous structured part (left) combined with an homogeneous unstructured part (right)

A major advantage is that the first pass allows to achieve faster convergence rates due to the structured nature of the discretization. For the second pass it is expected, that optimizations can be made, which take advantage of the fact, that the corresponding right hand side has relatively few non-zero entries (namely only along the internal obstructions). This method is also currently being tested and further evaluations regarding its computational efficiency will follow. Of particular interest is whether the staggered solution of two supposedly simpler problems is faster than the single solution of the global unstructured Poisson problem itself.

Figure (19) shows a simple square-shaped 2D case with an internal obstruction. While the first pass on the left doesn't know the obstruction, but satisfies the correct external boundary conditions, the second pass contains the respective correction for the internal obstruction but without contributing to the external boundary any more. Combining both gives the right solution. Note that the second pass is displayed in a more detailed range to better illustrate its special shape.



1. pass

2. pass

Combined solution

Figure 19: McKenney-Greengard-Mayo method for a simple 2D domain with obstruction

4 OUTLINE

Now that the implementation of the algorithmic components of the above-mentioned iterative solvers in SCARC is largely completed, a multitude of different strategies for solving the FDS pressure equation is available. A positive side effect is that individual elements from this kit can be combined in various ways, even including the computationally powerful components of the standard direct solvers. For example, the preconditioning of the CG method in its structured variant can be based on local FFT methods, in its unstructured variant on local LU decompositions. Furthermore, the coarse grids generated via multigrid techniques can be used to generate a 2-level CG method with a respective faster convergence rate. Even a complete multigrid method can be used as a preconditioner within the CG method. Likewise, different strategies can be combined for the two partial solutions of the MGM method.

Thus, the objective is to profitably merge the good properties of the individual strategies and to identify suitable combinations for certain problem classes. Of course, the increased computational effort for these variations is only justified if they can solve cases that would otherwise lead to inaccurate results or even longer computation times using the standard methods. In order to get a comprehensive overview, various test series are currently being carried out, which are particularly focused on various critical problem classes (such as e.g. long tunnel simulations). The results will be reported in due course.

References

[1] K. McGrattan, S. Hostikka, R. McDermott, J. Floyd, C. Weinschenk, and K. Overholt. Fire Dynamics Simulator, Technical Reference Guide. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, and VTT Technical Research Centre of Finland, Espoo, Finland, sixth edition, September 2013. Vol. 1: Mathematical Model; Vol. 2: Verification Guide; Vol. 3: Validation Guide; Vol. 4: Software Quality Assurance. 2, 5, 12

[2] Susanne Kilian and Stefan Turek. An example for parallel scarc and its application to the incompressible navier-stokes equations, 1997. 10

[3] Susanne Kilian. ScaRC: Ein verallgemeinertes Gebietszerlegungs-Mehrgitterkonzept auf Parallelrechnern. PhD thesis, Universität Dortmund, Berlin, 2001. 10

[4] Dominik Göddeke. Fast and accurate finite-element multigrid solvers for PDE simulations on GPU clusters. PhD thesis, TU Dortmund, May 2010. 10

[5] Y. Saad. Iterative methods for sparse linear systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003. 10

[6] W. Hackbusch. Iterative solution of large sparse systems of equations, volume 95 of Applied Mathematical Sciences. Springer-Verlag, New York, 1994. Translated and revised from the 1991 German original. 10

[7] Green Mountain Software, Boulder, Colorado. CRAYFISHPAK User's Guide, Cray Version 1.1, 1990. 11

[8] E.A. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof. Combined Immersed-Boundary Finite-Difference Methods for Three-Dimensional Complex Flow Simulations. Journal of Computational Physics, 161:35–60, 2000. 12