Bachelor Thesis
Applied Mathematics

SKEW PARTITIONING FOR THE HYBRID MULTILEVEL SOLVER

Mark van der Klok
s2324903

University of Groningen
July 13, 2017

First assessor: Dr. ir. F.W. Wubs
Second assessor: Prof. dr. M.K. Camlibel
Daily supervisor: Ir. S. Baars

Abstract

This bachelor thesis constitutes a research of skew partitioning in three dimensions for the hybrid multilevel solver (HYMLS), a parallel solver developed at the University of Groningen to solve incompressible (Navier-) Stokes problems. The separators in standard Cartesian partitioning isolate pressure nodes which possibly affects the convergence properties of HYMLS, which is why skew partitioning is considered as an alternative. After a brief overview of the HYMLS method, the currently employed partitioning mechanisms – 2D Cartesian, 3D Cartesian and 2D skew partitioning – are studied using a three-step approach that consists of (1) building a template, (2) finding the groups for a general domain and (3) distributing the results over the grid. This lead to insights on how to generalize the 2D situation to the 3D case and how to perform the transition from Cartesian to skew partitioning. Using these insights, domain shape candidates were identified for 3D skew partitioning. A prism shape was finally shown to yield a consistent partitioning without the presence of isolated pressure nodes.

Contents

1 Introduction

1.1 The solver

The Hybrid Multilevel Solver (HYMLS) is developed at the University of Groningen to solve the incompressible Navier-Stokes (NS) equations on parallel computers [1–4]. The method shows a grid-independent convergence rate and does not break down at high Reynolds numbers. The incompressible[1] NS equations describe the relations between velocity $\mathbf{u} = (u, v, w)$ and pressure p of a fluid with density ρ and viscosity μ and subject to external forces $\mathbf{g}$. They can be stated as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 \mathbf{u} + \mathbf{g}, \tag{1.1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{1.2}$$

where Equation (1.1) describes conservation of momentum and Equation (1.2) describes mass conservation under the incompressibility condition. HYMLS uses a combination of direct and iterative techniques to solve the NS equations numerically. The problem is usually discretized on an Arakawa C-grid, where for each grid cell the velocities are defined at the center of cell faces and the pressure at the cell center [5] (see Figure 1.1). This discretization (along with linearization for the nonlinear terms, by e.g. Newton's method) leads to systems of the form

$$\begin{bmatrix} A & B \\ B^T & O \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}, \tag{1.3}$$

where A is positive definite and B is a *gradient type* matrix [6], i.e. it has at most two nonzero entries per row and their sum is zero. The matrix occurring on the left-hand side in Equation (1.3) is called an $\mathscr{F}$-matrix [7]. In addition to the (Navier-) Stokes problems, HYMLS can also be used to solve strongly coupled fluid-transport equations [8], Poisson problems, or other problems in which $\mathscr{F}$-matrices occur, e.g. electrical network simulations. In this thesis, the focus lies on the steady (Navier-) Stokes problems.

[1] Throughout this text, only the incompressible Navier-Stokes equations are considered. Therefore, from here on the adjective *incompressible* will not explicitly be written.

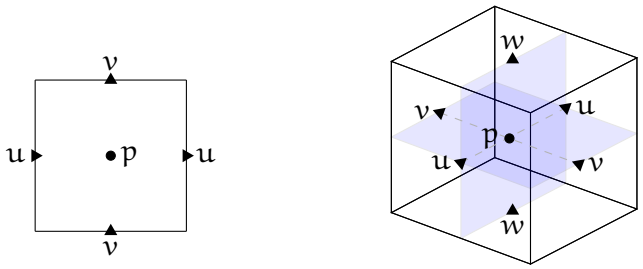


Figure 1.1. Schematics of the cells in the staggered Arakawa C-grid in 2D (left) and 3D (right), indicating the positions of the nodes of the variables u, v, w and p within the grid cells.

1.2 Partitioning and Schur complements

HYMLS is designed to solve the fluid flow problem in parallel, and it does so by partitioning the computational domain into a set of non-overlapping subdomains (the interiors) plus an interface (the separators). After solving the interface problem through the *Schur complement*, the problem is reduced to solving the independent systems associated with the subdomains. This allows for parallel computation. The general idea of partitioning is illustrated in the following example.

Example. For a general large (sparse) problem $A\mathbf{x} = \mathbf{b}$, partitioning into n subdomains amounts to reordering the unknowns to obtain a problem of the form

$$\begin{bmatrix} A_{11} & & & A_{1S} \\ & \ddots & & \vdots \\ & & A_{nn} & A_{nS} \\ A_{1S}^T & \dots & A_{nS}^T & A_{SS} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \\ \mathbf{x}_S \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \\ \mathbf{b}_S \end{bmatrix}.$$

The diagonal blocks A_{ii} (for $i = 1, \dots, n$) correspond to the interiors of the n subdomains (i.e. variables $\mathbf{x}_i$) and the blocks A_{iS} represent the couplings of these interior variables to the separator variables x_S. If for brevity the block diagonal matrix containing the interior blocks is contracted into a single block A_{II} and their couplings to the separators are contracted into A_{IS}, then elimination of the interior variables formally yields the block LU decomposition

$$\begin{bmatrix} A_{II} & A_{IS} \\ A_{IS}^T & A_{SS} \end{bmatrix} = \begin{bmatrix} I & O \\ A_{IS}^T A_{II}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{II} & A_{IS} \\ O & S \end{bmatrix}$$

where

$$S = A_{SS} - A_{IS}^T A_{II}^{-1} A_{IS} = A_{SS} - \sum_{i=1}^{n} A_{iS}^T A_{ii}^{-1} A_{iS}$$

is the Schur complement of A_{II}. It should be noted that wherever an inverse of a matrix is written, a solver should be employed rather than explicitly computing the matrix

inverse. From this LU decomposition it follows that the partitioned system can now be solved by first solving

$$Sx_S = b_S - \sum_{i=1}^{n} A_{iS}^T A_{ii}^{-1} b_i$$

followed by solving n independent systems

$$A_{ii}x_i = b_i - A_{iS}x_S, \qquad \forall i = 1, \ldots, n,$$

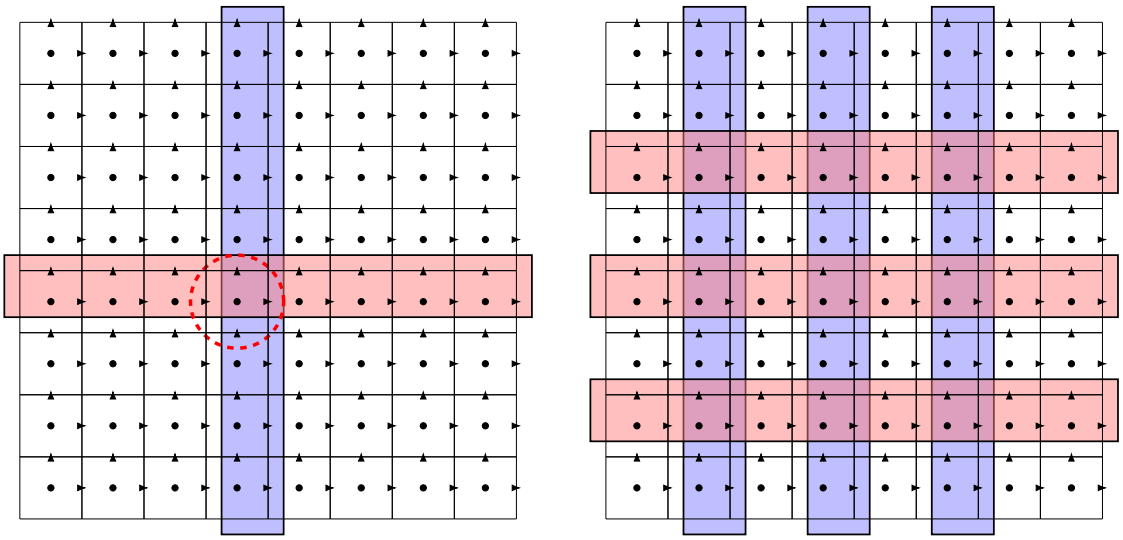
which can be done in parallel. ■

Partitioning of a linear system can be done based on the graph of the associated matrix. It is also possible to partition the grid geometrically, i.e. by subdividing the grid manually into equally sized subdomains. It is this latter case that is considered in this project. The partitioning can be thought of as if the grid is subdivided by placing lines (planes) over the velocity nodes in the 2D (3D) grid to define the separators (see e.g. Figures 1.2 and 1.3). All of the velocities on these separators will be retained in the Schur complement, along with for each subdomain (ideally) one pressure node from the interior, usually the first one. This pressure node is retained to prevent singularities in the Schur complement problem (i.e to fix the pressure inside the subdomain). As will be discussed in the next section, it may occur that additional pressure nodes have to be retained in the Schur complement, which is undesired since it leads to complicated code due to the many exceptions that have to be implemented. This complicated code is prone to bugs which may affect the convergence properties of the HYMLS program, even though the method is theoretically sound.

1.3 Geometric partitioning: standard versus skew

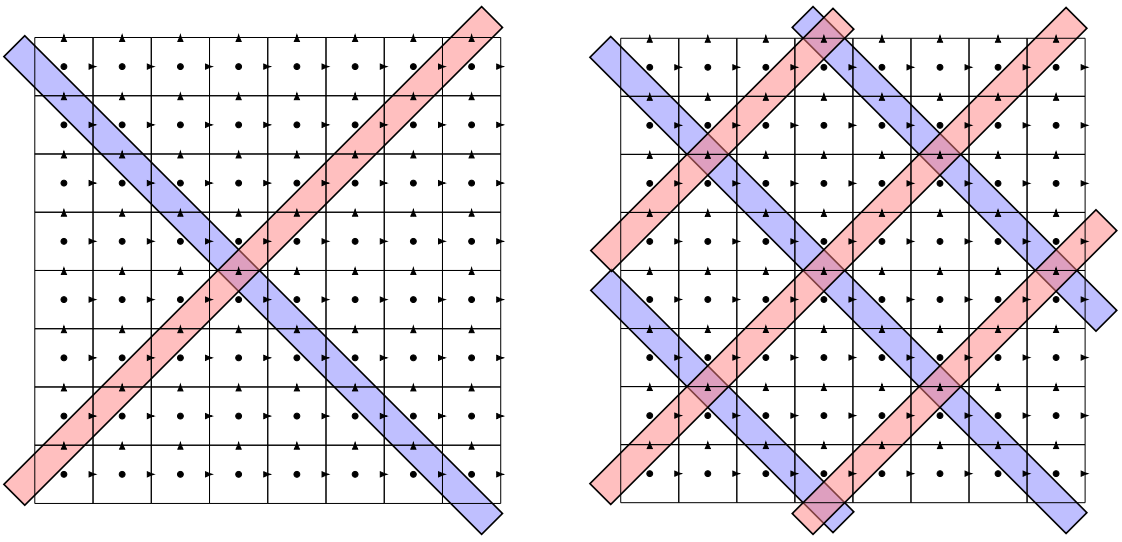
Arguably the most straightforward way of partitioning the grid geometrically, is to use *standard* Cartesian partitioning. Here the domain is partitioned into equally sized square subdomains (square in the number of unknowns), by placing equidistant horizontal and vertical subdomain separators on the grid. Two examples for the 2D case are shown in Figure 1.2. Here the separators are lines that intersect in points. The 3D case is similar, but then separators are planes intersecting in lines (where four subdomains meet) and in points (where eight subdomains meet). In Figure 1.2, an 8×8 grid is partitioned into four subdomains, each of size 4×4. The separators which separate the subdomains from each other are highlighted in red (horizontal separators) and blue (vertical separators). The separators consist of u and v nodes, which subsequently are grouped together based the set of subdomains they connect to. In other words, separator lines and points will belong to different groups. The nodes that do not belong to any separators will be said to belong to the interior of the subdomain. The reasons for grouping the nodes like this will become clear in the following chapter.

It can be observed in Figure 1.2 that the intersection of horizontal and vertical separators leads to isolated pressure nodes, meaning that all of its surrounding velocities



(a) Separator length 4. **(b)** Separator length 2.

Figure 1.2. Example of Cartesian partitioning of an 8×8 grid with different separator lengths. Horizontal separators are highlighted in red, vertical separators are highlighted in blue. The intersection of separators leads to the isolation of pressure nodes, as highligted in (a).



(a) Separator length $4\sqrt{2}$. **(b)** Separator length $2\sqrt{2}$.

Figure 1.3. Skew partitioning for an 8×8 grid with different separator lengths. By placing the separators diagonally on the grid, the intersections no longer isolate a pressure node, but domains along the grid boundary are asymmetric.

are separator velocities. This cell is called a *full conservation cell*. This pressure node can not be eliminated but instead should be retained in the Schur complement as it would otherwise lead to a singular matrix for the interior of the corresponding subdomain [4]. In addition to keeping this pressure node in the Schur complement (thereby placing it in a separate group), the four surrounding velocity nodes that are also in the full conservation cell should be placed in separate groups as well. (This ensures that these nodes are retained throughout the multilevel approach.) This workaround is, as turns out, unnecessarily complicated: it is better to prevent the occurrence of isolated pressure nodes at all. In three dimensions it gets even more complicated, since then separators become planes instead of lines, so that one could potentially get entire "tubes" of isolated pressure nodes at positions where four subdomains meet, and further complications can be imagined in cells where eight subdomains meet.

It has turned out that the treatment of full conservation cells is (unnecessarily) complicated and prone to bugs and errors in the HYMLS code. It may even affect the convergence properties of the method and therefore it may be better to partition the grid in such a way as to prevent any occurrences of isolated pressure nodes. In particular, it was proposed to use *skew* Cartesian separators, which are obtained by rotating the separators in standard Cartesian partitioning by 45 degrees. For the 2D case, this is shown in Figure 1.3. This figure shows that there are no isolated pressure nodes if the separators are placed diagonally on the grid. This means that there is no specific pressure node in the interior of the subdomains (except for the first one) that needs to be retained in the Schur complement and the interior can be eliminated in the regular way.

1.4 The goal of this project

Partitioning using skew separators for the 2D case has already been implemented and tested in the HYMLS code. The next step is to also implement skew separators in 3D. However, in 3D it gets hard to imagine/visualize what is happening. The fact that there are now three velocity components to form entire separator planes makes it complicated to construct such partitions without isolating pressure nodes. So, in particular intersections of separator planes should be carefully considered.

The main goal of this bachelor project is to visualize and analyze the possibility of skew domain partitioning in 3D. This starts by identifying candidate domain shapes. MATLAB will be used to write a program that for a given grid size and separator length, performs a skew partitioning on the 3D grid to obtain skew subdomains. The program should thus determine for each subdomain which nodes it contains. More precisely, it should give back a list of node indices of the interior nodes and of the nodes within the different separator groups that surround it. The final delivery will be MATLAB code which can be easily converted into C++ code to be used in HYMLS.

1.5 Overview of this thesis

This problem will be tackled step by step, which also defines the structure of this thesis. First, the HYMLS method is studied briefly in Chapter 2. Then, Chapter 3 contains a review on the standard Cartesian partitioning mechanisms, in two and three dimensions. This provides insight and a general systematic three-step approach to the problem of skew partitioning. In Chapter 4, the problem of skew partitioning is described, again using the three-step approach. The report ends with a discussion and some concluding remarks.

2 About HYMLS

The HYMLS method is based on a direct method developed for the Stokes $\mathscr{F}$-matrix [6]. This direct method maintains the $\mathscr{F}$-matrix properties of the equations and so preserving the structure of the equations. In particular, this method has the property that it introduces no additional fill in the B part of the matrix of Equation (1.3). To extend this method to an incomplete factorization, orthogonal transformations are used in combination with dropping of (only) velocity-velocity couplings. The resulting method, HYMLS, iterates in the kernel of B^T, meaning that it satisfies the incompressibility constraint in (the discrete version of) Equation (1.2) throughout the iterations. Furthermore, it should lead to a grid-independent convergence rate. A brief description of the various steps that are performed in the HYMLS method is given below, in order to exemplify why partitioning is required and how it is used HYMLS.

The domain decomposition. The first step is to split the computational domain into a number of subdomains without overlap. As has been discussed in the Chapter 1, this partitioning is based on the velocity nodes and can be done in various ways. For geometrical partitioning skew partitioning is preferable over standard Cartesian partitioning. The partitioning is done based on the separator length as the input parameter. In the case of skew partitioning, one can equivalently base the partitioning on the "cube length", i.e. the side length of the smallest cube which contains one subdomain. This has the advantage that one can define the separators with integer numbers in a straightforward way. The cube length can also be interpreted as the repeat distance of the separators. The terms "separator length" and "cube length" will be used interchangeably in this thesis.

The partitioning results in two sets of unknowns, the interiors and the separators. Separators are defined as the nodes which connect to more than one subdomain. The remaining nodes are said to belong to the interiors of the subdomain. In particular, all pressure nodes are inside the domain interiors as the partitioning is rather based on the velocity nodes. The domain the pressure nodes belong to is the one that contains the majority of its coupled velocity nodes [2].

The separator groups and retained variables. The separators defined by the domain decomposition are subdivided into separator groups. This particular grouping is done

in order for the multilevel approach to work. A separator group consists of all velocity nodes of the same type (so all u, v or w components) that connect to the same set of subdomains. So, an edge between two (four) domains in 2D (3D) would form a group, and so would a point between four (eight) subdomains, *et cetera*.

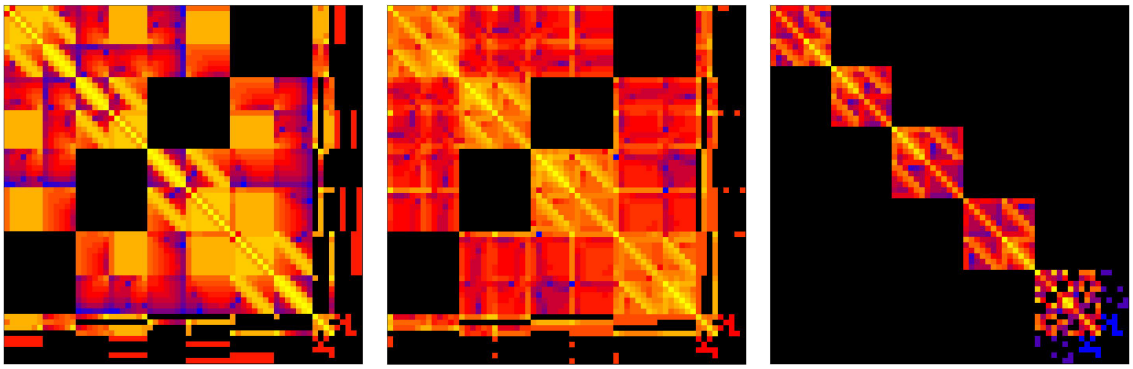
Each separator group contains information that has to be transferred to the next level in the multilevel approach. Additional information has to be retrieved from an arbitrary pressure node for each subdomain, which is also taken out of the interior and put into a separate 'group'. In the approaches described in this thesis, this pressure node is chosen to be the first pressure node in each subdomain interior. Furthermore, if there are isolated pressure nodes, this node as well as each of its surrounding velocities are put into separate groups as well in order to prevent singularities in the Schur complement. For the two-dimensional case, Figures 3.2b and 4.2b show the different groups for the standard Cartesian and skew Cartesian cases respectively.

The Schur complement. The remainder of interiors is eliminated by a direct method as described in [6]. A fill reducing order (such as approximate minimum degree [9]) is used on the fill pattern $F(A) \cup F(B^T B)$ and elimination is performed dynamically: whenever a velocity node that is coupled to a pressure node is to be eliminated, they are eliminated together using a 2×2 pivot. This ensures that the numerical and structural properties of the initial system are preserved throughout the elimination. In particular, it ensures that the Schur complement resulting from the elimination is again an $\mathscr{F}$-matrix. Therefore, it makes sense to talk about the A or B part in the Schur complement, cf. Equation (1.3).

The multilevel preconditioner. After elimination of the interior variables, the problem is reduced to first solving the Schur problem of the for $Sx_s = y_s$ after which the interior variables can be solved for independently.

An example of a Schur complement for standard partitioning of a 16×16 grid with separator length 8 is shown in Figure 2.1a. In the B part (i.e. the last few columns) the separators cause occurrence of dense columns, at most two per separator group. These correspond to the coupling of each separator to the retained pressures of the connecting subdomains. In order to prevent fill-in, the velocities on these separators should be decoupled as much as possible from the pressure nodes [2]. This can be done by applying orthogonal similarity transformations. These transformations should preserve the structural properties of the matrix. A suitable choice is the Householder transformation. When applied to the dense columns (and dense rows in B^T part) these columns can be reduced to only one nonzero entry (Figure 2.1b). In other words, *only one velocity node per separator group remains coupled to the pressures inside the subdomains on either side of the separator*. (This is, in essence, why the separator variables are grouped together in the first place.) The velocity nodes that are still coupled to pressure nodes are called V_Σ nodes and can be interpreted as the summed velocity through the separators.

After the similarity transformation, all V_Σ nodes are shifted to the end of the matrix (so that they will be retained in the next level Schur complement). Then, all couplings

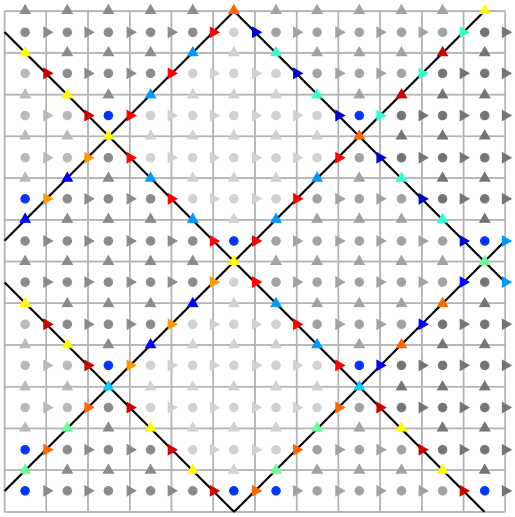


(a) The Schur complement. **(b)** After Householder transformations. **(c)** After shifting and dropping.

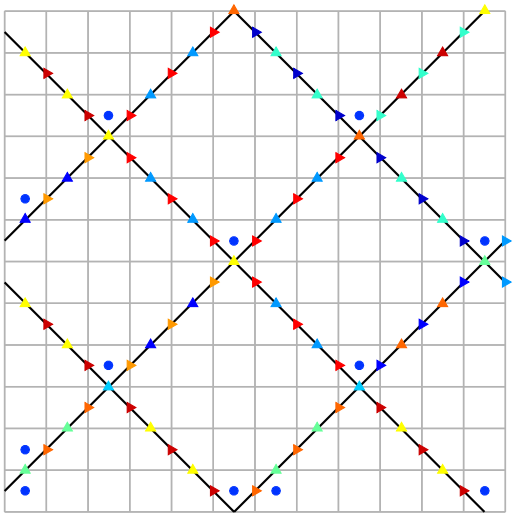
Figure 2.1. Example of Schur complement and the transform-and-drop strategy for standard partitioning of a 16×16 grid with separator length 8. After a Householder transformation on the dense columns of the B part, the V_Σ nodes are shifted to the end. Dropping then yields a block diagonal matrix with the next level Schur complement in the last block. *Image courtesy of Fred Wubs.*

between non-V_Σ nodes and V_Σ nodes are dropped, as well as all couplings between non-V_Σ nodes in different separator groups. The result (Figure 2.1c) is a block diagonal matrix with the next level Schur complement in the last block on which the process can be repeated. An illustration from the grid point-of-view is shown in Figure 2.2. This particular example shows the principle for a *coarsening factor* of 2, meaning that on each subsequent level the separator length becomes twice as long. Other values for the coarsening factor are possible within HYMLS.

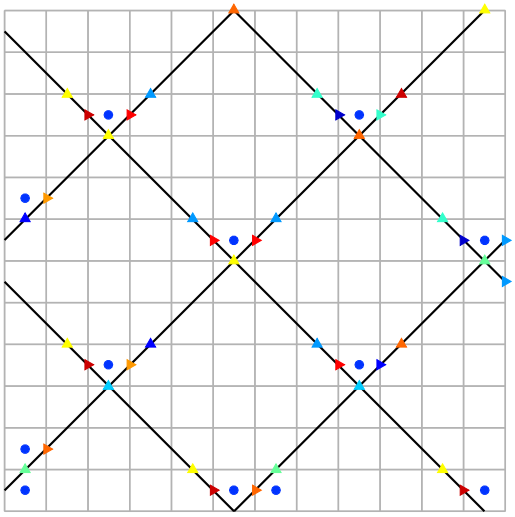
The solution. After construction of the preconditioner for the Schur complement, the problem can be solved by first solving the Schur problem with a preconditioned iterative Krylov subspace method (e.g. GMRES [10]). After this, the interior variables can be solved in parallel from the direct solvers that were computed in the first step.



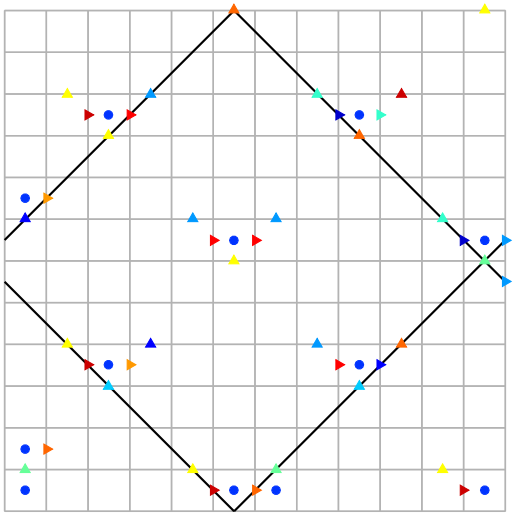
(a) Initial groups.



(b) After elimination of interiors only the separators remain.



(c) The Householder transformations leaves one V_Σ-node per separator group.



(d) Domains on next level.

Figure 2.2. Construction of the preconditioner from the grid point-of-view. This example uses a coarsening factor of 2, i.e. the separators on the next level have twice the length as on the previous level.

3 Cartesian partitioning

In the first implementation of HYMLS the standard Cartesian partitioning was used, where the grid is divided up into rectangular domains. This particular method of domain decomposition is studied in this chapter in order to get insight in the partitioning mechanisms. In particular any observations on the generalization from 2D to 3D could be helpful when the 3D skew partitioning is investigated. A three-step approach is used to tackle the partitioning problem and the results are verified visually.

3.1 The approach

Each of the partitioning mechanisms in this thesis can be described in pure mathematical terms. For example, for two-dimensional, Cartesian partitioning with separator length s, one can run through all grid cells and determine whether its variables u, v, p belong to the separators or not, by doing modulo arithmetic: in both directions, every s'th grid cell contains the separator velocities, except for the cells at the grid boundaries. Similarly, skew partitioning could be described by placing a skew separator every s'th grid cell (also in both directions).

The advantage to do the partitioning in this way is that it can be easily generalized to arbitrary grid sizes, and there are no restrictions on the subdomain sizes. However, in itself it will not fail if the cube length does not divide the grid size even though this would result in partial subdomains. Another disadvantage is that it is only possible to define the separators in this way if the exact shapes of the domains and hence the separator planes are known beforehand. This is obviously not the case for 3D skew partitioning since the shapes of the domains are not (yet) known. Therefore, the approach advocated here is more of a bottom-up approach. Since the exact shapes are not known for the three-dimensional skew case, the approach starts by defining a domain shape and it uses this shape to construct the partitioning for the entire grid. So, this approach can easily be used for subdomains of any shape and size. Furthermore, it reduces the partitioning problem to designing a general domain, and the entire grid information is extracted from this. Therefore the problem can be formulated in terms of the domain shape only. The approach consists of three successive steps, briefly delineated below. In general, the entire process is aided by means of plotting the (intermediate) results in order to (visually) verify the results.

1. Construct a domain template.
The first step is to create a domain template which consists of a general subdomain including all of its surrounding separators. This is the 'design process' for the problem and certainly the most important step. The domain template is basically an array of node numbers that make up the domain. The template can be plotted and the domain shape can be verified visually.

2. Set up a test problem and solve for the different groups.
With the domain template finished, a test problem can be set up. The test problem will be the only occasion where the group solver is employed. A central test domain is set up, and additional neighboring domains are placed around it as they would in the actual grid. The templates thus have some overlap: more precisely, only the separators which are part of the general template will belong to multiple domains in the test problem. The idea is now that all nodes (of the same velocity component) in the central domain that belong to the same set of neighbors will be put into separate separator groups. This generates all of the groups for a domain which is surrounded by a maximum number of possible neighbors in the actual grid.

3. Distribute the result from the test problem over the grid.
Now that the groups around a general domain are known, all that remains is to distribute this result over the actual grid. The results from the test problem should be translated such that they coincide with the actual domain in the grid. Since the templates are actually node numbers, this can be done by simply adding a number to the resulting groups of the test problem. For domains at the grid boundary, additional care should be taken since parts of the template may be outside the grid. The parts outside the grid should be thrown away. Furthermore, separators which exactly coincide with the grid boundaries may have to be put back into the interior. At the end of this step, the results should be verified.

3.2 Cartesian partitioning in 2D

In the following, the two-dimensional grid under consideration contains n_x by n_y cells. The grid is subdivided into square subdomains of side length s. This means that there will be $N = n_x n_y / s^2$ subdomains. In each direction, the grid size is thus restricted to be an integer multiple of s. This is a consequence of the three-step approach and it ensures that no partial (non-square) domains remain along the grid boundaries. Furthermore, the subdomains are numbered starting at 0 at the south-west corner and increasing from left to right, bottom to top. Each domain consists of s^2 cells which are numbered according to the same rules. Each cell contains u, v and p nodes, which are again numbered. In particular, nodes 0, 1 and 2 belong to cell c_0 and more generally c_i would contain nodes $3i$ (u), $3i+1$ (v) and $3i+2$ (p). An example of the cell and node numbering for $n_x = 8$, $n_y = 8$ and $s = 4$ is shown in Figure 3.1.

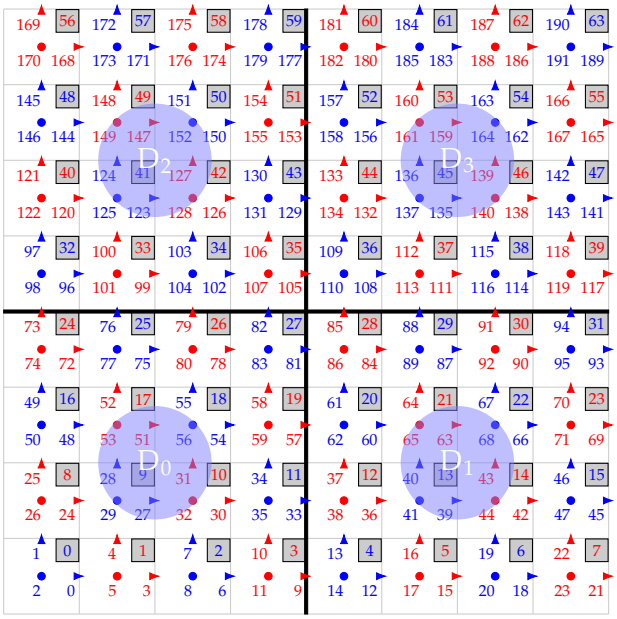


Figure 3.1. A grid of $n_x = 8$ by $n_y = 8$ cells is divided into $N = 4$ subdomains of separator length $s = 4$. Cell numbers are indicated in the north east corner of each cell. Each cell can be associated with three nodes (one u, v and p), indicated by the same color.

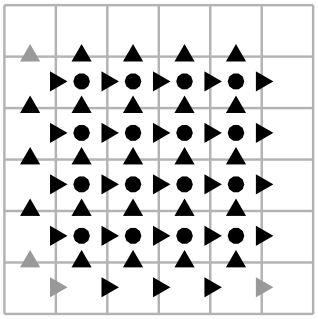
The template and test problem

The first step is to generate a template for the domain. The template contains all nodes in the domain together with all surrounding separator nodes. Here are distinction is made between the *minimal* template and the *maximal* template. The minimal template consists of the interior together with the minimal set of separators which completely close the interior from other domains. The maximal template is actually used in HYMLS and in the test problem to find all the groups properly. It is a square template (side length $s+1$) which is formed by the intersection of separator edges that run through the grid, up to and including all nodes that belong to both separators. The 2D Cartesian template is shown in Figure 3.2a, for the case where $s = 4$. The test problem is a 3×3 arrangement shown in Figure 3.3. The resulting groups, including the additional groups for the full conservation cell, are shown in Figure 3.2b.

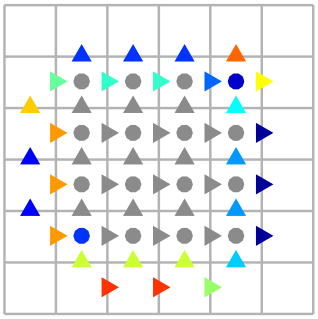
Full problem

With the groups for an arbitrary domain known, the full problem can be solved by shifting the results appropriately. However, for domains at the boundaries, special care should be taken. For example, for domains along the left boundary of the grid (D_0, D_3, D_6 in Figure 3.1) one should cut off the left-most cells from the template. Similarly, there are excess cells for domains along the bottom of the domain.

For cells along the top and right boundaries of the grid, separator groups that coin-



(a) Domain template. The minimal template is shown in black. The additional nodes used in the test problem are shown in grey.



(b) Result from test problem. Different colors represent different groups. This shows that there are at most 19 groups per domain.

Figure 3.2. MATLAB results for the domain template and the groups from the test problem for a domain with separator length $s = 4$.

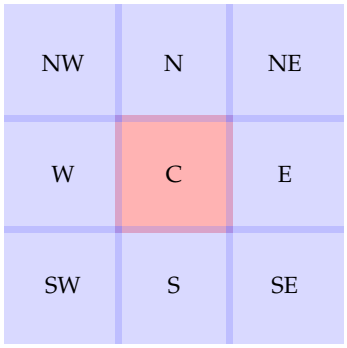


Figure 3.3. The configuration in the test problem for the 2D Cartesian partitioning. It consists of 9 domains, with the test domain located in the center.

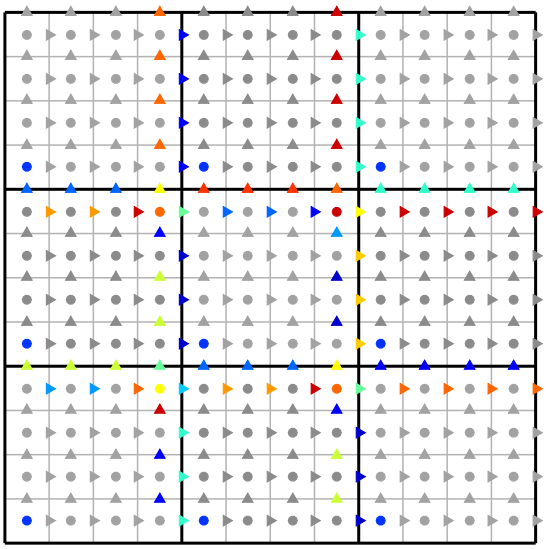


Figure 3.4. MATLAB results for problem with $n_x = 12$, $n_y = 12$, $s = 4$. Different colors represent different groups.

cide with the grid boundaries should be changed accordingly. Along the top boundary it does not make sense to have a horizontal separator (unless there are periodic boundary conditions) and therefore the v velocities should be added to the interiors of the corresponding domains whereas the u velocities from the full conservation cells should be added to the vertical separators. The separators along the right boundary should be updated analogously. So, there are two boundary cases to consider; either there are excess nodes corresponding to template cells that are outside the grid after shifting, or there are separators coinciding with the grid boundaries. To get the groups of the remaining domains, shifting the template groups is sufficient.

After running the MATLAB program, the results should of course be verified. At first hand, it was checked whether there was any overlap between different groups of the same domain. Any intersection of two groups of the same domain should be empty. Furthermore, it was made sure that all groups together contained not too many or too few nodes, i.e. that the disjoint union of nodes in the groups (including the interior) exactly coincided with all nodes in the domain template

A second, powerful way to verify the results is by plotting them. The result of the 12×12 grid is shown in Figure 3.4. For each domain, different groups are represented by different colors.

3.3 Cartesian partitioning in 3D

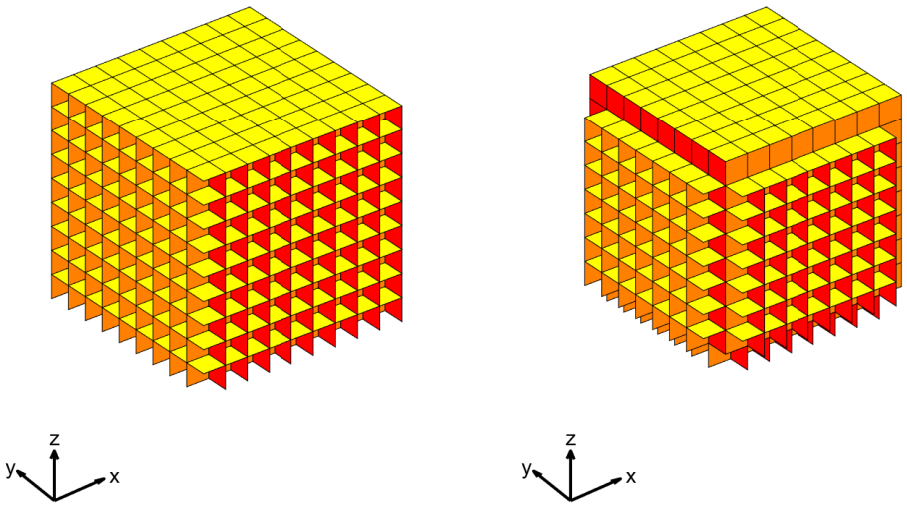
The three dimensional case can be considered an extension of the two dimensional case. To exploit this observation in building the 3D template, the starting point is the 2D do-

main template stacked on top of each other. Then, additional w-nodes should be placed to close the domain.

The visualization using markers and cells is no longer clear in three dimensions. Therefore, a different representation is chosen for the template, see Figure 3.5. This representation still uses the idea of grid cells, but instead of drawing markers, the nodes are indicated by colored faces of the grid cells. In this representation, u nodes are colored in red, v nodes in orange, w in yellow. Using this approach, it is easy to spot missing separators (there would be gaps in the separator planes, which could lead to connected interiors) or isolated pressure nodes (there would be grid cells that are completely closed off). Furthermore, the entire template also contains pressure nodes, which if plotted should be completely closed off by the separator nodes. A layer-by-layer representation of the separators surround the domain template is shown in Figure 3.7. In this picture, the two dimensional template can be recognized when looking closely to the u (red) and v (orange) nodes only. Furthermore, isolated pressure nodes can readily be spotted in the top-right grid cells on most layers, resulting in complete tubes of isolated pressure nodes.

In order to find the groups, a test problem can be set up which contains 27 subdomain templates. Their relative positions correspond to three layers of 3×3 arrangement similar to the 2D case, i.e. all possible combinations of displacements 1, 0 and -1 (in units of the separator length) in x, y and z directions (see Figure 3.6). The tubes of isolated pressure nodes will result in many additional groups of nodes that should be retained in the Schur complement problem. Finally, the full problem is solved similarly as in the two dimensional case, where domains along the boundary may contain excess grid nodes or unnecessary separators that should be either removed from the grid or added to the interior of the corresponding domains.

The fact that the two-dimensional case can be recognized in Figure 3.7 deserves some more attention. The figure shows a layer-by-layer representation in the z-direction and if the w-nodes are disregarded one sees exactly the two-dimensional Cartesian partitioning of Figure 3.2a in all layers except the first and last. More generally, projecting the template layers along either the x, y or z directions will yield the exact same figure (but with permuted colors) and hence the 2D case can be recognized from either direction, as can be expected from an extension of a square into a cube.



(a) Maximal template. **(b)** Minimal template.

Figure 3.5. MATLAB results for 3D Cartesian domain template with separator length $s = 8$.

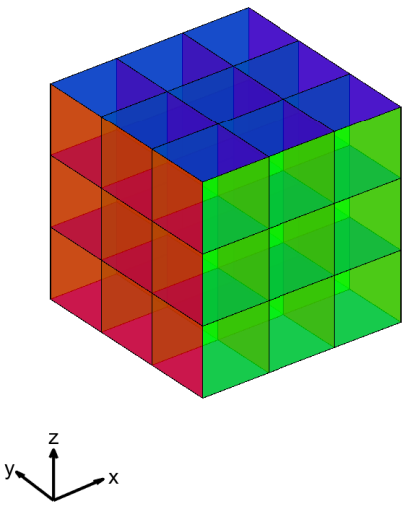


Figure 3.6. The configuration for the three-dimensional Cartesian test problem contains 27 domains. The test domain is located in the center.

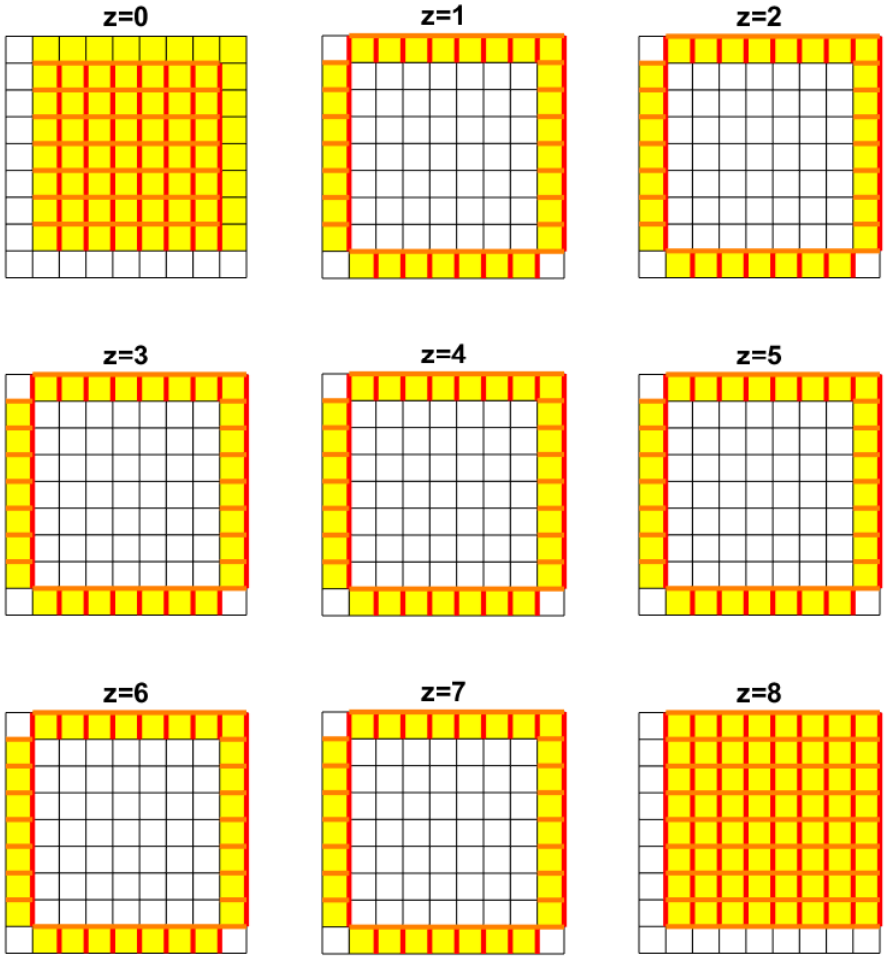


Figure 3.7. A layer-by-layer representation of the domain in Figure 3.5. The (yellow) *w*-nodes are located 'on top of' the u- and *v*-nodes. In the top-right corner, isolated pressure nodes are present.

4 Skew partitioning

Now that the standard Cartesian partitioning is discussed, it is clear how isolated pressure nodes arise. In the HYMLS code, an full conservation cell implementation was made such that isolated pressure nodes are retained in the Schur complement. Otherwise, danger exists that singular matrices are encountered for the domain interior or the matrices could no longer share the important properties of $\mathscr{F}$-matrices [4]. This implementation, however, is fragile and hence partitioning with skew separators is considered as an alternative. The general approach to find the separators groups for skew separators remains the same as in the previous case, as only the templates should be changed (and hence the corresponding test problem). For the 3D case, it gets really important to be able to plot the results accurately in order to verify the partitioning, since this partitioning is novel. As in the previous chapter, the problem will be discussed using the three-step approach: template, test problem, full problem.

4.1 Skew partitioning in 2D

The principle for skew partitioning in two dimensions has already been mentioned in Chapter 1. Placing the separators diagonally over the grid provides the ability to partition the domain without isolating any pressures. An example of a skew partitioned grid in 2D is shown in Figure 4.1, where $n_x = n_y = 8$ and the separator length is $s = 2\sqrt{2}$. Because it is easier to work with integer number, one can recognize that the slightly shaded area in the figure (square block with corner cells 0 and 27) actually forms a 'repeating unit' for the grid. That is, when placing the separator pattern inside this block repeatedly throughout the grid the full grid partitioning is generated. Hence, instead of using the separator length s one can equivalently work with the 'cube length' $c_\ell = \sqrt{2}s$ (so $c_\ell = 4$ in Figure 4.1) since in three dimensions this extends to a cube. This length will also be the input parameter for the MATLAB programs.

The template and test problem

The template for this problem with $c_\ell = 6$ is shown in Figure 4.2a and the resulting groups are shown in Figure 4.2b. One can recognize a square subdomain, rotated by 45 degrees. At its maximum width and height it spans $c_\ell + 1$ and c_ℓ grid cells respectively.

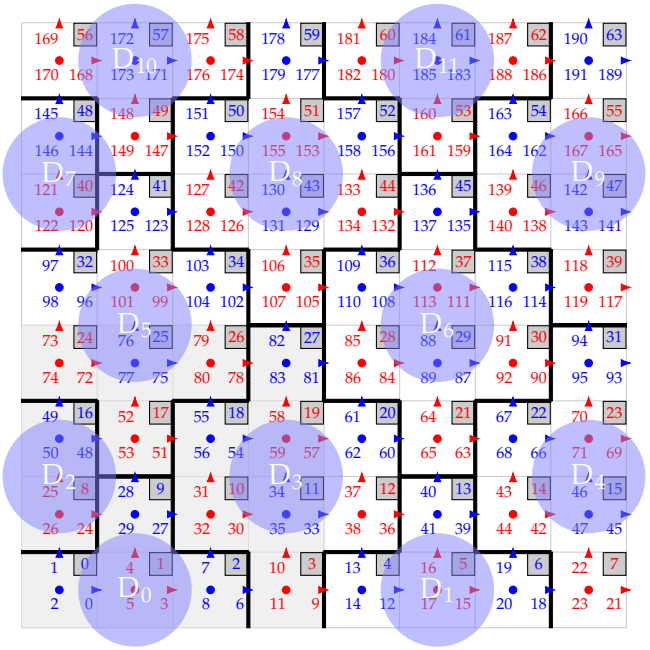
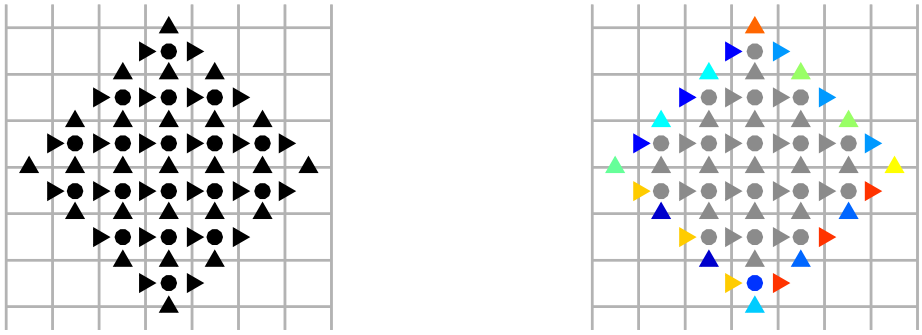


Figure 4.1. A grid of $n_x = 8$ by $n_y = 8$ cells is divided into 12 subdomains of separator length $s = 2\sqrt{2}$, or equivalently cube length $c_\ell = 4$ (shaded area). Cell numbers are indicted in the north east corner of each cell. Each cell can be associated with three nodes (one u, v and p), indicated by the same color.

The arrangement for the test problem is depicted in Figure 4.3. The arrangement is similar to the 2D Cartesian case and it also contains 9 domain templates. It results in less groups than in the Cartesian case, due to the fact that no longer full conservation cells have to be implemented since no isolated pressure nodes are present.

Full problem

The results from the test problem can now be applied to the full problem. For the problem in Figure 4.1, the MATLAB results are shown in Figure 4.4. It can be seen here that instead of only 'updating' the excess nodes or separator nodes along the grid boundaries, this time entire halves of the template have to be cut off. Contrary to the standard Cartesian case, where the excess nodes are always either the first or last row/column of the template, this requires some bookkeeping in the MATLAB program in order to determine which nodes exactly correspond to which half of the domain. This bookkeeping can be prevented by considering the absolute coordinates of the cells, making use of the node numbering introduced before. That is, for each node, the coordinates of its corresponding cell can be calculated and if either of its coordinates falls out of the specified grid its nodes should be removed. It may also occur that the first pressure node in the interior of the template, as shown in Figure 4.2b, has to be removed. Therefore, the step where for each subdomain the first pressure in its interior is put into a separate group is



(a) Domain template.

(b) Result from test problem.

Figure 4.2. MATLAB results for the skew domain template and the groups from the test problem for a domain with $c_\ell = 6$. Different colors represent different groups. From the test problem it can be found that there are at most 14 groups per domain.

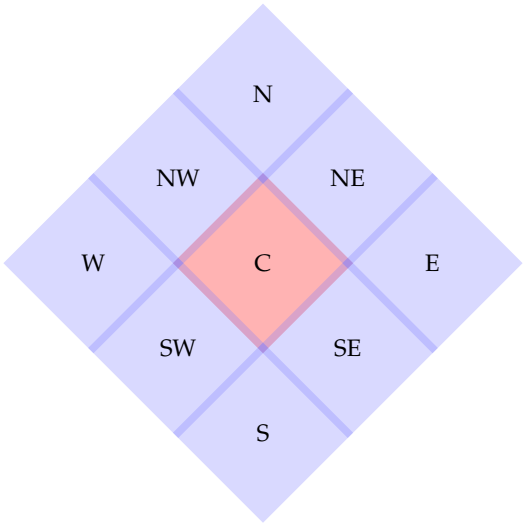


Figure 4.3. The arrangement in the test problem for the 2D skew partitioning.

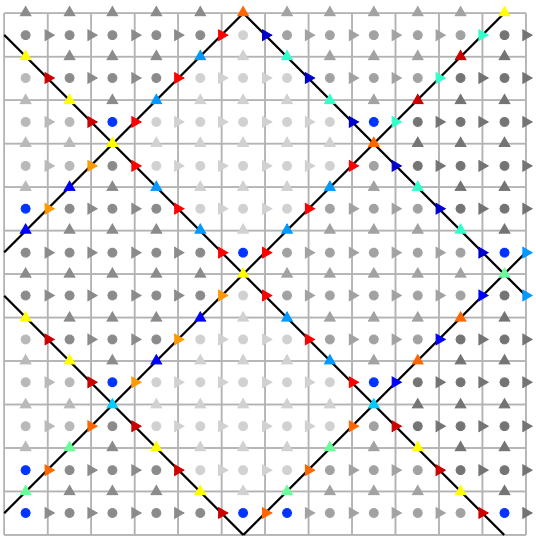


Figure 4.4. MATLAB results for the test problem and the complete problem with $n_x = 12$, $n_y = 12$, $c_\ell = 6$. Different colors represent different groups.

carried out only after the domains are distributed over the grid and hence the pressure nodes are part of the interior in the first stages of partitioning.

4.2 Skew partitioning in 3D

So far, each of the partitioning mechanisms currently employed have been studied in some detail. The most important findings can be summarized as follows. When studying the Cartesian partitioning, the 3D case can be considered as an extension of the 2D case, by applying the 2D partitioning in each of the three grid directions for the two complementary velocity components. Furthermore, when comparing the standard and skew partitioning, one can think of the latter being a distortion or rotation of the first. It is perhaps better to think of it as being a distortion, to prevent any restrictions on generalizing this to the 3D situation. That is, the 3D skew partitioning is not merely restricted to be a rotation of the cube that is used for 3D Cartesian partitioning, but rather it can be the result of distorting the cube. In fact, the final result is obtained by first rotating the cube by 45 degrees in the (arbitrarily chosen) z-plane and then distorting the result along the normal of this plane. Alternatively this can be imagined as performing the 2D partitioning to each layer in the grid and then use an additional third separator plane to create the subdomains.

4.2.1 Partitioning constraints

In order to ensure that the partitioning works properly, it should satisfy a number of constraints which are induced from the study of the existing partitioning mechanisms.

More precisely, the demands for the 3D skew partitioning are:

- There should not be any isolated pressure nodes. This is the main problem that is tackled in this project to begin with as it possibly defects the convergence properties of HYMLS.
- The interiors of different domains should not overlap or touch each other. This requires a careful construction of the separators and it is preferable to define the domains by subdividing the grid with entire (straight) separator planes.
- The separators of surrounding neighboring domains should coincide. If two domains are neighbors then they should share (precisely) one layer of separator velocities. This can make the template seem somewhat asymmetric. For example, the top separators of D_0 in Figure 3.1 should coincide with the bottom separators of D_3. Also this can be easily achieved by using straight separator planes. The test problem further ensures that the groups that are found are indeed the result from the overlapping separators, if the domain template is defined correctly.
- The domains should be scalable for the multilevel approach. The HYMLS method is based on the principle of multilevel preconditioning of the Schur complement. In order to do so, for each level a partitioning is performed on the retained nodes from the previous level. From a grid point-of-view (e.g. Figure 2.2) this comes down to having a domain shape which can be combined to construct a larger, scaled version of itself. In three dimensions this can readily be achieved by using three types separator planes.
- The domain shape should be space filling. That is, using only the domains one should be able to completely fill 3D space without leaving gaps. An easy way to achieve this is by making sure the domain can be built up from a 'repeating unit', similary to the shaded area of Figure 4.1. This further allows to continue working with the cube length instead of the separator length.
- For a proper partitioning, all nodes in the grid should belong to precisely one unique group (including the interiors of subdomains). These groups may occur multiple times as they surround multiple domains.

4.2.2 Identifying possible domain shapes

It should be clear from all previous considerations, that the most important ingredient in the domain decomposition is the domain template. This is where all the tweaking occurs, the rest is mainly the same as the previous cases: solve test problem, translate results. Therefore, the problem starts by finding candidate domain shapes that fulfill the partitioning constraints. This can be done based on the principle of extending the 2D skew case to three dimensions in the same way was the extension for Cartesian partitioning, i.e. by using the 2D partitioning and applying it in each of the three principal directions. Alternatively this can be done by stacking the 2D situation and using a cut

plane, or by rotating the Cartesian domains and then distorting it. An overview of some domain candidates is shown in Figure 4.5.

4.2.3 First design: octahedrons / tetrahedrons

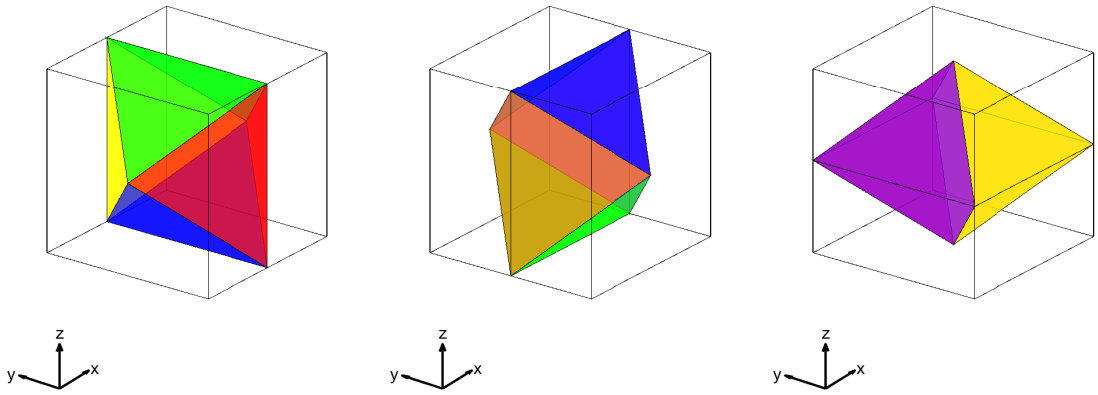
The first idea resulting from generalizing the 2D skew domain shape to a 3D setting was to use octahedron domains, as shown in Figure 4.5a. Partitioning with this design turned out to be unsuccessful, but it is still briefly discussed here since it lead to some new insights. Since regular octahedrons (the dual to cubes, having their vertices at the centers of the cube faces) in itself are not space tiling, the octahedrons are slightly distorted to make them fit within a single cubic repeat unit. The resulting domains are space tiling, but only by using three mutually orthogonal domain types. For clarity, these types will be referred to by the normal of the central square plane of the domain. So, the x-domain has a central square plane parallel to the yz plane and with normal in the x direction, *et cetera*. The fact that it requires the use of three types of templates increases the programming efforts significantly since it introduces a lot of exceptional cases that should be considered, e.g. for boundary domains.

When viewed along the y and z axes, the x-domain resembles the 45 degree rotated square plane as for the 2D domain shape. Moreover, a central, horizontal cross-section of the x-domain exactly yields the 2D case, which is a desired property for ease of implementation. Similar observations can be made for the other two domain types. However, a general observation is not true, for example the central, horizontal cross-section of the z-domain yields the 2D Cartesian case.

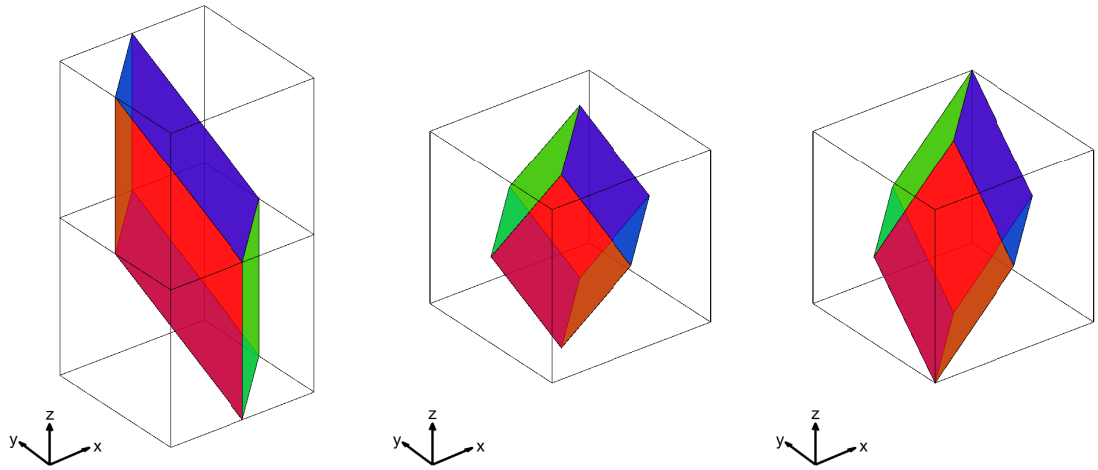
The octahedron domains are not scalable, which is a demand for the multilevel approach. However, scalability can be achieved by splitting the octahedrons into four smaller tetrahedrons, of which six different types are required to fill 3D space. This is indicated by the different colors in Figure 4.5a. This would introduce additional separator planes that are similar to the 2D skew case and hence it increases the risk of isolating a pressure node when such planes intersect. Especially planar intersections which are parallel to either of the Cartesian axes have a high risk of isolating pressure nodes (for similar reasons as isolated pressure nodes occur for the 3D Cartesian case, see Figure 3.7).

The octahedron domains from Figure 4.5a resulted either in isolated pressure nodes or in interiors which where not completely separated from each other and hence after a fair amount of effort, this idea was put to rest. It was briefly considered to rotate all three octahedrons by 45 degrees around the z axis, but since this results in entire planes parallel to the yz- or zx-planes (the rotated x- and y-domains), also this idea was disregarded.

Despite the fact that the implementation of octahedron domains was unsuccessful, it was very insightful since it provided the following observations. First of all, the complexity of the domain shape should be reduced as much as possible. Whereas the octahedron domains are bounded by *four* planes that do *not* run continuously throughout the grid, it should be possible to define domains that are bounded by three continuous planes. In fact, the 3D Cartesian partitioning is an example of this. Decreasing the



(a) Octahedron domains. Three mutually orthogonal domains (from left to right referred to as the x, y and z type) can be used to tile 3D space. In order to make this design scalable, each octahedron should be split up into four tetrahedrons. This would lead to six different domain types, indicated by the different colors.



(b) The final design. It is the result of stacking the 2D skew partitioning in the *z*-direction (red and green) and bounding it by a third, diagonal plane (blue).

(c) By tilting the green plane from the design in (b), a design is obtained that fits within a single cube.

(d) Subsequently tilting the red plane from the design in (c) yields a regular rhombohedron domain.

Figure 4.5. Possible domain shapes for 3D skew partitioning.

number of planes means that the risk of isolating pressure nodes is also decreased. Furthermore, by only using three separator planes, this creates one template types, which reduces the amount of programming effort in terms the number of exceptional cases for domains along the grid boundaries. It also makes it easier to get an impression of how the different domains are located relative to each other. If, in turn, this single domain type has the property that a cross-section yields the 2D skew case, this would also benefit the implementation in the source code. Lastly, by using only three types of planes, the resulting domain is guaranteed to be scalable: this is for example achieved by considering the domain that is obtained by simply removing every second plane of each type.

The three domain shapes shown in Figures 4.5b to 4.5d all satisfy these constraints. The latter two can be obtained by subsequently tilting the green and red planes. Each of these was briefly studied, and since the first is most easily obtained from the 2D skew partitioning, namely by 'stacking' the two-dimensional template in the z direction and then using a third plane to bound the three-dimensional domains, it was decided to continue working on this design. The other two designs showed isolated pressure nodes in the first try and are hereafter disregarded in this work.

4.2.4 Final design

The final design is the prism in Figure 4.5b. This domain is obtained from the 2D skew partitioning by stacking the 2D skew separators in the $z$direction splitting this into subdomains by using a third, skew plane as shown in Figure 4.6. So the partitioning is defined by three skew planes throughout the grid. These planes can be characterized by their normal vectors:

$$\begin{aligned} \text{red}: &\quad n_r = \tfrac{1}{2}\begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^T, \\ \text{green}: &\quad n_g = \tfrac{1}{2}\begin{bmatrix} 1 & -1 & 0 \end{bmatrix}^T, \\ \text{blue}: &\quad n_b = \tfrac{1}{3}\begin{bmatrix} 1 & -1 & 1 \end{bmatrix}^T, \end{aligned}$$

where the length of each vector is normalized to match the planar spacings (in units of c_ℓ).

The template for the final design is derived from the prism in Figure 4.5b. Using the same three-dimensional visual representation as before, with $c_\ell = 8$ this yields the domains shown in Figure 4.7. A layer-by-layer representation is shown in Figure 4.9. The maximal template is used to solve for the different groups, whereas the minimal template shows all of the separators that directly connect to the interior nodes. The configuration for the test problem is shown in Figure 4.8; it contains 27 domains. In fact, the test problem is equivalent to the 3D Cartesian case but with rotated principal directions: instead of stacking in the x-, y- and z-directions, the stacking occurs in the (x/2+y/2)-, (x/2-y/2+z)- and z-directions. There are 65 groups for each subdomain.

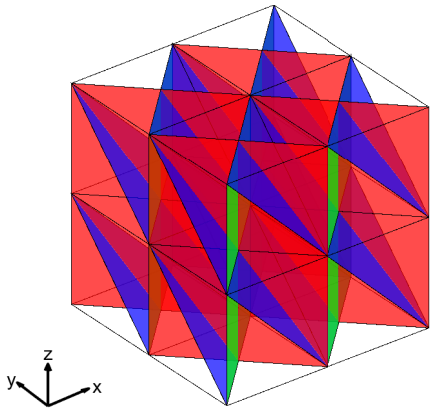


Figure 4.6. Three types of planes define the entire partitioning for the final design. Furthermore, the grid can be build up from a cubic repeat unit, of which eight are drawn here. This implies that the design is suitable for the multilevel approach.

At the same time, Figure 4.8 illustrates that indeed the domain shape is scalable. This makes it a suitable choice for the multilevel approach. In fact, the figure shows the domain that would be obtained if a coarsening factor of 3 is applied.

An interesting property of the final domain template is that its central cross-section resembles the skew partitioning in the two-dimensional case. In Figures 4.7 and 4.9 this can be seen at the eighth layer of grid cells ($z = 8$). Indeed, the red (u) and orange (v) nodes on this layer correspond to the same template as Figure 4.2a, albeit for a different c_ℓ. This means that in the implementation, the first layer of cells can be made to correspond exactly to the 2D case and the skew 3D partitioning can be regarded as an extension of the 2D skew partitioning. Another way of seeing this is by comparing the separator edges on the bottom plane (or top for that matter) in Figure 4.6 to the separator lines in Figure 4.4 and recognizing that they are equivalent. The HYMLS implementation was adapted to make the case $n_z = 1$ correspond to the 2D skew partitioning by only taking the central layer of the domain template.

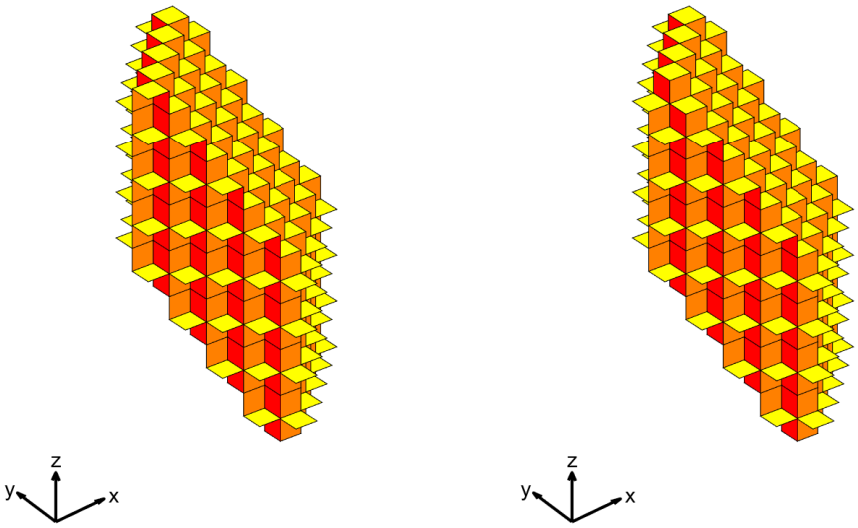
What is interesting to see in Figure 4.5b is that in the z direction, the (yellow) w nodes follow an oscillating pattern. This is done to match the w nodes in the third plane corresponding to the blue plane in Figure 4.6 on each z layer. This minimizes the number of separator velocities as well as the risk of isolating pressure nodes. The cost of this is that the w nodes on the inside of the domain have to be put in a different group than the ones on the outside of the template, since each will couple to different pressure nodes in the Schur complement. Hence, if this is not done, the $\mathscr{F}$-matrix structure of the Schur complement is lost and the method does not work anymore. These kinds of tweaks where implemented in the C++ code during the implementation process but not in the MATLAB code that is included in Appendix A.

The first test of the new partitioning approach was performed on the steady Stokes equations in absence of body forces. These equations are a simplification of the Navier-

Stokes equations whereby $Re \to 0$. This yields (cf. Equations (1.1) and (1.2))

$$\begin{aligned} \mu\nabla^2\mathbf{u} - \nabla p &= 0, \\ \nabla \cdot \mathbf{u} &= 0. \end{aligned}$$

Both the 2D (the first layer of the 3D case) and 3D Stokes equations were tested for different combinations of the (first-level) separator length c_ℓ, the coarsening factor c_x and the number of levels of the Schur preconditioner. The results are shown in Figures 4.10a and 4.10b. For the 2D case it is clear that the number of iterations becomes independent of n_x. The 3D case looks promising as well since the lines seem to flatten as they do for the 2D problem. However, probably larger grids are required to really see if the lines become independent of n_x. An interesting observation is that the lines for $c_\ell = 8$, $c_x = 4$ have similar slopes as the lines for $c_\ell = 4$, $c_x = 8$. This can be explained from the fact that at each level the domains grow in a similar way, meaning that essentially the same information is retained on successive levels. In the first stage, however, more information is 'lost' for $c_\ell = 8$ (since there are less separators throughout the grid), and hence more iterations are required to solve the problem accurately.



(a) Maximal template. **(b)** Minimal template.

Figure 4.7. Representation of the separator nodes in the domain template in the final design for cube length $c_\ell = 8$. The domain is 15 grid cells high, the u and v nodes on the eighth layer exactly coincide with the 2D skew partitioning discussed in Section 4.1.

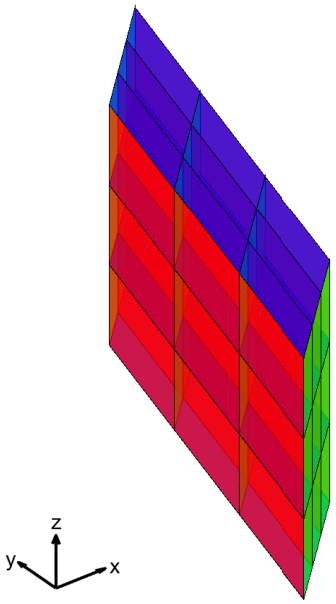


Figure 4.8. The configuration in the test problem consists of 27 domains.

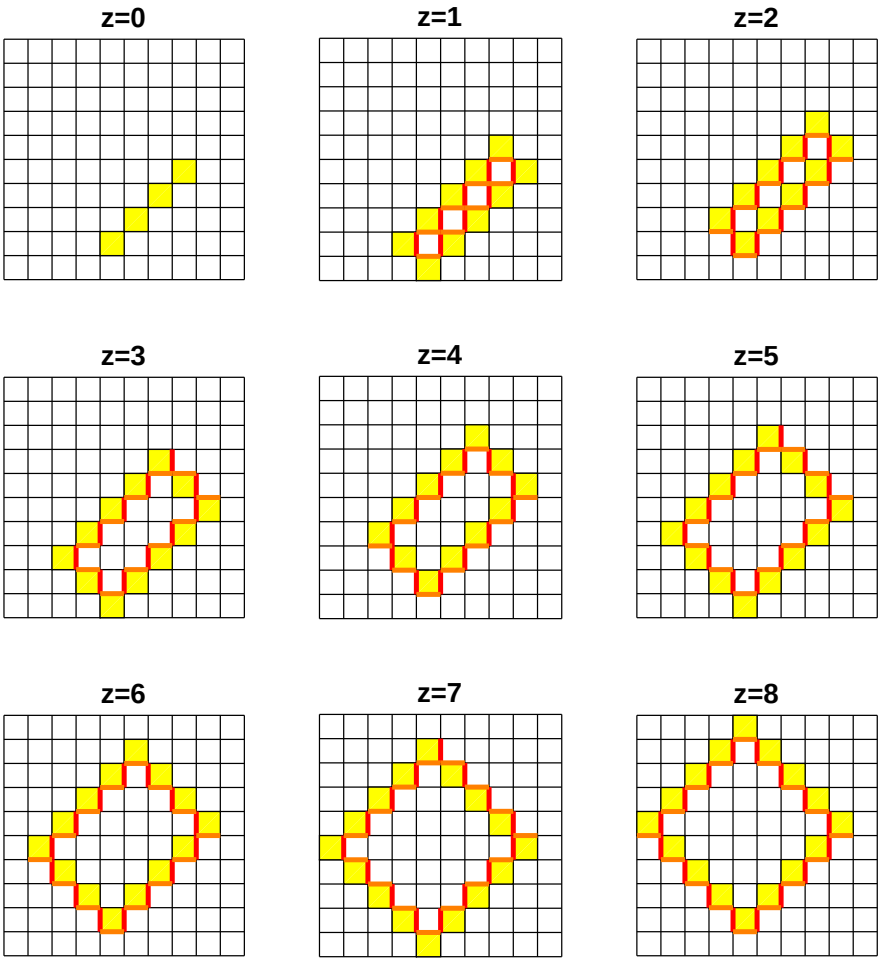


Figure 4.9. Layer by layer representation of the final design for $c_\ell = 8$. At $z = 8$, the 2D skew partitioning is retrieved.

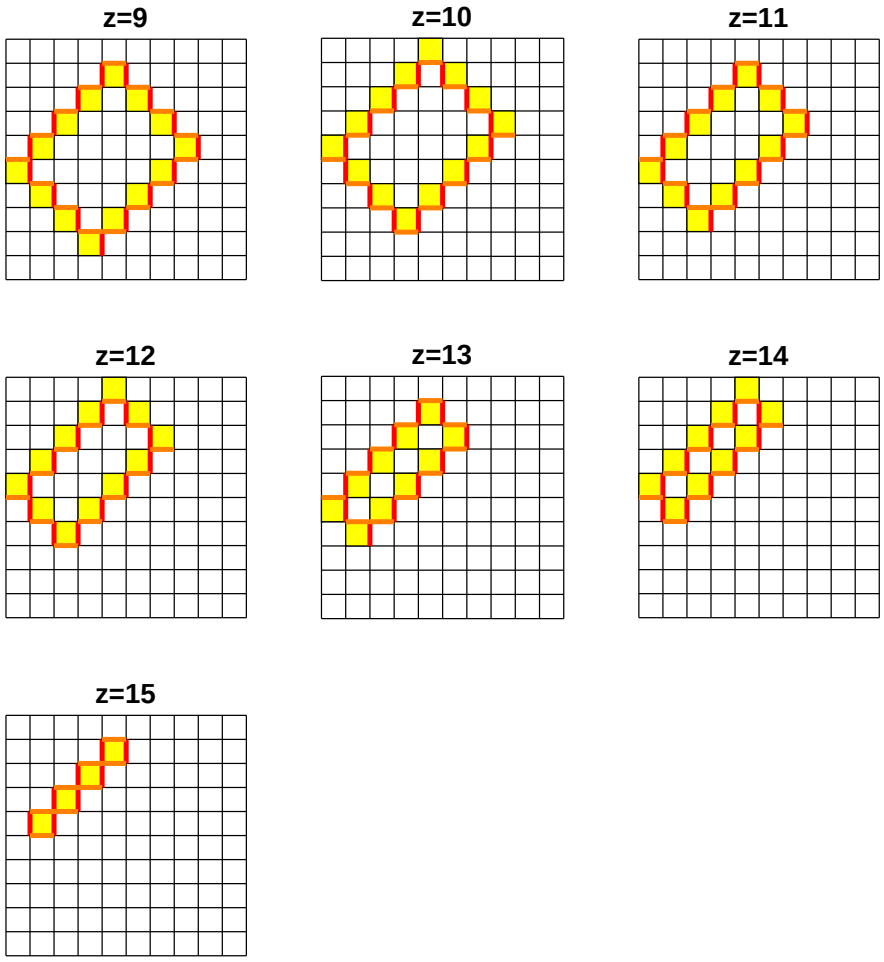
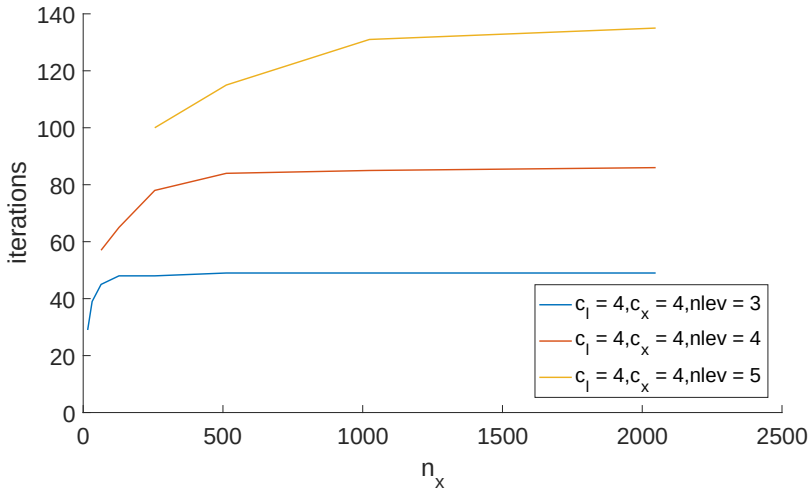
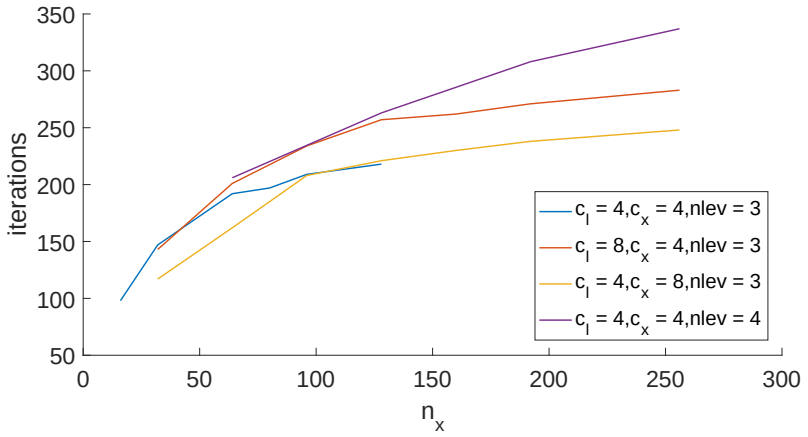


Figure 4.9. *(continued)* Layer by layer representation of the final design for $c_\ell = 8$. At $z = 8$, the 2D skew partitioning is retrieved.



(a) Results for 2D Stokes using central layer of the new 3D skew domains.



(b) Results for 3D Stokes.

Figure 4.10. Results for the steady Stokes equations using the new partitioning. The grid size is n_x in all directions.

5 Discussion

The MATLAB code presented in Appendix A was handed over to be implemented in the HYMLS C++ code. After some optimization, the code was tested successfully on a single level, i.e. using the Schur preconditioner only once before solving. At this stage, the code did not work in parallel nor on multiple levels. The parallel implementation was eventually achieved and also the problems considering multilevel preconditioning were solved, leaving a fully functional multilevel preconditioner.

During the implementation, it was found out that the oscillating w nodes in the vertical separator walls posed a problem for the multilevel approach. The w nodes on either side of the walls would couple to different sets of pressure nodes leading to four dense columns in the Schur complement instead of two. To make sure no similar cases would occur, a run can be performed over all nodes in the separator groups to check whether the cells of the corresponding nodes would belong to the same domains. Nodes that belong to different domains (i.e. velocities that would couple to a different set of pressure nodes) can then be split into separate groups.

The partitioning method performed as in the provided MATLAB code suffers the limitation that the `solveGroups` script only works if c_ℓ is chosen to be smaller than or equal to both $n_x/2$ and $n_y/2$. This happens because the height of the template has twice the size of the width. The positioning part in the `solveGroups` script will fail if c_ℓ is chosen too large. This problem can be circumvented by reprogramming the script to make it independent of the grid that is used. This has not yet been corrected in the C++ implementation, which means that the number of levels is restricted.

Furthermore, the bookkeeping used in lines 78–173 of `SKEW_PARTITIONER.m` should be replaced by a script that removes nodes based on their global coordinates, so that it automatically removes nodes that fall outside the grid. This would both be safer code (since it treats all boundary cases simultaneously) and consequently the code would be cleaner to read. This was changed in the actual HYMLS implementation as well.

The first results for the Stokes problem look promising. However, the 3D partitioning without isolated pressure nodes does not yet yield compelling evidence for grid-independent convergence. It is likely that this is due to the small range of n_x values, so experiments on finer grids should be performed in order to confirm grid independence.

6 Conclusion

Skew partitioning in three dimensional problems is complicated but possible. Correct partitioning is essential for HYMLS since its entire working principle is based on partitioning. Preferably the partitioning is done without any isolated pressure nodes or full conservation cells, since these may affect the convergence properties of the method. The search for ways to partition the three-dimensional grid started off by studying the currently employed partitioning methods in some detail. The three-step approach provided means to tackle each of the problems systematically. This lead to insights on generalizing from 2D to 3D and from Cartesian to skew partitioning. These insights in turn lead to candidate domain shapes. With the prism domain shape, it is possible to have skew partitioning without isolated pressure nodes. Moreover, the implementation in the HYMLS code showed promising results and hence the 3D skew partitioning in HYMLS provides many new opportunities to novel numerical experiments.

Bibliography

[1] Fred W Wubs and Jonas Thies. A robust parallel ilu solver with grid-independent convergence for the coupled steady incompressible navier-stokes equations. *Proceedings 5th ECCOMAS CFD*, 2010, 2010.

[2] Fred W Wubs and Jonas Thies. A robust two-level incomplete factorization for (navier–) stokes saddle point matrices. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1475–1499, 2011.

[3] Jonas Thies and Fred Wubs. Design of a parallel hybrid direct/iterative solver for cfd problems. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 387–394. IEEE, 2011.

[4] Jonas Thies. *Scalable algorithms for fully implicit ocean models*. PhD thesis, University of Groningen, 2011.

[5] Akio Arakawa and Vivian R Lamb. Computational design of the basic dynamical processes of the ucla general circulation model. *Methods in computational physics*, 17:173–265, 1977.

[6] Arie C De Niet and Fred W Wubs. Numerically stable ldlt-factorization of f-type saddle point matrices. *IMA Journal of Numerical Analysis*, 29(1):208–234, 2009.

[7] Miroslav Tuma. A note on the ldlt decomposition of matrices from saddle-point problems. *SIAM journal on matrix analysis and applications*, 23(4):903–915, 2002.

[8] Weiyan Song, Fred W Wubs, and Jonas Thies. A highly parallel code for strongly coupled fluid-transport equations. In *Proceedings of the 11th world congress on computational mechanics (WCCM XI)*, pages 199–210, 2014.

[9] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[10] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

A MATLAB code

The three main MATLAB files for 3D skew partitioning, corresponding to the three-step approach are included in this appendix.

Listing A.1. SKEW_PARTITIONER.m

```
%% 3D SKEW PARTITIONER by Mark van der Klok
% performs 3d skew partitioning for HYMLS

clearvars

nx = 16; ny = 16; nz = 16;% Grid settings
cl = 8;                   % Cube length. NOTE: cl <= nx/2 , ny/2
nVars = 4;                % Number of node types

% Principal directions.
dirX = nVars*cl; dirY = nVars*nx*cl; dirZ = nVars*nx*ny*cl;

% Get the template and solve test problem
template = getTemplate(cl, nx, ny, nz, nVars);
templateList = template2list(template);
groups = solveGroups(templateList, cl, nx, ny, nz, nVars);
fprintf('Found %d groups ... \n', length(groups))

% Carry out first test to see if all nodes are in exactly one group
testGroups(templateList, groups);

% Get different halves of the domain to treat boundary cases
bottomhalf = template2list(template(1:cl));
tophalf = template2list(template(cl+1:end));
for ii = 1:cl+2
  testMatrix(ii,:) = (0:nVars:nVars*cl) + (ii-1)*nVars*nx ...
    - cl*nVars*nx*ny - nVars*(cl/2+1)*nx;
end
testSouth = testMatrix(1:cl/2+1,:);
testWest = testMatrix(:,1:cl/2+1);
removeCol = testMatrix(1,cl/2+1):nVars*nx*ny:testMatrix(1,cl/2+1) ...
  + dirZ+nVars*nx*ny;

halves = {'bottomhalf', 'tophalf'};
for h = [1,2]
  currentHalf = eval(halves{h});
```

```
  NSintersect = []; EWintersect = [];
  for type = 0:3
    for jj = 1:cl % length of the half
      NSintersect = [NSintersect;
        testSouth+(jj-1 + (h-1)*cl)*nVars*nx*ny+type];
      EWintersect = [EWintersect;
        testWest+(jj-1+ (h-1)*cl)*nVars*nx*ny+type];
    end
  end

  north{h} = removeFromList(currentHalf, NSintersect(:));
  south{h} = removeFromList(currentHalf, north{h});
  east{h} = removeFromList(currentHalf, EWintersect(:));
  west{h} = removeFromList(currentHalf, east{h});

end

% Get one layer in the grid (same as skew 2d case)
ncx = nx/cl; ncy = ny/cl; ncz = nz/cl;
totNum2DCubes = ncx * ncy; % number of cubes for fixed z
numPerLayer = 2 * totNum2DCubes + ncx + ncy; % domains for fixed z
numPerRow = 2*ncx + 1; % domains in a row (both lattices); fixed y
totNumDomains = (ncz+1)*numPerLayer - 1;

domains = cell(1,numPerLayer);
for D = 0:totNumDomains
  Z = floor(D/numPerLayer);

  % Get domain coordinates and its first node
  % Considers 'superposed' lattices
  X = mod(D - Z*numPerLayer, numPerRow); lattice = 1;
  Y = floor((D - Z*numPerLayer) / numPerRow) - 0.5;
  if X >= ncx
    X = X - ncx  - 0.5;
    Y = Y + 0.5;
    lattice = 2;
  end

  % First node of cube
  firstNode = X*dirX + Y*dirY + Z*dirZ - nVars + nVars*nx*(cl/2);

  % Boundary cases. Can be optimized by basing on cell coordinates
  toRemove = [];
  if lattice == 1
    if Y == -0.5
      if Z == 0
        currentNodes = north{2};
        toRemove = [toRemove; bottomhalf; south{2}];
      elseif Z == ncz
        currentNodes = north{1};
        toRemove = [toRemove; tophalf; south{1}];
      else
        currentNodes = [north{1}; north{2}];
        toRemove = [toRemove; south{1}; south{2}];
```

```
      end
    elseif Y == ncy - 0.5
      if Z == 0
        currentNodes = south{2};
        toRemove = [toRemove; bottomhalf; north{2}];
      elseif Z == ncz
        currentNodes = south{1};
        toRemove = [toRemove; tophalf; north{1}];
      else
        currentNodes = [south{1}; south{2}];
        toRemove = [toRemove; north{1}; north{2}];
      end
    else
      if Z == 0
        currentNodes = tophalf;
        toRemove = [toRemove; bottomhalf];
      elseif Z == ncz
        currentNodes = bottomhalf;
        toRemove = [toRemove; tophalf];
      else
        currentNodes = templateList;
      end
    end

    if X == 0
      currentNodes = removeFromList(currentNodes, ...
        [removeCol - nVars*(cl/2 + (cl/2)*nx + (cl-1)*nx*ny), ...
        removeCol - nVars*(cl/2 + (cl/2)*nx + (cl-1)*nx*ny + 1), ...
        removeCol - nVars*(cl/2 + (cl/2)*nx + (cl-1)*nx*ny + 2), ...
        removeCol - nVars*(cl/2 + (cl/2+1)*nx + (cl-1)*nx*ny), ...
        removeCol - nVars*(cl/2 + (cl/2+1)*nx + (cl-1)*nx*ny + 2)]);
      toRemove = [toRemove;
        removeCol' - nVars*(cl/2 + (cl/2)*nx + (cl-1)*nx*ny);
        removeCol' - nVars*(cl/2 + (cl/2)*nx + (cl-1)*nx*ny + 1);
        removeCol' - nVars*(cl/2 + (cl/2)*nx + (cl-1)*nx*ny + 2);
        removeCol' - nVars*(cl/2 + (cl/2+1)*nx + (cl-1)*nx*ny);
        removeCol' - nVars*(cl/2 + (cl/2+1)*nx + (cl-1)*nx*ny + 2)];
    end
  elseif lattice == 2
    if X == -0.5
      if Z == 0
        currentNodes = east{2};
        toRemove = [toRemove; bottomhalf; west{2}];
      elseif Z == ncz
        currentNodes = east{1};
        toRemove = [toRemove; tophalf; west{1}];
      else
        currentNodes = [east{1}; east{2}];
        toRemove = [toRemove; west{1}; west{2}];
      end
    elseif X == ncx - 0.5
      if Z == 0
        currentNodes = west{2};
        toRemove = [toRemove; bottomhalf; east{2}];
```

```
      elseif Z == ncz
        currentNodes = west{1};
        toRemove = [toRemove; tophalf; east{1}];
      else
        currentNodes = [west{1}; west{2}];
        toRemove = [toRemove; east{1}; east{2}];
      end
    else
      if Z == 0
        currentNodes = tophalf;
        toRemove = [toRemove; bottomhalf];
      elseif Z == ncz
        currentNodes = bottomhalf;
        toRemove = [toRemove; tophalf];
      else
        currentNodes = templateList;
      end
    end
    if Y == 0
      currentNodes = removeFromList(currentNodes, ...
        [removeCol+1, removeCol+2]);
      toRemove = [toRemove; removeCol'+1; removeCol'+2];
    elseif Y == ncy-1;
      currentNodes = removeFromList(currentNodes, ...
        removeCol + 2 + nVars*((cl-1)*nx*ny + (cl+1)*nx));
      toRemove = [toRemove;
        removeCol' + 2 + nVars*((cl-1)*nx*ny + (cl+1)*nx)];
    end
  end

  % Get the groups, remove nodes
  currentGroups = groups; numEmpty = 0;
  for ii = 1:length(currentGroups)
    G = ii - numEmpty;
    currentGroups{G} = removeFromList(currentGroups{G}, toRemove) ...
      + firstNode;
    if isempty(currentGroups{G})
      currentGroups(G) = [];
      numEmpty = numEmpty + 1;
    end
  end

  % Get first pressure node from interior to a new group
  interior = sort(currentGroups{1}); % Assumes interior first!
  for i = 1:length(interior)
    node = interior(i);
    if mod(node,nVars) == 3
      interior = removeFromList(interior, node);
      currentGroups{1} = interior;
      currentGroups = [currentGroups, {node}];
      fprintf('First pressure node : %3d \n', node)
      break
    end
  end
```

```
199
200   % For visualization only: get v_sum nodes for next level
201   vsum = zeros(1,length(currentGroups));
202   for j=1:length(currentGroups)
203     vsum(j) = currentGroups{j}(1);
204   end
205
206   % Save results
207   domains{D+1}.nodes = currentNodes + firstNode;
208   domains{D+1}.groups = currentGroups;
209   domains{D+1}.first = firstNode;
210   domains{D+1}.xVal = X;
211   domains{D+1}.yVal = Y;
212   domains{D+1}.zVal = Z;
213   domains{D+1}.lattice = lattice;
214   domains{D+1}.vsum = vsum;
215
216   % Check function xyz2subdom for all nodes in interior
217   for i= currentGroups{1}
218     x = mod(floor(i / nVars), nx);
219     y = mod(floor(i / nVars / nx), ny);
220     z = mod(floor(i / nVars / nx / ny), nz);
221     assert(xyz2subdom(x,y,z,nx,ny,nz,cl) == D);
222   end
223 end
224
225 % Additional tests
226 performChecks(domains,nx,ny,nz)
```

Listing A.2. `getTemplate.m`

```
1  function nodes = getTemplate(cl, nx, ny, nz, nVars)
2  % Principal directions
3  dirX = nVars; dirY = nVars*nx; dirZ = nVars*nx*ny; % Cartesian directions
4  dir1 = dirY+dirX; dir2 = dirY-dirX; % Skew directions in xy-plane
5
6  % Info for each node type
7  firstNode = [nVars*cl/2 + dirY, nVars*cl/2 + 1, nVars*cl/2 + 2 - dirZ, ...
8    nVars*cl/2 + 3 + dirY] - dirY*(cl/2+1);
9  baseLength = [cl/2, cl/2 + 1, cl/2 + 1, cl/2];
10
11 % Get the nodes
12 for type = 1:nVars
13   % Current node type
14   nodes{type} = cell(1, 2*cl + 1);
15
16   % Get central layer
17   [central, ptr] = buildPlane45(firstNode(type), baseLength(type));
18   nodes{type}{cl+1} = central;
19
20   % Used in the loop to determine which nodes to assign to each layer
21   idxList = 1:length(central); rowLength = diff(ptr) - 1;
22   activePtrs = 1:baseLength(type);  offset = rowLength(activePtrs);
23
24   % Vertical offset for p layers
```

```
  if type == nVars, pAdd = 1; else pAdd = 0; end

  bottom = [];
  for i = 1:cl
    % Get indices for the layers
    bottom = [bottom, ptr(activePtrs) + offset];
    top = idxList; top(bottom) = [];

    % Add layers
    if type == 3 % w layers are an exception with odd/even layers
      if mod(i,2) == 0
        nodes{type}{i} = nodes{type}{i-1} + dirY + dirZ;
        nodes{type}{cl + 1 + i} = central(top) + i*dirZ;
      else % mod(i,2) == 1
        nodes{type}{i} = central(bottom) - (cl + 1 - i)*dirZ;
        nodes{type}{cl + 1 + i} = nodes{type}{cl + i} + dirY + dirZ;
      end
    else
      nodes{type}{i + pAdd} = central(bottom) - (cl + 1 - pAdd - i)*dirZ;
      nodes{type}{cl + 1 + i} = central(top) + i*dirZ;
    end

    if i < cl
      % Update pointers and offset
      offset = offset - 1;
      if type == nVars % pressure nodes are an exception
        if offset(1) < 0
          offset(1) = []; activePtrs(1) = [];
          activePtrs = [activePtrs, activePtrs(end)+1];
          offset = [offset, rowLength(activePtrs(end))];
        end
      else
        if offset(1) < 0
          offset(1) = []; activePtrs(1) = [];
        elseif offset(1) == 0,
          activePtrs = [activePtrs, activePtrs(end)+1];
          offset = [offset, rowLength(activePtrs(end))];
        end
      end
    end
  end
end

% Remove top and bottop single wall, which are unnecessary separators
nodes{1} = nodes{1}(2:end-1);
nodes{2} = nodes{2}(2:end-1);
nodes{3} = nodes{3}(2:end);
nodes{4} = nodes{4}(2:end-1);

% Merge the template layers
nodes = mergeTemplateLayers(nodes);

%% Nested function, to build a square, rotated plane
  function [plane,ptr] = buildPlane45(firstNode, lngth)
```

```
    left = firstNode; right = firstNode; height = 2*lngth;
    extraLayer = false;
    if (mod(firstNode,nVars) == 0) % correction for u nodes
      left = left - dirX; height = height + 1; extraLayer = true;
    elseif (mod(firstNode,nVars) == 3)
      extraLayer = true; height = height + 1;
    end

    % Build the plane
    plane = []; ptr = [1, zeros(1,height-1)];
    for ii = 1:height-1
      plane = [plane, left:dirX:right];
      ptr(ii+1) = ptr(ii) + length(left:dirX:right);

      if ii < lngth
        left = left + dir2; right = right + dir1;
      elseif (extraLayer) && (ii == lngth)
        left = left + dirY; right = right + dirY;
      else
        left = left + dir1; right = right + dir2;
      end
    end
  end

end

function newTemplate = mergeTemplateLayers(template)
newTemplate{1} = template{3}{1};
for jj = 1:length(template{1})
  newTemplate{jj+1} = [template{1}{jj}, template{2}{jj},...
    template{3}{jj+1}, template{4}{jj}];
end
end
```

Listing A.3. `solveGroups.m`

```
% Finds the groups for the 'test problem'.

function groups = solveGroups(template, cl, nx, ny, nz, nVars)
% Principal directions for domain displacements
dirX = nVars*cl; dirY = nVars*nx*cl; dirZ = nVars*nx*ny*cl;
dir1 = (dirY + dirX)/2;
dir2 = (dirY-dirX)/2 + dirZ;
dir3 = dirZ;

% Create model problem
positions = [
  0    0      0
  0    0      -1
  0    0      1
  0    -1     0
  0    -1     -1
  0     -1    1
  0     1     0
  0     1     -1
```

```
  0     1     1
  -1    0     0
  -1    0     -1
  -1    0     1
  -1    -1    0
  -1    -1    -1
  -1    -1    1
  -1    1     0
  -1    1     -1
  -1    1     1
  1     0     0
  1     0     -1
  1     0     1
  1     -1    0
  1     -1    -1
  1     -1    1
  1     1     0
  1     1     -1
  1     1     1
  ]*[dir1; dir2; dir3];


% Setup all domains in the test problem
domain = cell(1, length(positions));
for p = 1:length(positions)
  domain{p} = template + positions(p);
end


% Find groups. Note that domain{1} is the domain that we want to solve for
groups = {};
groups{1}.doms = [1]; % Ensure that interior is first group
groups{1}.nodes = [];
grpCntr = 1;
for N = 1:length(domain{1})
  % For each node, first check to which domains it belongs
  node = domain{1}(N);
  listOfDomains = [];
  for D = 1:length(domain)
    if ismember(node,domain{D})
      listOfDomains = [listOfDomains, D];
    end
  end

  newgroup = true;
  % Check if a group already exists for this combination of domains. If
  % not, we have to make a new group (below)
  for G = 1:length(groups)
    if length(groups{G}.doms) == length(listOfDomains)
      if groups{G}.doms == listOfDomains
        newgroup = false;
        groups{G}.nodes = [groups{G}.nodes, node];
      end
    end
```

```
  end

  if (newgroup)
    grpCntr = grpCntr + 1;
    groups{grpCntr}.doms = listOfDomains;
    groups{grpCntr}.nodes = node;
  end
end

% Only keep the nodes field of the struct; doms it not required later
groups = cellfun(@(STRUCT) sort(STRUCT.nodes), groups, ...
  .'UniformOutput', false);

% Separate different node types
numGroups = length(groups);
newGroups = 0;
for i = 2:numGroups %skip interior! (first group = interior)
  j = i + newGroups;
  if length(groups{j}) > 1
    listU = groups{j}(mod(groups{j},nVars) == 0);
    listV = groups{j}(mod(groups{j},nVars) == 1);
    listW = groups{j}(mod(groups{j},nVars) == 2);

    % Update groups; replace groups{j} with 3 separate groups for u,v,w
    groups = [groups(1:j-1), {listU}, {listV}, {listW}, groups(j+1:end)];
    newGroups = newGroups + 2;
  end
end

% Keep only nonempty
groups = groups(~cellfun('isempty', groups));
end
```