



Parallel line identification for line-implicit-solvers

Arne Rempke¹

Received: 28 September 2022 / Accepted: 5 May 2023 © The Author(s) 2023

Abstract

Line-implicit preconditioners are well known in computational fluid dynamics (CFD) solvers and are an essential component to handle meshes with cells of very high aspect ratio (> 1000:1). Such anisotropic cells are commonly used to resolve steep gradients in the boundary layer of a turbulent flow with high Reynolds number. To date, this technique has rarely been used to solve other partial differential equations. We show that the advantages of such preconditioners do not depend on the partial differential equation or discretization used, but also apply to other problems like a node-based mesh deformation with linear elasticity on such meshes. We show the influence of the selection of these lines, and present a new algorithm for identifying lines for line-implicit preconditioners. This new algorithm makes better use of parallel processors and leads to more homogeneous lines. Finally, we see that using the same line-implicit preconditioner, but the new line identification algorithm, even leads to faster convergence for the mesh deformation problem based on linear elasticity.

Keywords Linear algebra \cdot Parallel preconditioner \cdot Line-block preconditioner \cdot Tridiagonal-matrix-algorithm (TDMA) \cdot Block-Thomas-algorithm \cdot Line identification \cdot CFD Solver \cdot Viscous boundary layer mesh \cdot Anisotropic cells \cdot High aspect ratio \cdot Linear elasticity mesh deformation

Mathematics Subject Classification $15\text{-}04 \cdot 65N22 \cdot 65F08 \cdot 05C50$

1 Introduction

When solving the linear equation system Ax = b for a sparse matrix A, a typical ingredient as preconditioner is to apply iterative smoothers. These smoothers reduce

Communicated by Jan Nordström.

Arne Rempke arne.rempke@dlr.de

¹ Institute of Software Methods for Product Virtualization, German Aerospace Center (DLR), Dresden, Germany

the defect $d^{(i)} := b - Ax^{(i)}$ in each iteration *i* by inverting an approximation *P* to the matrix *A*: $x^{(i+1)} := x^{(i)} + P^{-1}d^{(i)}$. When *P* is a part of *A*, *A* – *P* has less nonzero entries, and we have the basic *splitting operator* iteration scheme

$$x^{(i+1)} := P^{-1}(b - (A - P)x^{(i)}).$$
⁽¹⁾

E.g. for a Jacobi/Richardson scheme P is the diagonal part D, for a forward Gauss–Seidel scheme P = L + D with the matrix A = D + L + U split to diagonal, lower and upper triangular matrix parts. For simplicity, we skip any relaxation parameters or Gauss–Seidel-scheme modifications and just concentrate on the basic Jacobi/Richardson scheme in this overview. A simple point/block-implicit preconditioner inverts the diagonal part of the matrix, P = diag(A).

In contrast to this, the tridiagonal or lines inversion chooses a different approximation P for A: here P is a tridiagonal matrix, which still can easily be inverted exactly using the Thomas algorithm or tridiagonal matrix algorithm (TDMA). P is no longer made up of just the diagonal entries of A, but of additionally up to two nonzero entries more per row. This change from point-implicit to line-implicit solvers can be interpreted as moving up to two off-diagonal entries from the explicit defect evaluation on the right-hand side $(b - (A - P)x^{(i)})$ to the implicit inversion (P^{-1}) . Of course the selection of which off-diagonal entries to be moved when splitting the operator A has an influence on the convergence of the method. This choice of the tridiagonal entries can geometrically be described as line identification and will be covered in this work.

Such line-implicit methods have successfully been applied in the context of implicit computational fluid dynamics (CFD), see e.g. [2, 6, 7, 9, 13] for applications using the finite volume methods (FVM), [4] using the Discontinuous Galerkin (DG) method or [8] using the finite element method (FEM). These studies depict a slow convergence or even divergence of the iterative solvers when using point-implicit methods instead of the line-implicit method.

In this paper, we will apply different line identification algorithms to a FEM discretization for linear elasticity. This is of importance in the context of CFD mesh deformation where, following the approach [10, 12], elasticity equations need to be solved. Since the mesh to be deformed is designed for CFD applications with a viscous boundary layer and strongly anisotropic cells, similar numerical problems occur as in CFD which indicate the use of line-implicit solvers may be beneficial. At the same time, there are also some differences between CFD and linear elasticity resulting from the different discretization approaches, which make the transfer not so straightforward. For FEM, in contrast to CFD codes based on FVM or DG, the non-zero blocks of the matrix to be solved are based on the vertices instead of the volumes of the mesh. This leads to significantly more entries in the sparsity pattern, e.g. in a structured hexaeder region each node has 26 neighbors instead of 6 neighbors in FVM, see Fig. 1. Also in FEM in contrast to FVM, computing the quotient of longest by shortest neighbor's distance is no longer equivalent to the aspect ratio of the cell, it is already $\sqrt{3}$ for a perfectly isotropic hexaeder cell.

Due to these differences, it turns out that a straightforward application of the lineimplicit approach used in [2, 4, 6, 7, 9, 13] is not possible for the FEM code. Also the approach of [8] does not work for this application: there line-implicit solvers are



Fig. 1 Direct neighbors (black vertices) of the central vertex (outlined in black) and their distances in an isotropic structured 3d mesh (dashed): Finite Volume (left) vs. Finite Element (right) Method

applied to a FEM discretization, but the line identification is done using the solution of a linear convection-diffusion equation which works well for CFD problems. The linear elasticity problem however is not convection dominated, and therefore this methodology is not applicable. Instead, we modify the methods to identify lines that were already applied in [6, 7] and which are based on the geometric properties of the mesh,¹ and therefore independent of the partial differential equation to be solved. This paper sketches how the transfer can be achieved and what is necessary to make it work, including a new algorithm to identify the lines.

The outline of this paper is as follows: Sect. 2 gives an introduction on what lines in the context of sparse linear equation systems are, and defines important properties in order to operate on a clear vocabulary. In Sect. 3, we will then introduce two algorithms in order to identify such lines. These algorithms are finally applied to the linear elasticity problems for mesh deformation in Sect. 4, compared and the results are discussed. We will end with some final remarks in Sect. 5.

2 Lines in a sparse linear equation system

For the sparse linear equation system Ax = b, we will consider the graph² G = (V, E) with vertices V and edges E where each component of the solution vector represents a vertex and the adjacency matrix has the same sparsity pattern as A - diag(A). We will presume that the sparsity pattern of A is symmetrical—otherwise add explicit zeros— so the graph is undirected. We skip the diagonal entries of A in the adjacency matrix since they are already part of P (and therefore not relevant for line identification) and they would just add debiliating loops at each vertex in the graph.

¹ Note that although Fig. 1 illustrates the case for hexahedral cells, both the line-implicit method and the identification algorithms work on unstructured meshes with many different cell types, including triangular or tetrahedral cells. The illustration is given for hexahedral cells because for CFD meshes the viscous boundary layer often consists of highly anisotropic hexahedral cells.

 $^{^2}$ The terminology used here for graph theory will not be introduced in detail, see e.g. [1] for a general introduction.



On such a graph associated to the linear equation system, we can now define lines, which describe the off-diagonal entries that are additionally inverted in line-implicit solvers.

Definition 1 For an undirected graph G = (V, E), the subset $L \subseteq E$ is called *lines for G*, when

$$|\{e \in L : v \in e\}| \le 2 \quad \forall v \in V,$$

i.e. the maximum degree of (V, L) is 2, and there is no cycle in (V, L).

As an example, consider the matrix

$$A = \begin{bmatrix} 6 - 1 & 6 & & \\ 1 & 6 & 4 & & \\ 4 & 6 - 1 & 3 & & \\ 2 & 1 & 5 & -1 & 5 & \\ 3 & 1 & 2 & -1 & 6 & 1 \\ 5 & 1 & 3 & 2 & \\ & 6 & 7 & 4 & 4 & \\ & & -1 & 2 & 4 & 4 & 1 \\ & & & & 2 & 1 & 5 \end{bmatrix}.$$
 (2)

The corresponding graph G is depicted in Fig. 2. One can easily see that $\{(1, 2)\}$ and $\{(3, 4), (4, 6), (6, 8)\}$ are lines for G, but $\{(1, 4)\}$, $\{(3, 4), (3, 5), (4, 5)\}$ and $\{(5, 7), (7, 8), (7, 9)\}$ are not because $(1, 4) \notin E$, 3,4,5 form a cycle, and the degree of 7 in the subgraph is 3.

For identifying lines that help to solve a given linear equation system, it is essential to have some measure on how strong two vertices are coupled, their willingness to form a line segment. We will formulate this as a weight function $w : E \to (0, \infty)$ for the edges of G. When such a weight function is given, we can compare different lines for the same graph by comparing their weights. Generally it is preferred to achieve a higher weight for a better approximation of the linear equation system by the lines-implicit preconditioner.

Definition 2 For a weighted undirected graph G = (V, E, w) and lines L for the graph (V, E), we define the weight of L as

$$w(L) := \sum_{e \in L} w(e).$$



Another important measure to qualify lines that work well for line-implicit solvers is tridiagonality. This was already recognized in [8].

Definition 3 For an undirected graph G = (V, E) and lines L for G, the lines are called *tridiagonal*, iff there is no $e \in E \setminus L$ such that $(V, L \cup \{e\})$ has a cycle and a maximum degree of ≥ 3 .

In the example, $\{(1, 3), (3, 5)\}$ and $\{(6, 8), (8, 7)\}$ are tridiagonal, because there is no edge that could be added to form a cycle, but $\{(4, 3), (3, 5), (5, 7)\}$ is not tridiagonal because adding $(4, 5) \in E$ forms a cycle and leads to deg(5) = 3 in the subgraph.

If the lines are tridiagonal, any additional connection from $E \setminus L$ only connects different lines, not two vertices of the same line. For the corresponding matrices, this means when cropping the matrices to those entries of a single line, P and A are identical, wheras A would contain additional entries connecting different parts of this line, if the lines were not tridiagonal.

For our example (2), we may use $w(v_i, v_j) = |a_{i,j}| + |a_{j,i}|$ as a weight for each edge. The edges in Fig. 2 represent the weights as thicknesses and Fig. 3 shows two possible identifications of lines for the weighted graph:

$$L_1 := \{(5, 7), (7, 8), (8, 6), (6, 4), (4, 2), (2, 1), (1, 3)\}$$

and

$$L_2 := \{ (1,3), (3,5), (5,7), (7,9), (2,4), (4,6), (6,8) \}.$$

In the next section, we will see, that these lines are actual results from the line identification algorithms introduced there: L_1 is the result of Algorithm 1 and L_2 that of Algorithm 2 for (V, E, w) and $\alpha = 1.^3$ We postpone the detailed discussion on how to determine these lines to Sect. 3, and close this section with a comparison of

 $^{^{3}}$ This parameter for the line identification algorithms will be introduced in Sect. 3 and controls how different the weights need to be in order to form a line.

Choice of L	Р	w(L)	Tridiagonal?	$ \lambda_{\min} $	$ \lambda_{max} $
ø	$P(A, \emptyset) = \operatorname{diag}(A)$	0	Yes	0.107	2.016
L_1	$P(A, L_1)$	52	No	0.0	1.001
<i>L</i> ₂	$P(A, L_2)$	54	Yes	0.0	0.714

 Table 1
 Properties of different line choices and Eigenvalues of the iteration matrix for different splittings of the example matrix A from (2)

the effects of using these two different line selections in the iterative approach (1) of (2).

Definition 4 For a quadratic matrix $A = (a_{i,j})_{i,j=1,...,n}$, the associated graph $G = (\{1, ..., n\}, E)$ and lines L for G, the corresponding tridiagonal matrix $P(A, L) := (p_{i,j})_{i,j=1,...,n}$ is given as

$$p_{ij} := \begin{cases} a_{ij} & \text{if } i = j \text{ or } (i, j) \in L \text{ or } (j, i) \in L \\ 0 & \text{else.} \end{cases}$$

When using the two choices L_1 and L_2 for lines in the lines-implicit solver, we get a much more stable solver for the splitting scheme on A than using the point-implicit approach: Inspecting the Eigenvalues of the iteration matrix $P^{-1}(A - P)$ for various choices of P, we get the results in Table 1. Note that these Eigenvalues should be less than 1 for a stable method. So we can see that for this example, the effect of using lines or not is significant, and the choice L_2 even gives a safe convergence without introducing an additional relaxation parameter. In Sect. 4, we will see more verification of these effects for more complex numerical examples.

3 Line identification

After we have seen the influence of line solvers and the choice of lines, we will now compare different methods to identify lines. Several different proposals for line identifications have been discussed (see [2, 4, 6–8]), and all are provided with an edge-weighted undirected graph G = (V, E, w). The weights give a user-prescribed strength of how strong the coupling between the involved nodes is. Such a coupling can originate from the magnitude of the off-diagonal entry of a matrices (like in the scalar-values example in section 2), or from a geometric information like the inverse Euclidean distance (see [2, 6, 7]):

$$w(v_1, v_2) := \frac{1}{||\text{coord}(v_1) - \text{coord}(v_2)||},$$
(3)

with $coord(v_i)$ the geometric coordinates of node v_i within the mesh. Other approaches use weights from the expected fluxes (see [2]) or the solution of a simpler substitute problem, e.g. the scalar linear convection problem (see [8]). Such weights do not seem

helpful for our application of linear elasticity, and since we need to handle anisotropic mesh properties, we employ the geometric weights (3). The line identification algorithm is then applied to find the strongest differences in these weights per vertex, and to form lines of vertices containing the strongest weighted edges.

Both algorithms presented in this paper employ the anisotropy criterion, which means they compare how much larger than other weights the largest weight of a node is, and if this differs by more than a given ratio, the edge to this neighbor is considered to be a continuation of a line. The minimum ratio of the weights of the edges required for line continuation is the parameter $\alpha > 1$, which is used in both algorithms presented. [6, 7] showed that using the smallest weight as denominator for this ratio works well for finite volume-based schemes and inverse distances as weights. However, since we want to apply the algorithms on a vertex-based finite-element setting, we have significantly more neighbors, since e.g. also vertices on diagonally opposite sides of an element are connected. In general, this leads to much higher differences in the distances (and therefore also weights) even in an isotropic mesh region (factor \sqrt{d} in d dimensions), see Fig. 1 for an illustration. In order to transfer this criterion for this setting and removing the dependency of minimum aspect ratio from dimensionality, we use the third largest weight instead of the smallest weight in order to compute the aspect ratio: since the line is only able to connect at most two neighbors (probably those with the two strongest weights), the third strongest coupling will still be not part of a line. Expecting the largest two connections being part of the line, the third largest weight indicates the distance to the next neighbor that is not part of the line, and therefore works well for vertex and cell-based discretizations and varying dimensions. See Fig. 4 for an illustrating example: although the mesh view is in a quite isotropic region (quadrilaterals have aspect ratios of 1.25:1 or 1.33:1), the original check for this topology would lead to a ratio of $\frac{w(v,n_{v,1})}{w(v,n_{v,7})} \approx 1.67$. The improved check expects line segment to be formed between the nodes $n_{v,1}$, v and $n_{v,2}$, and uses the ratio of strongest expected in-line coupling to strongest expected not-in-line coupling as $\frac{w(v,n_{v,1})}{w(v,n_{v,3})} \approx 1.25$, which is a much better approximation to the aspect ratio of the cells.

3.1 The greedy line identification algorithm

As baseline algorithm and for comparison with the new algorithm, we will start with a slightly modified version of the algorithm used in [6, 7]. The difference consists in the change in the evaluation of the anisotropy criterion as explained in Sect. 3. Note that this change can partly be interpreted as a scaling of the parameter α : scaling α by \sqrt{d} for vertex-based discretizations instead of volume-based dicretizations.

Algorithm 1 gives the formal representation of the line identification algorithm. The aforementioned modification was done in line 16.

Algorithm 1 Depth-first line identification

Input: A weighted undirected graph G = (V, E, w) with vertices V, edges $E \subseteq V \times V$, weights $w : E \to (0, \infty)$, and the parameter $\alpha \ge 1$.

Output: Lines *L* for *G* 1: for all $v \in V$ do



Fig.4 Cells (dashed) and connections (solid) in the local neighborhood of a vertex v from FEM discretization with edge weights proportional to inverse Euclidean distances. The ratio of highest to lowest weight is $\frac{w(v,n_{v,1})}{w(v,n_{v,7})} \approx 1.67$, although the mesh is quite isotropic in this area, where we would like to have a ratio of ≈ 1 . The improved check ratio $\frac{w(v,n_{v,1})}{w(v,n_{v,3})} \approx 1.25$ is a much better approximation for the aspect ratio of the quadrilateral cells

2: Set $k_v := \deg(v)$

3: Sort *v*'s neighbors $\{n_{v,i} : (v, n_{v,i}) \in E\}$ by $w(v, n_{v,i})$:

$$w(v, n_{v,1}) \ge w(v, n_{v,2}) \ge \cdots \ge w(v, n_{v,k_v})$$

```
4: end for
5: Set M := V, L := \emptyset
6: while |M| > 0 do
       Set v := \operatorname{argmax}_{v \in M} w(v, n_{v,1}) / (\frac{1}{k_v} \sum_{i=1}^{k_v} w(v, n_{v,j}))
7:
8:
       Set s := v
9:
       Update M := M \setminus \{v\}
       Set r = 0
10:
11:
       repeat
          Set u := n_{v,1}, i := 1
12:
          while u \notin M and i < k_v do
13:
              Update i := i + 1, u := n_{v,i}
14:
          end while
15:
16:
          if u \in M and w(v, u) \ge \alpha w(v, n_{v, \max(k_v, 3)}) then
              Update L := L \cup \{(v, u)\}
17:
              Update M := M \setminus \{u\}
18:
              Set v := u
19:
          else if r = 0 then
20:
```



```
      21:
      Set r := 1

      22:
      Set v := s

      23:
      else

      24:
      Set r := 2

      25:
      end if

      26:
      until r = 2

      27:
      end while

      28:
      return L
```

Figure 5 shows what happens when Algorithm 1 is applied to the example graph with $\alpha = 1$: node 5 is identified to contain the largest aspect ratio. A line is built from there by following the strongest weights recursively. In this case this directly leads to the full line, no continuation in the reverse direction or additional line identification from a different seeding point is done.

This algorithm generally works well in situations where the decision which edge should be selected for a line is easy: when we generate the same lines regardless of its seeding point or which neighboring nodes are already taken. However, it has problems, when there are conflicts, e.g. because more than 2 adjacent edges per vertex have a large weight, and therefore different lines may be built, depending on which neighbor the line is continued from. It also does not involve any checks for tridiagonality. Due to its overall design, this algorithm can be described as a greedy, sequential or depth-first line-identification.

3.2 The parallel line identification algorithm

This new algorithm allows to identify lines in parallel. Instead of being dependent on the choice of a seed point, and sequentially constructing one line after another, it independently builds small line segments within the whole mesh, potentially all at the same time. With only local knowledge of the neighbors of a vertex, their preferences for line participation and the weights on the edges within this neighborhood, it allows an intrinsically parallel identification of lines. Even though the overall time spent in the line identification at present is typically an insignificant part of solving a linear equation system, achieving a good performance and scalability here still can become more relevant for massively parallel computations.

The basic idea of this algorithm is to start with a greedy approach where each vertex v is free to choose its favourite two neighbors for a line. In a second step, the choices need to be consolidated to ensure just lines and no trees are formed. Afterwards, each vertex knows whether it is part of a line, and if it is, the immediate neighbors within

Fig. 6 Mappings p (solid) and q (dashed) for the example graph after greedy selection



Fig. 7 Mappings p (solid) and q (dashed) for the example graph after consolidation

the line are also known. From these local informations, the full lines just have to be formed by following the already established connections.

The result of the first step can be visualized using two maps: $p: V \rightarrow V$ for the primary choice for a line neighbor and $q: V \rightarrow V$ for the secondary choice. Each vertex may also map to itself to indicate it does not want to use this connection. Since this first greedy mapping does not necessarily fit the requirements for lines, it needs to be updated in a second step. This step tries to solve conflicts where e.g. on a hub more than two edges are indicated to be chosen for a line by their adjacent vertices (e.g. 7 in Fig. 6).

After this consolidation, the two mappings exactly map to the two neighbors within the lines. As seen in Fig. 7 this results in a double-linked list of nodes, and these lists form the actual lines.

Since all the steps until now are done with a very decentral data knowledge (only direct neighbors and possibly their neighbors are involved), there is a problem: we can't ensure that the line is not actually a cycle with no endpoints at all. Either the tridiagonal solver needs to also support cycles (e.g. using the Sherman-Morrison formula), or the cycle needs to be broken. We choose to break the cycle at its weakest coupling here.

Algorithm 2 Bottom-up line identification

Input: A weighted undirected graph G = (V, E, w) with vertices V, edges $E \subseteq V \times V$, weights $w : E \to [0, \infty)$, and the parameter $\alpha \ge 1$.

Output: Lines L for G

1: for all $v \in V$ do

2: Set
$$k_v := \deg(v)$$

3: Sort *v*'s neighbors $\{n_{v,i} : (v, n_{v,i}) \in E\}$ by $w(v, n_{v,i})$:

$$w(v, n_{v,1}) \ge w(v, n_{v,2}) \ge \cdots \ge w(v, n_{v,k_v})$$

4: Set
$$m_v := w(v, n_{v, \max(k_v, 3)})$$

5: Set
$$p(v) := \begin{cases} n_{v,1} & \text{if } w(v, n_{v,1}) \ge \alpha m_v, \\ \dots & \dots & \dots \end{cases}$$

$$v$$
 else

6:
$$S := \{i \in \{2, \dots, k_v\} : w(v, n_{v,i}) \ge \alpha m_v \text{ and } (p(v), n_{v,i}) \notin E\}.$$

```
Set q(v) := \begin{cases} n_{v,\min(S)} & \text{if } S \neq \emptyset, \\ v & \text{else} \end{cases}
7:
 8: end for
 9: for all v \in V do
10:
         Set u := q(v)
11:
         if (u \neq v \text{ and } ((q(u) \neq v \text{ and } p(u) \neq v) \text{ or } (q(u) = v \text{ and } (p(u), p(v)) \in E))
        then
            Update q(v) := v (Lose connection)
12:
         end if
13.
        if q(v) = v then
14:
            S := \{i \in \{2, \dots, k_v\} : p(n_{v,i}) = v \text{ and } (p(v), n_{v,i}) \notin E\}.
Update q(v) := \begin{cases} n_{v,\min(S)} & \text{if } S \neq \emptyset, \\ v & \text{else} \end{cases}
15:
16:
17:
         end if
18: end for
19: for all v \in V do
         if p(v) \neq v and p(p(v)) \neq v and q(p(v)) \neq v then
20:
            Update p(v) := v (Lose connection)
21:
         end if
22.
23: end for
24: Set L := \{(u, v) \in E : (p(u) = v \text{ or } q(u) = v) \text{ and } (p(v) = u \text{ or } q(v) = u)\}
25: for all cycles C = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\} \subset L do
26:
         Set m := \operatorname{argmin}_{e \in C} w(e)
         Update L := L \setminus \{m\}
27.
28: end for
29: return L
```

Algorithm 2 shows the new algorithm. In lines 1–8, the initial preferences for each vertex are computed. The consolidation of conflicts is done in lines 9–23. The gathering of the double-linked lists to full lines is done in line 24. In lines 25–28 the detection and breaking of cycles is done.

The greedy selection of the best neighbors for a vertex v in lines 2–7 is quite simple: the strongest coupled neighbor is selected to be mapped to in p, if the anisotropy criterion, ratio of the weights of at least α , is fulfilled. For the second choice in q, additionally to this anisotropy criterion a local tridiagonality criterion needs to be fulfilled: only those vetices are considered that are not connected to p(v). If no matching vertex is found, the maps are set to point to v itself, which means that this vertex is an ending position and the line does not continue in this direction.

The consolidation is more complex: we start from a vertex v and inspect its second choice vertex u = q(v). In line 12 we remove this second choice in all those cases, where something is wrong: Either because u does not point to v with neither p nor q, or because the link (u, v) is only second choice from both perspectives and the first choices have a direct connection (violate a slightly less local check for tridiagonality). If now the second choice q(v) points back to v, we try to make it point to the strongest coupled neighbor that points to v as primary choice and still fulfill the tridiagonality

criterion (line 15–16). After all second choices have been updated (line 18), the first choices are checked again and potentially updated: if the first choice is not reciprocally answered, it is removed in line 21.

Afterwards we have a double-linked list of nodes for the lines, which is gathered in line 24. If there are any cycles, the weakest connection is just removed in line 27.

Due to the conflict-solving phase of this algorithm, it identifies nice lines even in situations where the lines to be selected are not that clear at the beginning. Note that most parts of the algorithm do not need dynamic (re)allocation of arrays and the "for all" loops can be executed in parallel, which enable performance advantages with current hardware. Due to its overall design, this algorithm can be described as a consolidating, parallel or bottom-up line-identification. *Bottom-up* here means that initially only those line segments stand that are undisputed, while the conflicting segments resolve themselves in later steps, forming longer or more lines if successful, or isolated vertices if not.

4 Results

The line identification algorithms are applied to a mesh deformation method based on linear elasticity analogy described in [10]. It numerically solves the partial differential equation

$$\nabla \cdot (\mathbf{C} \nabla \mathbf{u}) = \mathbf{f} \tag{4}$$

for $\mathbf{u}: \Omega \to \mathbb{R}^3$ with **C** the 4th level strain–stress transformation tensor. The discretization employs a FEM method with 3 degrees of freedom for each vertex of the mesh, thus the assembled matrix has a sparse block structure with blocks of 3×3 scalar entries. The non-zero pattern of these blocks is analyzed by the line identification algorithm. Therefore the scalar entries in Sects. 2 and 3 become dense matrix blocks in this application and the tridiagonal solver is in fact a block-tridiagonal solver.

Regarding the weights of the edges, we can't use the trivial value-based weighting as in the example. Since the differential equations to be solved are not convection dominated, the approaches in [3, 4, 8] also do not seem beneficial. Instead, we use the inverse distance weights as done in [2, 6, 7], since the inspected CFD meshes do contain a viscous boundary layer with cells of high aspect ratio. Also the artificial stiffening of elements according to their volume (see [12]) leads to a very stiff boundary layer and difficult to solve linear equation systems with increased condition number (see [10]).

We demonstrate the effects on 4 different meshes: the RAE2822 and DU210 cases are 2d meshes of airfoils with approximately 65k nodes each and a very structured approach with just quadrilateral elements, see Figs. 10 and 11. The other two test cases are 3d meshes: the Onera M6 wing is meshed with hexahedron and prismatic elements, 122k nodes in total, see Fig. 12. The Common Research Model (CRM) wing has 3.7M nodes and consists of hexahedrons only, see Fig. 13.

The implementation of the line identification as well as the other involved linear solver components are done in Spliss, described in [5].



Fig. 8 Comparison of the run times for identification of lines using both algorithms with different parameters on several meshes. Algorithm 2 is faster and scales better with increasing mesh size

Choice of lines	RAE2822	DU210	M6	CRM
Ø	> 3000	> 3000	> 2000	> 20,000
Algorithm 1 with $\alpha = 1$	237	718	> 2000	>20,000
Algorithm 1 with $\alpha = 1.25$	85	243	1130	88
Algorithm 1 with $\alpha = 1.33$	86	282	1365	83
Algorithm 1 with $\alpha = 2$	100	297	1561	88
Algorithm 1 with $\alpha = 4$	169	440	> 2000	77
Algorithm 2 with $\alpha = 1$	89	259	541	60
Algorithm 2 with $\alpha = 1.25$	88	249	799	86
Algorithm 2 with $\alpha = 1.33$	78	241	1076	85
Algorithm 2 with $\alpha = 2$	100	292	1764	81
Algorithm 2 with $\alpha = 4$	168	430	1740	88

 Table 2
 Iteration counts when applying GMRES(100) preconditioned by a line-based Gauss–Seidel for different choices of lines

Comparing the run times of the different line identification algorithms gives us Fig. 8.⁴ Here you can see that the influence from the parameter α is much smaller than that of the mesh structure and the chosen algorithm. The measurements for these meshes also indicate that both algorithms scale in $O(n^{\gamma})$, with $\gamma \approx 1.25$ for Algorithm 1 and $\gamma \approx 1.05$ for Algorithm 2. So both the absolute run time as well as the scaling behaviour is better for Algorithm 2 compared to Algorithm 1.

⁴ The additional measurement points here come from coarser variants of the M6 mesh and are added to provide a larger range of mesh sizes.



Fig. 9 Distribution of the vertices to lines of different lengths when applying the different line identification algorithms to the M6 case. For large α the portion of vertices in trivial lines of length 0 is large, this portion decreases when decreasing α . Major differences between Algorithms 1 and 2 occur for $\alpha = 1$

In Table 2 the influence of different line identification algorithms and parameter choices on the effectiveness of the line inversion preconditioner is shown. It shows the number of linear iterations when using a GMRES Krylov solver, left-preconditioned by a line-implicit multi-color Gauss–Seidel smoother (see e.g. chapter 12.4.3 in [11]), lowest iteration counts per case highlighted in bold. This smoother applies two Gauss–Seidel sweeps using the iterative line-implicit approach (1) discussed in the previous sections. Except for the CRM case, we request a relative reduction of the residual by 1e-16 for the GMRES outer linear solver. Note that for the CRM case, we only request a reduction by 1e-8 for this solver configuration without utilizing a multigrid solver for better comparison with the smaller meshes. Since the work per iteration is approximately the same for all inspected preconditioners, the number of iterations behaves proportional to the wall clock time.

It is obvious that the point-implicit methods (using empty set of lines) does not work well at all for these meshes. For $\alpha > 1$ the results for the two different line identification algorithms are quite similar. The new bottom-up algorithm often is slightly better, but is not a game changer in these cases. However, we can also note that typically it is beneficial to decrease α . And for $\alpha = 1$, there are very large differences between the two inspected algorithms: while the greedy algorithm has massive problems, we get very smooth extension of the values for larger α for the bottom-up algorithm. For the 3d cases, Algorithm 2 with $\alpha = 1$ turned out the best, and it also produced good results for the 2d cases.

In Fig. 9 the share of nodes in the mesh that are agglomerated to lines of different lengths are shown. A line of length 0 means the vertices are not part of a line at all and this part of the mesh is handled point-implicit. Both algorithms produce almost identical characteristics with only 30% of the nodes actually handled line-implicit for



Fig. 10 Line identification for the RAE2822 airfoil using Algorithm 1 (left) and 2 (right) with $\alpha = 1$. Note the line selections that don't follow the meshes edges but crosses cells diagonally

 $\alpha = 4$. For $\alpha = 1.33$, approximately 80% of the mesh is part of a line (slightly less for the greedy algorithm), and the bottom-up algorithm was able to identify slightly longer lines. For $\alpha = 1$, the characteristics of the lines from both algorithms massively differ: while both algorithms are able to connect almost all vertices to lines, the lines from Algorithm 1 are much longer. The longest identified line contains 726 edges, while for Algorithm 2 much more lines of lengths 10–29 were identified, and the longest line still was shorter than 100 edges.

Note that apart from the algorithmic effects of a better or worse convergence, shorter lines have a better potential for more parallelization, since each line needs to be traversed sequentially when applying the Thomas algorithm. This also means that very inhomogenous line lengths can lead to load imbalances. So also from the point of view of parallel efficiency and scalability, the results from Algorithm 2 look more favourable.

In Figs. 10, 11, 12 and 13 some detailed comparison of the line identifaction in the mesh is shown: both algorithms select lines that cross cells diagonally in the RAE2822 case, which seems to be no problem at all. However, Algorithm 1 connected many line segments that Algorithm 2 left split in separate parts.



Fig. 11 Line identification in the DU210 case using Algorithm 1 (left) and 2 (right) with $\alpha = 1$. Note the artifacts from the greedy line continuation of Algorithm 1

Since the used weight is the same for all algorithms, the lines (region and direction) look quite similar on a global level. However, we can see that the details like simple connections of two line segments or slight modifications at one end can have a huge impact on convergence properties as seen in Table 2. Especially connections around a corner or even connecting two parallel lines at their ends seems to be disadvantageous. These properties correspond to the tridiagonality property introduced in Definition 3.

Algorithm 1 with $\alpha = 1$ behaves particularly unfavorably, because it selects few very long lines which violate the tridiagonality property in at least a few points. However, $\alpha = 1$ is a desirable parameter for two reasons: first, we have seen in Table 2 that this can give the fastest convergence, especially for the 3d test cases. And second, $\alpha = 1$ is a canonical choice and does not cause any dependency from properties of a specific mesh, e.g. the growth rate of the inflation layer or the scaling of different axes. That means that the algorithm could be used parameter-free, which significantly simplifies its usage. In this regard, only Algorithm 2 is able to give reasonable results and should highly be preferred to Algorithm 1.



Fig. 12 Line identification for the Onera M6 wing using Algorithm 1 (left) and 2 (right) with $\alpha = 1$ (top) and $\alpha = 1.33$ (bottom)

5 Conclusion

We demonstrated that line solvers are a helpful ingredient when solving FEM based elasticity equations. They are essential for meshes with high anisotropy, regardless of a cell-based or vertex-based discretization. When comparing different line selections for the same equation system, it seems beneficial for convergence to connect more nodes to lines. However, absurdly long lines revisiting already contained neighbors have a negative influence on convergence. Tridiagonality is an important criterion to ensure this property. We introduced a new parallel line identification algorithm that employs local tridiagonal checks. It identifies lines according to the above mentioned criteria for good convergence. Its line detection runs faster and scales better in parallel than a baseline algorithm, and it allows to connect most vertices to lines which can result in a reduction of iteration counts of 50% until convergence.



Fig. 13 Line identification for the CRM mesh using Algorithm 1 (left) and 2 (right) with $\alpha = 1$ (top) and $\alpha = 1.33$ (bottom). Although it is hard to spot differences between the two algorithms in these plots, the convergence rate for $\alpha = 1$ changes dramatically, see Table 2

Acknowledgements Not Applicable.

Author Contributions All authors contributed to the study conception and design. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. The authors gratefully acknowledge funding from the DLR project COANDA.

Availability of data and materials The datasets generated during and/or analysed during the current study are not publicly available due to unclear publication rights but are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication All authors consent to publish this manuscript.

Human and animal ethics No humans or animals were harmed for this manuscript.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

- Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. Elsevier Science Publishing (1976). ISBN 0-444-19451-7
- Eliasson, P., Weinerfelt, P., Nordström, J.: Application of a line-implicit scheme on stretched unstructured grids. In: Proc. 47th AIAA Aerospace Sciences Meeting (2009)
- Fidkowski, K.J.: A High-order discontinuous Galerkin multigrid solver for aerodynamic applications. Master's thesis, Massachusetts Institute of Technology (2004)
- Fidkowski, K.J., Oliver, T.A., Lu, J., Darmofal, D.L.: p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations. J. Comput. Phys. 207, 92–113 (2005)
- Krzikalla, O., Rempke, A., Bleh, A., Gerhold, T.: Spliss: a sparse linear system solver for transparent integration of emerging HPC technologies into CFD solvers and applications. In: Jahresbericht 2020 zum 22. DGLR-Fach-Symposium der STAB (2020)
- 6. Langer, S.: Application of a line implicit method to fully coupled system of equations for turbulent flow problems. Int. J. Comput. Fluid Dyn. **27**(3), 131–150 (2013)
- Mavriplis, D.J.: Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes. J. Comput. Phys. 145, 141–165 (1998)
- 8. Okusanya, T.O.: Algebraic multigrid for stabilized finite element discretizations of the Navier–Stokes equations. PhD dissertation, M.I.T., Department of Aeronautics and Astronautics (2002)
- Pierce, N.A., Giles, M.B., Jameson, A., Martinelli, L.: Accelerating three-dimensional Navier–Stokes calculations. In: 13th AIAA Computational Fluid Dynamics Conference, no. 1997–1953 in Conference Proceeding Series, AIAA (1997)
- Rempke, A.: Netzdeformation mit Elastizitätsanalogie in multidisziplinärer FlowSimulator-Umgebung. In: Jahresbericht 2016 zum 20. DGLR-Fach-Symposium der STAB (2016)
- 11. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM (2003)
- Stein, K., Tezduyar, T., Benney, R.: Mesh moving techniques for fluid-structure interactions with large displacements. J. Appl. Mech. 70(1), 58–63 (2003)
- Swanson, R.C., Turkel, E., Rossow, C.-C.: Convergence acceleration of Runge–Kutta schemes for solving the Navier–Stokes equations. J. Comput. Phys. 224, 365–388 (2006)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.