J. Parallel Distrib. Comput. 107 (2017) 101–113

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Domain decomposition approach for parallel improvement of tetrahedral meshes

Jianjun Chen [a,b,c], Dawei Zhao [a,b], Yao Zheng [a,b], Yan Xu [a,b,*], Chenfeng Li [c], Jianjing Zheng [a,b]

[a] Center for Engineering and Scientific Computation, Zhejiang University, Hangzhou 310027, China
[b] School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China
[c] Zienkiewicz Centre for Computational Engineering, Swansea University, Swansea SA2 8PP, UK

HIGHLIGHTS

- The proposed algorithm ensures the inter-domain boundary contains no features that might negatively impact element quality.
- The parallel mesh improvement algorithm fixes the inter-domain boundary without compromising mesh quality.
- The parallel algorithm can fully reuse existing sequential mesh improvement codes.
- The parallel algorithm takes only 90 seconds to improve 170 million elements using 128 computer cores in a test.

ARTICLE INFO

Article history:
Received 8 October 2014
Received in revised form
29 April 2016
Accepted 18 April 2017
Available online 27 April 2017

Keywords:
Parallel algorithms
Mesh generation
Quality improvement
Domain decomposition
Dual graph

ABSTRACT

Presently, a tetrahedral mesher based on the Delaunay triangulation approach may outperform a tetrahedral improver based on local smoothing and flip operations by nearly one order in terms of computing time. Parallelization is a feasible way to speed up the improver and enable it to handle large-scale meshes. In this study, a novel domain decomposition approach is proposed for parallel mesh improvement. It analyses the dual graph of the input mesh to build an inter-domain boundary that avoids small dihedral angles and poorly shaped faces. Consequently, the parallel improver can fit this boundary without compromising the mesh quality. Meanwhile, the new method does not involve any inter-processor communications and therefore runs very efficiently. A parallel pre-processing pipeline that combines the proposed improver and existing parallel surface and volume meshers can prepare a quality mesh containing hundreds of millions of elements in minutes. Experiments are presented to show that the developed system is robust and applicable to models of a complication level experienced in industry.

1. Introduction

For numerical simulations with complex geometries, mesh generation typically represents a large portion of the overall analysis time. The ability to perform computations on tetrahedral elements has always been regarded as very important, since the meshing time required for them is usually at least an order of magnitude smaller than for hexahedral or other types of elements. The fundamental reason is related to the fact that tetrahedral elements can be automatically generated for complex geometries using a theoretically guaranteed procedure [43,19,12], while this is not the case for other types of elements. The weakness of tetrahedral elements is that simulations on them may suffer from stability issues and need more elements given a similar node set. However, these difficulties related to solution efficiency and stability have been resolved to some extent, thanks to the rapid advance of high performance computing (HPC) technologies and new numerical methods on tetrahedral elements.

The Delaunay triangulation (DT) [43,19,12,7,37,20,25,38,40,4] and the advancing front technique (AFT) [27,30,14,42] are two of the most successful tetrahedral mesh generation approaches, although both approaches may generate low-quality elements. Firstly, they usually rely on surface inputs, and as a result the quality of a volume mesh is limited by the quality of its surface mesh. Secondly, both approaches are still far away from being perfect. The AFT mainly considers creating an element in each step

* Corresponding author at: School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China.
E-mail address: xyzs@zju.edu.cn (Y. Xu).

http://dx.doi.org/10.1016/j.jpdc.2017.04.008
0743-7315/

of forwarding a front. After a number of front-forwarding steps, the fronts that define the unmeshed region may contain undesired geometry features. Low-quality elements have to be introduced to ensure the termination of the mesh generation process. With respect to the DT based approach, quality guaranteed algorithms have been proposed [38,40]. However, their 3D versions are still problematic due to the issues of sliver elements and boundary integrity. Therefore, an improvement procedure must be followed after calling an AFT or a DT mesher to ensure the mesh quality meets the requirement of downstream simulations.

Although various improvement approaches have been proposed for tetrahedral meshes, the prevailing ones involve two types of local operations. One is smoothing, which repositions mesh points to improve adjacent elements [13,3,11,15,17]. The others are topological transformations [17,24,22,26,32,9,10,18,39], which replace a local mesh with another mesh that fills up the similar region but has different point connections. A general purpose improver usually needs to combine both types of operations and execute them iteratively [17,24]. This process is demonstrated to be very time-consuming, in particular when the simulation needs a quality mesh containing hundreds of millions of elements. In our experience, a sequential Delaunay mesher can now generate one hundred million elements in about ten minutes [28]. Owing to the rapid advance of parallel algorithms, the time cost for a parallel Delaunay mesher can be further reduced to a very low level [2,1,5,41]. However, the following improvement procedure may take several hours to improve this mesh. If a higher standard is set for mesh quality, the time cost for mesh improvement can even grow to be larger than the sequential meshing time by several orders [24]. In this sense, the real performance bottleneck of generating tetrahedral meshes for complicated models lies in the phase of quality improvement rather than mesh generation itself.

Parallelization is a feasible way to speed up the mesh improvement procedure and enable it to handle large-scale meshes, where a prevailing approach is to subdivide the original problem into many subproblems by a domain decomposition (DD) procedure and then solve these subproblems in parallel. The DD tools for parallel solutions of partial differential equations (PDEs) mainly focus on reducing load imbalance and interface communication, and are incompetent to avoid the generation of a poorly shaped inter-domain boundary. If configured with such DD tools, the parallel improver has to introduce time-consuming inter-process operations to improve the elements in the neighbourhood of the inter-domain boundary [2]. Besides, these operations will complicate the process of parallelizing a sequential improver, where an ideal approach is to fully reuse the existing sequential codes as a black box.

In this study, a DD approach [45] that can ensure good geometric properties for the inter-domain boundary is examined for three-dimensional mesh improvement. Instead of directly sending the input mesh to a graph partitioner [31,23,34,35], an intermediate procedure is proposed to simplify the mesh through local operations defined on the dual graphs of the mesh. Partitioning the simplified mesh rather than the initial mesh produces a distributed mesh that not only fulfils the dual goal of balancing loads and minimizing communications, but also contains an inter-domain boundary with desirable geometric shapes. Therefore, the subsequent parallel improvement procedure can fit the boundary without compromising mesh quality, and any sequential improver that respects domain boundary can be reused. Meanwhile, because the improvement procedure involves no communications, its parallel efficiency is very high.

To demonstrate the efficiency and effectiveness of the DD approach, the open-source mesh improver Grummp [17,33] is revised and incorporated. The DD approach is parallelized to accommodate large-scale meshes. Furthermore, a parallel preprocessing pipeline that combines this improver and the existing parallel surface and volume meshers [5,45] is set up, which can reduce the time for preparing a mesh that contains hundreds of millions of elements from hours (for a sequential pipeline) to minutes.

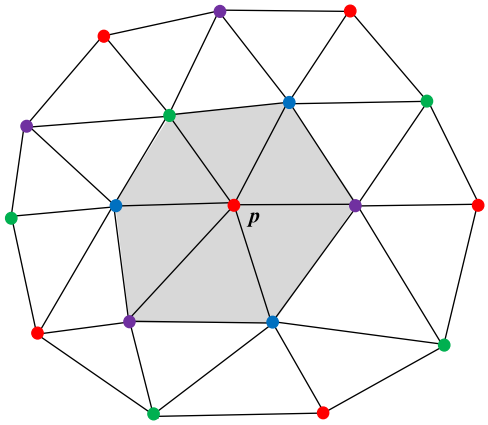


Fig. 1. Independent sets of mesh points for parallel mesh smoothing. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

2. Related work

Parallel meshing algorithms have been extensively investigated in the literature [2,1,5,41,45,29], but much fewer algorithms have been reported on parallel mesh improvement. In general, the existing approaches for parallel mesh improvement could be classified into two types: algorithm parallel approaches [15,16] and problem parallel approaches [5,29,21].

The pioneering work for algorithm parallel approaches was conducted by Freitag et al. [15]. Their parallel smoothing algorithm considers the region covered by elements adjacent to one single mesh point as an individual *submesh*, e.g., the shaded elements around a mesh point p in Fig. 1. In order to avoid the synchronization costs required by the operations of repositioning adjacent mesh points, the mesh points are classified into many *independent sets*. The points belonging to a similar set must not be adjacent to each other, as shown in Fig. 1, where mesh points of different independent sets are differently coloured. Based on independent sets, the smoothing procedure is rescheduled as Algorithm 1, where the main computation lies in the inner loop described by Lines 5 and 6. This computation is parallelizable because the smoothing function callings in Line 6 can be executed concurrently.

For most applications, smoothing is usually not sufficient to ensure a qualified mesh. Topological transformations must be included in the improvement procedure [17,24]. It is possible to extend the concept of independent sets for topological operations. Freitag et al. demonstrated the extension schemes for the 2D edge flip and edge bisection [16]. But to our knowledge, no extension schemes for 3D flips have been reported so far.

Algorithm 1. The mesh smoothing algorithm based on independent sets.

1. $k = 0$
2. Let S_0 be the initial set of mesh points marked for smoothing
3. **while** $S_k \neq \emptyset$
4. Choose an independent set I from S_k
5. **for** each $v \in I$
6. $\mathbf{x}'_v = \text{smooth}(\mathbf{x}_v, \mathbf{x}_{adj(v)})$
7. $S_{k+1} = S_k \setminus I$
8. $k = k + 1$

On a shared memory computer, the schemes like Algorithm 1 can be parallelized easily. For instance, if OpenMP is adopted, a line of codes '#pragma omp parallel for' before Line 5 of Algorithm 1 will dispatch concurrent tasks onto available threads. Besides, a synchronization barrier is required after Line 8 to ensure all threads can enter the smoothing step for the next independent set simultaneously. Although it is possible to tailor the parallel

schemes like Algorithm 1 onto a distributed environment, the tailoring process is rather complicated. Here, the main difficulty is to ensure:

(1) The workloads are well distributed on all processes;
(2) The mesh is well distributed so that a process does not need to query global data when conducting the workloads distributed on this process.

Two extreme options exist to meet the above goals when switching from one independent set to another or from one local operation to another: repartitioning the mesh or fitting a copy of the mesh into all processes. For parallel smoothing, Freitag et al. suggested a trade-off strategy [15]. It records workload distributions with a coloured *vertex adjacency graph* and fits this graph into each process. Before switching into another independent set, each process queries the new workload distributions and sends the mesh data it owns to other processes in need. However, two more difficulties arise when extending this strategy to topological transformations. Firstly, more dual graphs of the mesh need to be defined, which complicates the design of a suitable distributed data structure. In addition, since the mesh topology is updated after transformations, it is more difficult to maintain a coherent mesh on all processes.

Furthermore, the recent advance of mesh improvement techniques sets more obstacles for the above parallel strategy, where more aggressive local operations have been suggested such as *aggressive* vertex insertion [24], small polyhedron reconnection (SPR) [4,26] and recursive flips [22,39,46]. Unlike smoothing where a submesh can be predetermined by clustering all elements adjacent to a point, these operations impact more elements, and the affected elements are determined in the run time.

The parallel algorithm based on independent sets can improve the mesh quality to a similar level, and it is possible to execute a variant of this algorithm on GPU [36,8]. However, the drawbacks of this algorithm are also evident. Firstly, its implementation on distributed platforms is far more complicated. Secondly, this algorithm cannot reuse the sequential algorithm as a black box because no general schemes can apply the concept of independent sets for all local operations. In contrast, these schemes have to be revised case by case. Thirdly, it remains an outstanding issue to extend the method for local operations where the affected elements are determined dynamically [24,22,26,39,46].

In the light of the above issues in relation with algorithm parallel approaches, problem parallel approaches are preferred in some other studies for their ability to employ sequential algorithms as a black box [29,21]. In these studies, the meshes to be improved (in most cases, these meshes are the outputs of a parallel mesher) are usually subdivided into the same number of submeshes as the number of parallel processes involved in the mesh improvement task. Then, the input meshes could be improved concurrently by employing the sequential mesh improvement algorithm on each submesh with the inter-domain boundary fixed. However, the main issue is that tetrahedral elements may not be in shape near the inter-domain boundary. A possible solution to this issue is to introduce inter-process local operations to improve the elements in the neighbourhood of the inter-domain boundary, based on the same idea as that introduced by Chrisochoides et al. for their parallel Delaunay mesher [6]. These inter-process operations could be time-consuming because they involve a huge amount of communication and synchronization costs, not to mention the complication of their implementation. As a compromise, Ito et al. suggested a two-stage strategy to deal with this issue [21]. In the first stage, the submeshes are improved concurrently with the inter-domain boundary fixed. In the second stage, a few layers of elements adjacent to the inter-domain boundary are collected into a single mesh, and then this single mesh is improved sequentially. Evidently, the second stage could become a performance bottleneck due to its sequential nature. Differently, Löhner suggested redistributing the submeshes after the first pass of mesh improvement and then performing a second pass of mesh improvement on the redistributed submeshes [29]. The redistribution algorithm takes the first distribution as a starting point, and then adds 1–2 extra layers of elements to each domain (idomn) from the neighbouring domains (jdomn) for which idomn $<$ jdomn.

In this study, a problem parallel approach is proposed for quality improvement of large-scale meshes on distributed memory computers, which partitions the input mesh into many submeshes and then improves these submeshes in parallel by fixing the inter-domain boundary. So doing avoids the drawbacks of the algorithm parallel algorithm mentioned above. Compared with those existing problem parallel approaches [29,21], the proposed algorithm ensures the inter-domain boundary contains no features that might negatively impact element quality. Therefore, it provides a better guarantee on the quality of elements near the inter-domain boundary. Meanwhile, the proposed algorithm is more efficient because it does not need a second mesh improvement pass.

3. The domain decomposition approach

3.1. Basic terms and definitions

Definition 1 (*Non-overlapping Mesh*)**.**
A non-overlapping d-dimensional mesh is a group of mesh entities whose dimensions range from 0 to d, i.e.

$$M = \{E^i | i = 0, 1, \ldots, d\},$$

where

$$E^i = \{e^i_k | k = 0, 1, \ldots, n_i\} \quad (i = 0, 1, \ldots, d)$$

refers to the set of i-dimensional mesh entities, n_i is the cardinality of E^i, and

(1) $\forall e^i_k, e^i_l \in E^i (0 \le i \le d)$, e^i_k does not intersect e^i_l, or intersect e^i_l at a mesh entity whose dimension is less than i;
(2) $\forall e^{i-1}_k \in E^{i-1} (0 < i \le d)$, e^{i-1}_k must be a boundary entity of one or more i-dimensional entities.

In a non-overlapping mesh, elements and sides refer to mesh entities with $i = d$ and $i = d - 1$, respectively. If an $(i-1)$-dimensional entity e^{i-1}_l is the boundary entity of an i-dimensional entity e^i_k, then name e^i_k the domain entity of e^{i-1}_l. Each mesh side may have one or two domain entities. If the side has one domain entity, it is a boundary side; otherwise, it is an interior side. Two mesh entities e^i_k and e^i_l are neighbours if both of them are the domain entities of an $(i-1)$-dimensional entity e^{i-1}_m.

More strictly, the sides of a 3D mesh must be defined by co-planar vertices in order to define the angles adjacent to neighbouring sides accurately. If two boundary sides of an element are neighbours, the angle they form interior of the element is the interior angle of the element.

Definition 2 (*i-Dimensional Dual Graph*)**.** Given a d-dimensional non-overlapping mesh

$$M = \{E^i | i = 0, 1, \ldots, d\},$$

an i-dimensional dual graph $(1 \le i \le d)$ of M is denoted as $G = \{V, A\}$, where V is the set of graph nodes, and A is the set of graph edges. Each graph node $v_k \in V$ corresponds to one mesh entity e^i_k in M, i.e. $v_k = \phi_v(e^i_k)$, and $e^i_k = \phi_v^{-1}(v_k)$.

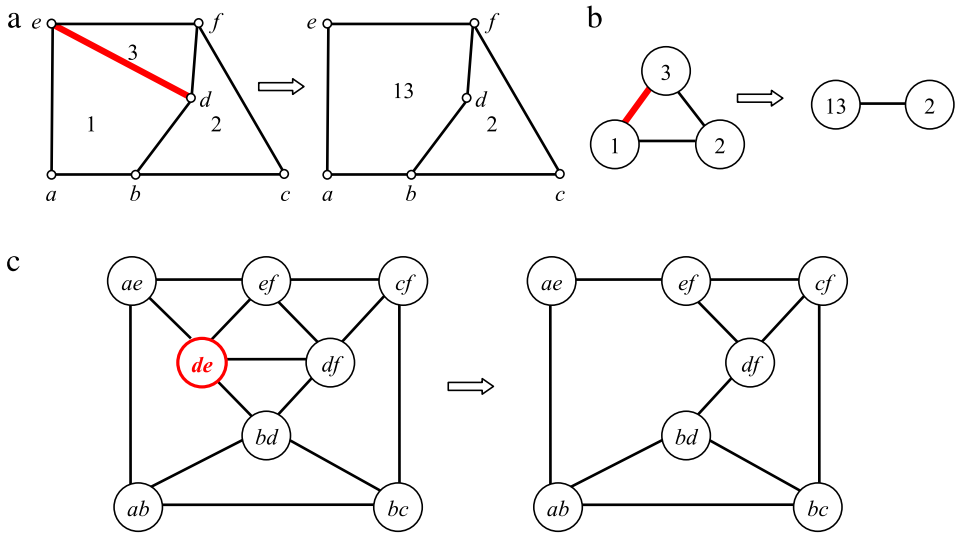


Fig. 2. Meshes and their dual graphs. (a) A side-deletion operation simplifies the mesh to a new one that has fewer small angles. (b) An edge-contraction operation transforms the EDG dual to the original mesh to a dual of the simplified mesh. (c) A node-deletion operation transforms the SDG dual to original mesh to a dual of the simplified mesh.

If two mesh entities of M, e_k^i and e_l^i, are neighbours across a shared boundary entity e_m^{i-1}, a graph edge $a_{kl} \in A$ exists, and

$$a_{kl} = \langle v_k = \phi_v(e_k^i),\ v_l = \phi_v(e_l^i)\rangle.$$

Particularly, a_{kl} is a graph edge classified on the mesh entity e_m^{i-1}.

The nodes and edges of a dual graph may be weighted for some application-specific purposes. Assuming that $w_v(v)(v \in V)$ and $w_a(a)(a \in A)$ are the node and edge weight functions, respectively, the weighted dual graph is a tetrad, i.e.

$$G = \{V, A, w_v(v), w_a(a)\}.$$

With respect to Definition 2, it needs to be emphasized that a graph is a pure combinatorial object. The term i-dimensional dual graph is an abbreviation for the dual graph with its graph nodes and edges respectively classified on the i-dimensional and $(i-1)$-dimensional mesh entities. It does not mean the graph defined in this study is attached with any geometrical information.

According to Definition 2, the **S**ide **D**ual **G**raph (**SDG**) and **E**lement **D**ual **G**raph (**EDG**) are two specific cases of the dual graphs of a mesh, i.e. the $(d-1)$-dimensional and d-dimensional dual graphs of the mesh, respectively.

Definition 3 (*Node Deletion*)**.** Node deletion defined on a graph $G = \{V, A\}$ is a local modification operation with respect to a node v_k of G. After deletion, a new graph $G' = \{V', A'\}$ is generated. The set of new graph nodes is defined as $V' = V \setminus \{v_k\}$. The set of new graph edges is defined by removing all of the edges in G that are adjacent to v_k.

Definition 4 (*Edge Contraction*)**.** Edge contraction is a local modification operation defined on a weighted graph $G = \{V, A, w_v(v), w_a(a)\}$ with respect to an edge $a_{kl} = \langle v_k, v_l\rangle$ of G. After contraction, a new graph $G' = \{V', A', w'_v(v'), w'_a(a')\}$ is generated. The set of new graph nodes is defined as $V' = V \setminus \{v_k, v_l\} \cup \{v'_m\}$, where v'_m is a new graph node, and $w'_v(v'_m) = w_v(v_k) + w_v(v_l)$. The set of new graph edges is defined as follows: the edge a_{kl} is removed; and each edge $a_{kn} = \langle v_k, v_n\rangle$ (or $a_{ln} = \langle v_l, v_n\rangle$) that is adjacent to v_k (or v_l) and another graph node v_n is removed, and instead, an edge $a'_{mn} = \langle v'_m, v_n\rangle$ is inserted. In the case that both v_k and v_l are adjacent to v_n,

$$w'_a(a'_{mn}) = w_a(a_{kn}) + w_a(a_{ln});$$

otherwise

$$w'_a(a'_{mn}) = w_a(a_{kn}) (\text{or } w'_a(a'_{mn}) = w_a(a_{ln})).$$

Fig. 2 presents a simple example to explain the above definitions. The left part of Fig. 2(a) is a 2D non-overlapping mesh, which consists of one triangular and two quadrilateral elements. The left parts of Fig. 2(b) and (c) correspond to the EDG and SDG of this mesh, respectively. Only the EDG is weighted, and its node and edge weights are initially set to be 1.0. The right part of Fig. 2(a) shows the simplified mesh obtained by deleting the side *de*, which results in the deletion of some small angles as well. Consequently, the elements 1 and 3 are merged to a new pentagonal element, labelled as 13. To obtain the respective duals of the simplified mesh, an edge-contraction operation and a node-deletion operation are conducted on the EDG and SDG, as illustrated in Fig. 2(b) and (c).

3.2. The domain decomposition flowchart

Fig. 3 presents the basic flowchart of the proposed DD approach, where the frames, solid arrows and dotted arrows represent the data, algorithms and inputs of the algorithms, respectively. The input is a non-overlapping mesh composed of any type of polyhedral or polygonal elements (a tetrahedral mesh in this study). The output is a partitioned result of the input mesh, where no subdomains (submeshes) contain artificial small angles and poorly shaped sides in their boundaries. If the input mesh is appropriate, the dual goal of load balancing and communication minimization is achieved at the same time.

The intermediate steps that connect the input and output are as follows.

(1) Build the SDG and EDG of the input mesh.
(2) Identify mesh sides with undesirable shapes as *removable*, and simplify the SDG by deleting the graph nodes dual to the removable sides.
(3) Simplify the SDG by deleting some graph nodes and identify the mesh sides they correspond to as removable, to ensure that the mesh without these sides contains no interior angles smaller than a predefined threshold.
(4) Simplify the EDG by contracting all of the edges that are identified as removable in Steps 2 and 3.
(5) Decompose the simplified EDG using a graph-partitioning tool, aimed at the dual goal of load balancing and communication minimization. In this study, the open-source tools Metis [31,23] and its parallel counterpart ParMetis [34,35] are employed for graph partitioning.

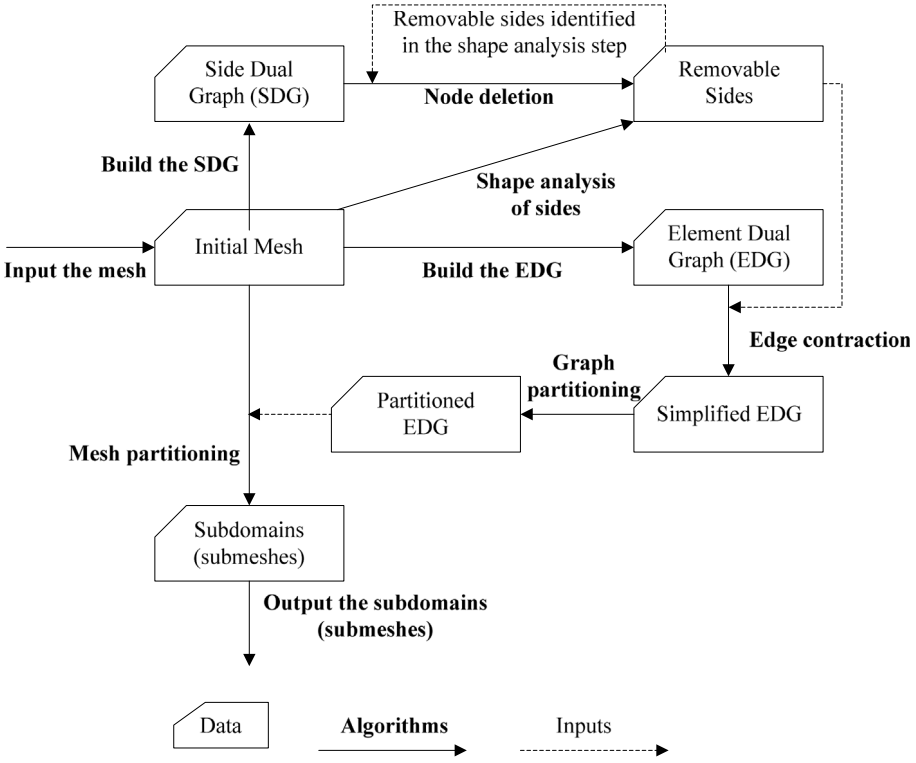


Fig. 3. The flowchart of the proposed domain decomposition approach.

(6) Based on the dual relation between the simplified mesh and its EDG, decompose the input mesh into subdomains according to the partitioning result of the simplified EDG.

3.3. Shape analysis of mesh sides

Step 2 of the proposed DD approach identifies mesh sides with undesirable shapes and removes their counterparts from the SDG. It depends on the specific application purpose to label which mesh sides as undesirable. For tetrahedral mesh improvement, mesh faces with small interior angles are not favoured to appear on the inter-domain boundary because badly shaped volume elements are usually incident to these faces. Therefore, given a user specified threshold α_{th}, we check the minimal interior angle of each interior face. If this value is smaller than α_{th}, the face is identified as removable. Correspondingly, the SDG node dual to this face is deleted.

3.4. Node deletion of the SDG

Step 3 of the proposed DD approach can ensure that no small interior angles (except those incident to two boundary sides) will survive in the simplified mesh by deleting a minimal number of mesh sides. To clarify it, two definitions are presented as follows.

Definition 5 (*Weighted SDG*)**.** Given a d-dimensional non-overlapping mesh M and a predefined angle threshold β_{th}, the weighted SDG is the $(d-1)$-dimensional dual graph of M, with its nodes and edges weighted as follows.

(1) For each graph edge, its weight equals the interior angle bounded by two sides that the end nodes of the graph edge represent. In the case that there are two such angles (for example, in the right of Fig. 2(a), both angles bounded by bd and df are interior angles), the smaller value is chosen.

(2) For each graph node, if it corresponds to a boundary side, its weight equals zero; otherwise, its weight equals the number of its incident edges whose weights are less than β_{th}.

Here, a graph node with a non-zero weight is an *unqualified node*, and an SDG containing unqualified nodes is an *unqualified SDG*, which corresponds to a mesh having small interior angles.

Because Definition 3 only applies for the unweighted graph, a new definition is introduced for the node-deletion operation of a weighted SDG.

Definition 6 (*Node Deletion of a Weighted SDG*)**.** Node deletion defined on a weighted SDG is a local modification operation related to a graph node v_k. The graph topologies are updated as in Definition 3. The weights of remaining nodes and edges are kept unchanged except for the following particular cases:

(1) For each unqualified node adjacent to v_k, its weight is decreased by 1.
(2) For each graph edge ended with the nodes adjacent to v_k, if it is the only edge classified on one $(d-2)$-dimensional mesh entity (d is the mesh dimension), reset its weight as the smaller one of w_e and $2\pi - w_e$, where w_e is the original edge weight.

Fig. 4 illustrates the above definitions with the mesh examples shown in Fig. 2(a). It is supposed that three angles are less than β_{th} in the unsimplified mesh, i.e., $\angle dfc$, $\angle fde$ and $\angle def$. The interior sides that bound these angles are de and df, and either of the sides is incident to two of the three small angles. Therefore, the graph nodes dual to both sides are weighted to be 2, and the other nodes are all weighted to be 0. After the graph node de is removed, the edges incident to de are removed as well, and the weight of the node df decreases from 2 to 1. In the meantime, the graph edge ending with bd and df becomes the unique edge classified on the mesh point d; therefore, the weight of this edge is set to be $\angle fdb$, which is the smaller one of the two angles formed by the mesh edges bd and df.

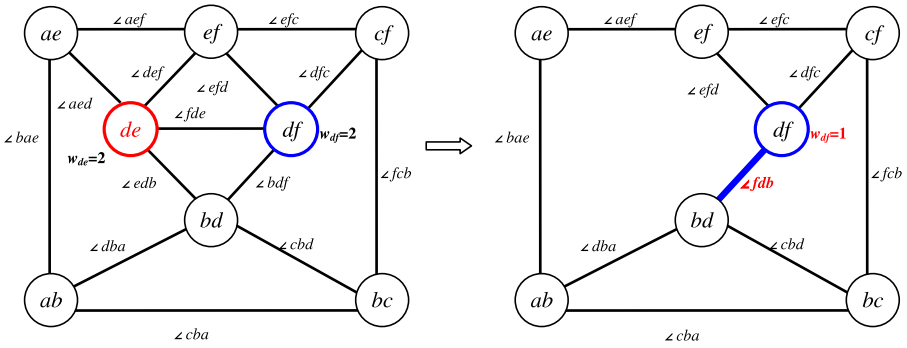


Fig. 4. Illustration for the weighted SDG and the node-deletion operation defined on this graph. The dual meshes are shown in Fig. 2(a).

The input of the node-deletion step is an unqualified SDG, and the output is a subgraph of the input that contains only qualified nodes. The simplest solution is to delete all unqualified nodes. However, the preferred solution shall attempt to delete as few graph nodes as possible:

(1) For the interior sides that form less-than-threshold angles with boundary sides, delete their corresponding unqualified nodes in the SDG.
(2) Initialize a priority queue for the remaining unqualified nodes in a descending order of the node weights, and repeat the following steps until the queue is empty:
   (a) Remove the head node of the queue by a node-deletion operation; consequently, the weights of adjacent unqualified nodes decrease by 1.
   (b) If the nodes adjacent to the head node become qualified ones, remove them from the priority queue; otherwise, insert them in the right positions of the priority queue.
(3) Label the interior mesh sides dual to the deleted graph nodes as removable.

3.5. Edge contraction of the EDG

Step 4 of the proposed DD approach inputs the EDG of the input mesh, and outputs the EDG of the simplified mesh. The EDG is also a weighted graph, and the weighting strategy depends on the application purposes. For mesh improvement, the weights are all initialized to be 1. Instead of sending the EDG that corresponds to the input mesh for partitioning, a variant of this EDG is needed to prevent removable sides recognized in Steps 2 and 3 from appearing in the inter-domain boundary. Here, two types of variant EDGs are considered.

(1) The first one is generated by *penalizing* the dual graph edges of removable sides with very large weights [5]. Because the graph partitioner intends to minimize the size of the cutting edge weights [31,23,34,35], it may prevent these penalized edges from becoming cutting edges.
(2) The second one is generated by contracting the dual graph edges of the removable sides. For the initial EDG, each graph edge only represents one mesh side. The graph edge is identified as *contractable* if the corresponding mesh side is removable. However, in the intermediate process of edge contraction, one graph edge may represent more than one mesh side. The graph edge is identified as contractable if any of these sides are removable. For instance, the graph edge between the nodes 13 and 2 shown in Fig. 2(b) represents the mesh sides *bd* and *df*, and if either of the nodes is removable, the graph edge will be contracted later.

It is evident that the setup of the first type of EDG is much simpler. Nevertheless, the following consideration justifies why the second one is selected as the default. The graph partitioner does intend to minimize the weights of cutting edges, but its performance highly relies on the input. If a small percentage of edges have very large weights, the partitioner will work as expected; otherwise, the partitioner may occasionally fail to prevent these edges from becoming cutting edges. When the partitioner works abnormally, the goal of minimization of cutting edge weights may lead to an undesirable result because the punitive weights stray a lot from the actual meaning of the edge weights.

3.6. A 2D example

Fig. 5 shows the DD result of a 2D pipe model. The input mesh shown in Fig. 5(a) is generated through Delaunay triangulation with constrained points on the model boundary. As this is a 2D mesh, Step 2 of the proposed DD flowchart is skipped over. Mesh edges and faces are weighted by their lengths and areas, respectively. The angle threshold β_{th} is set to be 80° to ensure that no interior angles of the simplified mesh are smaller than 80°, and therefore no dihedral angles on the inter-domain boundary (referred to as the inter-domain dihedral angle hereafter) are smaller than 80° after decomposing the simplified mesh into 8 subdomains. Note that the subdomains have roughly equal areas because the built-in graph partitioning tool always attempts to balance the loads between different subdomains.

4. Parallel mesh improvement

4.1. Parallel domain decomposition

The input of a parallel mesh improver may be a large-scale mesh composed of many submeshes that are distributed on different computer nodes. An option is to first assimilate the submeshes and then partition the assimilated mesh by using a sequential domain decomposer. However, this approach is unsuitable, if storing the entire mesh is beyond the accessible memory size of a single computer node or the computational scalability is a primary focus. Instead, a parallel domain decomposer is necessary in this circumstance.

The node-deletion and edge-contraction operations discussed in Sections 3.4 and 3.5 need to access mesh data incident to one mesh side. Therefore, if both operations are executed in the input submeshes concurrently, and the sides they access are inter-domain sides, inter-processor operations are necessary to ensure a qualified output. To avoid implementing these time-consuming inter-processor operations, an alternative scheme is proposed as follows:

(1) Simplify the EDG of each submesh concurrently by labelling inter-domain sides as boundary sides.

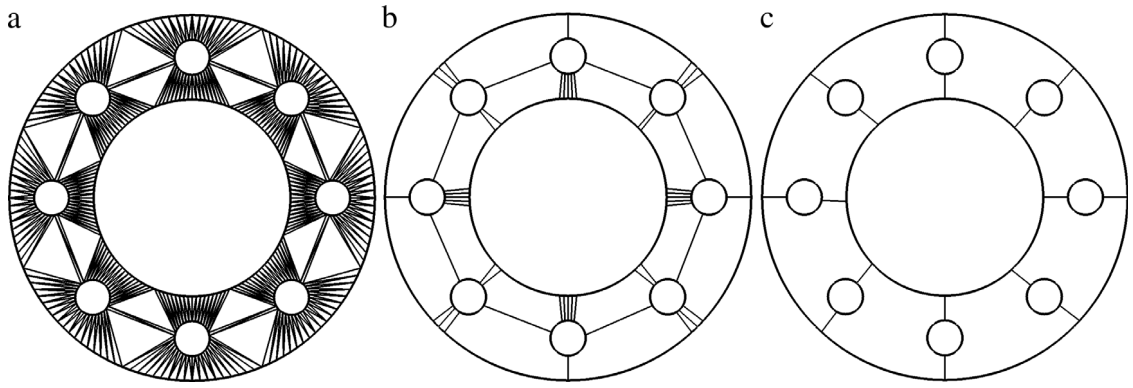


Fig. 5. A 2D example to illustrate the proposed DD approach, where no angles smaller than 80° are formed on the inter-domain boundary. (a) The input mesh. (b) The simplified mesh. (c) The DD result of the simplified mesh.

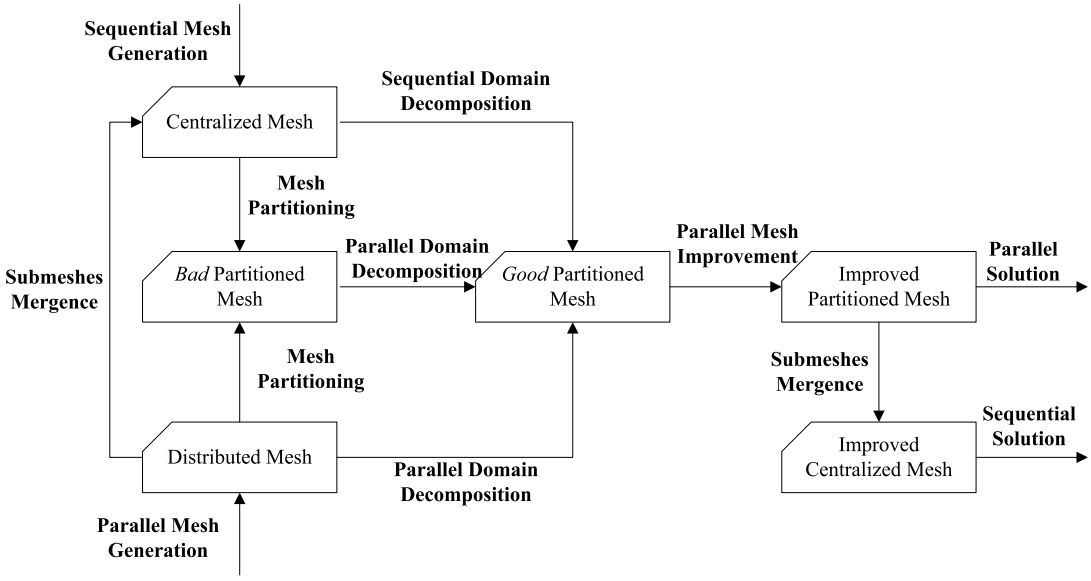


Fig. 6. The flowchart of the proposed parallel mesh improvement approach.

(2) For those inter-domain sides with undesirable shapes or those bounding small dihedral angles, *penalize* their dual EDG edges with large weights.
(3) Repartition the EDG by ParMetis [34,35].
(4) Redistribute the submeshes to comply with the graph partitioning result.

In the above scheme, an optional repartitioning step by ParMetis needs to be executed first if the input mesh is not well balanced or the number of inter-domain sides is too large. The better-balanced submeshes will benefit the timing performance of the subsequent parallel steps. Meanwhile, if the number of inter-domains sides is minimized, the possibility of an abnormal output due to the punitive weighting strategy is minimized as well.

In addition, if the input is a *centralized mesh*, i.e., a mesh stored in the private memory space of a single computer node, it can be repartitioned by Metis [31,23] first and then input to the proposed parallel DD scheme.

4.2. Flowchart of the parallel mesh improvement approach

Fig. 6 presents the proposed parallel mesh improvement approach. If the input is a distributed mesh, it can be split into a *good* partitioned mesh that contains no small dihedral angles and poorly shaped faces on the inter-domain boundary by directly using the parallel DD scheme, or as we suggest in most cases, employing the parallel scheme after a repartitioning step using ParMetis. If the user requires, the distributed mesh can also be merged into one centralized mesh, and then partitioned sequentially. If the input is a centralized mesh, a sequential DD scheme is adopted, or the parallel DD scheme is adopted after partitioning the centralized mesh first using Metis.

The number of submeshes output by the proposed DD approach is set to be equal to the number of computer cores in default. Therefore, after DD, a simple one-to-one mapping scheme is adopted to distribute the submeshes onto available computer cores. Any sequential improver can be fully reused in the subsequent step if it respects the inter-domain mesh. Each submesh is improved individually without communication or synchronization. The negative effect of a fixed inter-domain boundary on element quality is minimized because this boundary contains no small dihedral angles or poorly shaped faces. Moreover, the built-in graph partitioning tool of the proposed DD scheme ensures the sizes of the output submeshes are well balanced. Therefore, the mesh improvement procedure can be speeded up remarkably.

4.3. The sequential mesh improver

The generic flowchart shown in Fig. 6 can be utilized to improve any types of elements. In this study, we only consider tetrahedral mesh improvement, for which the open-source mesh improver Grummp (V0.3.4) is revised and incorporated.

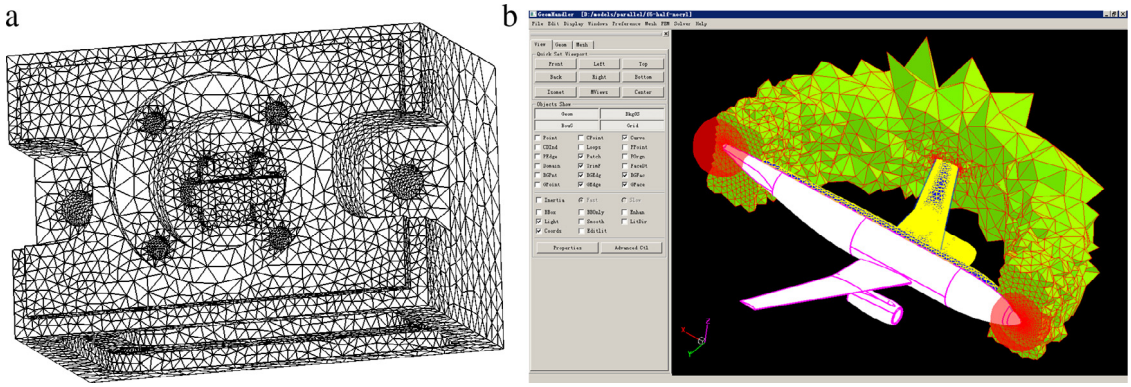


Fig. 7. Two selected models. (a) The surface mesh of part. (b) Visual representation of the geometry, sources and the coarsest surface and volume meshes of the F6 aircraft in the graphical user interface of HEDP/PRE.

Grummp is a cost-effective tetrahedral mesh improver that combines smoothing and topological operations [17,33]. These operations do not intend to remove redundant points or insert points to refine the mesh. Therefore, if the input mesh conforms to a well-graded sizing function, as those generated by a state-of-the-art tetrahedral mesher, Grummp can improve it to an acceptable level for simulations in most cases. Occasionally, Grummp may fail when many poorly shaped elements meet locally. To overcome this drawback, the SPR technique [4,26] is complementarily included. Given an empty polyhedron, the SPR routine seeks an optimal mesh by connecting the boundary nodes of this polygon among all the possibilities. This exhaustive search nature explains why the SPR routine can be more successful in some cases than the flips implemented in Grummp. However, the SPR routine is more expensive than the flips in terms of computing time although many enhancement techniques have been suggested [4,26]. Therefore, in the revised version of Grummp, the SPR routine only targets the elements that survive the original Grummp and have the worst shapes.

4.4. Parallel pre-processing pipeline

Given a CAD model and an element sizing function, it takes the following steps to prepare a large-scale tetrahedral mesh in parallel:

(1) Generate the surface mesh in parallel [45].
(2) Decompose the volume domain represented by the surface mesh into many subdomains by inserting inter-domain faces inside the domain [5].
(3) Mesh subdomains in parallel by employing a Delaunay mesher that respects subdomain boundaries in a constrained manner [4,5,46].
(4) Repartition the volume mesh using the proposed parallel DD approach, and then improve the repartitioned volume mesh in parallel.

The only sequential part of the above parallel pipeline lies in Step 2, where the surface mesh generated in parallel is unified and sent to a sequential DD procedure for parallel tetrahedral mesh generation [5]. The running time of this sequential procedure is proportional to the surface mesh size, which is lower than the volume mesh size by several orders.

In general, the parallel pipeline runs very efficient, and only consumes minutes to prepare a mesh containing hundreds of millions of elements, as we will demonstrate in the next section.

5. Numerical results

All numerical experiments presented below were conducted on the TH-1A system managed by the National Supercomputer Center in Tianjin (http://www.nscc-tj.gov.cn/en/), China. Each node contains two six-core CPUs whose clock frequency is 2.93 GHz, and the local memory for each node is 24 GB.

5.1. Experiments on the parallel improver

5.1.1. The selected geometries

In the experiments of this subsection, a mechanical part model (referred to as Part hereafter) and an exterior flow simulation model of the DLR-F6 wing–body-nacelle-pylon aircraft (referred to as F6 hereafter) are selected, shown in Fig. 7(a) and (b), respectively.

The volume meshes of both models are generated by either a Delaunay mesher [4] or its parallel version [5]. Both meshers are components of an in-house pre-processing system HEDP/PRE [44]. Grid sources are configured for F6 to generate small elements locally that are required for a better resolution of some geometrical and physical details. Four pairs of surface and volume meshes with various sizes are generated for F6 by adjusting a factor s to scale the spacing values of sources. Accordingly, the meshes generated with the scale factor s are named F6-s. The two smallest volume meshes can be improved sequentially, whereas the other two can only be improved in parallel because the data structure adopted by Grummp is rather memory-consuming.

5.1.2. Performance data of the DD approach

In the proposed DD approach, the thresholds α_{th} and β_{th} are set to limit the interior angles of inter-domain faces and the inter-domain dihedral angles, respectively. They have direct impacts on the DD results. Much bigger thresholds are preferred to provide a better shaped inter-domain boundary; however, they also result in a smaller EDG because more edges of the EDG are contracted. According to Definition 2, the weights of some EDG nodes and edges increase along with edge-contraction operations. Therefore, if too many edges are contracted, the magnitude of the EDG decreases rapidly, and the node and edge weights become highly imbalanced. An overly simplified EDG will set obstacles for achieving a well-balanced partitioning, whereas this goal is mandatory to an efficient implementation of the proposed parallel improvement algorithm.

Here, an experiment is designed to verify whether an initial mesh output by a tetrahedral mesher can be decomposed into enough well balanced partitions. This is done by setting acceptable

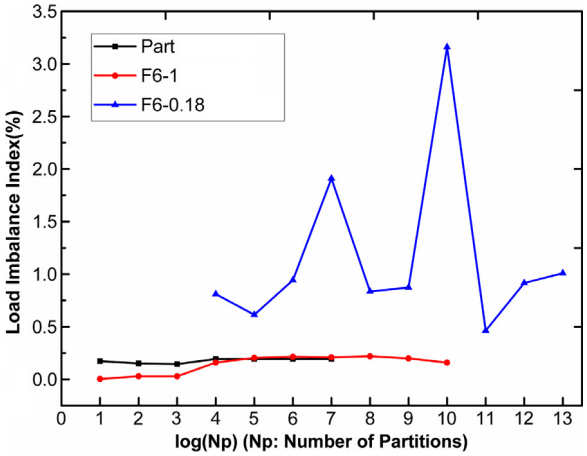


Fig. 8. Variation of the load imbalance index against the number of partitions.

α_{th} and β_{th} values to ensure the inter-domain boundary has an acceptable shape quality. Note that a general purpose mesh improver mainly focuses on elements with worst shapes and the dihedral angles of these elements are very small; therefore, α_{th} and β_{th} are not necessary to be very high. In our experiments, α_{th} and β_{th} are set to be 30° in default.

Three meshes, i.e., Part, F6-1 and F6-0.18, are tested. The former two meshes are generated sequentially, including 66,257 and 1,285,633 elements, respectively. The mesh F6-0.18 is generated in parallel, including 128 submeshes and totally 185,198,575 elements. The sequential DD procedure is adopted for the former two meshes and the parallel procedure is adopted for the last one. By fixing α_{th} and β_{th} being their default values, i.e., 30°, the variation of a load imbalance index l_{imb} against the number of partitions (n_p) is drawn in Fig. 8. $l_{\text{imb}} = (l_{\max} - l_{\min})/l_{\text{ave}}$, where $l_{\max}$, $l_{\min}$ and l_{ave} are the maximal, minimal and average numbers of elements on the submeshes output by the sequential or parallel DD procedure.

In the tests for Part and F6-1, l_{imb} remains very small (less than 0.5%) even when n_p increases up to 256 and 1024, respectively. In the test for the F6-0.18 model, more imbalances are observed because ParMetis instead of Metis is employed for DD. Nonetheless, l_{imb} remains an acceptable value (under 3.5%) in this test even when n_p increases up to 8192, although it fluctuates more widely due to the heuristic nature of the graph partitioning algorithm employed in ParMetis.

The above performance data reveal that we can use at least 256, 1024 and 8192 computers cores to efficiently improve the three meshes in parallel, respectively. In practice, so high parallelism is more than enough. For instance, it is unnecessary to improve a mesh of the magnitude like F6-1 using more than 128 computer cores (about ten thousand elements are managed by one core averagely). A sequential run consumes about one minute, and this time cost is reduced to about one second on 128 cores. Investing more cores is of little significance.

In addition, we compare the shape quality of the inter-domain boundaries resulted from the proposed DD approach and the approach that employs ParMetis directly. The input mesh is F6-0.18, and 128 computer cores are utilized. In Table 1, the angles $\alpha_{\min}$ and $\beta_{\min}$ refer to the minimal values of interior angles of inter-domain face and inter-domain dihedral angle, respectively. As we expect, when the proposed approach is adopted, the angles $\alpha_{\min}$ and $\beta_{\min}$ surpass the predefined thresholds slightly. However, when the mesh F6-0.18 is partitioned by ParMetis directly, 9043 inter-domain dihedral angles and 27,398 minimal interior angles of inter-domain faces are smaller than 30°. Meanwhile, $\alpha_{\min}$ and $\beta_{\min}$ are very close to zero.

Table 1
Comparison of shape quality of the inter-domain boundary (input mesh: F6-0.18).

DD Mtd.	#elems.	#cores	$\alpha_{\min}$	$\beta_{\min}$	$\alpha < 30°$	$\beta < 30°$
Proposed	185,198,575	128	30.0	30.0	0	0
ParMetis			2.1	0.37	27,398	9043

5.1.3. Performance data of the parallel mesh improver

Because the inter-domain mesh is fixed in the improvement procedure, a badly shaped inter-domain boundary will limit the quality of the improved mesh remarkably. To clarify this, Table 2 lists the quality statistics of the improved meshes of the F6-0.18 model based on the DD results shown in Table 1. Because the focus is on the worst elements, the distribution of the minimum dihedral angles (θ) of volume elements in the range of 0 to 24° is listed in Table 2. A tetrahedron is classified as a *low-quality* element if its θ value is smaller than 24° or as a *bad* element if its θ value is smaller than 12°. The angle $\theta_{\min}$ refers to the minimal dihedral angle of all volume elements. If the input mesh is decomposed directly by ParMetis, the output mesh contains 848 bad elements and 28,252 low-quality elements. Many of them are adjacent to the inter-domain boundary. However, these two numbers are reduced by one order if the proposed DD approach is employed, being 35 and 2365, respectively. Meanwhile, as listed in Table 1, the former DD procedure introduces a dihedral angle of 0.37° on the inter-domain boundary. This angle is contained in the improved volume mesh. If not improved further, this angle will be very harmful for simulations.

Table 2 also evaluates the *stability* of the parallel improver, referring to whether it can improve the mesh quality to a comparable level with the result of the sequential improver using the Part model. Although the majority of elements of the two input meshes are well shaped, a certain number of bad and low-quality elements are generated by the Delaunay mesher. Some elements even have dihedral angles close to zero. Remarkable improvements are observed for both the sequential and parallel improvers, and the meshes output by them have a comparable level of element quality. The output mesh of the parallel improver contains 20 bad elements and 757 low-quality elements, and $\theta_{\min}$ is 3.32°. Likewise, the output mesh of the sequential improver also contains 20 bad elements and its $\theta_{\min}$ is 3.32°. The number of low-quality elements is 837, slightly bigger. This difference is reasonable because the results are dependent on the execution orders of local improvement operations.

In the third experiment, our focus is on the efficiency and scalability of the parallel improver. Firstly, the input mesh is fixed to be the F6-0.18 mesh generated by our in-house parallel mesher using 128 computer cores. At least 16 computer cores are used to improve the mesh, and the parallel efficiencies are demonstrated by doubling the number of computer cores used and evaluated with the timing data related to the experiment where 16 cores are assigned. When the number of cores is less than 128, more than one submesh is input to a core. For instance, if the number of cores is 16, each core will read 8 submeshes. Table 3 details the timing data of this experiment, where the parallel DD approach is subdivided into eight steps and the timing performance of each step is also detailed. The time cost of the improvement procedure dominates the entire scheme. A super linear speedup (γ_3) is observed for this procedure. Meanwhile, because a parallel efficiency of above 0.7 is achieved by the parallel DD procedure, a nearly linear or super linear speedup is observed for the entire scheme.

A super linear speedup for γ_3 is attributed to the nonlinear running performance of the sequential improver. In other words, if a mesh is partitioned into submeshes, the time cost spent in improving the original mesh is usually bigger than the total time costs spent in improving the submeshes. Therefore, if the

Table 2
Element quality comparison of the sequential and parallel improvers.

Input mesh	Part			F6-0.18		
DD Mtd.	–	None	Proposed	–	Proposed	ParMetis
Imprv. mode	Unimproved	Sequential	Parallel	Unimproved	Parallel	Parallel
#elems	66,257	61,628	60,466	185,198,575	173,189,701	173,420,641
#subdomains	–	1	4	–	128	128
$\theta_{\min}(^\circ)$	4.4e−4	3.3	3.3	≈0.0	5.6	0.37
$0^\circ < \theta \leq 6^\circ$	347	7	7	676,374	1	172
$6^\circ < \theta \leq 12^\circ$	997	13	13	2,067,274	34	676
$12^\circ < \theta \leq 18^\circ$	1943	37	26	3,735,560	181	3998
$18^\circ < \theta \leq 24^\circ$	3459	780	711	6,027,106	2184	23,406
$\theta \leq 12^\circ$ (and percentage)	1344 (2.0%)	20 (0.03%)	20 (0.03%)	2,743,648 (1.48%)	35 (2e−5%)	848 (5e−4%)
$\theta \leq 24^\circ$ (and percentage)	6746 (10.2%)	837 (1.36%)	757 (1.25%)	12,506,314 (6.75%)	2365 (1.4e−3%)	28,252 (0.016%)

Table 3
Parallel efficiencies of the parallel improver and its major steps when the input mesh is fixed and the number of computer cores varies.

Input mesh		F6-0.18			
General data	#elems before	185,198,575 (composed of 128 submeshes initially)			
	#elems after	173,242,842	173,237,367	173,220,582	173,189,701
	#cores	16	32	64	128
Time costs of major steps (s)	Entire scheme	716.60	357.45	177.18	90.51
	Domain decomposition	112.04	57.79	33.42	19.45
	Quality improvement	582.13	288.15	137.83	68.03
	I/O	22.43	11.51	5.93	3.03
Time costs of different steps of domain decomposition (s)	First repartitioning	35.02	18.73	12.44	7.84
	Build mesh structure	28.03	13.78	6.82	3.62
	Shape analysis	7.22	3.66	1.82	0.90
	Build SDG	3.99	1.55	0.77	0.42
	Build EDG	2.62	1.27	0.64	0.32
	Node-deletion	5.45	2.17	1.06	0.56
	Edge-contraction	5.11	2.48	1.38	0.98
	Second repartitioning	24.60	14.15	8.49	4.81
Parallel efficiencies	Entire scheme with I/O (γ_1)	–	1.00	1.01	0.99
	Domain decomposition (γ_2)	–	0.97	0.84	0.72
	Quality improvement (γ_3)	–	1.01	1.06	1.07

submeshes are well distributed and improved in parallel without any communication or synchronization costs, a super linear efficiency is possible.

To test the scalability further, the number of computer cores is fixed at 16, but the four meshes of the F6 model with various sizes are selected, which contain 1,285,633, 6,653,586, 53,206,270 and 185,198,575 elements, respectively. To improve them, the parallel improver consumes 4.4, 23.1, 216.0 and 714.9 s, respectively. The velocity values, referring to how many elements the parallel improver can manage per core per second, are 18,439, 17,996, 15,395 and 16,192, respectively. No evident decline is observed when the mesh size grows. It varies under a reasonable range considering the timing performance of a mesh improver always depends on the input mesh to some extent.

5.1.4. Performance comparison with existing algorithms

In this experiment, we re-implement two parallel mesh improvers for the comparison purpose, following the ideas suggested in [21,29], respectively.

The first mesh improver takes the following steps to improve a distributed mesh [21]:

(1) Input the distributed mesh, and then repartition the mesh by ParMetis for load balancing.
(2) Improve the redistributed submeshes in parallel with their inter-domain interfaces fixed.
(3) Collect two layers of elements near inter-domain interfaces into a single mesh.
(4) Improve this single mesh sequentially.

Here, the first layer of elements refers to those containing at least one inter-domain mesh vertex, and the second layer of elements refers to those sharing at least one mesh vertex with the first layer of elements.

Table 4 reports the timing performance data of the first mesh improver. The input is the F6-0.18 mesh (with 128 submeshes). The single mesh formed by collecting elements near inter-domain interfaces contains 26,362,570 elements. A sequential run of the revised Grummp on this mesh consumed 1231.52 s. As a result, the entire mesh improvement process consumed 1320.87 s, which is about 14.6 times slower than the proposed algorithm.

The second mesh improver follows the idea suggested in [29]. It performs the first mesh improvement pass as the first improver does, but performs the second mesh improvement pass very differently [29]. After the first mesh improvement pass, the second mesh improver redistributes resulting submeshes by adding two layers of elements to each domain (idomn) from the neighbouring domains (jdomn) for which idomn < jdomn, and the redistributed submeshes are then improved in parallel with the inter-domain boundary fixed.

Table 4 also reports the timing performance data of the second mesh improver by using the same input as the test for the first mesh improver. In the step of redistributing elements near inter-domain interfaces, many elements are sent from the processes with high rank values to neighbouring processes with small rank values. As a result, the processes with small rank values usually have to treat many more elements than the processes with high rank values. In this test, it was observed that the largest submesh is placed on the process whose rank value is 3 after the redistribution step. This submesh contains about 3,251,625 elements, nearly 2.4

Table 4
Timing performance (unit: second) of the re-implemented parallel mesh improvers following the ideas suggested by Ito et al. (namely the first improver) [21] and Löhner (namely the second improver) [29], respectively. The input mesh is the F6-0.18 mesh (containing 128 submeshes). 128 computer cores were employed in mesh improvement.

Algorithm	Ito et al. [21]	Löhner [29]
Entire scheme	1320.87	247.61
Mesh repartitioning	7.96	7.85
The first mesh improvement pass	69.21	69.05
Collection of elements (for the first improver) or redistribution of elements (for the second improver) near inter-domain interfaces	12.18	14.90
The second mesh improvement pass (executed in sequential for the first improver while in parallel for the second improver)	1231.52	155.81

Table 5
Quality of elements produced by the mesh improvement techniques suggested by Ito et al. [21] and Löhner [29]. The input mesh is the F6-0.18 mesh (containing 128 submeshes). 128 computer cores were employed in mesh improvement.

Algorithms	Ito et al. [21]	Löhner [29]
$\theta_{\min}(^\circ)$	5.6	5.6
$0^\circ < \theta \leq 6^\circ$	1	1
$6^\circ < \theta \leq 12^\circ$	38	36
$12^\circ < \theta \leq 18^\circ$	198	182
$18^\circ < \theta \leq 24^\circ$	3543	3026
$\theta \leq 12^\circ$	39	37
$\theta \leq 24^\circ$	3780	3245

times larger than the average size of the redistributed submeshes. This process was therefore the busiest one during the second mesh improvement pass, consuming 155.81 s. As a result, the entire mesh improvement process consumed 247.61 s, which is about 5.3 times faster than the first mesh improver, but about 2.7 times slower than the proposed algorithm.

Element quality is another concern in this comparison. Table 5 lists the quality data of the improved meshes by two mesh improvers mentioned above. It was observed that both mesh improvers could improve the quality of the F6-0.18 mesh to the comparable level as the proposed algorithm, in terms of the value of the minimal dihedral angle and the numbers of bad and low-quality elements (see Table 2 for the quality data produced by the proposed algorithm). However, it is worth of noting that, during the second mesh improvement pass, the first mesh improver needs to fix the boundary of the single mesh, and the second mesh improver needs to fix the inter-domain boundary of the redistributed mesh. The possibility exists that bad or low-quality elements may survive in the final mesh if features that might negatively impact element quality are contained in these boundaries, although this undesirable situation did not happen in this test.

5.2. Experiments on the overall parallel pipeline

The entire pipeline of preparing the F6-0.18 mesh is parallelized by combining the parallel improver with the parallel surface and volume meshers we developed previously [5,45]. 128 computer cores are involved in this experiment for the entire pipeline. The surface mesher takes 12 s to get a mesh containing 1,318,726 triangles. The volume mesh generation includes two steps, as we mention in Section 4.4. The first step is the only sequential part of the parallel pipeline. It decomposes the volume domain represented by the surface mesh into many subdomains, and consumes 125 s in this experiment. The second step generates volume meshes on all subdomains in parallel. It runs very fast, and only consumes 10 s. Averagely, the Delaunay mesher generates 144,686 elements per core per second. The timing data of the parallel improver has been listed in Table 3, totally 90.51 s are consumed. Excluding the proposed DD step, the improvement of 128 submeshes consumes about 68 s. Averagely, the improver manages 21,277 elements per core per second, slower than the Delaunay mesher by nearly seven times.

The sequential generation of the surface mesh takes about 10 min. If we assume the Delaunay mesher and the improver maintain a speed of 144,686 and 21,277 elements per core per second, then they take 1280 and 8704 s to generate and improve the F6-0.18 mesh, respectively. Roughly speaking, it takes 3 h to prepare this mesh sequentially. The real cost might be larger because it remains an issue to enhance a sequential mesher or improver to obtain a linear timing performance when the mesh size increases to the magnitude of F6-0.18. As a comparison, the proposed parallel pre-processing pipeline consumes about 238 s to prepare this mesh on 128 computer cores, achieving a speedup value of about 45.

Finally, two additional mesh examples for outflow simulations are selected to show that the developed system is robust and applicable to geometries of a complication level experienced in industry. The first example is a fully loaded F16 aircraft model. Fig. 9(a) presents a volume mesh of the F16 aircraft model generated in parallel, where the elements painted in different colours were generated on different computer cores. The second example is the London Tower Bridge model. This model is the benchmark model adopted by the 23rd International Meshing Roundtable for meshing contest (imr.sandia.gov/23imr/MeshingContest.html). Fig. 9(b) presents the surface mesh, volume mesh and the pressure map on the surface of the bridge model. Here, the pressure is induced by a crosswind at the speed of 34 m/s. 32 computer cores participate in the entire processes of producing both meshes, which contain 30,185,079 and 40,247,936 tetrahedral elements, respectively. The two pre-processing processes consume 154 and 191 s, respectively, of which parallel mesh improvement consumes 79 and 97 s, respectively.

The distributions of interior angles of the tetrahedral elements for the F16 aircraft and Tower Bridge models are presented in Fig. 10(a) and (b), respectively. It is observed that the adopted Delaunay mesher always generates a certain number of elements with extremely small or large interior angles (close to 0° or 180°) due to the issues of sliver elements and boundary constraints. After improvement, both meshes are improved evidently.

6. Conclusions

A novel domain decomposition approach for tetrahedral mesh improvement is proposed in this study. Given a tetrahedral mesh generated by the state-of-the-art mesher, the proposed DD approach is able to decompose it into many partitions of similar sizes. Moreover, the inter-domain boundary is ensured to contain no small dihedral angles and poorly shaped faces. An enhanced open-source improver, i.e., Grummp, is reused to improve each partition in parallel by fixing the inter-domain boundary. As expected, the parallel algorithm can improve the mesh quality to a comparable level with its sequential counterpart. Meanwhile, because the DD procedure is parallelized efficiently, plus the improvement procedure involves no communications, the efficiency of the parallel improver is very high. For instance, if applying the sequential improver on a mesh containing about 170 million elements, the time cost is roughly estimated to be at least 8000 s, which is reduced to only 90 s by the parallel improver when 128 computer cores are employed. Furthermore, the parallel improver also enables us to set up a parallel preprocessing pipeline for large-scale simulations. It reduces the time for preparing a

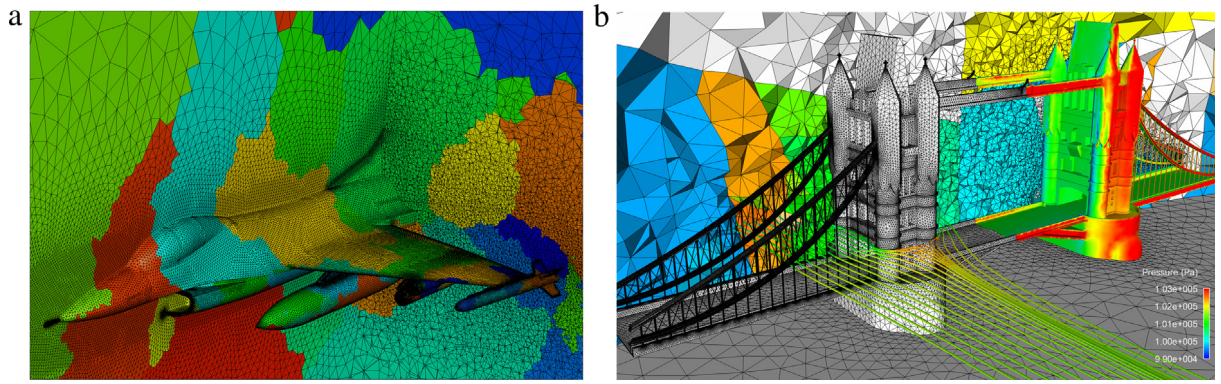


Fig. 9. Two mesh examples for outflow simulations. (a) A mesh of the F16 aircraft model. (b) A mesh of the London Tower Bridge model and the pressure map on the bridge surface (the pressure is induced by a crosswind at the speed of 34 m/s). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

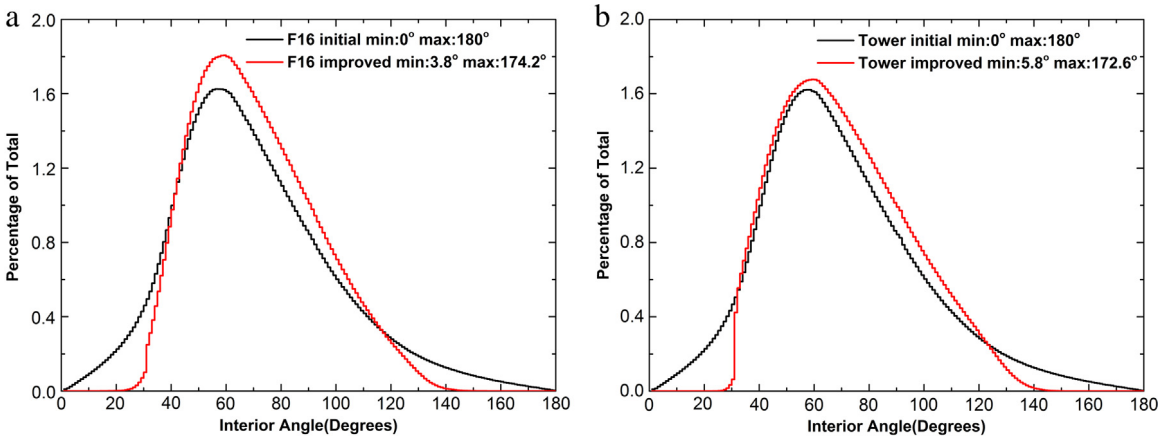


Fig. 10. The distributions of interior angles of the tetrahedral elements for the initial meshes and the meshes after quality improvement on (a) the F16 aircraft model and (b) the London Tower Bridge model.

mesh that contains hundreds of millions of elements from hours (for a sequential pipeline) to minutes.

Acknowledgments

The authors appreciate the joint support for this project by the National Natural Science Foundation of China (Grant Nos. 11172267, 11432013, U1630121, 11402229), Science Challenge Project of China (No. JCKY2016212A502) and Zhejiang Provincial Natural Science Foundation (Grant Nos. LR16F020002, Y1110038 and LQ14A020003). The authors also appreciate Dr. Hongtao Wang in Zhejiang University for his help in accessing a parallel computer. The first author acknowledges the joint support from Zhejiang University and China Scholarship Council and the host of Professor Oubay Hassan and Professor Kenneth Morgan for his visiting research at Swansea University, UK.

References

[1] C.D. Antonopoulos, F. Blagojevic, A.N. Chernikov, N.P. Chrisochoides, D.S. Nikolopoulos, Algorithm, software, and hardware optimizations for Delaunay mesh generation on simultaneous multithreaded architectures, J. Parallel Distrib. Comput. 69 (2009) 601–612.
[2] C.D. Antonopoulos, F. Blagojevic, A.N. Chernikov, N.P. Chrisochoides, D.S. Nikolopoulos, A multigrain Delaunay mesh generation method for multicore SMT-based architectures, J. Parallel Distrib. Comput. 69 (2009) 589–600.
[3] M. Brewer, L.F. Diachin, P. Knupp, T. Leurent, D. Melander, The mesquite mesh quality improvement toolkit, in: 12th International Meshing Roundtable, Santa Fe, NM, USA, 2003, pp. 239–250.
[4] J. Chen, D. Zhao, Z. Huang, Y. Zheng, S. Gao, Three-dimensional constrained boundary recovery with an enhanced Steiner point suppression procedure, Comput. Struct. 89 (2011) 455–466.
[5] J. Chen, D. Zhao, Z. Huang, Y. Zheng, D. Wang, Improvements in the reliability and element quality of parallel tetrahedral mesh generation, Internat. J. Numer. Methods Engrg. 92 (2012) 671–693.
[6] N. Chrisochoides, D. Nave, Parallel Delaunay mesh generation kernel, Internat. J. Numer. Methods Engrg. 58 (2003) 161–176.
[7] G. Compère, J.F. Remacle, J. Jansson, J. Hoffman, A mesh adaptation framework for dealing with large deforming meshes, Internat. J. Numer. Methods Engrg. 82 (2010) 843–867.
[8] J.P. D'Amato, M. Vénere, A CPU–GPU framework for optimizing the quality of large meshes, J. Parallel Distrib. Comput. 73 (2013) 1127–1134.
[9] H.L. de Cougny, M.S. Shephard, Refinement, derefinement, and optimization of tetrahedral geometric triangulations in three dimensions. 1995 (unpublished manuscript).
[10] E.B. De l'Isle, P.L. George, Optimization of tetrahedral meshes, IMA Vol. Math. Appl. 75 (1995) 97–128.
[11] L.F. Diachin, P. Knupp, T. Munson, S. Shontz, A comparison of two optimization methods for mesh quality improvement, Eng. Comput. 22 (2006) 61–74.
[12] Q. Du, D. Wang, Recent progress in robust and quality Delaunay mesh generation, J. Comput. Appl. Math. 195 (2006) 8–23.
[13] J.M. Escobar, R. Montenegro, G. Montero, E. Rodríguez, J.M. González-Yuste, Smoothing and local refinement techniques for improving tetrahedral mesh quality, Comput. Struct. 83 (2005) 2423–2430.
[14] G. Foucault, J.C. Cuillière, V. François, J.C. Léon, R. Maranzana, Generalizing the advancing front method to composite surfaces in the context of meshing constraints topology, Comput.-Aided Des. 45 (2013) 1408–1425.
[15] L.A. Freitag, M. Jones, P. Plassmann, A parallel algorithm for mesh smoothing, SIAM J. Sci. Comput. 20 (1999) 2023–2040.
[16] L.A. Freitag, M.T. Jones, P.E. Plassmann, The scalability of mesh improvement algorithms, in: M. Heath, A. Ranade, R. Schreiber (Eds.), Algorithms for Parallel Processing, in: The IMA Volumes in Mathematics and its Applications, vol. 105, Springer, New York, 1999, pp. 185–211.
[17] L.A. Freitag, C. Ollivier-Gooch, Tetrahedral mesh improvement using swapping and smoothing, Internat. J. Numer. Methods Engrg. 40 (1997) 3979–4002.
[18] P.L. George, H. Borouchaki, Back to edge flips in 3 dimensions, in: 12th International Meshing Roundtable, Santa Fe, NM, USA, 2003, pp. 393–402.

[19] P.L. George, H. Borouchaki, E. Saltel, 'Ultimate' robustness in meshing an arbitrary polyhedron, Internat. J. Numer. Methods Engrg. 58 (2003) 1061–1089.
[20] P.L. George, F. Hecht, E. Saltel, Automatic mesh generator with specified boundary, Comput. Methods Appl. Mech. Engrg. 92 (1991) 269–288.
[21] Y. Ito, A.M. Shih, A.K. Erukala, B.K. Soni, A. Chernikov, N.P. Chrisochoides, K. Nakahashi, Parallel unstructured mesh generation by an advancing front method, Math. Comput. Simulation 75 (2007) 200–209.
[22] B. Joe, Construction of three-dimensional improved quality triangulations using local transformations, SIAM J. Sci. Comput. 16 (1995) 1292–1307.
[23] G. Karypis, V. Kumar, Multilevel algorithms for multi-constraint graph partitioning, in: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, San Jose, CA, USA, 1998, pp. 1–13.
[24] B. Klingner, J.R. Shewchuk, Aggressive tetrahedral mesh improvement, in: 16th International Meshing Roundtable, Seattle, WA, USA, 2007, pp. 3–23.
[25] R.W. Lewis, Y. Zheng, D.T. Gethin, Three-dimensional unstructured mesh generation: part 3. volume meshes, Comput. Methods Appl. Mech. Engrg. 134 (1996) 285–310.
[26] J. Liu, B. Chen, S. Sun, Small polyhedron reconnection for mesh improvement and its implementation based on advancing front technique, Internat. J. Numer. Methods Engrg. 79 (2009) 1004–1018.
[27] S.H. Lo, Volume discretization into tetrahedral-II. 3D triangulation by advancing front approach, Comput. Struct. 39 (1991) 501–511.
[28] S.H. Lo, Delaunay triangulation of non-uniform point distributions by means of multi-grid insertion, Finite Elem. Anal. Des. 63 (2013) 8–22.
[29] R. Löhner, Recent advances in parallel advancing front grid generation, Arch. Comput. Methods Eng. 21 (2014) 127–140.
[30] R. Löhner, P. Parikh, Generation of three-dimensional unstructured grids by the advancing front method, Internat. J. Numer. Methods Fluids 8 (1988) 1135–1149.
[31] METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. Feb-07–2013. URL: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.
[32] M. Misztal, J. Bærentzen, F. Anton, K. Erleben, Tetrahedral mesh improvement using multi-face retriangulation, in: 18th International Meshing Roundtable, Salt Lake City, UT, USA, 2009, pp. 539–555.
[33] C. Ollivier-Gooch, GRUMMP, Jan-21–2012. URL: http://tetra.mech.ubc.ca/GRUMMP/.
[34] ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering. Feb-07–2013. URL: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.
[35] K. Schloegel, G. Karypis, V. Kumar, Parallel multilevel algorithms for multi-constraint graph partitioning, in: 6th International Euro-Par Conference on Parallel Processing, Munich, Germany, 2000, pp. 296–310.
[36] E. Shaer, Z. Cheng, R. Yeh, G. Zagaris, L. Olson, Simple and effective GPU-based mesh optimization. Research Notes of the 20th International Meshing Roundtable, Paris, France, 2011.
[37] J.F. Shepherd, C.R. Johnson, Hexahedral mesh generation constraints, Eng. Comput. 24 (2008) 195–213.
[38] J.R. Shewchuk, Delaunay refinement mesh generation (Ph.D. thesis), Carneigie Mellon University, Pittsburgh, PA, USA, 1997.
[39] J.R. Shewchuk, Two discrete optimization algorithms for the topological improvement of tetrahedral meshes. 2002 (unpublished manuscript). Mar-05–2014. URL: https://www.cs.berkeley.edu/~jrs/papers/edge.pdf.
[40] H. Si, TetGen, a Delaunay-based quality tetrahedral mesh generator, ACM Trans. Math. Softw. 41 (2015) 11:1–11:36.
[41] S. Soner, C. Ozturan, Generating multibillion element unstructured meshes on distributed memory parallel machines, Sci. Program. 2015 (2015) Article ID 437480.
[42] S. Song, M. Wan, S. Wang, D. Wang, Z. Zou, Robust and quality boundary constrained tetrahedral mesh generation, Commun. Comput. Phys. 14 (2013) 1304–1321.
[43] N.P. Weatherill, O. Hassan, Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints, Internat. J. Numer. Methods Engrg. 37 (1994) 2005–2039.
[44] L. Xie, Y. Zheng, J. Chen, J. Zou, Enabling technologies in the problem solving environment HEDP, Commun. Comput. Phys. 4 (2008) 1170–1193.
[45] D. Zhao, J. Chen, Y. Zheng, Z. Huang, J. Zheng, Fine-grained parallel algorithm for unstructured surface mesh generation, Comput. Struct. 154 (2015) 177–191.
[46] J. Zheng, J. Chen, Y. Zheng, Y. Yao, S. Li, Z. Xiao, An improved local remeshing algorithm for moving boundary simulations, Eng. Appl. Comput. Fluid Mech. 10 (2016) 405–428.

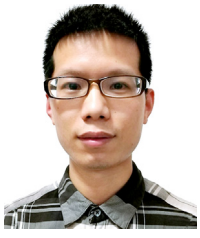
Jianjun Chen received the B.Eng. degree in civil engineering from Nanjing Institute of Civil Engineering and Architecture, China, the M.Eng. degree in computational mechanics and Ph.D. degree in computer science from Zhejiang University, China. He has been working at Zhejiang University since 2006 and is currently a full professor at School of Aeronautics and Astronautics. From 2012 to 2014, he conducted a visiting research in Swansea University, UK. His main research interests include high performance computing, mesh generation and computational engineering and science.

Dawei Zhao received the B.Eng. degree in computer science from China University of Mining and Technology and Ph.D. degree in computer science from Zhejiang University, China. The subject of his Ph.D. thesis is parallel mesh generation.

Yao Zheng received the B.Sc. degree in Mathematics from Hangzhou University, China, the M.Sc. (Eng) degree in Mechanics from Harbin Institute of Technology, China, and the Ph.D. degree from University of Wales Swansea, UK. At present he is a Cheung Kong chair professor (2001-) and the vice dean (2014-) of Faculty of Engineering with Zhejiang University, China, and is directing Center for Engineering and Scientific Computation (2002-), Zhejiang University. Also he is the founding deputy dean (2007–2013) of School of Aeronautics and Astronautics, Zhejiang University. He had been a Senior Research Scientist for NASA Glenn Research Center, Cleveland, Ohio, USA (1998–2002). His main research interests include high performance computing, and aerospace computing engineering.

Yan Xu received the B.Eng. degree and Ph.D. degree in civil engineering from Zhejiang University, China. He has been working at Zhejiang University since 2011 and is currently an associate professor at School of Aeronautics and Astronautics. From 2007 to 2008, he conducted a one-year visiting research in Tokyo University, Japan. His main research interests include high performance computing, computational science and engineering, and architectural/aerospace structure engineering.

Chenfeng Li is currently an Associate Professor at the College of Engineering of Swansea University, UK. He received his B.Eng. and M.Sc. degrees from Tsinghua University, Beijing in 1999 and 2002, respectively, and received his Ph.D. degree from Swansea University in 2006. His research interests include computational solid mechanics, computational fluid dynamics, and computational graphics. He is the Editor-in-Chief of Engineering Computations.



Jianjing Zheng received the B.Eng. degree in engineering mechanics and Ph.D. degree in computer science from Zhejiang University, China. At present, he is a postdoctoral research fellow at Zhejiang University. His main research interests include computational fluid dynamics, mesh generation and high performance computing.