Scalable generation of large-scale unstructured meshes by a novel domain decomposition approach

Jianjun Chen[a,b], Zhoufang Xiao[c,], Yao Zheng[a,b], Jianfeng Zou[a,b,], Dawei Zhao[a,b], Yufeng Yao[d]

[a] Center for Engineering and Scientific Computation, Zhejiang University, Hangzhou 310027, China
[b] School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China
[c] School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China
[d] Faculty of Environment and Technology, University of the West of England, Bristol BS16 1QY, United Kingdom

A B S T R A C T

A parallel algorithm is proposed for scalable generation of large-scale tetrahedral meshes. The key innovation is the use of a mesh-simplification based domain decomposition approach. This approach works on a background mesh with both its surface and its interior elements much larger than the final elements desired, and decomposes the domain into subdomains containing no undesirable geometric features in the inter-domain interfaces. In this way, the most time-consuming part of domain decomposition can be efficiently parallelized, and other se-quential parts consume reasonably limited computing time since they treat a very coarse background mesh. Meanwhile, the subsequent parallel procedures of mesh generation and improvement are most efficient because they can treat individual subdomains without compromising element quality. Compared with published state-of-the-art parallel algorithms, the developed parallel algorithm can reduce the clock time required by the creation of one billion elements on 512 computer cores from roughly half an hour to less than 4 minutes.

Keywords: Mesh generation; Domain decomposition; Parallel algorithms; Dual graph; Large-scale meshes.

1. Introduction

Various simulation codes have been parallelized to exploit their potentials of running efficiently on increasingly powerful parallel computing machines. However, the clock time to finish a complex si-mulation is much longer than that expected by the end-user. A sig-nificant percentage of this time is consumed by the mesh generation step. For instance, it takes weeks or more to prepare a block-structured mesh for an external flow simulation over a complete aircraft model, even by an engineer with expertise in applying state-of-the-art meshing tools [1]. Unstructured mesh generation does not require a painful block-decomposition process; hence, it is more straightforward and automatic. Nevertheless, the process of generating an unstructured mesh in excess of 10^9 elements still consumes about 1.5 days if being executed sequentially [2]. In practice, when the input geometry is be-coming very complicated, or the resultant mesh quality is set at a high standard, mesh generation is often becoming a trial-and-error process. Its execution possibly needs to be repeated several times before an ideal mesh is obtained. Therefore, the clock time for preparing a large-scale unstructured mesh is usually comparable with or more than the time consumed for conducting parallel simulations. To fully exploit the computing power of the present and future emerging parallel computers, an environment where both the simulation and its mesh generation codes are parallelized is now highly demanded [2-7] by research developers as well as industry end-users.

Various parallel mesh generation approaches have been developed since 1990s. In general, these approaches could be classified into the algorithm-parallel ones and the problem-parallel ones [8]. Presently, the problem-parallel approaches are preferred in many studies because of their better capability of reusing state-of-the-art sequential mesh generation codes. Domain decomposition is an essential step of the problem-parallel approaches. It subdivides a problem domain into many subdomains such that the following meshing procedure could be conducted on these subdomains in parallel. The performance of a par-allel mesh generation algorithm is thus highly dependent on its domain decomposition approach.

Among various performance indices, the scalability of a parallel mesher should be highlighted nowadays since the computers used in present simulations may be configured with hundreds of thousands of computer cores. It is not uncommon that the required computational mesh by such simulations may contain billions of elements or more. To reduce the meshing time into the order of minutes, the parallel mesher must demonstrate its scalability on several hundred or more computer cores. However, in most existing parallel mesh generation studies, the principal motivation was to overcome the memory bottleneck of large-scale mesh generation, and the scalability is not a major concern. Consequently, the achievable scalability of existing parallel meshers is not always desirable. A typical example is the parallel mesher we de-veloped a few years ago [9]. Its domain decomposition approach inputs a surface mesh of the problem domain, and decomposes this mesh by inserting interface meshes in the interior of the domain. After resolving the intersections in subdomain boundaries properly, the reliability of this approach has been well demonstrated for complicated configura-tions. Nevertheless, the efficient parallelization of this domain decom-position approach remains a challenging issue. No matter how many additional computer cores are invested, the time cost of the domain decomposition step scales up in accordance with the increase of the required computational mesh size. As a result, a poor scalability is observed for this parallel mesher when hundreds of computers cores are invested to create large-scale meshes. It must be emphasized that this poor scalability is not only observed for the early parallel mesher de-veloped by us [9]. For one reason or another, the scalability issue is not well resolved in many other existing parallel meshers. For instance, in one test of the parallel advancing front technique (AFT) proposed in [2], 121 million elements could be created on 8 computer cores at a relative speed (els/sec/core) of 9,423. However, it was reported that this speed value was reduced by more than one order (being 788) when 512 computer cores are used to create one billion elements.

In this study, a novel domain decomposition approach is proposed to achieve the scalable generation of large-scale tetrahedral meshes. It inputs a problem domain depicted by a CAD model, and then decom-poses the domain into the required number of subdomains depicted by the same fine surface triangulation as the final mesh desired. To be clear, Fig. 1 presents an example illustrating the main steps involved in the proposed approach.

(1) With the input sizing function scaled up, a coarse surface back-ground mesh (SBM) is created by meshing the CAD surfaces. After that, a volume background mesh (VBM) is created with the SBM and the scaled-up sizing function as inputs (see Fig. 1a).
(2) Both the SBM and the VBM are simplified and their simplifications are decomposed into subsurfaces and subdomains, respectively. Here, a subsurface refers to the set of surface faces bounding a sub-domain. By adopting the mesh simplification approach developed in [10], not only badly shaped faces and small dihedral angles are prevented from appearing on inter-domain interfaces, but also those small angles are prevented from appearing on inter-surface interfaces (see Figs. 1b and 1c).
(3) Subdomain boundaries are remeshed according to the input sizing function. A 2D Delaunay mesher is used to triangulate the inter-domain faces individually, while an advancing front surface mesher is reused to mesh subsurfaces to ensure the generation of surface mesh fitting into the original CAD model.

This novel domain decomposition approach has enabled us to build up an efficient and scalable parallel pre-processing pipeline. This pi-peline inputs a CAD model and then executes the subsequent steps of surface meshing, volume meshing and mesh improvement in a fully parallel and automatic manner. Numerical experiments will be pre-sented to demonstrate that the developed approach is robust and ap-plicable to geometry models of a complicated level experienced in in-dustry. Further scalable parallel performance will be revealed by detailed comparisons with other state-of-the-arts approaches reported in the public domain.

The remaining sections are organized as follows. In Section 2, the typical problem-parallel mesh generation approaches are briefly re-viewed and compared, followed by a summary of our contributions. Sections 3 and 4 present the techniques adopted in the domain de-composition step and the parallel procedures of mesh generation and quality improvement, respectively. Section 5 provides various numerical examples demonstrating the effectiveness and the efficiency of proposed approach. Finally, Section 6 concludes the study.

2. Related work

2.1. Literature review

Depending on how the inter-domain interfaces are treated in par-allel mesh generation, the problem-parallel mesh generation ap-proaches are thus summarized [11] as: (1) those meshing interfaces as they mesh subdomains [12], (2) those postmeshing the interfaces, and (3) those premeshing the interfaces.

Chrisochoides and Nave proposed a typical approach that meshes interfaces as it meshes subdomains [12]. Delaunay refinement is exe-cuted on each submesh and a remote data gathering is required to enforce the mesh conformity if the formed cavity during the insertion of a new point crosses inter-domain interfaces. Here, the cavity refers to all elements whose circumcircles include the new point. Fig. 2a illus-trates a cavity in relation with the insertion of a new point P belonging to the submesh M_0, in which the cavity triangles ABC and BDC belong to M_0, but the cavity triangles DEC and EFC belong to the submeshes distributed on different processes (M_1 and M_2, respectively). Thus, the time-consuming inter-process gathering operations must be executed to insert P, which could degrade the scalability of the parallel point in-sertion algorithm substantially.

The second approach, which meshes interfaces after it meshes subdomains, follows totally different workflow with the first approach. Fig. 2b shows the interface after meshing a simple rectangular domain by using 4 processes. Note that buffer zones are set up between adjacent subdomains, in this case, around 4 interface lines and 1 interface point. Evidently, it remains a challenging task to robustly and efficiently treat these buffer zones.

Cougny and Shephard [13] adopted the second approach to paral-lelize their octree-based tetrahedral mesher. It meshes most interior octants in parallel and defines a cavity zone on each process by combing unmeshed octants. Consequently, different buffer zones are set up be-tween adjacent cavities, around inter-domain faces, lines and points, respectively. The cavity zones are filled up to an order of meshing non-buffer zones at first, and then meshing face-related, line-related and point-related buffer zones successively. Desirable scalability perfor-mance was reported when 32 processes were executed [13]; however, it is not clear how this algorithm would perform when more computing resources are invested.

Löhner [2,14] presented two parallel advancing front techniques (AFT) in which the meshing stage of subdomain interiors takes the precedence to that for interfaces. The first technique [14] distributes the workloads by using an octree background mesh. Many buffer zones are formed near inter-domain interfaces by prohibiting new elements starting from active fronts crossing over interfaces, and these zones will be meshed after shifting the octants to cover them. The second tech-nique [2] distributes workloads by using a domain defining grid (DDG), which has the same fine surface triangulation as the final mesh desired, but a much coarser interior mesh. The scalability of the early algorithm is limited since its applied parallelism is at a local level of the active front. By comparison, the scalability of the new algorithm is much higher by applying the volume-level parallelism, although further im-provement is possible. As reported in [2], the relative speed value of the new parallel AFT at a case of creating one billion elements on 512 computer cores is smaller by more than one order than its counterpart at a case of creating 121 million elements on 8 computer cores.

With respect to the third parallel mesh generation approach, the meshing procedure of each subdomain is completely decoupled; thus, the subdomain mesh generation step could achieve very high parallel efficiencies. However, the efficient parallelization of domain decom-position step may become very difficult. For instance, the algorithms proposed in [9,15-16] adopts a recursive bi-division scheme to

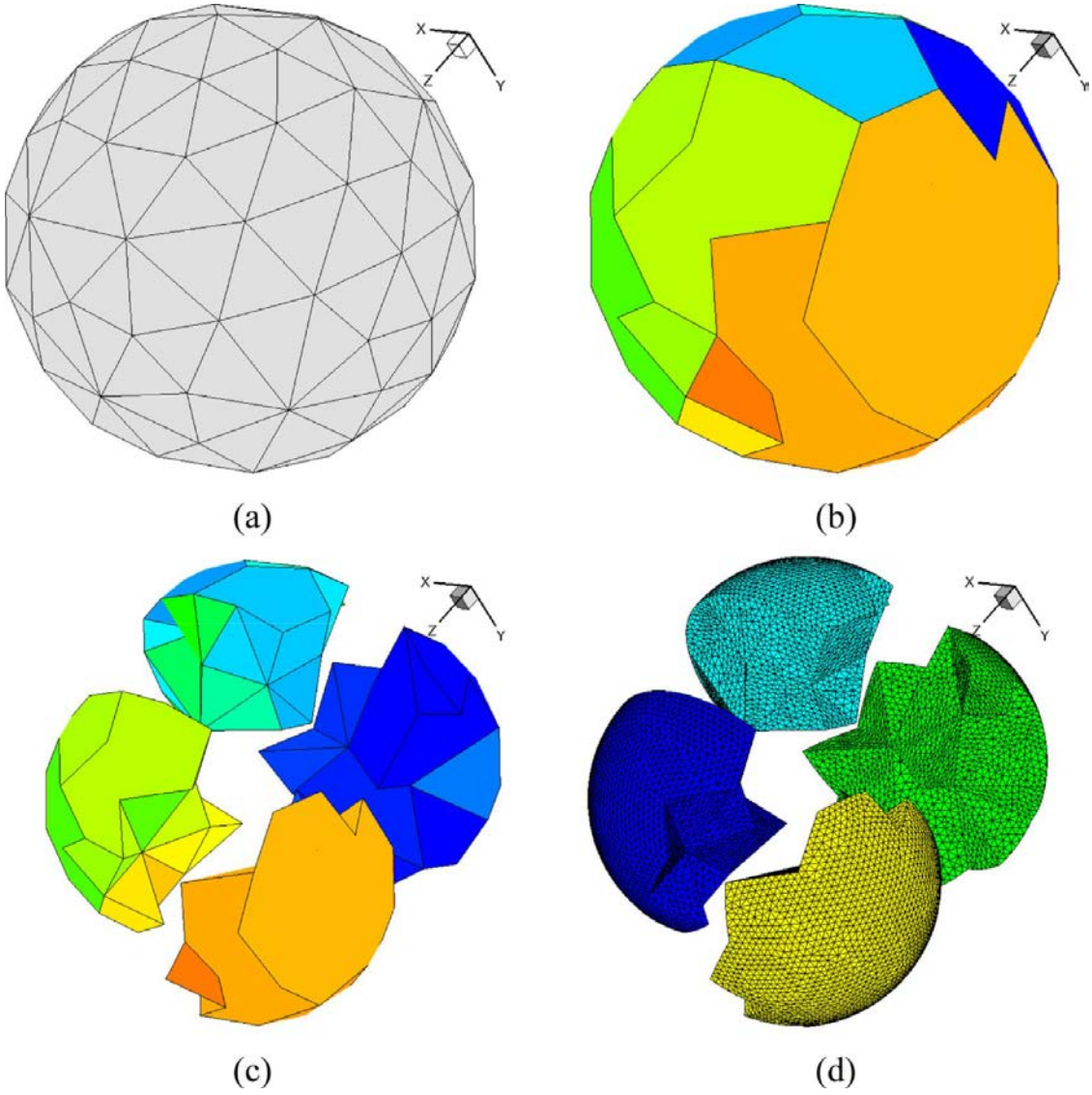

Fig. 1. An example illustrating the proposed domain decomposition approach. (a) The background mesh. (b) The simplified mesh. (c) The domain decomposition result. (d) The subdomains after remeshing their boundaries.

decomposes the problem domain depicted by a surface mesh into sub-domains. If each bi-division process represents a task, the task de-pendency graph of domain decomposition is a binary tree. The root task is most time-consuming, but it can only be performed sequentially. Moving down of this binary tree, the tasks become smaller and smaller, although the number of tasks that can be performed concurrently in-creases.

Instead of using the surface mesh to depict the problem domain, the domain decomposition approach proposed in [17] inputs a volume background mesh that covers the domain interior. Limited scalability was reported on this algorithm, possibly because a large percentage of the domain decomposition step still runs in sequential. Besides, the domain decomposition approach inputs the same fine surface triangu-lation as the final mesh desired. When the size of a background mesh scales up to meet the requirement of a finer computational mesh, the time cost by domain decomposition may scale up accordingly.

Therefore, Ito et al. [18] suggested inputting a background mesh with both its boundary and interior meshes much coarser than that of the final requirement. The domain decomposition approach directly subdivides this background mesh into submeshes and then refines the boundaries of submeshes in parallel. Nevertheless, the scalability of the entire parallel mesh generation algorithm is still limited by its mesh quality improvement algorithm, which first assimilates elements in the neighbourhood of inter-domain interfaces and then improves them se-quentially. Besides, this approach adopts face-wise refinement tem-plates to mesh the domain surface. If the problem domain is originally depicted by a CAD model, this approach cannot ensure the loyalty of the refined domain surface to the original CAD model. Thus, this ap-proach is inapplicable to simulations with stringent requirements on the geometry accuracy of the model.

2.2. Summary of contributions

In summary, the first and the second approaches reviewed in Section 2.1 need to change inter-domain interfaces during different phases of mesh generation and the full reuse of sequential codes cannot be easily achieved. Besides, many inter-processor operations are re-peatedly employed in these approaches. Nevertheless, the third parallel mesh generation approach could overcome these drawbacks to some extents and is thus widely adopted in recent studies. Nevertheless, the scalability of existing algorithms of this type is still limited due to various issues, as summarized in Section 2.1.

A third type of parallel mesh generation approach is proposed in this study, in order to achieve a full reuse of existing sequential surface

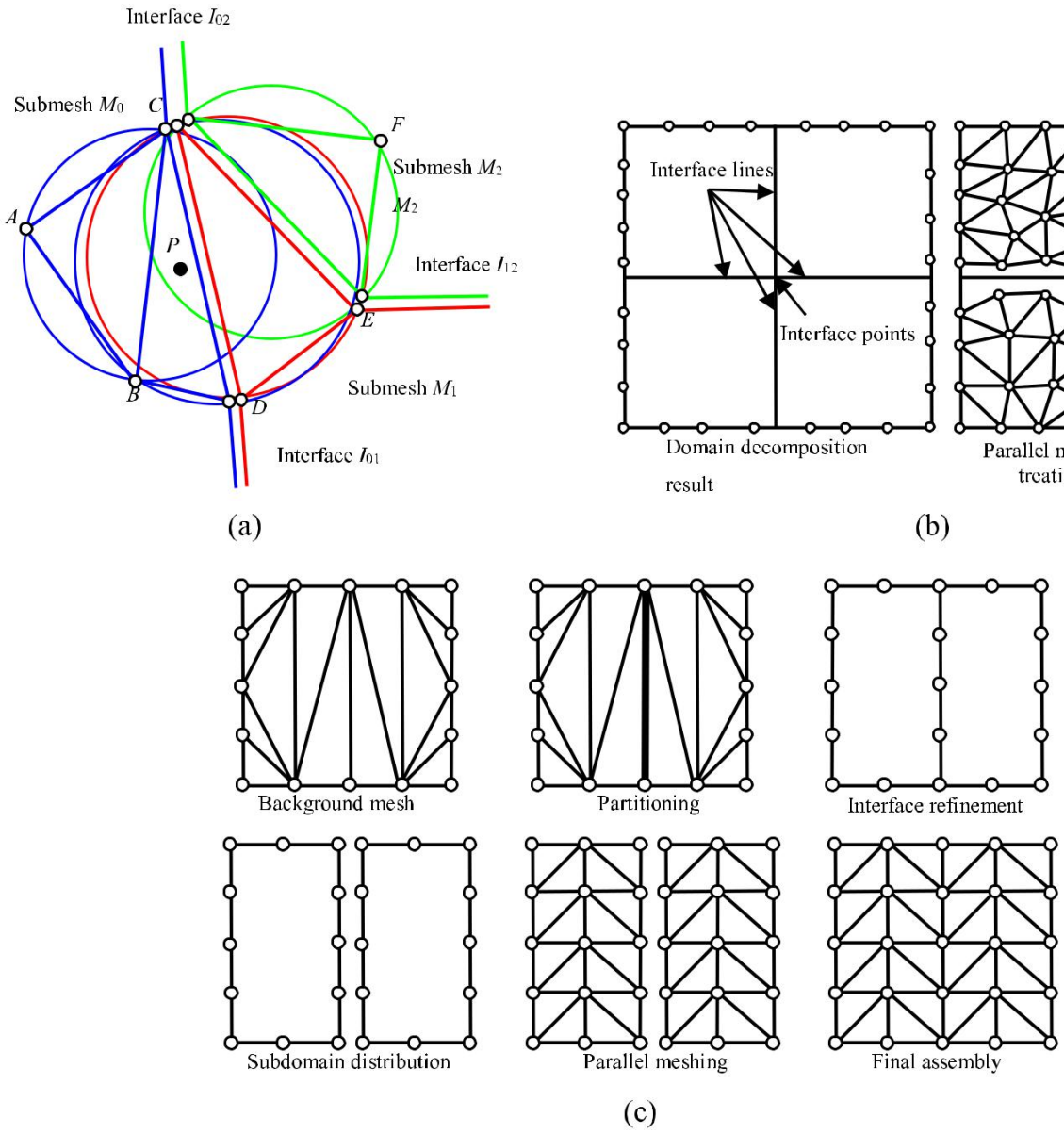

Fig. 2. Graphical illustrations of three parallel mesh generation approaches. (a) is a remake of Fig. 2 in [12], illustrating a cavity crossing multiple processes. (b) is a remake of Fig. 24.8 in [11], illustrating the basic idea of the parallel approach that meshes interfaces after meshing subdomains. (c) is a remake of Fig. 24.3 in [11], illustrating the main steps involved in the parallel approach that premeshes the interfaces.

meshers and tetrahedral meshers. A novel domain decomposition ap-proach is suggested, which accounts for the high scalability perfor-mance of the proposed parallel mesher because of the desirable features listed as below.

(1) Both the surface and the interior background elements are much larger than the final elements. Thus, the runtime of the steps of background mesh creation, simplification and partitioning are not so evident.

(2) The time cost of a sequential run of the subdomain boundary re-meshing step certainly increases when a finer computational mesh is required. However, an efficient parallel scheme exists for this step. Thus, the parallel runtime of this step is reasonably limited.
(3) The yielded inter-domain interfaces do not contain any feature that might have negative impact on element quality. Therefore, the subsequent parallel procedures of mesh generation and improve-ment will be efficient because they could treat subdomains in-dependently without compromising element quality.

For comparison purpose, Table 1 lists a few features of the proposed algorithm and the reviewed parallel mesh generation algorithms, with a summary of their scalabilities.

3. Domain decomposition

3.1. Overview of the workflow

As illustrated in Fig. 1, the domain decomposition approach inputs a problem domain depicted by a CAD model, and finally outputs a set of subdomains depicted by the same fine surface triangulation as the final mesh desired. Fig. 3 presents the detailed workflow of the proposed domain decomposition approach. It scales up the sizing function to create a background mesh with both its boundary and interior meshes much coarser than that of the final requirement. After that, it simplifies the surface and volume parts of the background mesh [10], respectively. Different with those approaches that directly decompose the non-simplified background mesh into subdomains (such as the approach proposed in [18]), the proposed approach decomposes the simplified mesh instead, which not only prevents badly shaped faces and small dihedral angles from appearing on inter-domain interfaces, but also prevents small angles from appearing on inter-surface inter-faces. Finally, subdomain boundaries are remeshed according to the input sizing function. In this step, the subsurfaces classified on the CAD model are extracted first such that a set of trimmed surfaces could be reconstructed. An advancing front surface mesher is thus reused to mesh these reconstructed sub surfaces to ensure the generation of sur-face mesh fitting into the original CAD model. After that, a face-wise triangulation scheme could be applied in the inter-domain faces but with a particular treatment on those faces containing more than one edges classified on CAD surfaces, given the fact that no inter-domain faces contain angles smaller than a user-specified threshold. Note that the suggested remeshing technique for subsurfaces may change sub-domain shapes slightly. Consequently, local intersections may occur between a pair of inter-domain faces or between a surface face and an inter-domain face. Since these intersections are only related with a few inter-domain faces, they can be removed by simply merging sub-domains adjacent to those intersecting inter-domain faces. Meshes treated in these steps are very coarse. Nevertheless, the steps of subdomain boundaries remeshing and local intersections removal are parallelized since they need to treat much larger mesh data. Presently, the generation, simplification and decomposition steps of background meshes are executed in sequential since the background meshes treated in these steps are very coarse. Nevertheless, the steps of subdomain boundaries remeshing and local intersections removal are parallelized since they need to treat much larger mesh data.

Table 1
Typical parallel mesh generation algorithms and their scalability.

Type[a]	Approach	Sequential mesher[b]	Backg. mesh[c]	Scalability
I	Chrisochoides & Nave [12]	Delaunay	Type I	Negatively impacted by intensively executed inter-process operations.
II	de Cougny & Shephard [13]	Octree-based	Type II	Desirable data was reported on 32 processes. No data are reported on more processes.
	Löhner [14]	AFT	Type II	Apply the parallelism at a local level, and yield a speed up by an order of magnitude
	Löhner [2]	AFT	Type III	Much higher by applying the volume-level parallelism, but further improvement is possible.
III	Galtier & George [15]	Delaunay	Type IV	Negatively impacted by the time-consuming domain decomposition step.
	Larwood, et al. [16]	Delaunay	Type IV	
	Chen, et al. [4]	Delaunay	Type IV	
	Said, et al. [17]	Delaunay	Type I	Limited scalability was reported.
	Ito, et al. [18]	AFT	Type V	Negatively impacted by the sequential mesh improvement procedure.
	Proposed	Delaunay	Type V	Three features account for its high scalability: very coarse background mesh; efficiently parallelized interface refinement step; efficiently parallelized mesh generation and mesh improvement procedures.

[a] Type I. Those meshing interfaces as they mesh subdomains. Type II. Those postmeshing the interfaces; Type III. Those premeshing the interfaces.

[b] The listed sequential meshers are only those verified in the literature. In principle, the type-III approaches can fully reuse any types of sequential meshers that respects mesh boundaries.

[c] Type-I. The constrained tetrahedralization of the same fine surface mesh as the final mesh desired. Type-II. Cartesian mesh created by the octree-based approach. Type-III. The refinement of the type-I mesh by inserting a few interior field points. Type-IV. The same fine surface mesh as the final mesh desired. Type-V. Both the domain boundary and interior are discretized with a scaled-up sizing function.

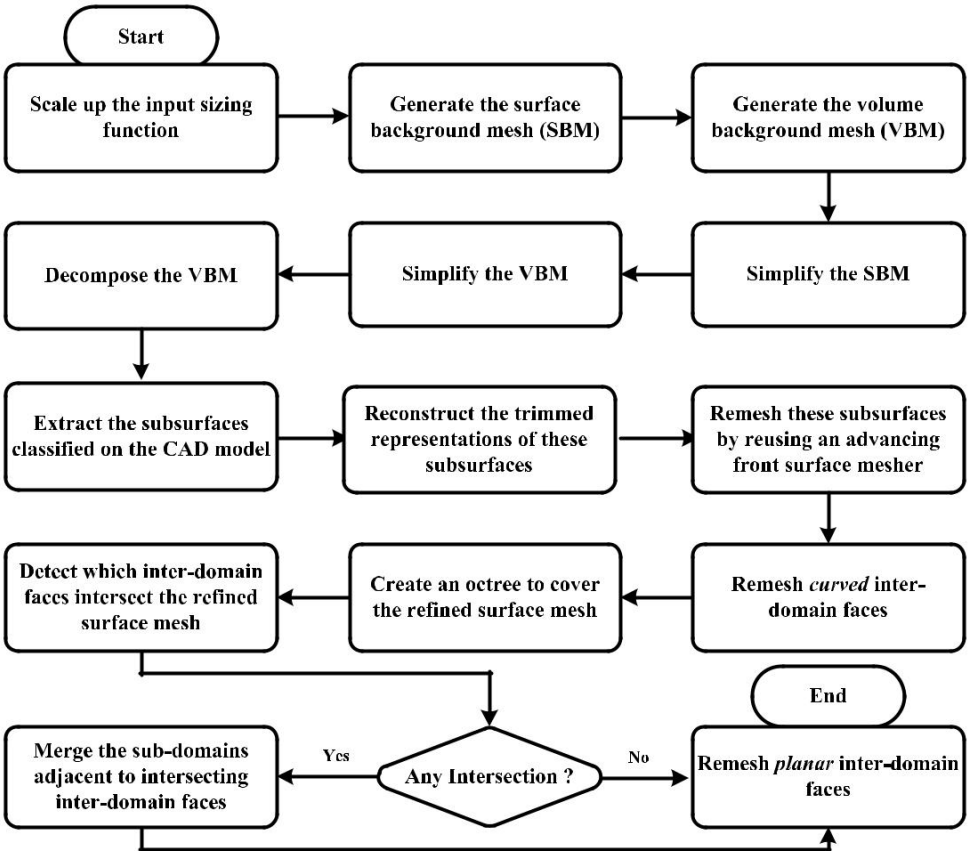

Fig. 3. The workflow of the proposed domain decomposition approach.

3.2. Background mesh generation

The element sizing function considered in this study is defined by a background mesh and many grid sources [19,20]. This sizing function can be roughly coarsened by scaling up the sizing values defined on background nodes and grid sources. In the region where geometrical proximity features are prominent [19], the scaled sizing values may be too large so that the resulting background mesh for domain decom-position contains badly shaped elements in this region. This is not an issue in practice because proximity features only have impact on a small fraction of elements, and the proposed mesh simplification approach can tolerate a background mesh composed of a certain percentage of badly shaped elements.

After coarsening the sizing function, a surface mesher [10] and a tetrahedral mesher [21,22] are employed to generate the SBM and the VBM, respectively.

3.3. Background mesh simplification

Fig. 4 presents a general flowchart of the mesh simplification algo-rithm [10] adopted in this step. The main computations are performed on the side-dual graph (SDG) and the element-dual graph (EDG) of the input mesh. Here, elements and sides refer to (d-1)-dimensional and (d-2)-dimensional mesh entities in a D-dimensional mesh, respectively. In an SDG, each graph node refers to one side of the dual mesh, and a graph edge exists between two neighboring sides. Likely, each graph node in a EDG refers to one element of the dual mesh, and a graph edge exists between two neighboring elements.

Firstly, a shape analysis step is executed on the input mesh to classify mesh sides that do not meet prescribed shape quality requirements as removable (denote this set of mesh sides by S_1). At the same time, the SDG of the input mesh is simplified by deleting nodes dual to mesh sides in S_1. After that, the node-deletion algorithm presented in [10] is em-ployed to simplify the SDG further such that the dual mesh after sim-plification contains no small dihedral angles below a user specified threshold (denoted by β_{th}). After this procedure, more removable mesh sides are classified (denote this set of mesh sides by S_2). Finally, with S_1 and S_2 as inputs, the simplified EDG is computed by contracting graph edges dual to these sides, and the mesh dual to this simplified EDG is exactly the desired simplified mesh.

Conceptually, the output is a simplified result of the input mesh generated by merging elements adjacent to removable mesh sides into super-elements. However, in the present studies, the goal of mesh sim-plification is to provide a simplified EDG to the graph partitioner for

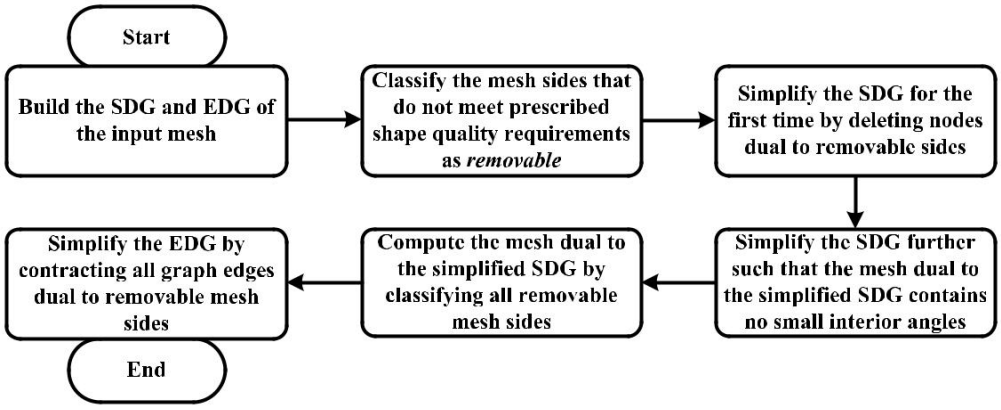

Fig. 4. The adopted mesh simplification approach.

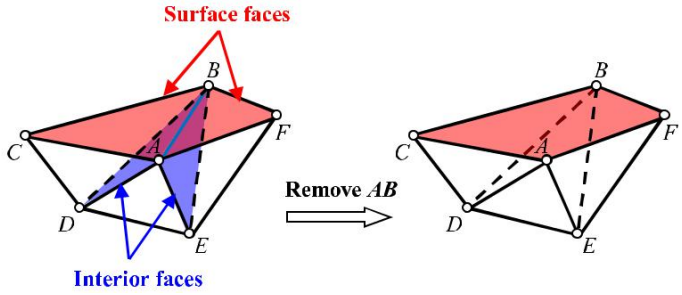

Fig. 5. An example illustrating how to simplify the surface background mesh by removing a surface mesh edge AB. To maintain the mesh conformity, two in-terior faces adjacent to AB are removed, which results in the simplification of the volume mesh: three tetrahedral elements meeting at AB are merged into a polyhedral super-element.

domain decomposition. Therefore, we do not remove those removable sides for real, but simply classify them.

As seen in Fig. 3, the SBM needs to be simplified at first. As a result, lots of super-faces are formed on the surface boundary by merging neighboring surface triangles. To ensure no super-faces contain angles below 50° (apart from those formed by boundary lines classified on CAD curves), the mesh simplification approach is executed by setting the dihedral angle threshold β_{th} to be 50°. Besides, to maintain the mesh conformity, the VBM needs be simplified at the same time by removing a surface mesh edge whilst removing all the mesh faces adjacent to this edge, as illustrated in Fig. 5.

Next, the VBM is simplified further by setting β_{th} to be 30° to ensure no super-elements contain dihedral angles below 30° (apart from those formed by surface triangles). Meanwhile, the shape analysis step is enabled to remove all interior faces having angles below 30°.

3.4. Background mesh decomposition

An EDG dual to the simplified VBM is computed after the mesh simplification step. It is then sent to a graph partitioner [23,24] for domain decomposition. To balance subdomain sizes and minimize sizes of inter-domain interfaces, the EDG needs to be weighted appropriately. Note that the initial VBM is created with the input sizing function scaled up. For simplicity, the weights of the EDG nodes and edges are in-itialized to be 1. Therefore, in the simplified EDG, the node weight refers to the number of tetrahedral elements being merged into the element dual to this graph node, and the edge weight refers to the number of triangular faces being merged into the face dual to this graph edge, respectively.

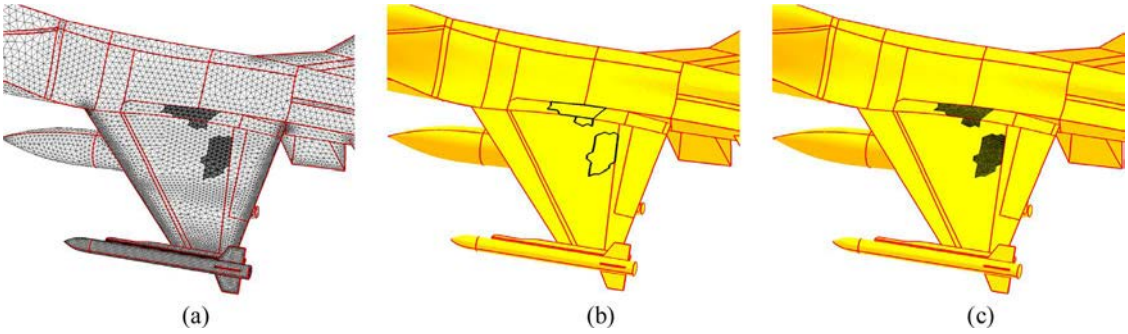

Fig. 6. Illustration of the remeshing procedure of regions filled up by surface faces. (a) Regions needed to be remeshed (in dark grey color); (b) Mesh edges bounding these regions (in dark grey color); (c) The fine elements after remeshing (in dark grey color).

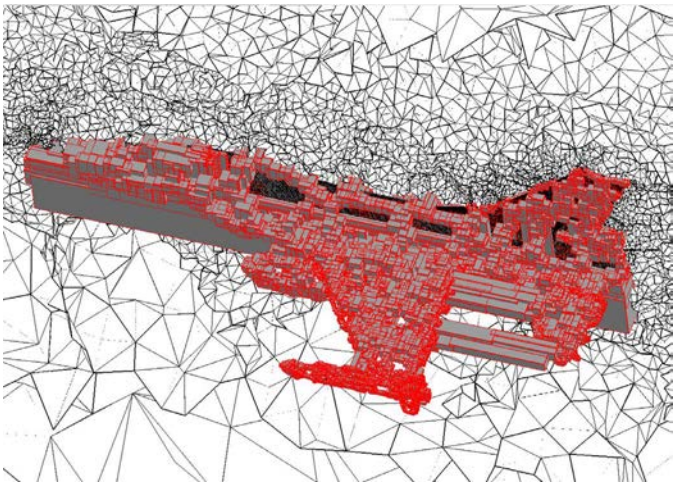

Fig. 7. Illustration of an octree built for detecting the possible intersections between the faces bounding subdomains of an aircraft model.

3.5. Remeshing of subdomain boundaries

After the mesh decomposition, each element of the simplified VBM is classified on a subdomain. Because a super-element refers to a set of neighbouring tetrahedral elements, the elements of the initial VBM can thus be classified in a similar manner. Then, boundary faces of each subdomain can be extracted and classified as the surface faces and inter-domain faces, referring to the faces classified on the input CAD surfaces and those shared by two subdomains, respectively. In the remeshing step, the surface faces are treated first, and the inter-domain faces are then refined individually.

Fig. 6 presents an example illustrating the remeshing procedure on the areas covered by the surface faces, taking the steps as below.

(1) A set of bounding surface faces is collected for each subdomain. In the case that these faces cover more than one CAD surface, the set of faces are further subdivided into smaller sets such that each set of faces, referred to as a region, only covers a single CAD surface. In brief, this surface is named the host surface of the region. In the rare case that the faces belonging to a region are multiple connected, the subdivision is continued until each region becomes single con-nected.

(2) A trimmed representation is defined for each region [10,25] with its supporting surface being the same as its host surface, and the boundary loop being depicted by a set of parametric curves corre-sponding to the mesh lines bounding the region. For each of these mesh lines, if it is classified as a CAD curve, its parametric re-presentation can be defined by trimming the curve; otherwise, a further procedure is necessary to project the mesh line onto the supporting surface [10].

(3) Finally, the artificially trimmed surfaces are meshed by an advan-cing front mesher [10].

The inter-domain faces can be further classified into curved or planar faces, corresponding to those with at least one boundary edge classified on the CAD model (referred to as curved edges hereafter) or those bounded by three interior mesh edges. A mesher for planar domains could be used to triangulate planar faces individually. However, an additional projection step is required to treat curved faces, in which the mesh points at curved edges are replaced by their counterparts located on the input CAD model. It is worth noting that the positions of these points are pre-computed in the remeshing procedure of surface faces.

3.6. Intersection detection and suppression

In this step, the possible intersections between the faces bounding subdomains are checked. For planar inter-domain faces, their shapes remain unchanged during the remeshing step. To save the time cost of intersection check, the intersection computation is executed before the remeshing step for planar inter-domain faces. Thus, the original planar inter-domain faces, whose number is far fewer than the number of their remeshed counterparts, are used in the intersection computation.

To speed up the intersection computation further, an octree that covers the problem domain is built by using a top-to-bottom recursive subdivision (see Fig. 7 for an example). At the beginning, surface faces and curved inter-domain faces after remeshing are related with the octants overlapping these faces. No further subdivision is conducted for an octant if the number of faces associated with the octant is smaller than 100. Once the subdivision stops, planar inter-domain faces are related to corresponding leaf octants.

Finally, a face-face intersection check routine is called on any pair of the faces related to the same octant. Any pair of surface faces will be excluded from these checks because the input CAD model is supposed to be free of self-intersections. Likely, any pair of planar inter-domain faces are excluded because these faces are extracted from a non-over-lapping volume mesh and surely not intersected by each other.

In the rare case where an intersection is detected, one intersected face must be an inter-domain face. We can simply remove this inter-section by merging the subdomains adjacent to this inter-domain face. Thus, all the inter-domain faces between the merged subdomains are removed. To save computing time, the subsequent intersection com-putations involving these removed inter-domain faces are skipped over.

4. Mesh generation and quality improvement

The domain decomposition approach has enabled us to build up a parallel mesh generation pipeline. The structure of this pipeline is based on a manager/worker model, as shown in Fig. 8. The volume domain is depicted by a watertight CAD model. On a manager process, this domain is partitioned into the required number of subdomains by the proposed domain decomposition approach. Here, the over-decomposi-tion techniques [9] is adopted to subdivide the domain into far more subdomains than the number of workers.

Next, boundary data of subdomains are passed from the manager process to all the worker processes. A dynamic load balancing strategy is used to ensure that no worker processes are idle [9,16]. Volume meshes of subdomains are created by a Delaunay-based tetrahedral mesher on the worker processes [21,22]. Over-decomposition surely leads to more than one subdomains being sent to the same worker. The volume meshes of these subdomains can be assimilated to a larger mesh on this worker process. Furthermore, if the memory size of the manager process allows, the assimilated submeshes on the worker processes can be sent back to the manager process for a further assimilation.

Followed with subdomain mesh generation and assimilation are mesh quality improvement. If one single large mesh is assimilated on the manager process, a sequential run of mesh improvement will end the mesh generation pipeline. However, the assimilation on the man-ager process is not encouraged because the focus of this study is on the parallel preprocessor for a parallel simulation that requires a dis-tributed mesh as input. Therefore, a parallel mesh improvement ap-proach is suggested in this study.

To ensure the sizes of submeshes are roughly equal and the number of inter-domain faces is minimized, we suggest performing a parallel mesh repartitioning process before parallel mesh improvement. This repartitioning process first simplifies the initial distributed volume mesh by the approach presented in Section 3.3, and then repartitions the simplified mesh to prevent faces containing angles below 30° and dihedral angles below 30° from appearing on inter-domain interfaces. Consequently, a sequential mesh improver [26] can be executed on all the processes in parallel, fitting into this boundary without compro-mising the mesh quality.

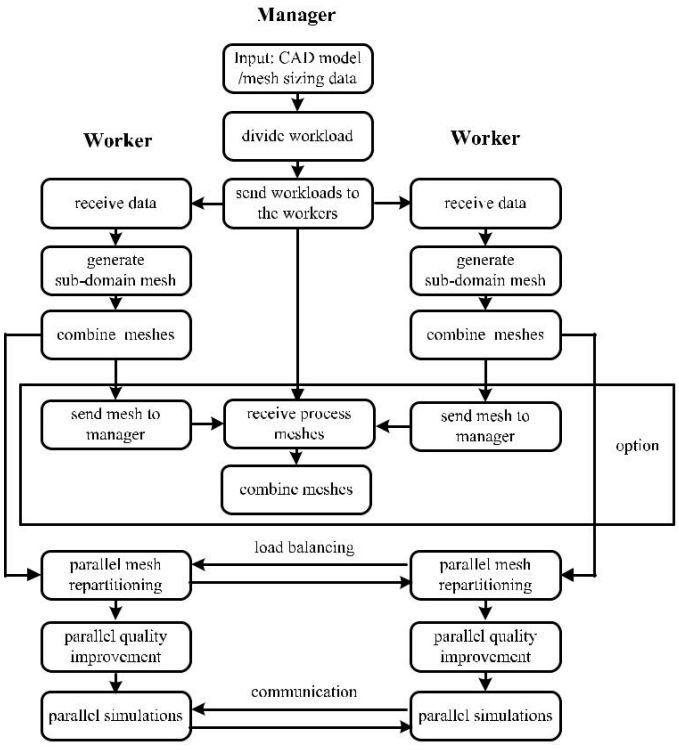

Fig. 8. The structure of the developed parallel mesh generation pipeline.

Note that a parallel implementation of the mesh simplification ap-proach is needed in this stage, which takes the steps as below [4].

(1) Simplify the EDG of each submesh concurrently by regarding inter- domain sides as boundary sides.
(2) For those inter-domain sides with undesirable shapes or those bounding small dihedral angles, penalise their dual EDG edges with large weights.

(3) Repartition the EDG by ParMETIS [23,24].
(4) Redistribute the submeshes to comply with the graph partitioning result.

5. Results and discussion

All the numerical experiments presented here were conducted on a Dawning TC5000 Cluster composed of 36 computational nodes, and the memory size for each node is 32GB. Each computational node contains two eight-core CPUs and the CPU frequency is 2.6 GHz.

5.1. Performance data of the proposed algorithm

5.1.1. The F16 aircraft model

The primary test geometry is a fully loaded F16 aircraft model. Grid sources are configured to refine the mesh in the regions where smaller element sizes are required (see Fig. 9a) [20,27]. Four pairs of surface and volume meshes with various sizes are generated by adjusting a factor s to scale the spacing values of sources. In the ascending order of mesh sizes, these meshes are named F16-I, F16-II, F16-III and F16-IV, respectively. Fig. 9b presents the F16-I mesh created on 32 computer cores, where the elements in diff erent colors were generated on dif-ferent computer cores. Figs. 9c and 9d present some close-up views of the surface and volume meshes.

The first two tests are conducted to evaluate the scalability of the proposed algorithm. In the first test, the meshes with various sizes are generated by fixing the number of computer cores to be 32. In the second test, the meshes with roughly equal sizes are generated on dif-ferent numbers of computer cores by inputting the same sizing function. Tables 2 and 3 list the statistics of all the meshes and the timing data of the proposed algorithm in these two tests, respectively, along with the timing data of the four individual steps involved in this algorithm. These steps are consecutively referred to as domain decomposition, mesh generation (including subdomain mesh assimilation), mesh re-partitioning (including mesh redistribution) and quality improvement.

The relative speed of the algorithm and that of each step of the algorithm are suggested as a major performance index here, which equal to the number of elements dividing the product of the runtime and the number of cores. Figs. 10(a) and 10(b) depicts how these re-lative speed values change against mesh magnitudes (for the first test) and against computer core numbers (for the second test), respectively. In both figures, it could be seen that the relative speeds of the steps of mesh generation and quality improvement remain rather stable, which can be partially attributed to the communication ways in both pro-cesses: communications exist only between the manager processor and the worker processors in the mesh generation process, and no com-munications are involved in the improvement process at all. The mesh repartitioning step is accomplished by employing ParMETIS [23,24]. Slight drops are observed for the speed value of this step in both tests, whenever the mesh size increases, or the number of computer cores increases. Nevertheless, since the runtime of the mesh partitioning step takes only a small fraction of the total runtime, this drop does not impact the scalability of the entire parallel scheme evidently.

The focus of this study is on the domain decomposition approach, which in general takes four steps, referred to as background mesh generation, background mesh simplification and decomposition, inter-section detection and remeshing of subdomain boundaries. Most parts of the first two steps are executed in sequential, and their timing per-formance is in relation with the size of the initial coarse background mesh. The last two steps have been parallelized, since both steps need to treat much finer meshes and are thus much more timing-consuming if executed in sequential: the intersection detection step treats a mesh with its interior as coarse as the initial background mesh but with its surface part refined to the final resolution; and the remeshing step needs to further refine this mesh by triangulating the inter-domain boundaries into elements of the final resolution.

In all the cases of the first test, the last two steps of domain de-composition consume more computing time in the bigger mesh cases. However, the same background mesh is used in the first two steps of domain decomposition since a fixed number of computer cores are in-volved. These two steps thus consume roughly equal computing time in all the cases of the first test. As a result, the total runtime of domain decomposition does not increase as fast as the magnitude of the final mesh does. Comparing the smallest and largest mesh cases, although the size of the largest mesh (one billion elements) is about 35.8 times of that of the smallest one (28 million elements), the runtime of domain decomposition for the largest mesh case (93.6 s) is only 2.5 times of that for the smallest mesh case (36.8 s). Therefore, an increase of the re-lative speed against mesh magnitudes is observed for domain decom-position in this test, see Fig. 10a and Table 2.

As far as the second test is concerned, the timing performance of domain decomposition depends on more factors. To evaluate these factors more closely, Table 4 lists a breakdown of the total runtime of domain decomposition in this test. In the first two cases of this test, a background mesh including one and half million elements is used, and the runtime of domain decomposition goes down from 94.9 s to 70.6 s when the number of computer cores is doubled to 32 from 16. This is mainly attributed to the efficient parallelization of the remeshing step, as seen in Table 4. In the case of 64 computer cores, a finer background mesh including 2.9 million elements is adopted. Accordingly, the run-time of the first two steps of domain decomposition is roughly doubled in this case. As a result, the total runtime of domain decomposition

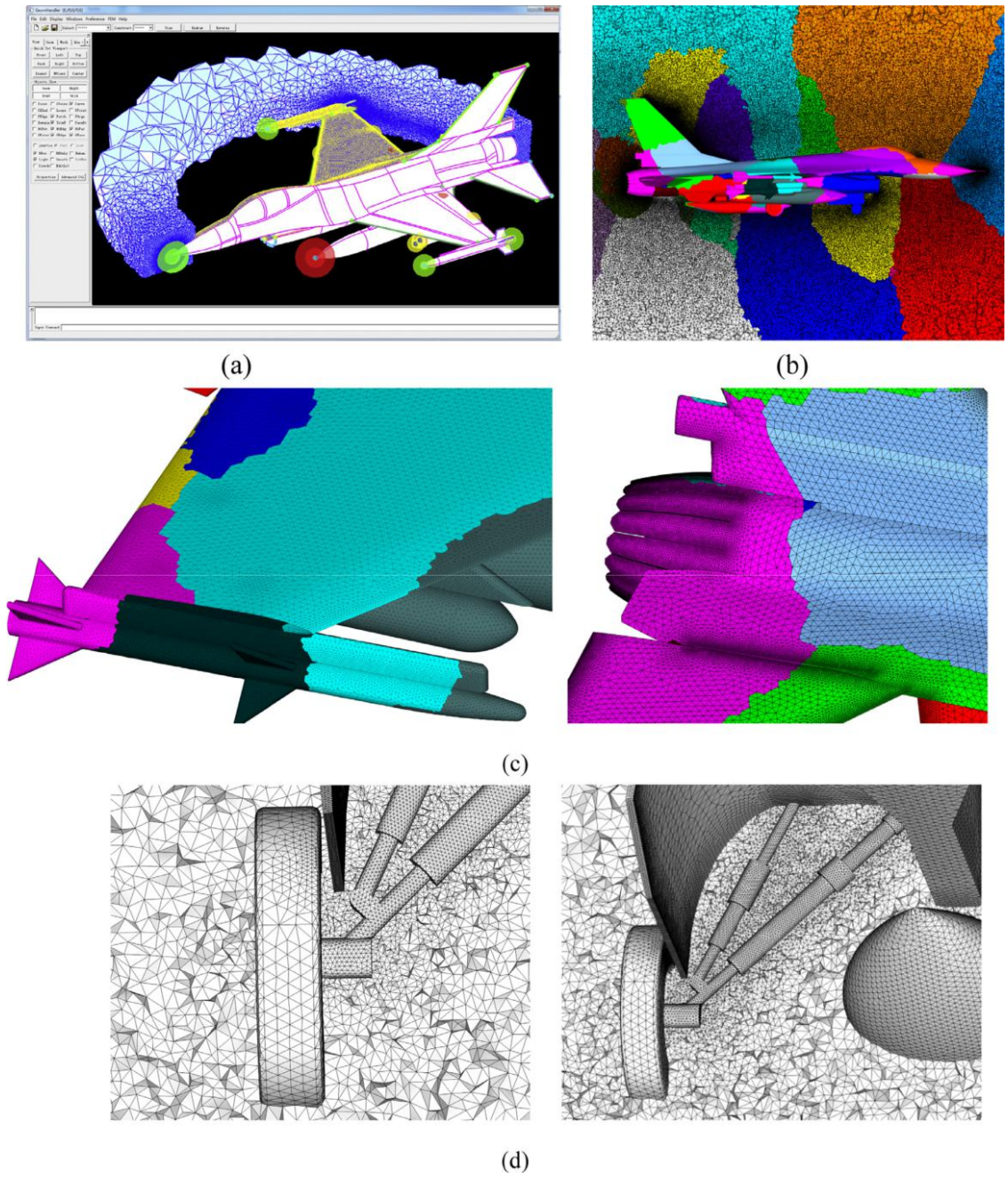

Fig. 9. The F16 aircraft model and the meshes on it. (a) Visual representation of the geometry, the sources and the surface and volume meshes in the graphic interface of HEDP/PRE [27]. (b) A mesh generated in parallel. (c) Close-up views of the surface mesh. (d) Close-up views of the volume mesh.

Table 2
Performance data when using the proposed algorithm to create four meshes with various sizes for the F16 aircraft model on 32 computer cores.

Model name		F16-I	F16-II	F16-III	F16-IV
General data	No. of cores	32			
	No. of background tetrahedra	1.5M			
	No. of tetrahedra	28M	121M	412M	1B
	No. of surface elements	362K	1M	2.5M	3.8M
Timing (seconds)	Entire scheme	102.5	341	1085	2648.7
	Domain decomposition	36.8	46.4	70.6	93.6
	Mesh generation	6.8	30.6	107.6	275.8
	Mesh repartitioning	2.4	13.0	46.7	116.8
	Mesh improvement	56.5	251.0	860.1	2162.5
Relative Speed (elements/second/core)		8.5K	11.1K	11.9K	11.8K

Table 3
Performance data when using the proposed algorithm to create the meshes with about 412 million tetrahedra on different number of computer cores.

Model name		F16-III			
General data	No. of tetrahedra	≈412M			
	No. of surface elements	2.5M			
	No. of cores	16	32	64	128
	No. of bkg. tetrahedra	1.5M	1.5M	2.9M	2.9M
Timing (seconds)	Entire scheme	2137.3	1085	576.9	325.6
	Domain decomposition	94.9	70.6	78.7	72.1
	Mesh generation	204.1	107.6	54.1	28.0
	Mesh repartitioning	85.2	46.7	25.2	14.4
	Mesh improvement	1753.1	860.1	418.9	211.1
Relative Speed (elements/second/core)		12.0K	11.9K	11.2K	9.9K

increases slightly by comparison with the first two cases. As expected, this runtime is reduced from 78.7 s to 72.1 s when we invest 64 more computer cores but keep using the same background mesh. Because the runtime of the sequential part of domain decomposition begins to dominate when 128 computer cores are employed, no clear benefits would be achieved by continuously doubling the number of computer cores. Nevertheless, it is worth noting that, since the most timing consuming step of domain decomposition (i.e., the remeshing step) has been parallelized, the runtime fraction of domain decomposition in the entire parallel meshing scheme is kept at a rather low level in all the cases of the second test. As a result, the benefit for the total meshing time owing to the investment of more computer cores is much more

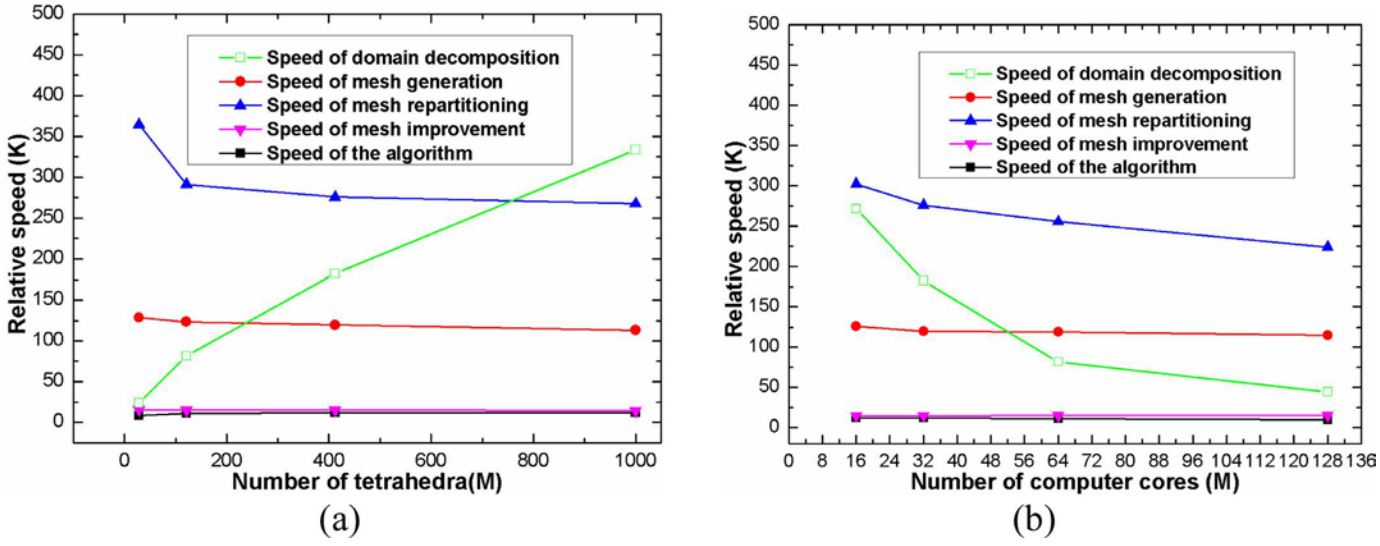

Fig. 10. Change of the relative speed of the algorithm and that of each step of the algorithm (a) against mesh magnitudes, and (b) against computer core numbers.

Table 4
Performance data of the proposed domain decomposition algorithm.

	Model name	F16-III			
General data	No. of tetrahedra	≈412M			
	No. of cores	16	32	64	128
	No. of bkg. tetrahedra	1.5M	1.5M	2.9M	2.9M
	No. of subdomains (before mergence)	160	320	640	1280
Timing (seconds)	Entire domain decomposition scheme	94.9	70.6	78.7	72.1
	Bkg. mesh generation	13.7	13.7	23.6	24.7
	Bkg. mesh simplification and decomposition	13.8	13.8	27.0	26.4
	Intersection detection	3.7	3.3	2.7	2.4
	Remeshing of subdomain boundaries	63.7	39.8	25.4	18.6
Data for intersection suppression	Diff erence of the number of subdomains	2	4	5	7
	Maximal size of subdomains	2	2	3	3

evident than that for the domain decomposition step. Hopefully, we can reduce the total meshing time further by doubling the number of computer cores.

Apart from the timing data of individual steps of domain decom-position, Table 4 lists two more indices in its last two rows to evaluate the solution to the intersection issue. As detailed in Section 3.6, the proposed solution removes intersections by merging subdomains ad-jacent to intersecting inter-domain faces. The first index refers to the diff erence of the number of subdomains before and after subdomain merging. The second index is the maximal size of subdomains. Here, the size of a subdomain equals to one plus the times of executing subdomain mergence operations in the process of forming this subdomain. As seen in Table 4, the intersection problems occur very rarely, and only a few subdomains are merged; thus, the maximal size of merged subdomains is limited. This observation gives us further confidence that the present solutions to the intersection issue might be enough, although this issue may still exist so that more complicated schemes, such as that based on local mesh modification [20] or other alternative approaches, might be necessary.

Table 5 presents the quality statistics of two meshes generated by the proposed algorithm. Because the focus is on the worst elements, the distribution of the minimum dihedral angles (α) in the range of 0° to 24° is listed. A tetrahedron is classified as a low-quality element if its α value is smaller than 24° or as a bad element if its α value is smaller than 12°.

Table 5
Statistics of volume mesh quality.

Model name	F16-I		F16-III
No. of computer cores	1	32	128
No. of tetrahedra	28M	28M	412M
Minimum dihedral angle (α); unit: degree	7.6	7.3	6.5
No. of tets with $0 < \alpha \leq 6$	0	0	0
No. of tets with $6 < \alpha \leq 12$	25	28	89
No. of tets with $12 < \alpha \leq 18$	58	71	504
No. of tets with $18 < \alpha \leq 24$	572	684	2681
No. of tets with $0 < \alpha \leq 12$ (bad elements)	25	28	89
No. of tets with $0 < \alpha \leq 24$ (low-quality elements)	655	783	3274

For the F16-I model, the sequential mesher produces a mesh slightly better than that by the proposed algorithm. Nevertheless, the quality diff erence of these two meshes is not evident. For the F16-III model, the sequential mesher fails because of the memory limitation, while the quality statistics of the mesh created by the proposed algorithm on 128 computer cores are provided.

5.1.2. Two more test models: the Combustor model and the Shuttle model To demonstrate the scalability and robustness of the proposed ap-proach further, two more test models are selected, namely the Combustor model and the Shuttle model, respectively. The combustion model contains plenty of small geometry features (see Fig. 11), and the distributions of element scales adapted to these features are computed by a fully automatic approach proposed in [19]. Nevertheless, grid sources [20,27] are configured on the Shuttle model (see Fig. 12) to reflect the requirement of exterior flow simulations on the distributions of element scales. A scaling factor could be set by the user in both cases to adjust the sizing values at any interior point of the meshing domain, aimed at providing meshes with similar distributions of element scales but various mesh sizes.

In the tests, the finally output mesh size, evaluated by the number of elements the mesh contains, varies from 7 M to 1B, and the number of computer cores varies from 8 to 512. Totally 14 test meshes are created in parallel on these two models, and 7 for each. Table 6 lists the sta-tistics of these meshes and the timing data of the meshing algorithm. The primary performance index is the relative speed, which, although changes slightly case by case, could always maintain at a very high level in all the test cases.

5.2. Comparison with a state-of-the-art domain decomposition approach

Another prevailing idea of domain decomposition is conducted by inputting the surface triangulation of the problem domain, and then subdividing the domain into two separate sub-domains by inserting an inter-domain triangulation inside of the domain. Recursively, the re-sulting sub-domains will be subdivided further if their sizes (e.g., evaluated by the numbers of boundary faces) are greater than a user-specified threshold [9,15,16]. After that, the same approaches of mesh generation, mesh repartitioning and quality improvement could be

.

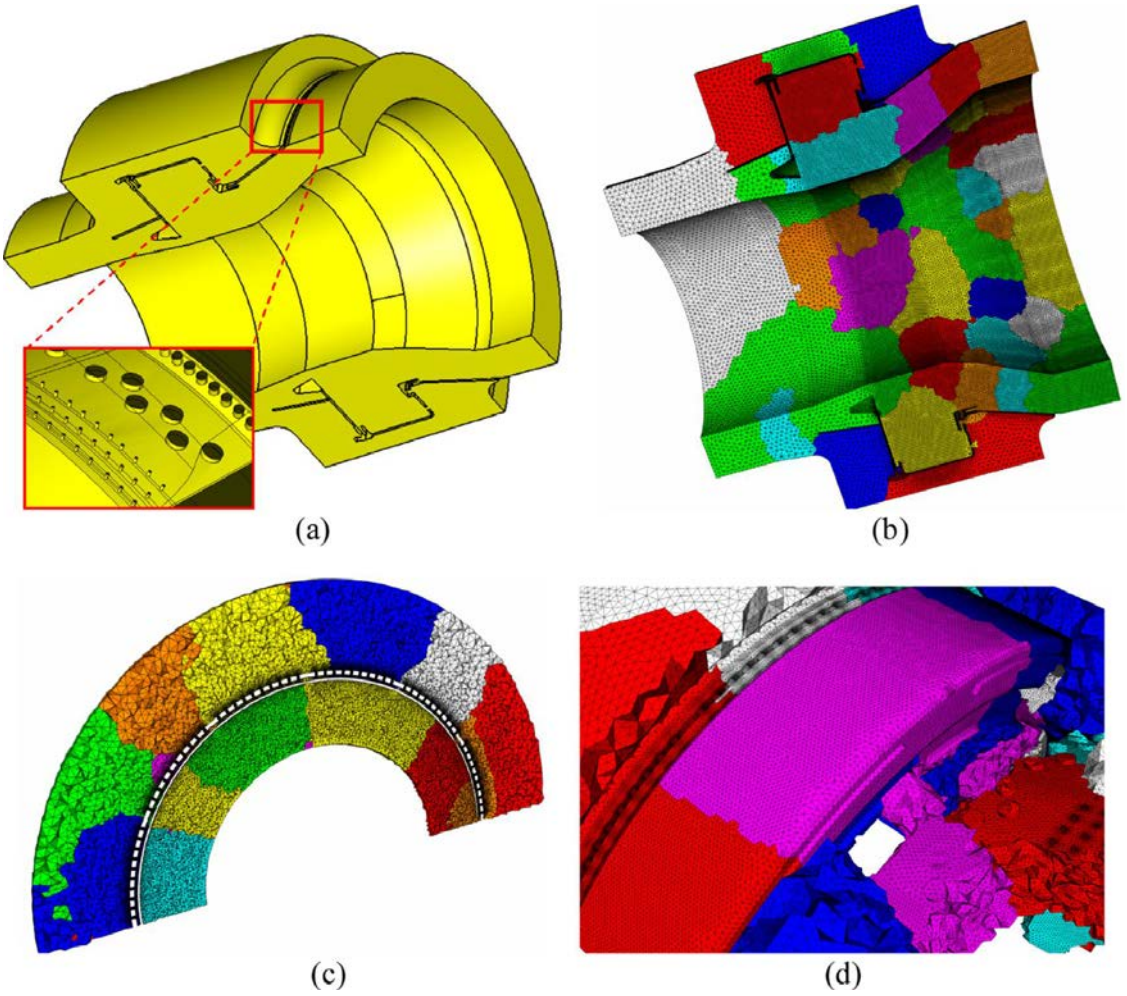

Fig. 11. The Combustor model and the meshes on it. (a) The geometry and a close-up view of the small features contained in the geometry. (b) The surface part of a mesh created in parallel. (c) A cut-view of the volume mesh. (d) A close-up view of the volume mesh.

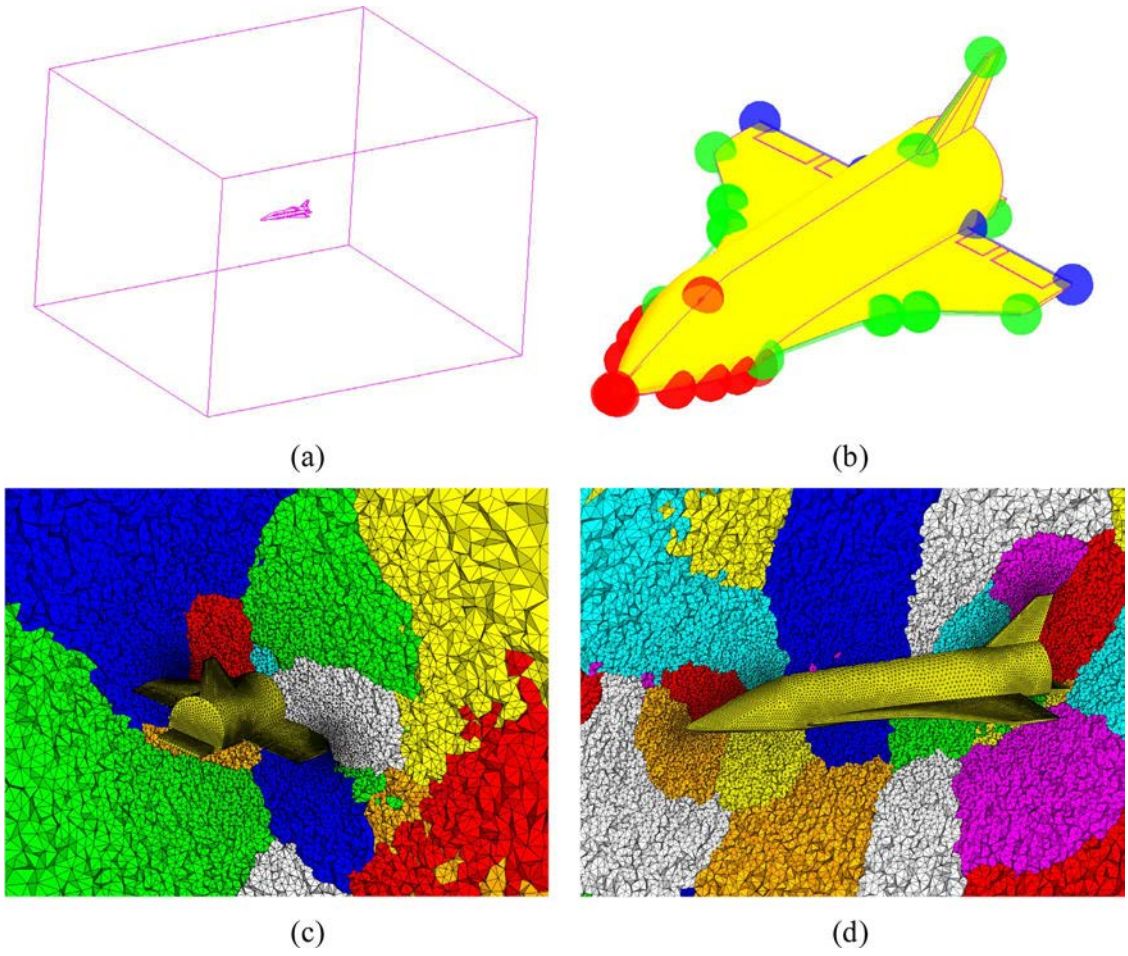

Fig. 12. The Shuttle model and the meshes on it. (a) The outline of the meshing domain. (b) The geometry of the Shuttle model and the grid sources configured on the model. (c) and (d): Two cut-views of a volume mesh created in parallel.

Table 6
Performance data when using the proposed algorithm to create meshes for the Combustor model and the Shuttle model.

Model	#cores	#bkg. elements	#elements	DD timing (seconds)	Total timing (seconds)	Relative speed (elements/second/core)
Combustor	8	1.9M	7M	23.4	82.4	10.7K
	16	1.9M	21M	32.1	123.2	10.8K
	32	1.9M	21M	26.7	75.6	8.8K
	64	3.2M	125M	49.8	159.2	12.3K
	128	3.2M	125M	31.6	82.9	11.8K
	256	3.2M	409M	54.7	174.9	9.1K
	512	3.2M	1B	64.3	224.2	8.8K
Shuttle	8	1.7M	6.8M	21.8	87.4	9.8K
	16	1.7M	25M	32.5	154.8	10.3K
	32	1.7M	25M	26.1	88.7	9.0K
	64	3.0M	130M	48.3	172.8	11.8K
	128	3.0M	130M	32.9	96.4	10.6K
	256	3.0M	418M	56.5	183.6	8.9K
	512	3.0M	1B	67.4	231.8	8.5K

followed as those proposed in this study to create a parallel meshing pipeline. In this section, we will compare the algorithm proposed in this study with this state-of-the-art parallel meshing pipeline, with the focus on the timing performance of domain decomposition and its impact on the total runtime. To be concise, the proposed algorithm and the al-gorithm selected for comparison are referred to as Algorithm 1 and Algorithm 2 in the following discussions, respectively. Note that

Algorithm 1 inputs a CAD model and creates a surface mesh in its do-main decomposition step. Diff erently, Algorithm 2 requires a stand-alone surface meshing step to provide its surface input. To present a fair comparison, we employ a parallel surface mesher [10] to create surface inputs for Algorithm 2.

The similar tests as those mentioned in Section 5.1.1 are conducted by using Algorithm 2: the meshes with various sizes are generated by fixing the number of computer cores to be 32 in the first test, while the meshes with roughly equal sizes are generated on diff erent numbers of computer cores in the second test. Tables 7 and 8 summarize the timing data of Algorithm 2 in these two tests, respectively, where the runtime of other steps refer to the sum of runtime consumed by the processes of

Table 7
Performance data when using the parallel meshing algorithm proposed in [9] to create four meshes with various sizes for the F16 aircraft model on 32 computer cores.

Model name		F16-I	F16-II	F16-III	F16-IV
General data	No. of cores		32		
	No. of tetrahedra	28M	121M	412M	1B
Timing (seconds)	Entire scheme	130.6	544.9	1734.6	3548.1
	Surface meshing	7.6	22.4	44.2	68.3
	Domain decomposition	59.2	231.5	674.0	922.6
	Other steps	63.8	291.0	1016.4	2557.2
Relative speed (elements/second/core)		6.7K	6.9K	7.4K	8.8K
Performance gap with Algorithm 1: = (t_1- t_2)/ t_1, where t_1 and t_2 are runtime of Alg. 1 and Alg. 2, respectively. (%)		27.4	59.8	59.9	34.0

Table 8
Performance data when using the parallel meshing algorithm proposed in [9] to create meshes with about 412 million tetrahedra on diff erent number of com-puter cores.

Model name		F16-III			
General data	No. of tetrahedra	≈412M			
	No. of surface elements	2.5M			
	No. of cores	16	32	64	128
	No. of bkg. tetrahedra	1.5M	1.5M	2.9M	2.9M
Timing (seconds)	Entire scheme	2693.8	1734.6	1286.6	1152.7
	Surface meshing	70.5	44.2	32.1	23.3
	Domain decomposition	580.1	674.0	754.4	874.3
	Other steps	2043.2	1016.4	500.1	255.1
Relative Speed (elements/second/ core)		9.5K	7.4K	5.0K	2.8K
Performance gap with Algorithm 1: = (t_1- t_2)/t_1, where t_1 and t_2 are runtime of Alg. 1 and Alg. 2, respectively. (%)		26.0	59.9	123.0	254.0

mesh generation, mesh repartitioning and quality improvement.

As mentioned in Section 5.1.1, the same background mesh could be used by Algorithm 1 in the first test since a fixed number of computer cores are involved. The increase of runtime consumed by domain de-composition is thus much smaller than the increase of mesh magni-tudes. For instance, the runtime of domain decomposition for the lar-gest mesh case is only 2.5 times of that for the smallest mesh case. However, as far as Algorithm 2 is concerned, we see the runtime of domain decomposition for the largest mesh case (922.6 s) is about 15.6 times of that for the smallest mesh cases (59.2 s). The main factor that accounts for this remarkable increase is that the resolution of the sur-face mesh treated by the domain decomposition approach of Algorithm 2, which must be the same as the final output mesh, increases about 11 times (from 362 K faces to 3.8 M faces).

To show the performance diff erence in the first test more clearly, Fig. 13 compares the runtime of both algorithms and their respective domain decomposition approaches. Meanwhile, the performance gap is computed by the relative diff erence between the runtime of both

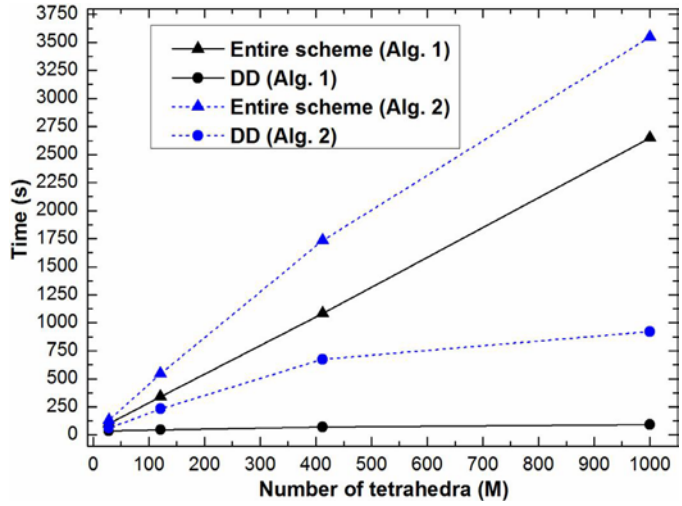

Fig. 13. Timing performance comparison of Algorithms 1 and 2 (DD is short for domain decomposition). The meshes with various sizes are generated on 32 computer cores.

Algorithms (see the last row of Table 7). The savings of computing time of Algorithm 1, by comparison with Algorithm 2, are mainly owing to a speeded-up domain decomposition approach. In this test, this value varies from 27.4% to 59.9%, mainly depending on the runtime fraction of domain decomposition in the entire parallel meshing scheme.

It is worth mentioning that the efficiency benefit due to the speedup of domain decomposition in the first test is not so evident because the mesh improvement step dominates the entire scheme in terms of computing time. Nevertheless, more benefits are observed in the second test, where the runtime fraction of domain decomposition increases with the increase of invested computer cores. In this test, Algorithm 1 performs 1.3 to 3.5 times faster than Algorithm 2 (see the last row of Table 8), mainly due to the performance diff erence of their domain decomposition approaches. For Algorithm 2, the adopted over-decom-position technique [9] requires more subdomains to be created by the domain decomposition approach when more computer cores are in-vested. Thus, the runtime of its sequential domain decomposition step goes up steadily when the number of computer cores is doubled (see Fig. 14a), and domain decomposition starts to dominate in terms of computing time when 64 computer cores are employed. The total runtime of Algorithm 2 is reduced slightly (from 2693.8 s to 1152.7 s) when the number of computer cores is increased from 16 to 128. Ac-cordingly, the achieved speedup, evaluated with the timing data related to the experiments on 16 computer cores, is only 2.34 (see Fig. 14b). Here, it is worth emphasizing again that an efficient parallelization of the recursive bi-division process of surface triangulations used by the domain decomposition approach of Algorithm 2 could be very difficult. If each bi-division process represents a task, the task dependency graph of domain decomposition is a binary tree. The root task is most time-consuming, but it can only be performed sequentially. Moving down of this binary tree, the tasks become smaller and smaller, although the number of tasks that can be performed concurrently increases. This fact justifies why we only provide a sequential domain decomposition ap-proach of Algorithm 2 for comparison. In contrast, the runtime taken by domain decomposition of Algorithm 1 remains at a low level always because the most time-consuming steps of domain decomposition have been efficiently parallelized (see Fig. 14a). The total runtime of Algo-rithm1 is reduced remarkably (from 2137.3 s to 325.6 s) when the number of computer cores is increased from 16 to 128. Accordingly, the achieved speedup is 6.56 (see Fig. 14b).

To compare the scalability of both algorithms further, a third test is conducted on up to 512 computer cores to create meshes containing roughly 1B elements. The least number of computer cores adopted in this test is 32. Therefore, the speedup values are calculated by doubling the number of computer cores used and evaluated with the timing data related to the experiments on 32 computer cores. Table 9 lists the timing data achieved in this third test.

In this test, Algorithm 1 performs 8.1 times faster than Algorithm 2 on 512 computer cores (see Fig. 15), thus reducing the clock time of creating one billion elements from roughly half an hour to less than 4 minutes. For Algorithm 2, the runtime of its domain decomposition step goes up steadily when the number of computer cores is doubled, and domain decomposition starts to dominate in terms of computing time when 128 computer cores are employed. The total runtime of Algo-rithm 2 is reduced slightly (from 1892.4 s to 1744.1 s) when the number of computer cores is doubled to 256. Moreover, when we continue to double the number of computer cores to be 512, the runtime of Algo-rithm 2 (from 1744.1 s to 1786.7 s) even goes up slightly because a more time-consuming domain decomposition step overtakes the entire meshing scheme.

5.3. Comparison with two state-of-the-art parallel mesh improvement approaches

In the above tests for the F16 model, we can see that an efficient parallel mesh improvement approach is another key for efficient and

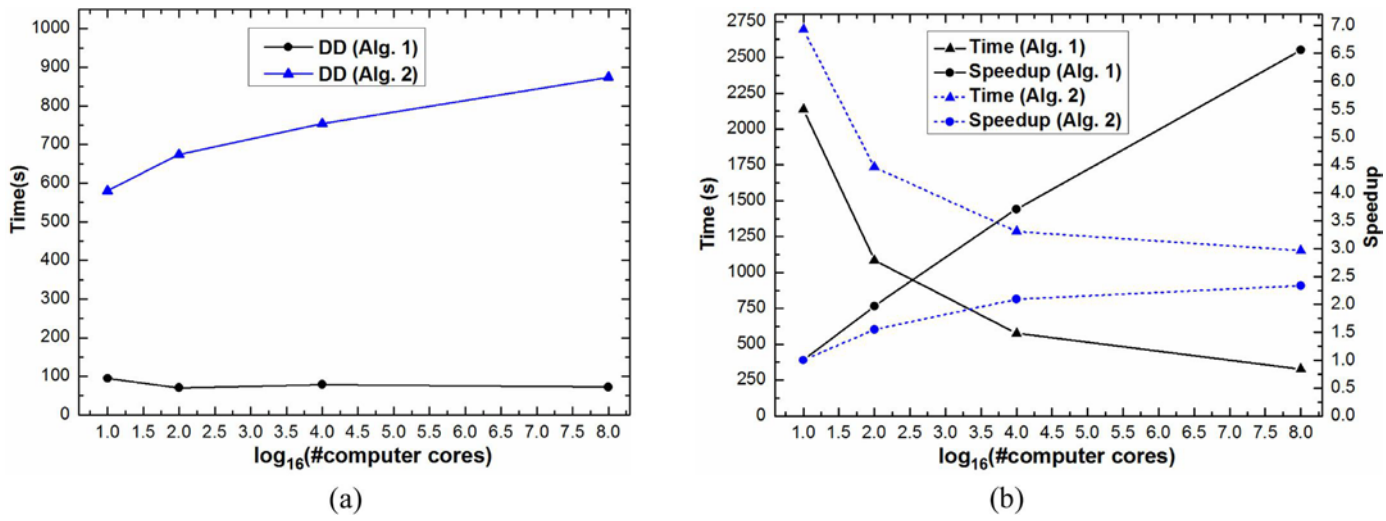

Fig. 14. Timing performance comparison for Algorithms 1 and 2 (DD is short for domain decomposition). The meshes with about 412 million elements are generated on diff erent numbers of computer cores. (a) Timing performance of the domain decomposition step. (b) Timing performance of the entire scheme and its speedup.

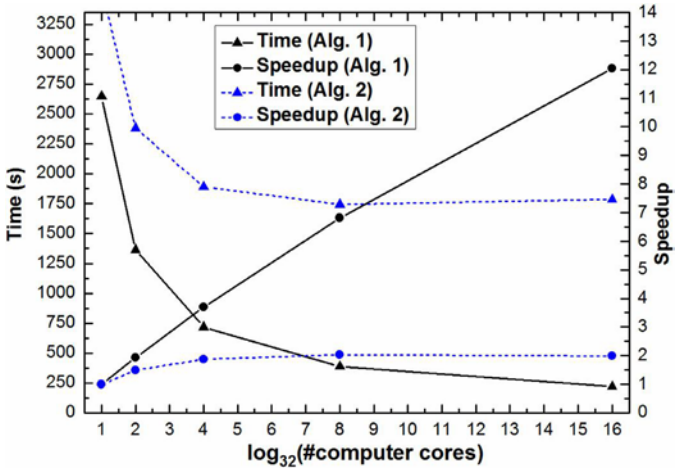

Fig. 15. Comparison for Algorithms 1 and 2: the timing performance of the entire scheme and its speedup in a test where the meshes with about one billion elements are generated on diff erent numbers of computer cores.

scalable generation of large-scale meshes. As seen in Tables 2 and 3, the parallel mesh improver developed in this study performs very well in terms of its scalability. A nearly linear speedup of the parallel mesh improver is observed (see Fig. 16). This desirable parallel performance is mainly due to the domain decomposition approach proposed in this study, which prevents any feature that might have negative impact on element quality from appearing near inter-domain boundaries. There-fore, the parallel mesh improvement procedure can fit into the inter-domain boundaries without compromising the mesh quality. This pro-cedure involves no additional parallel computing costs and therefore runs very efficiently.

In contrast, most existing parallel mesh improvers depend on ad-ditional steps to reduce the negative impact of low-quality inter-domain boundary on element quality. For the comparison purposes, we re-im-plement two parallel improvers of this type.

The first mesh improver takes the following steps to improve a distributed mesh [18]:

(1) Input the distributed mesh, and then repartition the mesh 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.

Table 9

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

The second mesh improver follows the suggestion in reference paper [2]. It performs the first mesh improvement pass as the first improver

Performance data of Algorithms 1 and 2: generating meshes with about one billion tetrahedra on different number of computer cores.

Model name			F16-IV				
General data	No. of tetrahedra		≈1B				
	No. of surface elements		3.8M				
	No. of bkg. tetrahedra		2.9M				
	No. of cores		32	64	128	256	512
Alg. 1	Timing (seconds)	Entire scheme (t_1)	2648.7	1365.5	717.0	388.4	219.8
		Domain decomposition	93.6	86.5	75.3	68.0	60.2
		Other steps	2555.1	1279.0	641.7	320.4	159.6
	Relative Speed (elements/second/core)		12.3K	11.9K	11.3K	10.5K	9.2K
Alg. 2	Timing (seconds)	Entire scheme (t_2)	3548.1	2383.7	1892.4	1744.1	1786.7
		Surface meshing	68.3	50.2	37.3	27.6	21.4
		Domain decomposition	922.6	1053.6	1209.5	1395.3	1604.5
		Other steps	2557.2	1279.9	645.6	321.2	160.8
	Relative Speed (elements/second/core)		8.8K	6.5K	4.1K	2.2K	1.1K
Performance gap: = (t_1-t_2)/t_1. (%)			34.0	74.6	163.9	349.0	712.9

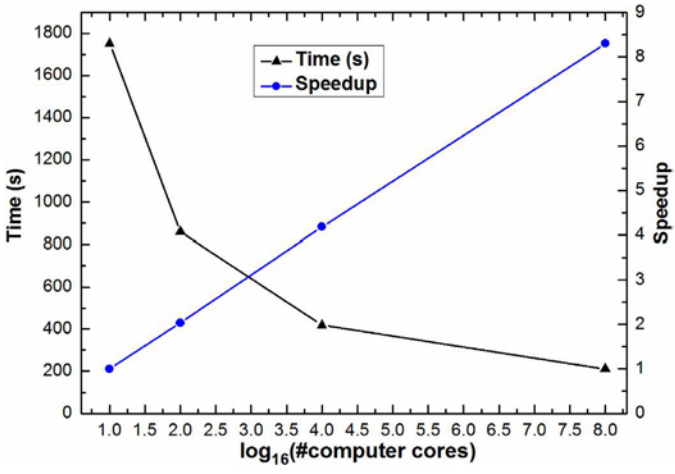

Fig. 16. Timing and speedup data of the proposed parallel mesh improvement approach when using it to improve meshes containing about 412 million ele-ments.

Table 10
Time performance (unit: second) of two re-implemented parallel mesh im-provers. The input mesh is the F16-III mesh (containing 128 submeshes).

Algorithm	Ito et al. [18]	Löhner [2]
Entire scheme	2997.8	556.8
Mesh repartitioning	18.3	18.7
The first mesh improvement pass	148.5	140.6
Collection of elements (for the first improver) or redistribution of elements (for the second improver) near inter-domain interfaces	29.4	36.4
The second mesh improvement pass (executed in sequential for the first improver while in parallel for the second improver)	2801.6	361.1

does, but performs the second mesh improvement pass very differently. After the first mesh improvement pass, the second mesh improver re-distributes 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 im-proved in parallel with the inter-domain boundary fixed.

Table 10 reports the time performance data of both re-implemented mesh improvers. The input is a mesh of the F16-III model (with 128 submeshes), and 128 computer cores were employed in mesh im-provement. The single mesh formed by collecting elements near inter-domain interfaces contains 59 M elements. A sequential run of the mesh improver on this mesh consumes 2801.6 s on 128 computer cores. Thus, the entire mesh improvement process consumes 2997.8 s, which is about 14.2 times slower than the proposed Algorithm 1 (that consumes 211.1 s).

For the second re-implemented mesh improver, many elements are sent from the processes with high rank values to neighbouring processes with small rank values in the step of redistributing elements near inter-domain interfaces. Consequently, the processes with small rank values usually must 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 2 after the redistribution step. This submesh contains about 6.7 M elements, nearly 2.1 times larger than the average size of the redistributed submeshes. This process was therefore the busiest one during the second mesh improvement pass, consuming 361.1 s. Thus, the entire mesh improvement procedure consumes 556.8 s, which is about 5.4 times faster than the first mesh improver, but still about 2.6 times slower than the proposed algorithm.

6. Conclusions

In this study, we propose a novel parallel approach for scalable generation of large-scale tetrahedral meshes. A key innovation of this study is the use of a mesh-simplification based domain decomposition approach. This approach works on a background mesh with both its surface and its interior elements much larger than the final elements, and then decomposes the domain into subdomains containing no un-desirable geometric features in the inter-domain interfaces. In this way, the most time-consuming part of domain decomposition (i.e., re-meshing subdomain boundaries) could be efficiently parallelized, and the other sequential parts consumes reasonably limited computing time since they treat a very coarse background mesh. Consequently, the overall time cost of the domain decomposition approach is limited at a very low level in general.

This novel domain decomposition approach has enabled the build-up of an efficient and scalable parallel pre-processing pipeline. This pipeline inputs a CAD model and then executes the subsequent steps of surface meshing, volume meshing and mesh improvement in a fully parallel and automatic manner. The scalability of this pipeline is well demonstrated by the cases involving hundreds or more computer cores and more than one billion elements. Compared with some other state-of-the-art parallel meshers, the new parallel preprocessor reduces the clock time required by the creation of one billion elements on 512 computer cores from more than half an hour to less than 4 minutes.

Acknowledgements

The authors appreciate the joint support for this project by Science Challenge Project of China (No. TZ2016002), Zhejiang Provincial Natural Science Foundation (Grant Nos. LR16F020002 and LQ14A020003) and the National Natural Science Foundation of China (Grant Nos. 11172267, 11432013, U1630121, 11402229, 11372276), Ningbo Innovative Team: The intelligent big data engineering appli-cation for life and health(Grant No. 2016C11024).

References

[1] Baker TJ. Mesh generation: art or science. Prog Aerosp Sci 2005;41(1):29–63.
[2] Löhner R. Recent advances in parallel advancing front grid generation. Arch Comput Meth Eng 2014;21(2):127–40.
[3] Weatherill NP, Hassan O, Morgan K, Jones JW, Larwood BG, Sorenson K. Aerospace simulations on parallel computers using unstructured grids. Int J Numer Methods Fluids 2002;40(1-2):171–87.
[4] Chen J, Zhao D, Zheng Y, Xu Y, Li C, Zheng J. Domain decomposition approach for parallel improvement of tetrahedral meshes. J Parallel Distrib Comput 2017;04:008. http://dx.doi.org/10.1016/j.jpdc. published online.
[5] Laug P, Guibault F, Borouchaki H. Parallel meshing of surfaces represented by collections of connected regions. Adv Eng Software 2017;103:13–20.
[6] Yilmaz Y, Ozturan C. Using sequential NETGEN as a component for a parallel mesh generator. Adv Eng Software 2015;84:3–12.
[7] Freitas MO, Wawrzynek PA, Cavalcante-Neto JB, Vidal CA, Martha LF, Ingraff ea R. A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains. Adv Eng Software 2013;59:38–52.
[8] Chrisochoides N. Parallel mesh generation. editors In: Bruaset AM, Tveito A, editors. Numerical solution of partial diff erential equations on parallel computersNew York (NY, USA: Springer; 2006. p. 237–66.
[9] Chen J, Zhao D, Huang Z, Zheng Y, Wang D. Improvements in the reliability and element quality of parallel tetrahedral mesh generation. Int J Numer Methods Eng 2012;92(8):671–93.
[10] Zhao D, Chen J, Zheng Y, Huang Z, Zheng J. Fine-grained parallel algorithm for unstructured surface mesh generation. Comput Struct 2015;154:177–91.
[11] de Cougny HL, Shephard MS. Parallel unstructured grid generation. editors In: Thompson JF, Soni BK, Weatherill NP, editors. Handbook of grid generationBoca Raton (FL, USA: CRC Press, Inc; 1999 Chapter 24.
[12] Chrisochoides N, Nave D. Parallel Delaunay mesh generation kernel. Int J Numer Methods Eng 2003;58(2):161–76.
[13] De Cougny HL ShephardMS. Parallel volume meshing using face removals and hierarchical repartitioning. Comput Meth Appl Mech Eng 1999;174(3):275–98.
[14] Löhner R. A parallel advancing front grid generation scheme. Int J Numer Methods Eng 2001;51(6):663–78.
[15] Galtier J, George PL. Prepartitioning as a way to mesh subdomains in parallel. Proceedings of the 5th International Meshing Roundtable. 1996. p. 107–22.
[16] Larwood BG, Weatherill NP, Hassan O, Morgan K. Domain decomposition approach for parallel unstructured mesh generation. Int J Numer Methods Eng 2003;58(2):177–88.
[17] Said R, Weatherill NP, Morgan K, Verhoeven NA. Distributed parallel Delaunay mesh generation. Comput Methods Appl Mech Eng 1999;177(1):109–25.
[18] Ito Y, Shih AM, Erukala AK, Soni BK, Chernikov A, Chrisochoides NP, Nakahashi K. Parallel unstructured mesh generation by an advancing front method. Math Comput Simul 2007;75(5):200–9.
[19] Chen J, Xiao Z, Zheng Y, Zheng J, Li C, Liang K. Automatic sizing functions for unstructured surface mesh generation. Int J Numer Methods Eng 2017;109(4):577–608.
[20] Zheng Y, Weatherill NP, Turner-Smith EA. An interactive geometry utility en-vironment for multi-disciplinary computational engineering. Int J Numer Methods Eng 2002;53(6):1277–99.
[21] Chen J, Zhao D, Huang Z, Zheng Y, Gao S. Three-dimensional constrained boundary recovery with an enhanced Steiner point suppression procedure. Comput Struct 2011;89(5):455–66.
[22] Chen J, Zheng J, Zheng Y, Si H, Hassan O, Morgan K. Improved boundary con-strained tetrahedral mesh generation by shell transformation. Appl Math Modell 2017;51:764–90.
[23] ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering. Jun-18-2017. URL: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.
[24] Schloegel K, Karypis G, Kumar V. Parallel multilevel algorithms for multi-constraint graph partitioning. Proceedings of the 6th international Euro-Par conference on parallel processing. 2000. p. 296–310.
[25] Peirò J. Surface grid generation editors In: Thompson JF, Soni BK, Weatherill NP, editors. Handbook of grid generationBoca Raton (FL, USA: Springer; 1999 Chapter 19.
[26] Chen J, Zheng J, Zheng Y, Xiao Z, Si H, Yao Y. Tetrahedral mesh improvement by shell transformation. Eng Comput 2017;33(3):393–414.
[27] Xie L, Zheng Y, Chen J, Zou J. Enabling technologies in the problem solving en-vironment HEDP. Commun Comput Phys 2008;4(5):1170–93.