Arch Computat Methods Eng (2014) 21:127–140
DOI 10.1007/s11831-014-9098-8

# Recent Advances in Parallel Advancing Front Grid Generation

**Rainald Löhner**

Received: 21 August 2013 / Accepted: 21 August 2013 / Published online: 19 February 2014


**Abstract** The quest for scalable, parallel advancing front grid generation techniques now spans more than two decades. A recent innovation has been the use of a so-called domain-defining grid, which has led to a dramatic increase in robustness and speed. The domain-defining grid (DDG) has the same fine surface triangulation as the final mesh desired, but a much coarser interior mesh. The DDG renders the domain to be gridded uniquely defined and allows for a well balanced work distribution among the processors during all stages of grid generation and improvement. In this way, most of the shortcomings of previous techniques are overcome. Timings show that the approach is scalable and able to produce large grids of high quality in a modest amount of clock-time. These recent advances in parallel grid generation have enabled a completely scalable simulation pipeline (grid generation, solvers, post-processing), opening the way for truly large-scale computations using unstructured, body-fitted grids.

**Keywords** Grid generation · Advancing front · Parallel computing

## 1 Introduction

The widespread availability of parallel machines with hundreds of thousands of cores and very large memory, solvers that can harness the power of these machines, and the desire to model in ever increasing detail geometrical and physical features has led to a steady increase in the number of points and elements used in field solvers. During the 1990s, grids in excess of $10^7$ elements became common for production runs in computational fluid dynamics (CFD) [4,5,28,50,69] and computational electromagnetics [17,53]. This tendency has continued during the first decade of the 21st century, roughly following Moore's law, i.e. gridsizes have increased by an order of magnitude every 5 years. Presently, grids in of the order of $10^9$ elements are commonly used for leading edge applications in the aerospace, defense, automotive, naval, energy and electromagnetics sectors.

While many solvers have been ported to distributed parallel machines, grid generators have, in general, lagged behind. One can cite several reasons for this:

(a) For many applications the CPU requirements of grid generation are orders of magnitude less than those of field solvers, i.e. it does not matter if the user has to wait several hours for a grid;
(b) (Scalar) grid generators have achieved a high degree of maturity, generality and widespread use, leading to the usual inertia of workflow ('modus operandi') and aversion to change;
(c) In recent years, low-cost machines with few cores but very large memories have enabled the generation of large grids with existing (scalar) software; and
(d) In many cases it is possible to generate a mesh that is twice ($2^d$ times) as coarse as the one desired for the simulation. This coarse mesh is then h-refined globally. Global h-refinement is easily ported to multicore and/or distributed memory machines. Moreover, many field solvers offer h-refinement as an option. With only one level of h-refinement a mesh of 125 Mels (which is easy to generate and split on any workstation) increases to 1 Bels, and to 15.6 Bels (which should suffice for many CFD runs) with two levels of h-refinement.

R. Löhner (✉)
CFD Center, SPACS, M.S. 6A2 College of Science, George Mason University, Fairfax, VA 22030-4444, USA
e-mail: rlohner@gmu.edu

For applications where remeshing is an integral part of simulations, e.g. problems with moving bodies [6,24,30,37,43,51,52] or changing topologies [7,8], the time required for mesh regeneration can easily consume a significant percentage of the total time required to solve the problem. This percentage increases drastically if the grid generation portion is not completely parallelized. Faced with this situation, a number of efforts have been reported on parallel grid generation [1,2,9–15,20,22,26,29,38,46,54,55,61,62,64,69].

The two most common ways of generating unstructured grids are the advancing front technique (AFT) [19,27,35,36,42,46,56–59] and the generalized Delaunay Triangulation (GDT) [1,3,9,10,12,21,49,64,67,68]. The AFT introduces one element at a time, while the GDT introduces a new point at a time. Thus, both of these techniques are, in principle, scalar by nature, with a large variation in the number of operations required to introduce a new element or point. While coding and data structures may influence the scalar speed of the 'core' AFT or GDT, one often finds that for large-scale applications, the evaluation of the desired element size and shape in space, given by background grids, sources or other means [47] consumes the largest fraction of the total grid generation time. Furthermore, the time required for mesh improvements (and any unstructured grid generator needs them) is in many cases higher than the core AFT or GDT modules. Typical speeds for the complete generation of a mesh (surface, mesh, improvement) on current Intel Xeon chips with 3.2 GHz and sufficient memory are of the order of 0.5–2.0 Mels/min. Therefore, it would take approximately 2,000 min (i.e. 1.5 days) to generate a mesh of $10^9$ elements. Assuming perfect parallelization, this task could be performed in the order of a minute on 2,000 processors, clearly showing the need for parallel mesh generation.

Unstructured grid generators based on the AFT may be parallelized by invoking distance arguments, i.e., the introduction of a new element only affects (and is affected by) the immediate vicinity. This allows for the introduction of elements in parallel, provided that sufficient distance lies between them.

Nearly two decades ago (when useful distributed memory parallel machines first appeared) Löhner et al. [38] introduced a parallel AFT for 2-D applications. This was extended shortly afterwards to 3-D by Shostko and Löhner [62]. The spatial distribution of work was based on the subdivision of a relatively fine background grid. While used for some demonstration runs, this scheme was not general enough for a production environment. The background grid had to be adapted in order to be sufficiently fine for a balanced workload. As only background grid elements covering the domain to be gridded were allowed, complex in/out tests had to be carried out to remove refined elements lying outside the domain to be gridded. Furthermore, element size specified at CAD entities could not be 'propagated' into the domain, as is the case in the scalar AFT, disabling an option favoured by many users and rendering many grid generation data sets unusable. The otherwise positive experience gained with this parallel AFT, and the rise of shared-memory machines, prompted the search for a more general parallel AFT. The key requirement was a parallel AFT that modified the mature, scalar AFT as little as possible, while achieving significant speedups on common parallel machines. This led to a shared-memory parallel AFT (based on OpenMP) that applied the parallelism at the **level of the current front**, and not globally [46]. This scheme has been used for more than a decade, and has yielded a means of speeding up grid generation by an order of magnitude. Given that the parallelism is invoked at the level of the front, the achievable scalability is clearly limited.

The advent of machines with hundreds of thousands of processors has led to a re-evaluation of parallel grid generation options. It is clear that for machines with such a high number of processors, every effort has to be made to extract the maximum parallelism possible at every stage of the grid generation. This means that the **parallelism should not be front-based, but volume-based**. The easiest form of achieving volume-based parallelism is by using a grid to define the regions to be meshed by each processor. Optimally, this domain-defining grid (DDG) should have the same surface triangulation as the desired fine mesh, but could be significantly coarser in the interior so that it can be stored in each processor. In this way, the definition of the domain to be gridded is unique, something that is notoriously difficult to achieve by other means (such as background grids, bins or octrees). This DDG is then split so that in each subdomain a similar number of elements is generated.

## 2 Desired Features for Parallel Meshers

Before describing parallel grid generators, we list the main characteristics such tools should offer:

- Use of the (fine) surface mesh specified by the user: this means that no global h-refinement can/ needs to be used; the key assumption is that this surface mesh can not be coarsened and then subsequently h-refined;
- Maximum re-use of existing scalar grid generation software: it takes a decade to build a robust, production- quality 3-D grid generator; therefore, being able to reuse existing software would be extremely desirable;
- AFT or GDT: the two main ways of generating general unstructured grids are the AFT and the GDT; the parallel grid generator should be able to use any of these techniques;
- Maximum re-use of existing grid generation features/ options, such as:

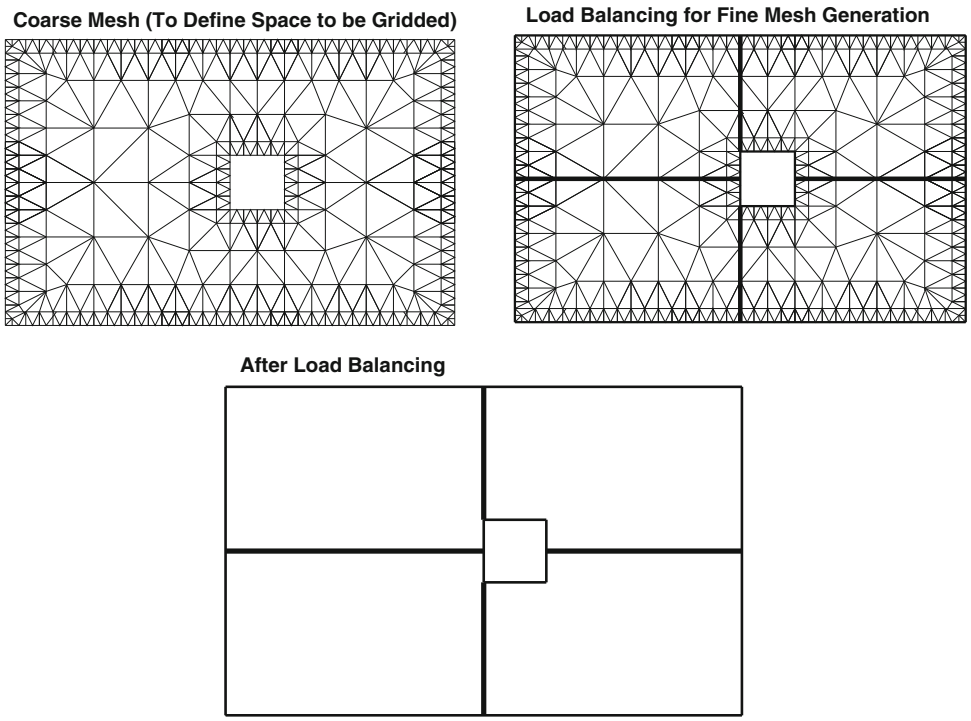


**Fig. 1** Splitting of domain defining grid

  - Mesh size specified via background grid;
  - Mesh size specified via sources;
  - Mesh size specified via CAD entities (points, lines, surfaces, domains);
  - Optimal space-filling tet options;
  - Link to boundary layer grids;
- Use of multicore parallel machines: given that massively parallel machines will be composed of multicore chips, it would be highly desirable to exploit effectively this type of architecture.

## 3 Two-Level Parallel Mesh Generation

The key idea of the most recent parallel mesh generators is the use of two levels of grid generation:

- One to define the domain to be gridded and subdivide space into regions that will generate approximately the same number of elements, and
- One that performs the parallel grid generation.

These two tasks, could, in principle, be carried out with different grid generation techniques/codes, making the approach very general. The procedure is shown conceptually in Figs. 1, 2.

## 4 Basic Advancing Front Technique

Before going on, we recall for the sake of clarity and completeness the main algorithmic steps of the AFT:

*Assume given*:

- AG1: A definition of the spatial variation of element size, stretchings, and stretching directions for the elements to be created. In most cases, this is accomplished via a combination of background grids, sources and CAD-based information [47].
- AG2: A watertight, topologically consistent triangulation that is commensurate with the desired element size and shape. This is the so-called initial front.
- AG3: The generation parameters (element size, element stretchings and stretching directions) for the faces of the initial front.

*Then*:
`While` there are active faces left in the front:

- AF1: Select the next face `ifout` to be deleted from the front; in order to avoid large elements crossing over regions of small elements, the face forming the smallest new element is selected;
- AF2: For the face to be deleted:
  - AF2.1: Select a 'best point' position for the introduction of a new point `ipnew`;
  - AF2.2: Determine whether a point exists in the already generated grid that should be used in lieu of `ipnew`; if there is such a point, set this point to `ipnew`;

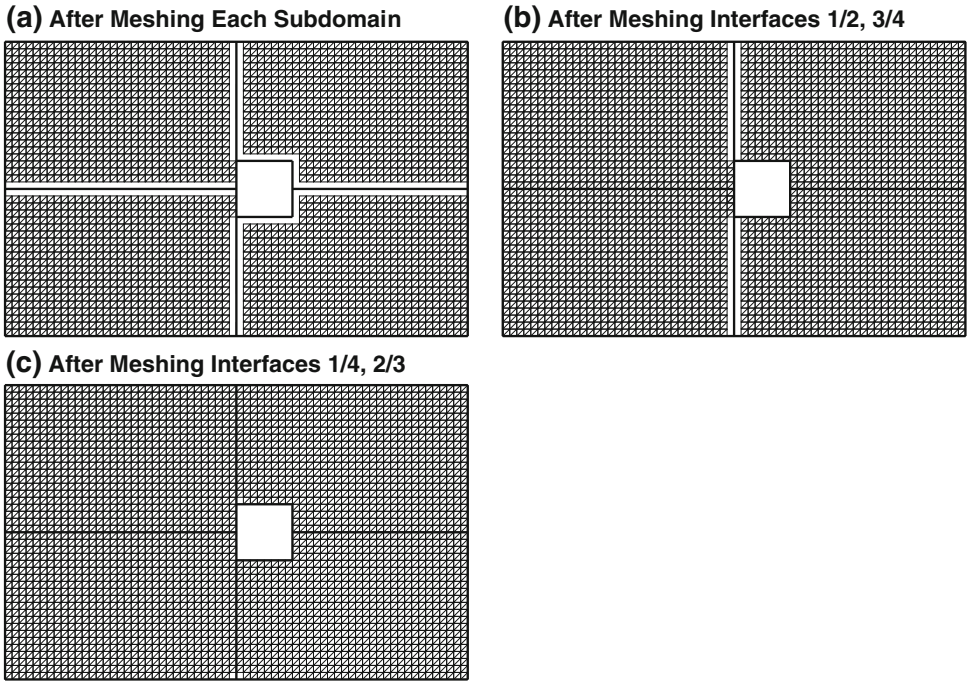


**Fig. 2** Parallel grid generation technique (Option 1)

  - AF2.3: Determine whether the element formed with the selected point `ipnew` crosses any given faces; if it does, select a new point as `ipnew` and try again; if none can be found: skip `ifout`;
- AF3: Add the new element, (point, faces) to their respective lists;
- AF4: Find the generation parameters for the new faces;
- AF5: Delete the known faces(s) from the list of faces;

`End While`

Individual aspects of the technique (such as optimal data structures for speed, robust checking of face intersections, filtering techniques to avoid unnecessary work, etc.) may be found in [40,47].

## 5 Generation of the Domain Defining Grid (Step 1)

Given that the number of elements and points decreases with the 3rd power of the element size, a mesh with elements whose side-lengths are $n$ times as large as the desired one will only contain $n^{-3}$ elements as the (fine) mesh desired. The idea is then to generate, starting from the fine surface mesh, a mesh whose elements are considerably larger than the grid desired. A factor of $n = 10$ will lead to a mesh that is generated in roughly 1/1000-th of the time required for the fine mesh. For $n = 20$, the factor is 1/8000. The mesh obtained, though, conforms to the general size distribution required by the user, i.e. is completely general. Moreover, it allows to determine exactly and easily which regions of space need to be gridded (one of the problematic aspects of earlier parallel grid generators [38,46,62]). In the following, we will denote this mesh as the DDG.

In order to generate the DDG, the changes required to the basic AFT are restricted to the desired element size, which has to increase rapidly as elements are generated in the volume:

- The generation parameters for the initial front (Step AG3 above) are multiplied by the increase factor $c_i$ allowed for each face removed from the front. Typical values are: $c_i = 1.5 - 1.7$.
- When a new point is added to the front, the grid generation parameters of the points belonging to the face being removed `ifout` are multiplied by $c_i$ and used instead of the usual ones (which are obtained from the background grid and sources, see step AF4 above).

Note that as the AFT always removes the face generating the smallest element from the front, no incompatibilities in element size appear when these changes are invoked. Thus, the generation of the DDG is of the same robustness as the basic underlying scalar AFT.

In practice one observes that the total number of extra points required to fill up the complete volume is of the order of the points on the boundary while element quality does not suffer.

## 6 Load Balancing the DDG (Step 2)

Given the DDG, the next task is to subdivide this mesh so as to obtain regions in which roughly the same numbers

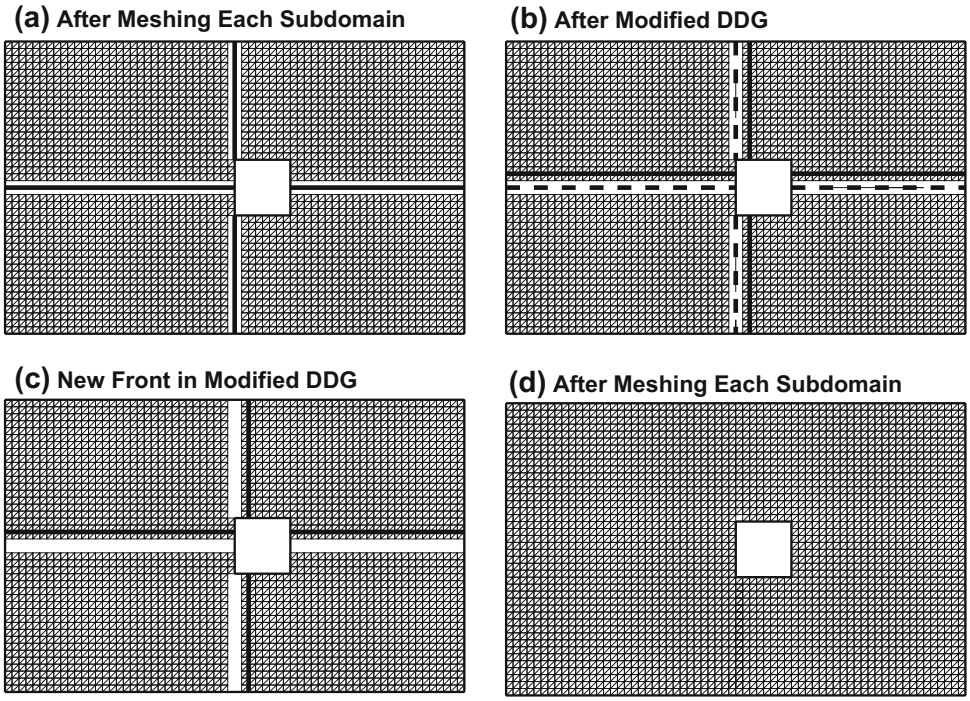


**Fig. 3** Parallel grid generation technique (Option 2)

of elements will be generated. A number of load balancing techniques and codes have been developed over the last two decades [23,31,32,39,65,66]. In principle, any of these can be used in order to obtain the subdivision required. For the results shown here, we used FESPLIT [39], which offers the possibility of subdividing grids based on the advancing front/greedy, recursive coordinate/moment bisection, or via spacefilling curves. Once an initial subdivision is obtained, FESPLIT improves the load balance (e.g. surface to volume ratios, continuity of subdivisions, etc.) using a diffusion technique [39].

## 7 Generation of the Final Mesh (Step 3)

Once the subdivision of space is obtained, the mesh is generated in parallel. The technique used here is the 'inside-out' procedure first described in [38,62]. Two variants are possible. The traditional one consists in 4 passes, which are shown in Fig. 2.

- *Pass 1*: Mesh, in parallel, the zones inside the subdivision domains (see Fig. 2a);
- *Pass 2*: Mesh, in parallel, the zones bordering the regions which have been left empty after pass 1, by pairing two domains at a time; by using a colouring technique, most of these inter-domain regions can be meshed completely in parallel (see Fig. 2b);
- *Pass 3*: Mesh, in parallel, the zones bordering more than two regions (groups of domains), which have been left empty after pass 2, by combining three or more domains at a time; as before, most of these inter-domain regions can be meshed completely in parallel by using a colouring technique;
- *Pass 4*: If required, mesh the remaining regions on processor 1.

This variant suffers from the so-called 'logarithmic trap'. While the first parallel grid generation pass (i.e. generating elements inside each domain) scales perfectly, the scaling can degrade quickly for the subsequent passes (i.e. those that mesh the inter-domain boundary regions). This is because the inherent parallelism is immediately diminished as either pairs (pass 2) or clumps (pass 3) are treated.

A second variant, based on the domain defining grid, offers much higher inherent parallelism. Denoting by DDG0 the initial subdivision of the DDG, it proceeds as follows:

- *Pass 1*: Mesh, in parallel, the zones inside the subdivision domains (see Fig. 3a);
- *Pass 2*: Modify the partition of DDG0 slightly by adding 1–2 extra layers of elements to each domain `idomn` from the neighbouring domains `jdomn` for which `idomn < jdomn`; redistribute the active front; then mesh in parallel all the (new) interior zones (see Fig. 3b, c);
- *Pass 3*: Modify again the partition of DDG0 slightly by adding 1–2 extra layers of elements to each domain `idomn` from the neighbouring domains `jdomn` for which `idomn > jdomn`; redistribute the active front; then mesh in parallel all the (new) interior zones;

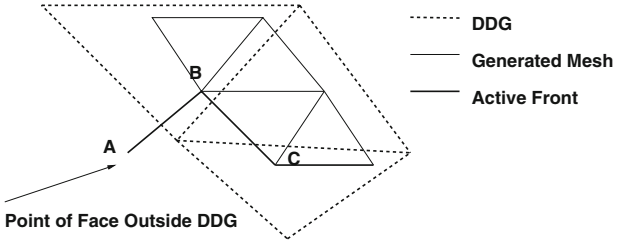


**Fig. 4** Reject if face outside DDG

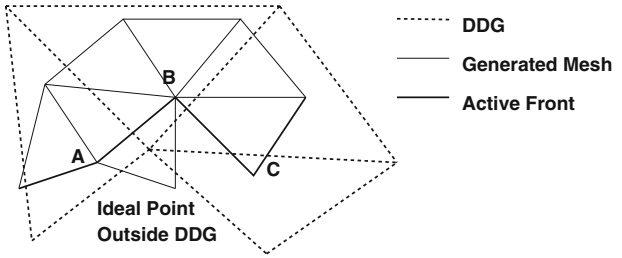


**Fig. 5** Reject if ideal point outside DDG

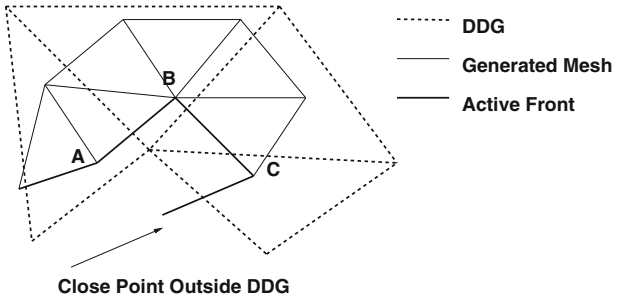


**Fig. 6** Reject if close point outside DDG

– *Pass 4*: If required, mesh the remaining regions on processor 1.

It is important to emphasize that regardless of the option used, all data is kept local. The list of elements and points being generated, the active front, and all other arrays are stored in the processor where they are being generated.

The following changes to the basic AFT are required in order to obtain a reliable parallel meshing algorithm:

– If any of the points of the face to be removed lies outside the local DDG, the face is marked as prohibited and skipped (Fig. 4);
– If the 'best point' position for the introduction of a new point lies outside the local DDG, the face is marked as prohibited and skipped (Fig. 5);
– If any 'close point' lies outside the local DDG, it is removed from the list of candidates to form a new element (Fig. 6);
– If any of the edges of the face to be removed `ifout` lies outside the local DDG, the face is marked as prohibited and skipped; this test is carried out by using a neighbour to neighbour traversal test between the points of the edge (see Fig. 7);
– If `ipnew` is on both the inter-processor boundary of the DDG and the actual surface of the domain: the face is marked as prohibited and skipped;
– When assigning the faces and points to the local DDG, a conservative approach is taken; i.e. should an active front point coincide with the points of the DDG, all the surrounding DDG elements are tested to see if the point should be assigned to the present domain.

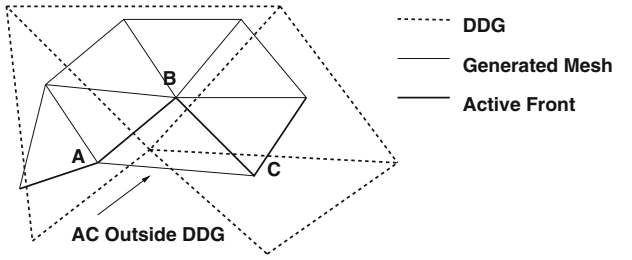


**Fig. 7** Reject if face/edge outside DDG

## 8 Mesh Redistribution (Step 4)

After the parallel advancing front has completed the mesh, the pieces generated in each of the individual passes will be scattered among the different processors. In order to arrive at a consistent mesh, the elements and points need to be redistributed and doubly defined points need to be removed.

Given that for each point the host element in the DDG is known, and that for each element of the DDG the processor it has been assigned to is also known, it is an easy matter so send the elements and the associated points to the processors they need to be. Each element is sent (if required; the majority already reside in the memory of the target processor) to the lowest processor assigned via points from the DDG. Doubly defined points are removed using an octree, so that this operation has $O(N \log(N))$ complexity locally (but runs completely parallel in a distributed setting).

The next step is to find the correlation between the points of neighbouring processors. In order to keep the procedure as general as possible, the following algorithmic steps are taken:

– The bounding box of each domain is computed;
– The bounding boxes of other domains that overlap the bounding box of each domain are determined; this determines a list of possible neighbouring domains;
– Each pair of possible neighbouring domains is tested in depth using octrees; in this way, the lists of neighbouring domains and points are obtained (so-called send/receive lists).

One could also have used the DDG to obtain this information. We remark that if one considers the overall parallel grid generation procedure, the time required for this step is negligible.

## 9 Mesh Improvement (Step 5)

After the generation of the mesh using the parallel advancing front technique (or any other technique for that matter) has been completed, the mesh quality is improved by a combination of several algorithms, such as:

- Diagonal swapping,
- Removal of bad elements,
- Laplacian/elasticity smoothing, and
- Selective mesh movement.

One should emphasize that mesh improvement may require CPU times that are comparable to those required by the basic grid generation technique, making it imperative to fully parallelize this necessary step as well. All of the procedures listed above have been implemented and run in parallel (shared locally via OMP and distributed globally via MPI).

### 9.1 Diagonal Swapping

Diagonal swapping attempts to improve the quality of the mesh by reconnecting locally the points in a different way [18]. The quality of every possible new combination is tested against the current connectivity. The number of cases to be tested can grow factorially with the number of elements surrounding an edge [47]. Given that these tests are computationally intensive, considerable care is required when coding a fast diagonal swapper. Techniques that are commonly used include:

- Treatment of bad, untested elements only (i.e. those whole quality measure falls above/below a certain threshold);
- Processing of elements in an ordered way, starting with the worst (highest chance of reconnection);
- Rejection of bad combinations at the earliest possible indication of worsening quality;
- Marking of tested and unswapped elements in each pass.

At the boundaries between processors, diagonal swapping would require a considerable amount of testing and information transfer. For this reason, it was decided not to allow any diagonal swapping for the external faces of each subdomain.

### 9.2 Removal of Bad Elements

A simple way to improve a mesh containing bad elements is to get rid of them. The bad elements are identified and compiled into a list. An element is removed by collapsing the points of one of the edges. This operation also removes all the elements that share this edge, implying that one has to check also all elements that contain the end-points of the edge being removed. This procedure of removing bad elements is simple to implement and relatively fast. On the other hand, it can only improve mesh quality to a certain degree. It is therefore used mainly in a pre-smoothing or pre-optimization stage, where its main function is to eradicate from the mesh elements of very bad quality.

At the boundaries between processors, edge removal would require a considerable amount of testing and information transfer. For this reason, it was decided not to allow any edge removal for the external faces of each subdomain.

### 9.3 Laplacian/Elasticity Smoothing

A number of smoothing techniques are lumped under this name. For the usual Laplacian smoothing, the edges of the triangulation are assumed to represent springs. These springs are relaxed in time using an explicit time stepping scheme, until an equilibrium of spring-forces has been established. Because 'globally' the variations of element size and shape are smooth, most of the non-equilibrium forces are local in nature. This implies that a significant improvement in mesh quality can be achieved rather quickly (5–6 timesteps or passes over the mesh). For Elasticity smoothing, the partial differential equations describing an elastic medium are solved. As before, a few relaxation passes over the mesh are usually enough to achieve a considerable mesh improvement. In both cases, no movement of points is allowed at the surface of the computational domain. The application of any smoothing technique can result in inverted or negative elements. The presence of even one element with a negative Jacobian will render most field solvers inoperable. Therefore, these negative elements are eliminated. For the AFT, it has been found advisable to remove not only the negative elements, but also all elements that share points with them. This element removal gives rise to voids or holes in the mesh, which are regridded using the advancing front technique.

At the boundaries between processors, point movement and possible removal/remeshing would require a considerable amount of testing and information transfer. For this reason, it was decided not to allow any change for the points of the external faces of each subdomain.

### 9.4 Selective Mesh Movement

Selective mesh movement tries to improve the mesh quality by performing a local movement of the points. If the movement results in an improvement of mesh quality, the movement is kept. Otherwise, the old point position is retained. The most natural way to move points is along the directions

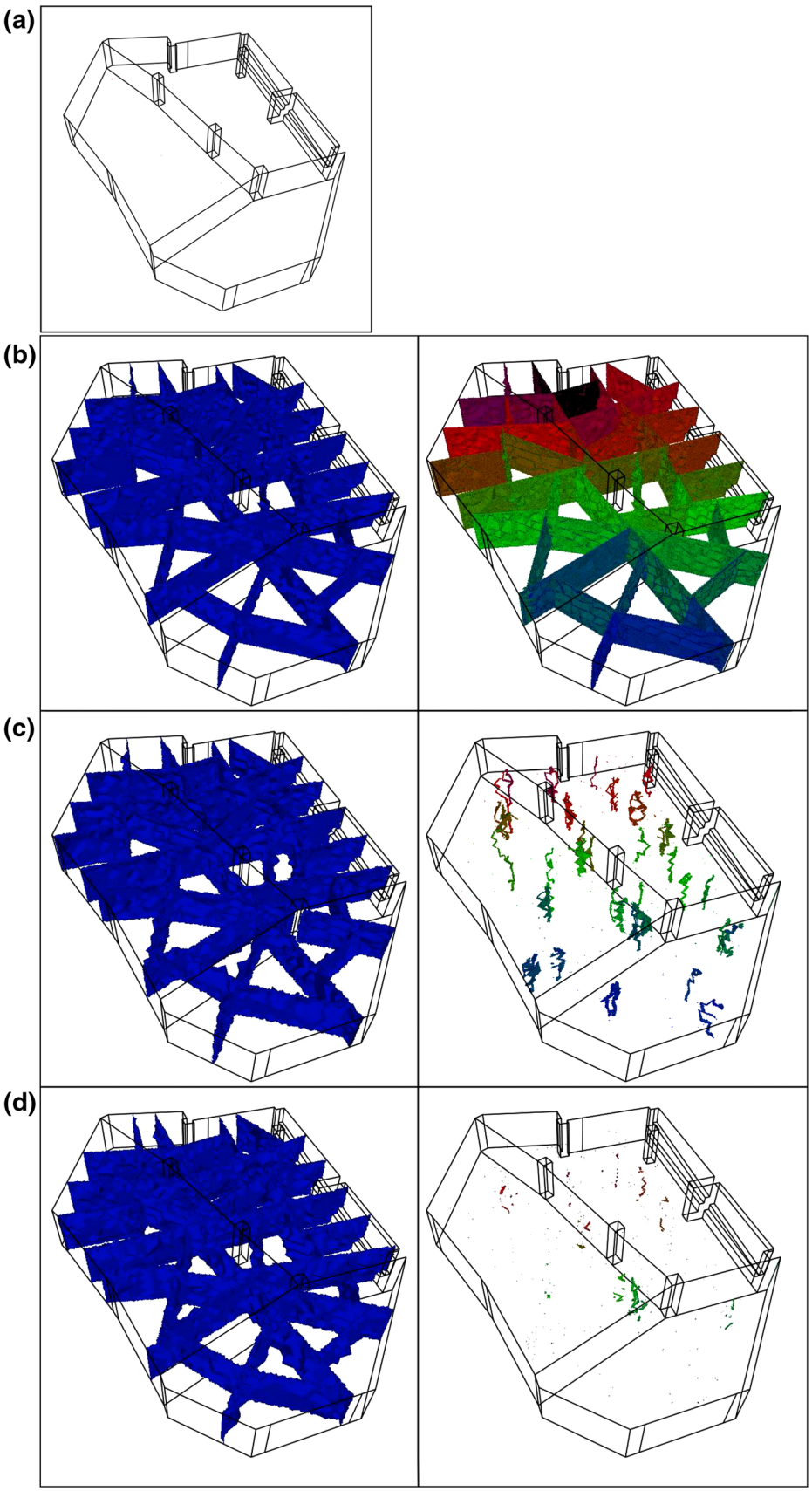


**Fig. 8** **a** Garage: Outline of geometry. **b**–**d** Garage: Internal surface of DDG partition and remaining front after each pass

**Table 1** Garage

| Machine | nproc | nprol | ncore | nelem | CPU (sec) | AbsSpeed (els/sec) | RelSpeed (els/sec/core) |
|---|---|---|---|---|---|---|---|
| Xeon(1) | 1 | 8 | 8 | 120 M | 2,293 | 52,333 | 6,542 |
| SGI ITL | 8 | 1 | 8 | 121 M | 1,605 | 75,389 | 9,423 |
| SGI ITL | 8 | 8 | 64 | 121 M | 516 | 234,496 | 3,664 |
| Cry AMD | 8 | 1 | 8 | 121 M | 2,512 | 48,169 | 6,021 |
| Cry AMD | 16 | 1 | 16 | 121 M | 1,954 | 61,924 | 3,870 |
| Cry AMD | 32 | 1 | 32 | 121 M | 1,118 | 100,082 | 3,128 |
| SGI ITL | 16 | 1 | 16 | 121 M | 1,048 | 115,458 | 7,216 |
| SGI ITL | 16 | 2 | 32 | 121 M | 667 | 181,409 | 5,669 |
| SGI ITL | 16 | 4 | 64 | 121 M | 407 | 297,297 | 4,645 |
| SGI ITL | 16 | 8 | 128 | 121 M | 329 | 367,781 | 2,873 |
| SGI ITL | 32 | 1 | 32 | 121 M | 646 | 187,306 | 5,853 |
| SGI ITL | 32 | 2 | 64 | 121 M | 427 | 283,372 | 4,427 |
| SGI ITL | 32 | 4 | 128 | 121 M | 346 | 349,710 | 2,732 |
| SGI ITL | 32 | 8 | 256 | 121 M | 316 | 383,030 | 1,496 |
| Cry AMD | 64 | 1 | 64 | 972 M | 6,048 | 160,714 | 2,511 |
| SGI ITL | 64 | 8 | 512 | 1010 M | 2,504 | 403,354 | 788 |

of the edges touching them. After each of these movements, the quality of each element containing the point being moved is checked. Only movements that produce an improvement in element quality are kept. This procedure, while general, is extremely expensive for tetrahedral meshes. This is because, for each pass over the mesh, we have approximately 7 edges for each point, i.e. 14 movement directions; approximately 22 elements (4 nodes per element, 5.5 elements per point) surrounding each point to be evaluated for each of the movement directions; i.e. approximately 308*`npoin` elements to be tested. To make matters worse, the evaluation of element quality typically involves arc-cosines (for angle evaluations), which consume a large amount of CPU time. The main strength of selective mesh movement algorithms is that they remove efficiently very bad elements. They are therefore used only for points surrounded by bad elements, and as a post-smoothing procedure.

At the boundaries between processors, selective point movement removal would require a considerable amount of testing and information transfer. For this reason, it was decided not to allow any edge removal for the external faces of each subdomain.

### 9.5 A Second Mesh Improvement Pass

As seen above, the mesh improvement techniques as implemented do not allow a change of the inter-processor boundary. This implies that the inter-processor regions will not be able to achieve the best possible mesh. In order to improve the mesh in these regions as well, the DDG is again redistributed among processors. The first distribution is taken as a starting point. Then, 1–2 extra layers of elements are added to the each domain `idomn` from the neighbouring domains `jdomn` for which `idomn < jdomn`. The elements of the generated (and smoothed) mesh are then redistributed as before based on the new DDG partition, and a second mesh improvement pass is performed using all the techniques discussed above.

## 10 Examples

The parallel advancing front grid generator described above has been in operation for approximately a year. It is still undergoing considerable changes and improvements, so the numbers quoted may improve over time. In the sequel `nproc` denotes the number of mpi processes (i.e. subdomains), while `nprol` denotes the number of shared-memory (OpenMP) cores used per mpi process/subdomain. The total number of cores employed is then given by `ncore=nproc*nprol`. The tables also quote the absolute (els/sec) and relative (els/sec/core) grid generation speeds achieved. Note that for perfect scaling, the relative grid generation speed should stay constant.

### 10.1 Garage

This example was taken from a blast simulation carried out for an office complex. The outline of the domain, as well as the trace of the domain defining grid partition on the surface is shown in Fig. 8a. Figure 8b–d show the trace of the domain

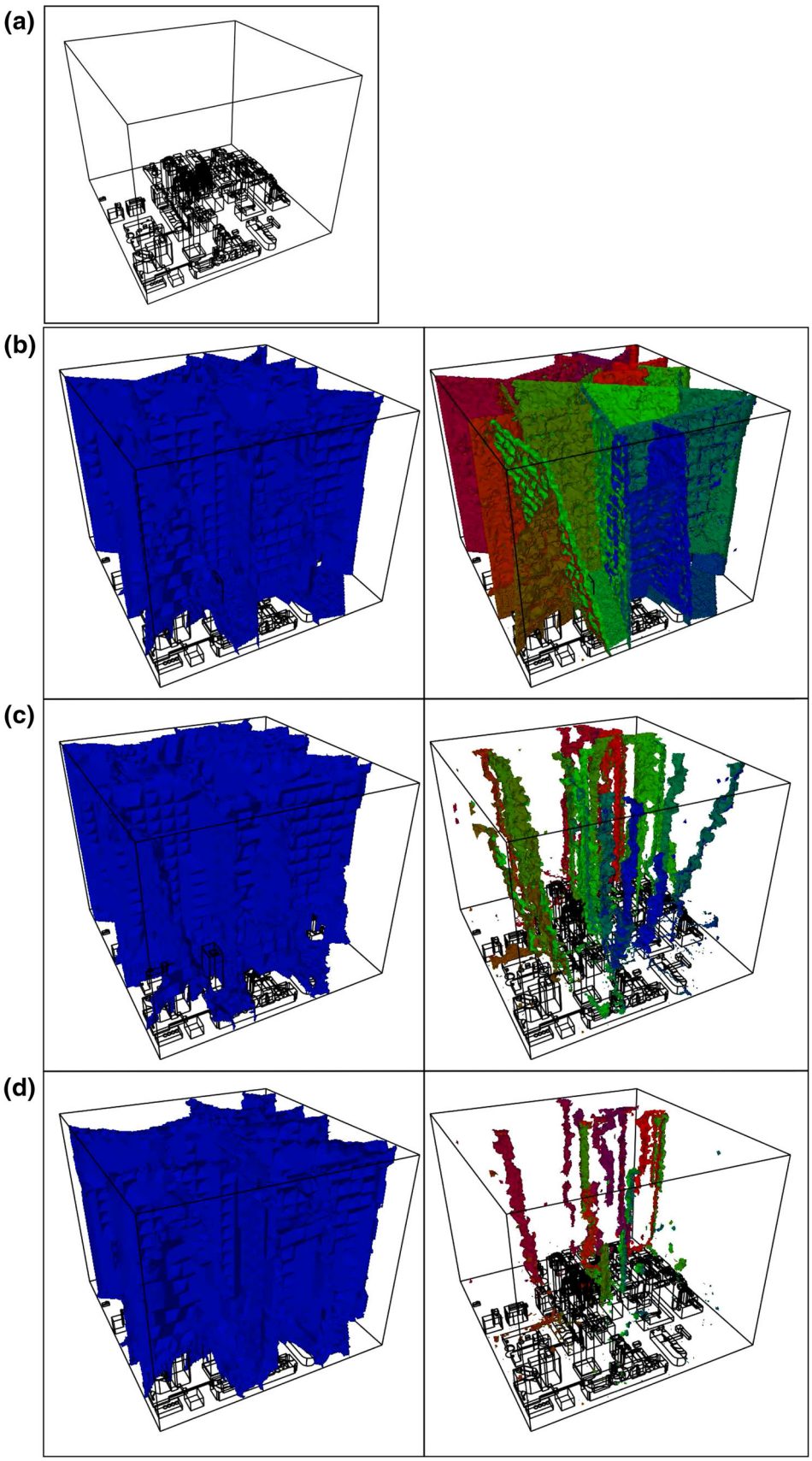


**Fig. 9** **a** Generic city center: Outline of geometry. **b–d** Generic city center: Internal surface of DDG partition and remaining front after each pass

**Table 2** Generic city center

| Machine | nproc | nprol | ncore | nelem | CPU (sec) | AbsSpeed (els/sec) | RelSpeed (els/sec/core) |
|---|---|---|---|---|---|---|---|
| Cry AMD | 32 | 1 | 32 | 135 M | 1,824 | 74,013 | 2,312 |
| SGI ITL | 16 | 8 | 128 | 135 M | 556 | 242,805 | 1,897 |
| SGI ITL | 32 | 1 | 32 | 135 M | 977 | 138,178 | 4,318 |
| SGI ITL | 32 | 2 | 64 | 135 M | 754 | 179,045 | 2,797 |
| SGI ITL | 32 | 4 | 128 | 135 M | 571 | 236,427 | 1,847 |
| SGI ITL | 32 | 8 | 256 | 135 M | 488 | 276,639 | 1,080 |

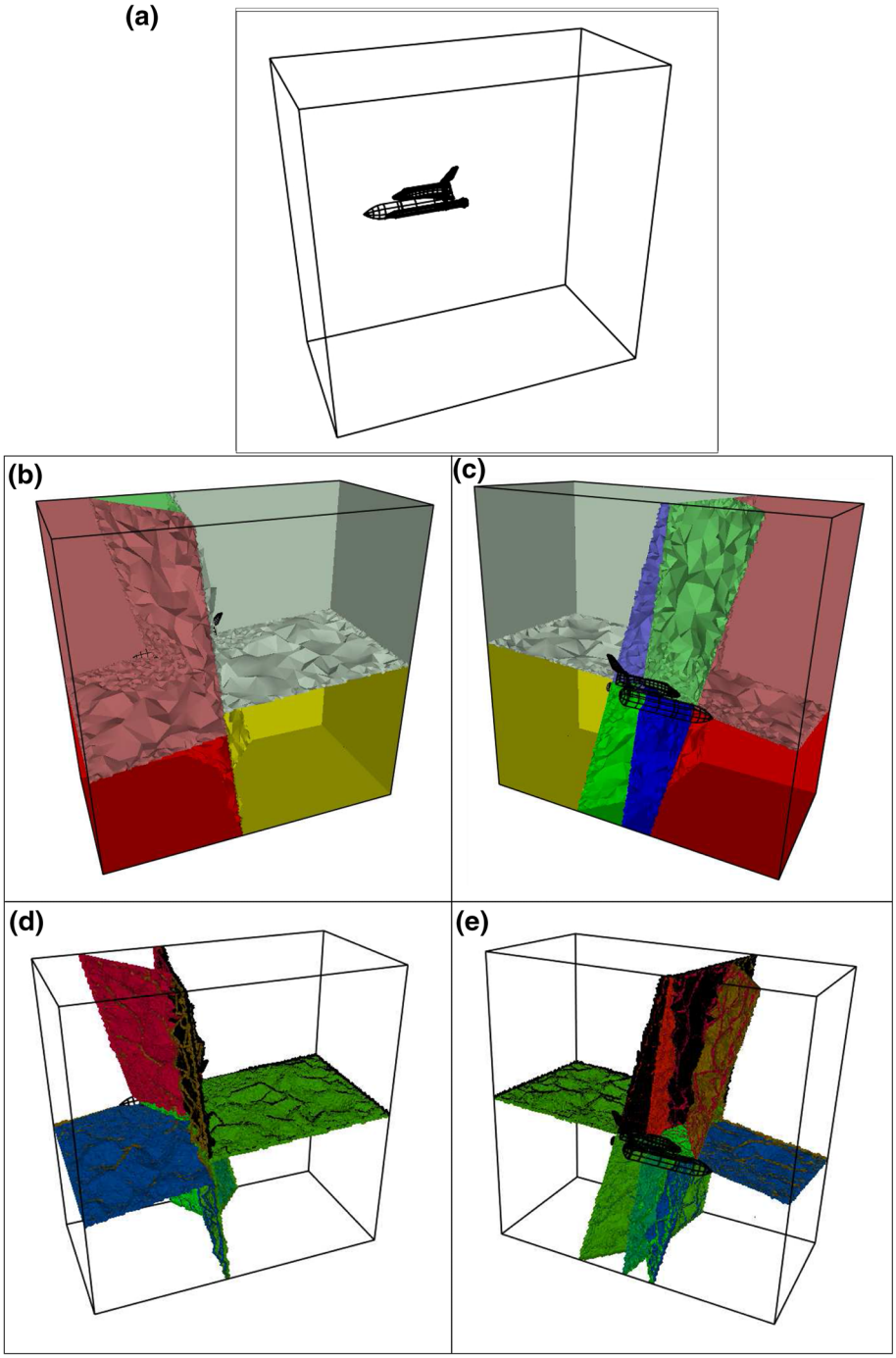


**Fig. 10** **a** Shuttle: Outline of domain. **b, c** Shuttle: Domain defining grid partition. **d, e** Shuttle: Front after 1st parallel grid generation pass

**Table 3** Shuttle

| Machine | nproc | nprol | ncore | nelem | CPU (sec) | AbsSpeed (els/sec) | RelSpeed (els/sec/core) |
|---|---|---|---|---|---|---|---|
| Xeon | 8 | 1 | 8 | 27 M | 872 | 30,963 | 3,870 |
| Xeon | 8 | 1 | 8 | 108 M | 3,128 | 34,526 | 4,315 |
| Cry AMD | 16 | 1 | 16 | 108 M | 2,204 | 49,002 | 3,062 |
| Cry AMD | 32 | 1 | 32 | 108 M | 1,458 | 74,074 | 2,315 |
| Cry AMD | 64 | 1 | 64 | 108 M | 1,344 | 80,357 | 1,255 |

defining grid partition on the surface as well as the fronts after the parallel grid generation passes using 64 domains (mpi processors) for a finer mesh. Table 1 gives a compilation of timings for different mesh sizes, domains and processors on different machines. One may note that:

(a) Generating the 121 M mesh on one 8-core shared memory node (i.e. `nproc=1, nprol=8`) is slower than the distributed memory equivalent (i.e. `nproc=8, nprol=1`);
(b) The number of elements per core should exceed a minimum value (typically of the order of 2–4 Mels) in order to reach a generation speed per core that is acceptable;
(c) The local OMP scaling improves as the number of elements in each domain is increased;
(d) It only takes on the order of five minutes to generate a mesh of 121 Mels on 256 cores (`nproc=32, nprol=8`).
(e) It only takes on the order of forty minutes to generate a mesh of 1 Bels on 512 cores (`nproc=64, nprol=8`).

### 10.2 Generic City Center

This example was taken from a recent blast and dispersion simulation. The outline of the domain, as well as the active front after each of the parallel grid generation passes for 32 processors (mpi domains) are shown in Fig. 9a–d. Table 2 gives a compilation of timings for different mesh sizes, domains and processors on different machines. One may observe the same general trends as seen for the previous case.

### 10.3 Shuttle Ascent Configuration

This example has also been used repeatedly for benchmarking purposes. The outline of the domain may be seen in Fig. 10a. The trace of the domain defining grid partition on the surface is shown in Fig. 10b, c. Figure 5d,e show the active front after the first generation pass using 8 domains (mpi processors). This mesh had approximately 120 Mels. Table 3 gives a brief compilation of timings.

## 11 Conclusions and Outlook

Recent advances in parallel advancing grid generation techniques for complex geometries and meshes with large size variations have been described. A key innovation that took place over the last three years has been the use of a DDG that has the same fine surface triangulation as the final mesh desired, but a much coarser interior mesh. In this way, the domain to be gridded is uniquely defined and a balanced distribution of work among the processors is achieved during all stages of grid generation and improvement, overcoming most of the shortcomings of previous approaches.

Timings show that the approach is scalable and able to produce large grids of high quality in a modest amount of clocktime.

These recent advances in parallel grid generation have enabled a completely scalable simulation pipeline (grid generation, solvers, post-processing), opening the way for truly large-scale computations using unstructured, body-fitted grids.

**Acknowledgments** This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725, and also resources of the DoD High Performance Computing Modernization Program. This support is gratefully acknowledged.

## References

1. Alleaume A, Francez L, Loriot M, Maman, N (2007) Large outof-Core tetrahedral meshing. In: Proceedings of the 16th international meshing roundtable, Sandia National Laboratory, Oct. 15–17
2. Andrae H, Ivanov E, Gluchshenko O, Kudryavtsev A (2008) Automatic parallel generation of tetrahedral grids by using a domain decomposition approach. J Comput Math Math Phys 48(8):1448–1457
3. Baker TJ (1989) Developments and trends in three-dimensional mesh generation. Appl Numer Math 5:275–304
4. Baum JD, Luo H, Löhner R (1993) Numerical simulation of a blast inside a Boeing 747; AIAA-93-3091
5. Baum JD, Luo H, Löhner R (1995) Numerical simulation of blast in the World Trade Center; AIAA-95-0085
6. Baum JD, Luo H, Löhner R, Yang C, Pelessone D, Charman C (1996) A coupled fluid/structure modeling of shock interaction with a truck; AIAA-96-0795
7. Baum JD, Luo H, Löhner R (1998) The numerical simulation of strongly unsteady flows with hundreds of moving bodies; AIAA-98-0788
8. Baum JD, Luo H, Mestreau E, Löhner R, Pelessone D, Charman C (1999) A coupled CFD/CSD methodology for modeling weapon detonation and fragmentation; AIAA-99-0794

9. Blelloch GE, Hardwick JC, Miller GL, Talmor D (1999) Design and implementation of a practical parallel Delaunay algorithm. Algorithmica 24:243–269
10. Chew LP, Chrisochoides N, Sukup F (1997) Parallel constrained Delaunay meshing; In: Proceedings 1997 workshop on trends in unstructured mesh generation, June
11. Chrisochoides N, Nave D (1999) Simultaneous mesh generation and partitioning for Delaunay meshes; In: Proceedings 8th international meshing roundtable, South Lake Tahoe, October pp. 55–66
12. Chrisochoides N, Nave D (2003) Parallel Delaunay mesh generation kernel. Int J Numer Methods Eng 58:161–176
13. Chrisochoides N (2005) Parallel mesh generation. In: Bruaset AM, Tveito A (eds) Numerical solution of partial differential equations on parallel computers. Springer, Norfolk
14. de Cougny HL, Shephard MS, Ozturan C (1994) Parallel three-dimensional mesh generation. Comput Syst Eng 5:311–323
15. de Cougny HL, Shephard MS, Ozturan C (1995) Parallel three-dimensional mesh generation on distributed memory MIMD computers. Tech. Rep. SCOREC Rep. # 7, Rensselaer Polytechnic Institute
16. de Cougny H, Shephard M (1999) Parallel volume meshing using face removals and hierarchical repartitioning. Comput Methods Appl Mech Eng 174(3–4):275–298
17. Darve E, Löhner R (1997) Advanced structured–unstructured solver for electromagnetic scattering from multimaterial objects. AIAA-97-0863
18. Freitag LA, Gooch C-Ollivier (1997) Tetrahedral mesh improvement using swapping and smoothing. Int J Numer Methods Eng 40:3979–4002
19. Frykestig J (1994) Advancing front mesh generation techniques with application to the finite element method; Pub. 94:10, Chalmers University of Technology; Göteborg, Sweden
20. Galtier J, George PL (1997) Prepartitioning as a way to mesh subdomains in parallel; In: Special Symposium on trends in unstructured mesh generation pp 107–122, ASME/ASCE/SES
21. George PL, Hecht F, Saltel E (1991) Automatic mesh generator with specified boundary. Comp Methods Appl Mech Eng 92:269–288
22. George PL (1999) Tet meshing: construction, optimization and adaptation. In: Proceedings of the 8th international meshing roundtable, South Lake Tahoe, October
23. von Hanxleden R, Scott LR (1991) Load balancing on message passing architectures. J Parallel Distrib Comput 13:312–324
24. Hassan O, Bayne LB, Morgan K and Weatherill N P (1998) An adaptive unstructured mesh method for transient flows involving moving boundaries; pp. 662–674 in Computational fluid dynamics '98 ( Papailiou KD, Tsahalis D, Périaux J and Knörzer D eds. ) Wiley
25. Ito Y, Shih AM, Erukala AK, Soni BK, Chernikov A, Chrisochoides N, Nakahashi K (2007) Parallel unstructured mesh generation by an advancing front method. J Math Comput Simul 75(5–6):200–209
26. Ivanov EG, Andrae H, Kudryavtsev AN (2006) Domain decomposition approach for automatic parallel generation of tetrahedral grids. Int Math J Comput Methods Appl Math 6(2):178–193
27. Jin H, Tanner RI (1993) Generation of unstructured tetrahedral meshes by the advancing front technique. Int J Numer Methods Eng 36:1805–1823
28. Jou W (1998) Comments on the feasibility of LES for commercial airplane wings. AIAA-98-2801
29. Kadow C, Walkington N (2003) Design of a projection-based parallel Delaunay mesh generation and refinement algorithm. In: Proceedings of the fourth symposium on trends in unstructured mesh generation
30. Kamoulakos A, Chen V, Mestreau E, Löhner R (1996) Finite element modelling of fluid/ structure interaction in explosively loaded aircraft fuselage panels using PAMSHOCK/ PAMFLOW coupling. Conference on spacecraft structures, materials and mechanical testing, Noordwijk, The Netherlands, March
31. Karypis G, Kumar V (1998) A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. J Parallel Distrib Comput 48:71–85
32. Karypis G, Kumar V (1999) Parallel multilevel k-way partitioning scheme for irregular graphs. SIAM Rev 41(2):278–300
33. Larwood BG, Weatherill NP, Hassan O, Morgan K (2003) Domain decomposition approach for parallel unstructured mesh generation. Int J Numer Methods Eng 58(2):177–188
34. Liu J, Kailasanath K, Ramamurti R, Munday D, Gutmark E, Löhner R (2009) Large-Eddy simulations of a supersonic jet and its near-field acoustic properties. AIAA J 47(8):1849–1864
35. Löhner R (1988) Some useful data structures for the generation of unstructured grids. Comm Appl Numer Methods 4: 123–135
36. Löhner R, Parikh P (1988) Three-dimensional grid generation by the advancing front method. Int J Numer Methods Fluids 8:1135–1149
37. Löhner R (1990) Three-dimensional fluid-structure interaction using a finite element solver and adaptive remeshing. Comput Syst Eng 1(2–4):257–272
38. Löhner R, Camberos J, Merriam M (1992) Parallel unstructured grid generation. Comput Methods Appl Mech Eng 95:343–357
39. Löhner R, Ramamurti R (1995) A load balancing algorithm for unstructured grids. Comput Fluid Dyn 5:39–58
40. Löhner R (1996) Extensions and improvements of the advancing front grid generation technique. Comm Numer Methods Eng 12:683–702
41. Löhner R (1996) Regridding surface triangulations. J Comput Phys 126:1–10
42. Löhner R (1996) Progress in grid generation via the advancing front technique. Eng Comput 12:186–210
43. Löhner R, Yang C, Cebral J, Baum JD, Luo H, Pelessone D, Charman C (1998) Fluid-structure-thermal interaction using a loose coupling algorithm and adaptive unstructured grids; AIAA-98-2419
44. Löhner R (1998) Renumbering strategies for unstructured-grid solvers operating on shared- memory, cache-based parallel machines. Comput Methods Appl Mech Eng 163:95–109
45. Löhner R, Yang C and Oñate E ( 1998) Viscous free surface hydrodynamics using unstructured grids; In: Proceedings 22nd symposium naval hydrodynamics, Washington DC, August
46. Löhner R (2001) A parallel advancing front grid generation scheme. Int J Numer Methods Eng 51:663–678
47. Löhner R (2008) Applied CFD techniques, 2nd edn. Wiley, Chichester
48. Löhner R, Cebral JR, Camelli FF, Appanaboyina S, Baum JD, Mestreau EL, Soto O (2008) Adaptive embedded and immersed unstructured grid techniques. Comput Methods Appl Mech Eng 197:2173–2197
49. Marcum DL, Weatherill NP (1995) Unstructured grid generation using iterative point insertion and local reconnection. AIAA J 33(9):1619–1625
50. Mavriplis DJ and Pirzadeh S (1999) Large-scale parallel unstructured mesh computations for 3-D high-lift analysis; ICASE Rep. 99–9
51. Mestreau E, Löhner R and Aita S (1993) TGV tunnel-entry simulations using a finite element code with automatic remeshing; AIAA-93-0890
52. Mestreau E and Löhner R (1996) Airbag simulation using fluid/structure coupling; AIAA-96-0798
53. Morgan K, Brookes PJ, Hassan O and Weatherill NP (1997) Parallel processing for the simulation of problems involving scattering of electro-magnetic waves; In: Proceedings Symposium advances in computational mechanics (Demkowicz L and Reddy JN eds)

54. Okusanya T, Peraire J (1996) Parallel unstructured mesh generation. In: Proceedings 5th international conference numerical grid generation in CFD and related fields, Mississippi, April
55. Okusanya T, Peraire J (1997) 3-D Parallel unstructured mesh generation. In: Proceedings of the joint ASME/ASCE/SES summer meeting
56. Peraire J, Vahdati M, Morgan K, Zienkiewicz OC (1987) Adaptive remeshing for compressible flow computations. J Comput Phys 72:449–466
57. Peraire J, Peiro J, Formaggia L, Morgan K, Zienkiewicz OC (1988) Finite element euler calculations in three dimensions. Int J Numer Methods Eng 26:2135–2159
58. Peraire J, Morgan K, Peiro J (1990) Unstructured finite element mesh generation and adaptive procedures for CFD; AGARD-CP-464, 18
59. Peraire J, Morgan K, Peiro J (1992) Adaptive Remeshing in 3-D. J Comput Phys 103:269–285
60. Pirzadeh SZ, Zagaris G (2008) Domain decomposition by the advancing-partition method for parallel unstructured grid generation. NASA/TM-2008-215350, L-19508
61. Said R, Weatherill NP, Morgan K, Verhoeven NA (1999) Distributed parallel Delaunay mesh generation. Comput Methods Appl Mech 177:109–125
62. Shostko A, Löhner R (1995) Three-dimensional parallel unstructured grid generation. Int J Numer Methods Eng 38:905–925
63. Tilch R, Tabbal A, Zhu M, Decker F, Löhner R (2008) Combination of body-fitted and embedded grids for external vehicle aerodynamics. Eng Comput 25(1):28–41
64. Tremel U, Sorensen KA, Hitzel S, Rieger H, Hassan O, Weatherill NP (2006) Parallel remeshing of unstructured volume grids for CFD applications. Int J Numer Methods Fluids 53(8):1361–1379
65. Vidwans A, Kallinderis Y and Venkatakrishnan V (1993) A parallel load balancing algorithm for 3-D adaptive unstructured grids; AIAA-93-3313-CP
66. Williams D (1990) Performance of dynamic load balancing algorithms for unstructured grid calculations. CalTech Report C3P913
67. Weatherill NP (1992) Delaunay triangulation in computational fluid dynamics. Comput Math Appl 24(5/6):129–150
68. Weatherill NP, Hassan O (1994) Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. Int J Numer Methods Eng 37:2005–2039
69. Yoshimura S, Nitta H, Yagawa G, Akiba H (1998) Parallel automatic mesh generation method of ten-million nodes problem using fuzzy knowledge processing and computational geometry. In: Proceedings of the 4th World CongComp. Mech. Buenos Aires, Argentina, July