INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2001; **51**:663–678

A parallel advancing front grid generation scheme

Rainald Löhner*,†

Institute for Computational Science and Informatics, M.S. 4C7, George Mason University, Fairfax, VA 22030-4444, U.S.A.

SUMMARY

A parallel advancing front scheme has been developed. The domain to be gridded is first subdivided spatially using a relatively coarse octree. Boxes are then identified and gridded in parallel. A scheme that resembles closely the advancing front technique on scalar machines is recovered by only considering the boxes of the active front that generate small elements. The procedure has been implemented on the SGI origin class of machines using the shared memory paradigm. Timings for a variety of cases show speedups similar to those obtained for flow codes. The procedure has been used to generate grids with tens of millions of elements. Copyright © 2001 John Wiley & Sons, Ltd.

KEY WORDS: unstructured grid generation; parallel computing; CFD

1. INTRODUCTION

The widespread availability of parallel machines with 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 lead to a steady increase in the number of points used in field solvers. Grids in excess of 10^7 elements have become common for production runs in computational fluid dynamics (CFD) [1–5] and computational electromagnetics (CEM) [6, 7]. The expectation is that in the near future grids in excess of 10^8–10^9 elements will be required. While many solvers have been ported to parallel machines, grid generators have lagged behind. For applications where remeshing is an integral part of simulations, e.g. problems with moving bodies [8–14] or changing topologies [15, 16], the time required for mesh regeneration can easily consume more than 50 per cent of the total time required to solve the problem. Faced with this situation, a number of efforts have been reported on parallel grid generation [17–25].

The two most common ways of generating unstructured grids are the advancing front technique (AFT) [26–34] and the generalized Delaunay triangulation (GDT) [35–39]. The AFT introduces one element at a time, while the GDT introduces a new point at a time. Thus,

*Correspondence to: Rainald Löhner, GMU/CSI, MS 5C3 Department of Civil Engineering, George Mason University, Fairfax, VA, 22030-4444, U.S.A.
†E-mail: rlohner@rossini.gmu.edu

Contract/grant sponsor: AFOSR, partially

Received 31 January 2000
Revised 31 July 2000

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 [34] consumes the largest fraction of the total grid generation time. 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.

Several years ago, the author and his colleagues introduced a parallel AFT based on the subdivision of the background grid [17, 19] 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 prompted the search for a more general parallel AFT. The key requirement was a parallel AFT that changes the current, evolved and mature scalar AFT as little as possible, while achieving significant speedups on common parallel machines. This implies that the parallelism should be applied at the level of the current front, and not globally.

2. PARALLEL GRIDDING SCHEME

The advancing front technique attempts to introduce an element at a time into an as yet ungridded domain by eliminating the face generating the smallest new element from the front [28, 29]. Given that the introduction of an element only affects its immediate neighbourhood, one could, in principle, introduce many elements at the same time, provided they are sufficiently far apart. A convenient way of delimiting the possible zones where elements may be introduced by each processor is via boxes. These boxes may be obtained in a variety of ways, i.e. via bins, binary recursive trees, or octrees. We have found the octree to be the best of these possibilities, particularly for grids with a large variation of element size. In order to recover a parallel gridding procedure that resembles closely the advancing front technique on scalar machines, only the boxes covering the active front in regions where the smallest new elements are being introduced are considered. After these boxes have been filled with elements, the process starts anew: a new octree is built, new boxes are created and meshed in parallel. The procedure is summarized schematically for a 2-D case in Figure 1.

At the end of each parallel gridding pass, each one of the boxes gridded can have an internal boundary of faces. For a large number of boxes, this could result in a very large number of faces for the active front. This problem can be avoided by shifting the boxes slightly, and then regridding them again in parallel, as shown in Figure 2. This simple technique has the effect of eliminating almost all of the faces between boxes with a minor modification of the basic parallel gridding algorithm.

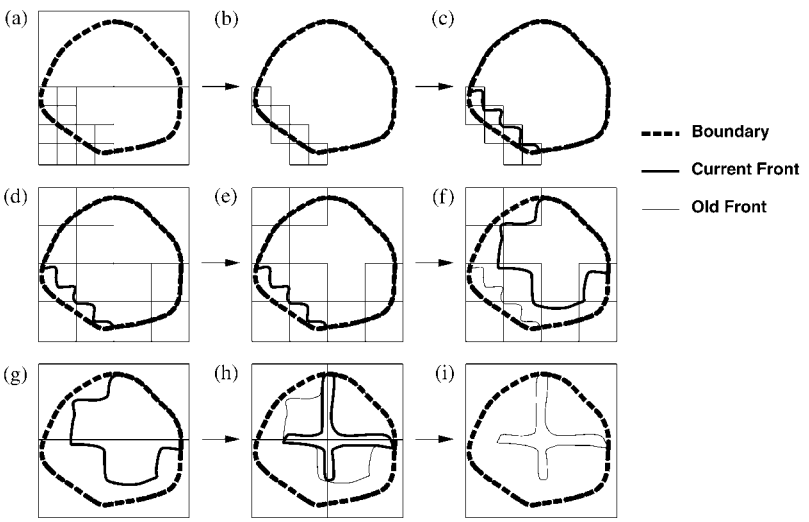

Figure 1. Parallel grid generation.

If we define as $d_{\min}$ the minimum element size in the active front, and as $s_{\min}$ the minimum box size in which elements are to be generated, the parallel AFT proceeds as follows:

```
WHILE: There are active faces left:
  Form an octree with minimum octant size s_min for the active points;
  Retain the octants that have faces that will generate elements of size d_min to c_l · d_min;
  If too many octants are left: agglomerate them into boxes;
  DO ISHFT=0,2:
    IF: ISHFT.NE.0:
      Shift the boxes by a preset amount;
    ENDIF
    Generate, in parallel, elements in these boxes, allowing only elements up to a
    size of c_l · d_min;
  ENDDO
  Increase d_min = 1.5 * d_min, s_min = 1.5 * s_min;
ENDWHILE
```

The increase factor allowed is typically in the range $c_l = 1.5$–2.0. The shift vectors are given by

$$\mathbf{s} = \delta_s(1,1,1), \quad \delta_s = \pm \min(0.5 * s_{\min}, 2.0 * d_{\min})$$

We remark that the octree used to compute the boxes for parallel grid generation is very coarse compared to the element size specified by the user. The edge-length of the finest octree box is of the order of 20–50 times the specified element size. This implies that its construction is very fast, and can be accomplished on a single processor without discernable CPU penalty.

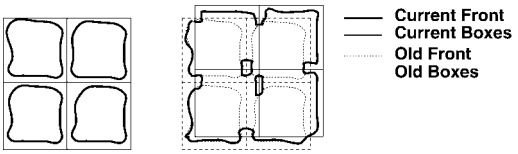

Figure 2. Shift and regrid technique.

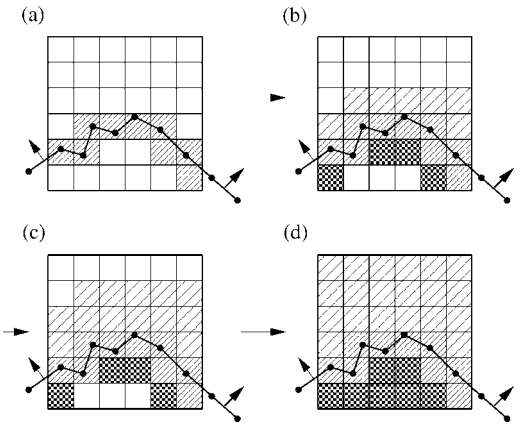

Figure 3. Estimation of volume to be gridded.

3. WORK ESTIMATION AND BALANCE

The procedure outlined above will work optimally if each box requires approximately the same CPU time to complete its grid. This implies that a good work estimate should be provided. Given that the boxes are not body conforming, even for uniform grids the volume to be gridded can vary drastically from box to box. A balanced workload can be obtained by starting with many boxes, estimating the work to be done for each of them, and then gridding in parallel *groups of boxes* with similar workload.

The volume to be gridded is estimated by the marching cubes procedure shown schematically in Figure 3. Given the dimensions of the box, and the list of active faces, the box is first subdivided into voxels (i.e. small cubes). In a first pass over the faces, the voxels cut by faces are marked as 'inside the domain', and an average normal is computed for each cut voxel. This normal information is used in a subsequent pass over the voxels in order to mark the neighbours of cut voxels as either inside or outside the domain to be gridded. The remaining voxels are then marked as inside or outside in several sweeps over the voxels. These sweeps are carried out until no further voxels can be marked. Finally, a work estimate is obtained by summing the expected number of elements in each of the voxels marked as inside the domain to be gridded. This work estimation procedure is done in parallel.

Given the estimated work in each of the boxes, the load is balanced in such a way that each processor receives a similar amount of work. The assumption is made that the number of boxes is always larger than the number of processors. Should this not be the case, the boxes are subdivided further (8 new boxes for each box). If any given box has a work estimate that lies above the average work per processor, this box is also subdivided further. This 'greedy' work balancing algorithm may be summarized as follows:

(a) Obtain a minimum nr. of active boxes:

```
WHILE: The number of boxes is smaller than the number of processors:
  Subdivide boxes (1:8);
ENDWHILE
```

(b) Balance the work:


```
WHILE: Work unbalanced
  Estimate, in parallel, the work for each box;
  Obtain average work per processor;
  IF: A box has a work estimate above average:
  Subdivide it further (1:8) and balance again;
  ENDIF
  Attempt to group boxes in such a way that the work in each group is close to average;
  IF: Good work balance impossible:
  Subdivide boxes with highest work estimate (1:8) and balance again;
  ENDIF
ENDWHILE
```

In many instances, boxes within a group will share a common face. In order to avoid the (unnecessary) buildup of many faces at the borders of boxes, an attempt is made to agglomerate these neighbouring boxes within groups. This is done recursively by checking, for any pair of boxes, if two of the dimensions are the same and the boxes are coincident in the remaining dimension. If so, the boxes are merged. Boxes are merged recursively, until no pair of boxes passes the test outlined above.

It was found that the procedure outlined above can sometimes underestimate the number of elements actually generated by each processor. This is particularly the case for complex geometries or problems with many merging fronts. It was found that the best way of dealing with this underestimation of work was to set a limit on the number of elements generated per processor. If this maximum number of elements estimated is exceeded, the processor returns to the main program with the current front, and the work is redistributed again.

4. MESH IMPROVEMENT

After the generation of the mesh using the parallel advancing front technique has been completed, the mesh quality is improved by a combination of several algorithms, such as:

(a) Diagonal swapping,
(b) Removal of bad elements, and
(c) Laplacian smoothing.

4.1. Diagonal swapping

Diagonal swapping attempts to improve the quality of the mesh by reconnecting locally the points in a different way. Examples of possible 3-D swaps are shown in Figures 4 and 5.

The optimality criterion used is the one proposed by George [40].

$$Q = \frac{h_{\max} S}{V}$$

where $h_{\max}, S$ and V denote the maximum edge length, total surface area and volume of a tetrahedron. The number of cases to be tested can grow factorially with the number of elements surrounding an edge. Figure 6 shows the possibilities to be tested for 4, 5 and 6 elements surrounding an edge.

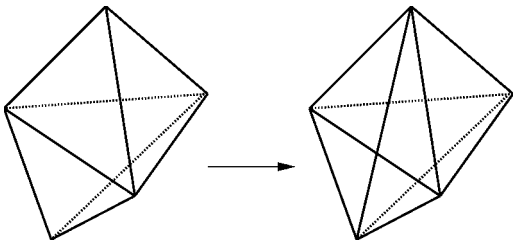

Figure 4. Diagonal swap case 2:3.

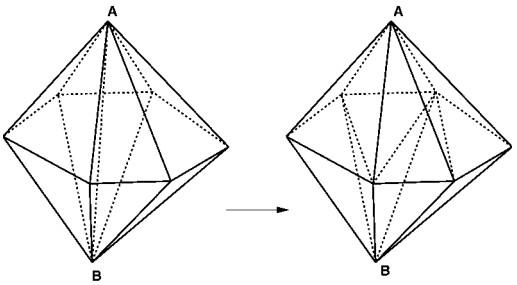

Figure 5. Diagonal swap case 6:8.

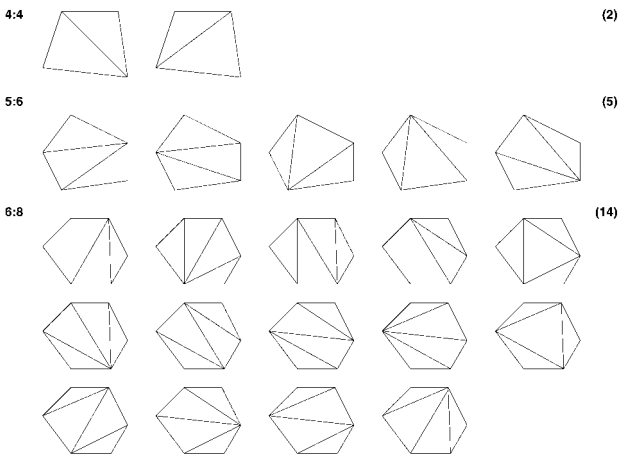

Figure 6. Swapping cases.

Given that these tests are computationally intensive, considerable care is required when coding a fast diagonal swapper. Techniques that were used in the present implementation include:

(i) Treatment of bad ($Q > Q_{\text{tol}}$), untested elements only;
(ii) Processing of elements in an ordered way, starting with the worst (highest chance of reconnection);
(iii) Rejection of bad combinations at the earliest possible indication of worsening quality;
(iv) Marking of tested and unswapped elements in each pass.

As stated before, the computationally intensive part of any diagonal swapping is sifting through all possibilities in order to find a better local point connectivity. One observes that a very large number of tests are required to obtain an improvement. This implies that parallelizing only the checking portion of a diagonal swapper, which can easily be done as no swapping actually occurs, will yield very good speedups. The parallel swapper can then be summarized as follows:

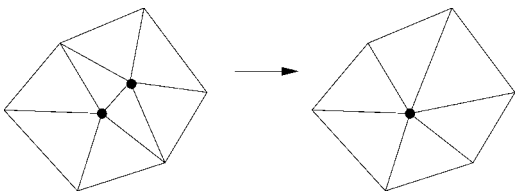

Figure 7. Removal of element.

```
WHILE: There are bad, untested elements in the heap:
    Check the worst elements in parallel;
    Reconnect if possible, and the neighbouring elements have not been reconnected
    (scalar, integer);
    Remember swapped elements;
ENDWHILE
```

4.2. *Removal of bad elements*

A straightforward way to improve a mesh containing bad elements is to get rid of them. For tetrahedral grids this is particularly simple, as the removal of an internal edge does not lead to new element types for the surrounding elements. Once the bad elements have been identified (in parallel), they are compiled into a list and interrogated in turn. An element is removed by collapsing the points of one of the edges, as shown in Figure 7 for a set of triangles in 2-D.

This operation also removes all the elements that share this edge. It is advisable to make a check which of the points of the edge should be kept: point 1, point 2, or a point somewhere on the edge (e.g. the mid-point). This implies checking all elements that contain either point 1 or point 2. This procedure of removing bad elements is simple to implement and relatively fast. On the other hand, it will not tend to improve the overall quality of the mesh. It is therefore used mainly in a pre-smoothing or pre-optimization stage, where its main function is to eradicate elements of very bad quality from the mesh.

4.3. *Laplacian smoothing*

A number of smoothing techniques are lumped under this name. 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. The force exerted by each spring is proportional to its length and along its direction. Therefore, the sum of the forces exerted by all springs surrounding a point can be written as

$$\mathbf{f}_i = c \sum_{j=1}^{ns_i} (\mathbf{x}_j - \mathbf{x}_i)$$

where c denotes the spring constant, $\mathbf{x}_i$ the co-ordinates of the point, and the sum extends over all the points surrounding the point. The time-advancement for the co-ordinates is

accomplished as follows:

$$\Delta \mathbf{x}_i = \Delta t \frac{1}{ns_i} \mathbf{f}_i$$

At the surface of the computational domain, no movement of points is allowed, i.e. $\Delta \mathbf{x} = 0$. Usually, the timestep (or relaxation parameter) is chosen as $\Delta t = 0.8$, and 5–6 timesteps yield an acceptable mesh. The parallelization of the Laplacian smoothing is similar to that of any other field solver. The loops over the elements/edges are coloured so that cash misses are minimized, pipelining does not lead to memory contingencies and cache-line overwrite is avoided [41]. The application of the Laplacian smoothing technique will yield an overall improvement of grid quality, but can result in inverted or negative elements. These negative elements are eliminated. 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. Another possible option is to check for negative elements while smoothing, never allowing a point movement that would result in negative elements. Our experience has been that for the majority of cases, it is faster to use a very simple smoother and to regrid a few holes with negative or small elements than to check at every instant for element quality.

5. SURFACE GRIDDING

For a scalar machine, the CPU required to generate the surface grids is small in comparison to that of the 3-D volume mesh. This situation changes as soon as the number of processors is increased. It was found that for some of the cases shown below, surface gridding required more than 25 per cent of the CPU time once the number of processors increased beyond 16. The situation deteriorates even further if one considers meshing surfaces that are defined in a discrete manner, i.e. the re-triangulation of a triangular mesh [42]. Here, the interpolations and checks required lead to a considerable increase in CPU as compared to planar surfaces or Coon's patches [29]. In order to achieve optimum scalability, the generation of surface grids was also parallelized. The approach taken here is the simplest possible: generate each surface independently on a processor. This approach will lead to load imbalances if some of the surfaces have considerably more elements than others, but its simplicity and ease of implementation make it very appealing.

6. IMPLEMENTATION

The procedure described above was implemented using the shared memory (i.e. `c$doacross`) paradigm on the SGI Origin 2000. Although this is a distributed-memory machine, it can be programmed as a shared-memory machine. This choice was adopted for the following reasons:

- Coding for a shared-memory environment is much simpler than for a distributed-memory environment. The operating system takes care of most of the inherent message passing, communication conflicts, etc., relieving the user from this task.

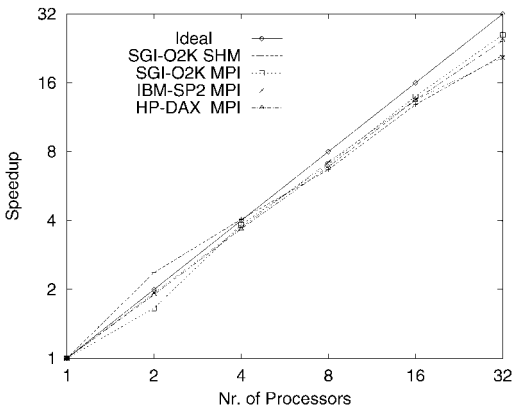

Figure 8. Performance of FEFLO on different platforms.

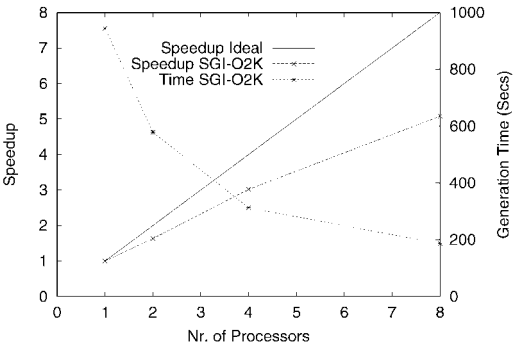

Figure 9. Cube: speedups obtained.

- The production codes used in conjunction with the grid generator scale well using the shared-memory paradigm [41]. Even on 32 processors, more than 50% of the theoretical speedup is achieved. Figure 8 shows the speedups obtained for a steady compressible flow problem using an edge-based, upwind solver on different computer platforms. Note that the speedups obtained using the shared and distributed memory paradigms are comparable.
- For the last years, all of the large-scale production runs carried out by the author and his colleagues [13, 15, 16, 43] were performed on these machines, using the shared-memory paradigm.
- The SGI Origin 2000 has become the dominant platform within the High-Performance Computing sites in the U.S.; at present, there are more SGI Origin 2000 processors than those of all other vendors combined; the expectation is that this trend will not change drastically in the foreseeable future.

While some of the reasons stated above have to do with particular circumstances, the basic ideas of the proposed algorithms are general, and may be coded within a distributed memory framework with explicit message passing.

7. EXAMPLES

The proposed parallel advancing front scheme has been used extensively over the last year in a production environment. We include a sampling of geometries gridded, highlighting the characteristics of the proposed scheme. The timings for all examples include surface gridding, mesh generation and mesh improvement, i.e. they can be considered representative of the timings obtained in a production environment. All timings were obtained on SGI Origin 2000 servers.

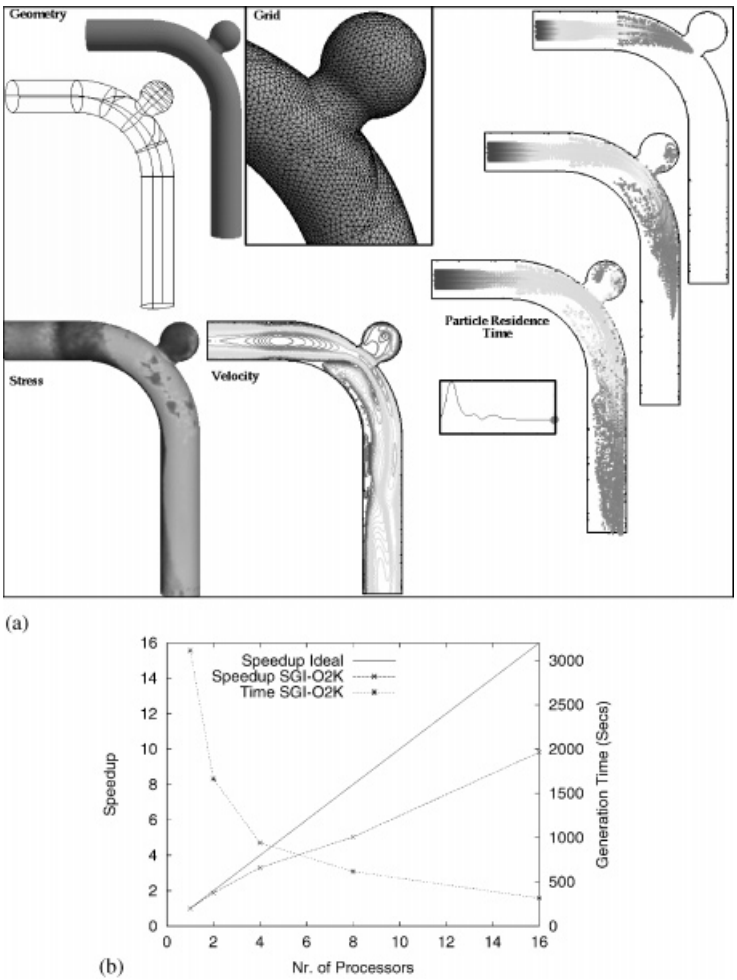

Figure 10. (a) Aneurism; (b) Aneurism: speedups obtained.

7.1. *Cube*

This academic example is included here to see how the procedure works in the 'best case scenario'. The unit cube is to be gridded with a uniform mesh of approximately 1 million tetrahedra. Although this is not a large grid, the timings shown in Figure 9 are illustrative.

7.2. *Aneurism*

This example is taken from a hemodynamic analysis recently conducted for this particular geometry. The outline of the domain, grid, as well as some sample results are shown in Figure 10(a).

The final uniform mesh had approximately 2.7 million tetrahedra. The speedup obtained is shown in Figure 10(b). As one can see, the parallel mesher was more efficient for this

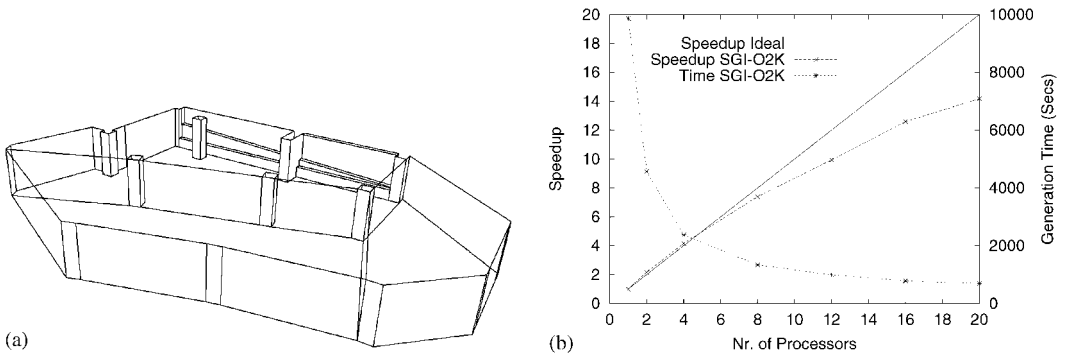

Figure 11. (a) Garage: wireframe; (b) Garage: speedups obtained.

finer grid than for the unit cube, even though the volume to be gridded in each box can vary widely due to geometric features.

7.3. Garage

This example was taken from a blast simulation recently carried out for an office complex. The outline of the domain is shown in Figure 11(a).

The final uniform mesh had approximately 9.2 million tetrahedra. The speedup obtained is shown in Figure 11(b). As before, the parallel mesher was more efficient for this finer grid than for the unit cube, even though the volume to be gridded in each box can vary widely due to geometric features.

7.4. Tysons Corner

This example was taken from a dispersion simulation recently carried out for this well-known shopping center. The outline of the domain is shown in Figures 12(a) and (b).

The final mesh had approximately 16 million tetrahedra. The smallest and largest specified element side lengths were 230 and 1400 cm, respectively. The speedup obtained is shown in Figure 12(c).

7.5. Space shuttle

This example is included here because it is typical of many aerodynamic data sets. The outline of the domain is shown in Figure 13(a).

The surface triangulation of the final mesh, which had approximately 4 million tetrahedra, is shown in Figure 13(b). The smallest and largest specified element side lengths were 5.08 and 467 cm, respectively, i.e. an edge-length ratio of approximately $1:10^2$ and a volume ratio of $1:10^6$. The spatial variation of element size was specified via approximately 200 sources [34]. Figure 13(c) shows the boxes used for parallel meshing as the element size increases during the grid generation process.

The speedup obtained is shown in Figure 13(d). Finally, the results of an Euler run for an incoming Mach number of $M = 2.0$ and angle of attack of $\alpha = 1.1^\circ$ are shown in Figure 13(e).

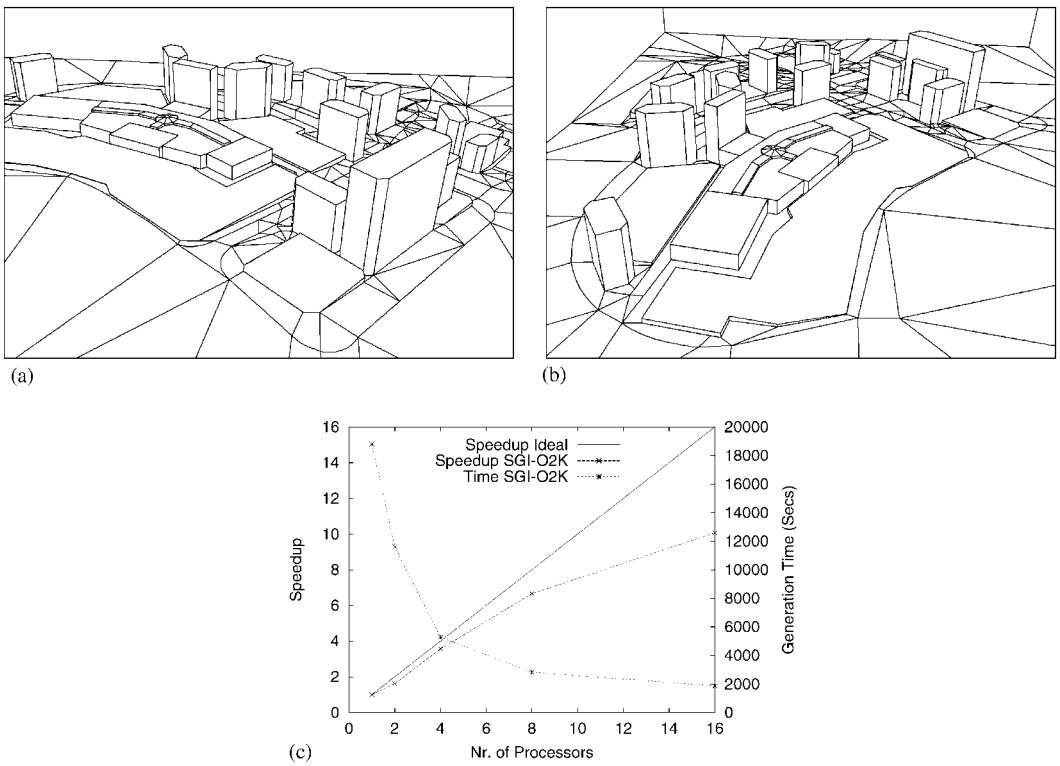

Figure 12. (a) Tysons Corner: wireframe (W); (b) Tysons Corner: wireframe (N); (c) Tysons Corner: speedups obtained.

7.6. Pilot ejecting from F18

Problems with moving bodies often require many remeshings during the course of a simulation. The case shown here was taken from a pilot ejection simulation recently conducted. The outline of the domain is shown in Figure 14(a).

The surface triangulation of the final mesh, which had approximately 14 million tetrahedra, is shown in Figure 14(b). The smallest and largest specified element side lengths were 0.65 and 250 cm, respectively, i.e. an edge–length ratio of approximately $1:4\times10^2$ and a volume ratio of $1:5.6\times10^7$. The spatial variation of element size was specified via approximately 110 sources [34]. The speedup obtained for two different grid sizes is displayed in Figure 14(c).

As could be seen from the previous examples, the grid generator certainly scales well with the number of processors. As with many other areas where parallel processing is being attempted, scalability improves with the amount of work required. The larger the grids, the better the scalability. One can also observe that for each case the 'scalability slope' is somewhat different, ranging from almost perfect for the garage to a $1:3$ asymptote for the shuttle. Closer inspection revealed that for the shuttle, although the final mesh contains 4 million elements, the number of elements created in each parallel pass over increasing element

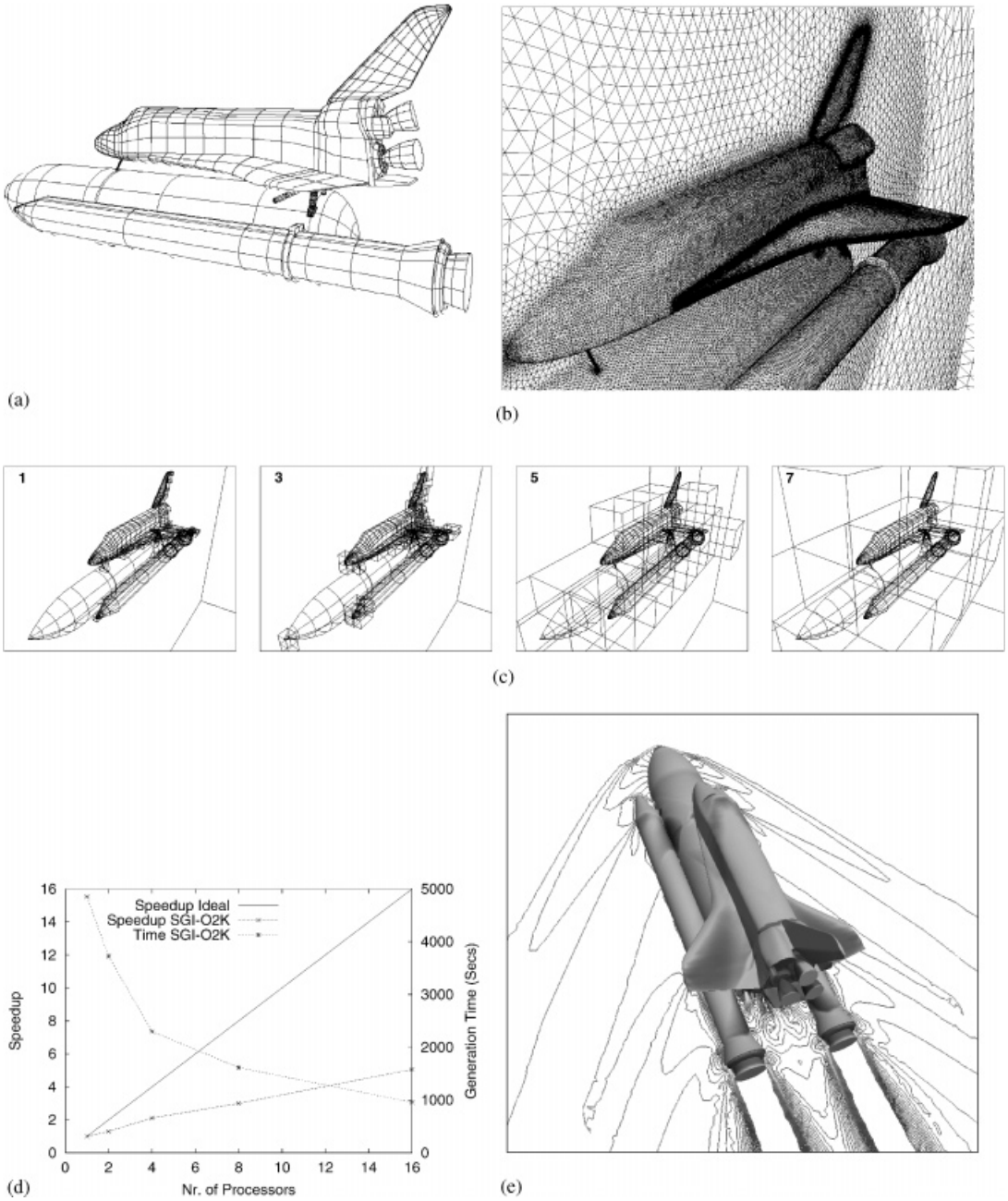

Figure 13. (a) Space shuttle: outline of domain; (b) Space shuttle: surface mesh; (c) Space shuttle: gridding boxes; (d) Space shuttle: speedups obtained; (e) Space shuttle: surface pressures.

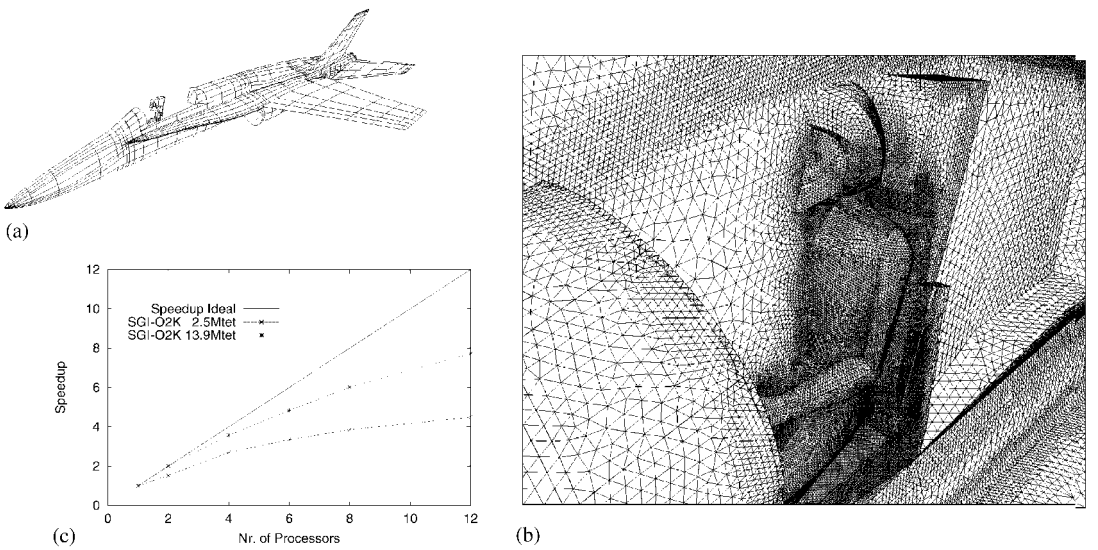

Figure 14. (a) F18 pilot ejection: outline of domain; (b) F18 pilot ejection: surface mesh (closeup); (c) F18 pilot ejection: speedups obtained.

sizes was rather modest, never exceeding 0.5 million elements. Thus, the inherent parallelism of this problem is rather low.

8. CONCLUSIONS AND OUTLOOK

A parallel advancing front scheme has been developed. The domain to be gridded is first subdivided spatially using a relatively coarse octree. Boxes are then identified and gridded in parallel. A scheme that resembles closely the advancing front technique on scalar machines is recovered by only considering the boxes of the active front that generate small elements. The procedure has been implemented on the SGI Origin class of machines using the shared memory paradigm. Timings for a variety of cases show speedups similar to those obtained for flow codes. The procedure has been used to generate grids for a large variety of cases, and is nearing production maturity.

Current work is focusing on improved work prediction algorithms, as it is found that in going to a larger number of processors, any small imbalance incurs a heavy CPU penalty.

ACKNOWLEDGEMENTS

This research was partially supported by AFOSR, with Dr. Leonidas Sakell as the technical monitor.

REFERENCES

1. Baum JD, Luo H, Löhner R. Numerical simulation of a blast inside a Boeing 747. *AIAA*-93-3091, 1993.
2. Baum JD, Luo H, Löhner R. Numerical simulation of blast in the World Trade Center. *AIAA*-95-0085, 1995.

3. Jou W. Comments on the feasibility of LES for commercial airplane wings. *AIAA*-98-2801, 1998.
4. Yoshimura S, Nitta H, Yagawa G, Akiba H. Parallel automatic mesh generation method of ten-million nodes problem using fuzzy knowledge processing and computational geometry. *Proceedings of the 4th World Congress on Computational Mechanics*, Buenos Aires, Argentina, July 1998.
5. Mavriplis DJ, Pirzadeh S. Large-scale parallel unstructured mesh computations for 3-D high-lift analysis. *ICASE Report* 99-9, 1999.
6. Darve E, Löhner R. Advanced structured–unstructured solver for electromagnetic scattering from multimaterial objects. *AIAA*-97-0863, 1997.
7. Morgan K, Brookes PJ, Hassan O, Weatherill NP. Parallel processing for the simulation of problems involving scattering of electro-magnetic waves. In *Proceedings of the Symposium on Advances in Computational Mechanics*, Demkowicz L, Reddy JN (eds), 1997.
8. Löhner R. Three-dimensional fluid–structure interaction using a finite element solver and adaptive remeshing. *Computer Systems in Engineering* 1990; **1**:(2–4) 257–272.
9. Mestreau E, Löhner R, Aita S. TGV tunnel-entry simulations using a finite element code with automatic remeshing. *AIAA*-93-0890, 1993.
10. Mestreau E, Löhner R. Airbag simulation using fluid/structure coupling. *AIAA*-96-0798, 1996.
11. Baum JD, Luo H, Löhner R, Yang C, Pelessone D, Charman C. A coupled fluid/structure modeling of shock interaction with a truck. *AIAA*-96-0795, 1996.
12. Kamoulakos A, Chen V, Mestreau E, Löhner R. 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 1996.
13. Löhner R, Yang C, Cebral J, Baum JD, Luo H, Pelessone D, Charman C. Fluid–structure-thermal interaction using a loose coupling algorithm and adaptive unstructured grids. *AIAA*-98-2419, 1998 (Invited).
14. Hassan O, Bayne LB, Morgan K, Weatherill NP. An adaptive unstructured mesh method for transient flows involving moving boundaries. *In Computational Fluid Dynamics '98*, Papailiou KD, Tsahalis D, Périaux J, Knörzer D (eds). Wiley: New York, 1998; 662–674.
15. Baum JD, Luo H, Löhner R. The numerical simulation of strongly unsteady flows with hundreds of moving bodies. *AIAA*-98-0788, 1998.
16. Baum JD, Luo, H, Mestreau E, Löhner R, Pelessone D, Charman C. A coupled CFD/CSD methodology for modeling weapon detonation and fragmentation. *AIAA*-99-0794, 1999.
17. Löhner R, Camberos J, Merriam M. Parallel unstructured grid generation. *Computer Methods in Applied Mechanics and Engineering* 1992; **95**:343–357.
18. de Cougny HL, Shephard MS, Ozturan C. Parallel three-dimensional mesh generation. *Computing Systems in Engineering* 1994; **5**:311–323.
19. Shostko A, Löhner R. Three-dimensional parallel unstructured grid generation. *International Journal for Numerical Methods in Engineering* 1995; **38**:905–925.
20. de Cougny HL, Shephard MS, Ozturan C. Parallel three-dimensional mesh generation on distributed memory MIMD computers. *Technical Report SCOREC Rep.* # 7, Rensselaer Polytechnic Institute, 1995.
21. Okusanya T, Peraire J. Parallel unstructured mesh generation. *Proceedings of the 5th International Conference on Numerical Grid Generation in CFD and Related Fields*, Mississippi, April 1996.
22. Chew LP, Chrisochoides N, Sukup F. Parallel constrained Delaunay meshing. *Proceedings of the 1997 Workshop on Trends in Unstructured Mesh Generation*, June 1997.
23. Okusanya T, Peraire J. 3-D parallel unstructured mesh generation. *Proceedings of the Joint ASME/ASCE/SES Summer Meeting*, 1997.
24. Said R, Weatherill NP, Morgan K, Verhoeven NA. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering* 1999, to appear.
25. Chrisochoides N, Nave D. Simultaneous mesh generation and partitioning for Delaunay meshes. *Proceedings of the 8th International Meshing Roundtable*, South Lake Tahoe, October 1999; 55–66.
26. Peraire J, Vahdati M, Morgan K, Zienkiewicz OC. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics* 1987; **72**:449–466.
27. Peraire J, Peiro J, Formaggia L, Morgan K, Zienkiewicz OC. Finite element Euler calculations in three dimensions. *International Journal for Numerical Methods in Engineering* 1988; **26**:2135–2159.
28. Löhner R. Some useful data structures for the generation of unstructured grids. *Communications in Applied Numerical Methods* 1988; **4**:123–135.
29. Löhner R, Parikh P. Three-dimensional grid generation by the advancing front method. *International Journal for Numerical Methods in Fluids* 1988; **8**:1135–1149.
30. Peraire J, Morgan K, Peiro J. Unstructured finite element mesh generation and adaptive procedures for CFD. *AGARD-CP-464*, 18, 1990.
31. Peraire J, Morgan K, Peiro J. Adaptive remeshing in 3-D. *Journal of Computational Physics*, 1992.
32. Jin H, Tanner RI. Generation of unstructured tetrahedral meshes by the advancing front technique. *International Journal for Numerical Methods in Engineering* 1993; **36**:1805–1823.

33. Frykestig J. Advancing front mesh generation techniques with application to the finite element method. *Pub*. 94:10, Chalmers University of Technology; Göteborg, Sweden, 1994.
34. Löhner R. Progress in grid generation via the advancing front technique. *Engineering with Computers* 1996; **12**:186–210.
35. Baker TJ. Developments and trends in three-dimensional mesh generation. *Applied Numerical Mathematics* 1989; **5**:275–304.
36. George PL, Hecht F, Saltel E. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering* 1991; **92**:269–288.
37. Weatherill NP. Delaunay triangulation in computational fluid dynamics. *Computational Mathematics Application* 1992; **24**(5/6):129–150.
38. Weatherill NP, Hassan O. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering* 1994; **37**: 2005–2039.
39. Marcum DL, Weatherill NP. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA Journal* 1995; **33**(9):1619–1625.
40. George PL. Tet meshing: construction, optimization and adaptation. *Proceedings of the 8th International Meshing Roundtable*, South Lake Tahoe, October 1999.
41. Löhner R. Renumbering strategies for unstructured-grid solvers operating on shared-memory, cache-based parallel machines. *Computer Methods in Applied Mechanics and Engineering* 1998; **163**:95–109.
42. Löhner R. Regridding surface triangulations. *Journal of Computational Physics* 1996; **126**:1–10.
43. Löhner R, Yang C, Oñate E. Viscous free surface hydrodynamics using unstructured grids. *Proceedings of the 22nd Symposium Naval Hydrodynamics*, Washington, DC, August 1998.

