15th International Symposium on Flow Visualization
June 25-28, 2012, Minsk, Belarus

ACCELERATION OF SOME PRE- AND POSTPROCESSING ALGORITHMS FOR CFD DATA ON 3-D UNSTRUCTURED GRIDS

P.A. VOINOVICH

Saint-Petersburg Branch of Joined Supercomputer Center
of the Russian Academy of Sciences, Saint-Petersburg, Russsia

Tel.: +7 812 297 2365; Email: vpeter@scc.ioffe.ru

KEYWORDS: CFD preprocessing, postprocessing, unstructured grids, algorithm acceleration, hash indexing

ABSTRACT: The 3-D unstructured grids based on tetrahedral grid elements are widely used in CFD simulations for decades. The modern computer hardware offers resources sufficient to deal with the grids containing many tens or even hundreds of millions of the grid objects N (grid nodes, tetrahedra, triangular faces, edges) using common personal computers, and several orders of magnitude more using powerful computer installations. Some intuitive algorithms for pre- and/or postprocessing of the grid and CFD data result in computational complexity of $O(N^2)$, which makes their application to the large grids impractical. Here belongs creation of data sets for grid edges or faces from the data base for tetrahedra referring to the grid nodes as their vertices. Extraction of domain boundary faces (surface triangulation) from a nodal-tetrahedral data base represents another typical example. This can be done using a temporary data set for all the triangular faces of the grid tetrahedra. For every tetrahedron, each of its 4 faces is compared to the existing faces in the current data set: if the face already exists, it is marked as an internal face of the computational domain, if not – the face is added to the data set. Finally, all the unmarked faces belong to the domain boundary, as they were encountered only once, in contrast to the internal faces which separate two adjacent tetrahedra. The above algorithm has a complexity of $O(N_{TETRAHEDRA} \cdot N_{FACES})$, i.e. $O(N^2)$. This makes it extremely inefficient in postprocessing, especially when an animated image is created from the data on a grid which varies in time, as in the case of adaptively refined grids applied to a transient problem. Hash-indexing the faces can essentially reduce the number of comparisons for each face to be tested. Several types of hash functions and hash tables were implemented. A drastic improvement in computational efficiency was achieved using a three-dimensional hash table and indexing the faces according to the Cartesian coordinates of their centers. However, even better result was obtained using a one-dimensional hash table and a hash function based on the numbers of the grid nodes corresponding to the face vertices. This reduced the computational complexity of the above algorithm to $O(N_{TETRAHEDRA})$.

INTRODUCTION. The unstructured grids are widely used in numerical simulation because they can naturally fit complex geometries and their flexible data structures support local grid modifications (e.g. solution-adaptive refinement) according to the resolution demands. A regular desktop or notebook computer with RAM of a few Gigabytes can process unstructured grids containing many millions of grid elements (grid nodes, faces, edges, control volumes etc.). Numerical simulations using much larger data sets can be performed on high-performance multiprocessor computer systems which are easily accessible via the modern telecommunication technologies. However, some pre- and postprocessing operations have still to be performed on the user side with essentially limited computer resources. For a grid composed of N elements and an algorithm of time complexity $O(N^k)$, the runtime can become inadmissibly long for $k>1$. A representative algorithm typical for pre- or postprocessing of grid data and approaches to accelerate it is considered in the present paper in application to a 3-D unstructured grid composed of tetrahedra.

UNSTRUCTURED GRID DATA SETS. A typical minimal data set of a tetrahedral unstructured grid contains a table of grid nodes with their Cartesian coordinates X_i, Y_i, Z_i, $i = 1 \ldots N_{VERT}$, defining grid geometry, and a table of grid tetrahedra with their references to the grid nodes as tetranedra vertices $iv1_i$, $iv2_i$, $iv3_i$, $iv4_i$, $i = 1 \ldots N_{TETR}$, defining grid connectivity. The grid data set may additionally contain some other grid objects and references: a table of grid edges with references to the nodes $iv1_i$, $iv2_i$, $i = 1 \ldots N_{EDGE}$; a table of grid faces with references to the nodes $iv1_i$, $iv2_i$, $iv3_i$, $i = 1 \ldots N_{FACE}$. The table of tetrahedra may contain references to the tetrahedra edges and/or faces.

The grid-related CFD data can be associated with any of the above grid objects (tetrahedra, faces, edges, nodes). In our 3-D unstructured adaptive CFD code [1], for instance, the following data sets have been used: a table of grid nodes with their Cartesian coordinates, control volumes, and dependent variables; a table of tetrahedra with references to the nodes as their vertices and references to the edges, a "child" tetrahedron has also a reference to its "parent" tetrahedron; a table of grid edges connecting two vertices of a tetrahedron with the references to the nodes as edge ends. The edge is associated with the respective interface separating two control volumes surrounding each of the involved nodes. Some additional work variables and flags are used to perform computations by the solver and to support the grid adaptation routines.

A SAMPLE ALGORITHM OF COMPLEXITY $O(N^2)$. Consider a representative example. Suppose we have a minimal grid data set as described above, i.e. a table of grid nodes and a table of tetrahedra referencing to the nodes as their vertices. The task is to create a table of triangular grid faces referencing to the nodes as their vertices and labeled to distinguish between the internal faces separating two adjacent tetrahedra and faces which belong to the computational domain boundary. This sample problem could be a part of preprocessing, e.g. to set up the boundary conditions, or to test the grid data integrity, as well as a post processing unit, e.g. for graphic presentation of the computed data at the domain surface, as illustrated in Fig. 1.

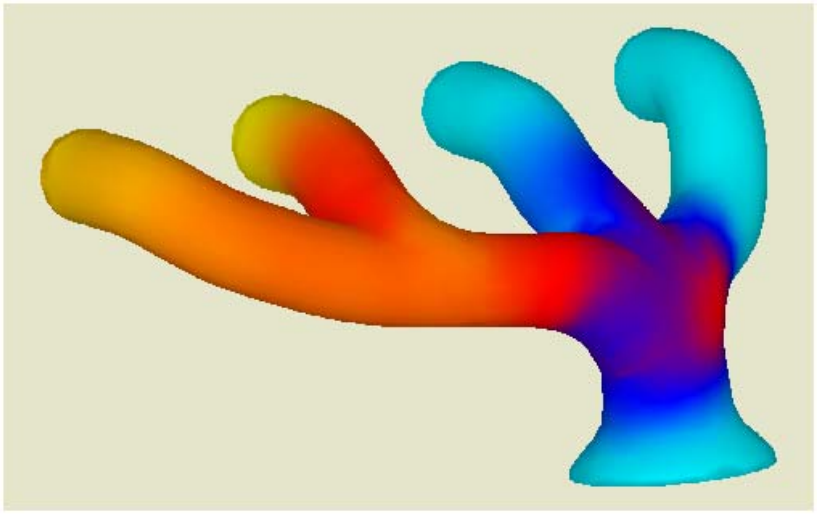

Fig. 1 Visualization of an instant pressure distribution at the surface of an exhaust manifold.

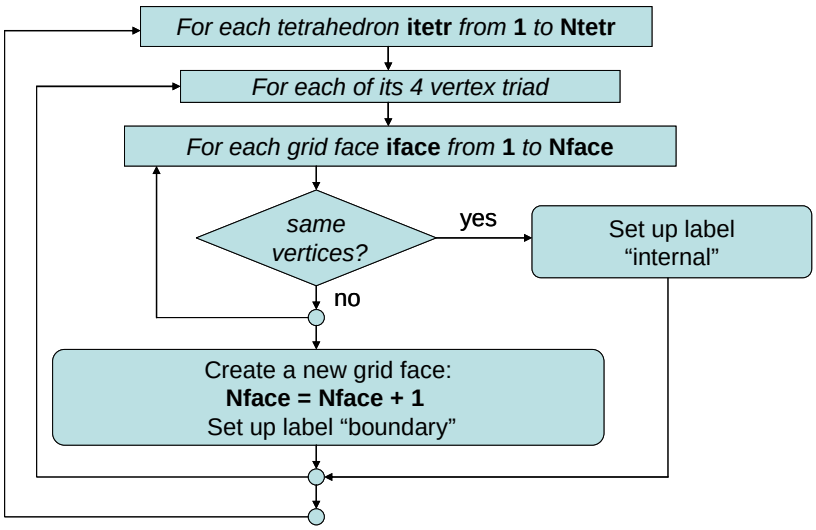

Fig. 2 A straightforward algorithm completing the grid faces table.

An intuitive algorithm to fulfill the above task is presented in Fig. 2 and given below in a symbolic algorithm language. For every tetrahedron, each of its 4 faces is compared to the existing faces in the current data set: if the face already

exists, it is labeled as an internal face of the computational domain, if not – the face is added to the data set as a boundary face. One can easily see that the time complexity of the algorithm is $O(N_{TETR} \cdot N_{FACE})$, or $O(N^2)$, where N is the number of all grid elements regardless of their type. The praxis shows that a straightforward implementation of this algorithm on a modern personal computer for a grid containing a few million elements may turn impractical because of a very long execution time. The problem becomes more acute in case of multiple post processing treatments, e.g. for dynamic visualization.

```
Nedge = 0
For each tetrahedron itetr from 1 to Ntetr do
        For each tetrahedron face itetrface from 1 to 4
                { Vt1=vertetr(itetr,1), Vt2=vertetr(itetr,2), Vt3=vertetr(itetr,3);
                  Vt1=vertetr(itetr,1), Vt2=vertetr(itetr,2), Vt3=vertetr(itetr,4);
                  Vt1=vertetr(itetr,1), Vt2=vertetr(itetr,3), Vt3=vertetr(itetr,4);
                  Vt1=vertetr(itetr,2), Vt2=vertetr(itetr,3), Vt3=vertetr(itetr,4) } do
            For each grid face iface from 1 to Nface do
               Vf1=verface(iface,1), Vf2=verface(iface,2), Vf3=verface(iface,3)
                        If (min(Vf1,Vf2,Vf3)=min(Vt1,Vt2,Vt3) and
                           mid(Vf1,Vf2,Vf3)=mid(Vt1,Vt2,Vt3) and
                           max(Vf1,Vf2,Vf3)=max(Vt1,Vt2,Vt3) ) then
                                facetype(Nface)='internal'
                                go to LABEL
                        end if
                                Nface = Nface + 1
                                verface(Nface,1)=Vt1
                                verface(Nface,2)=Vt2
                                verface(Nface,3)=Vt3
                                facetype(Nface)='boundary'
                end do
LABEL:
        end do
end do
```

HASH-INDEXING DATA FOR ALGORITHM ACCELERATION. *Hash implementation.* An essential reduction in time complexity of the considered algorithm can be achieved by the elimination of unnecessary comparisons of faces

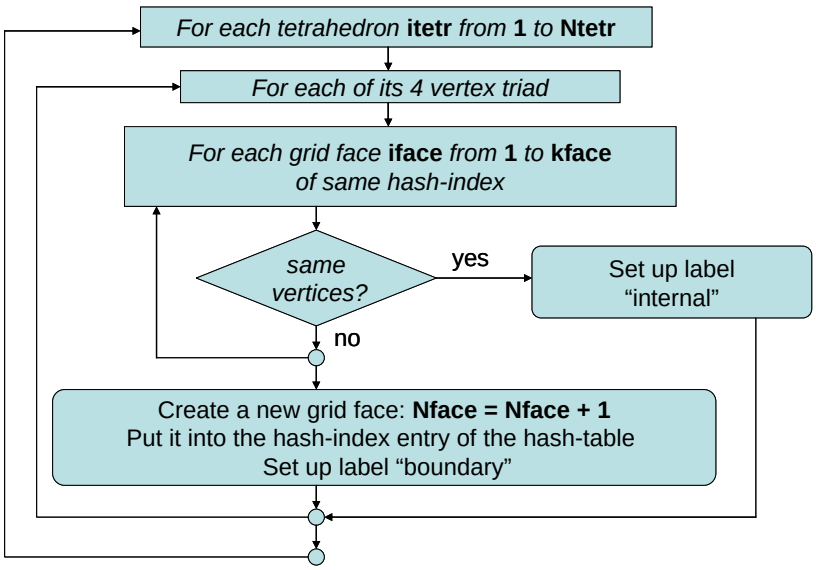

Fig. 3 An algorithm completing the grid faces table with hash-indexing.

which are far from one another in some sense. This can be accomplished through implementation of hashing [2]. The items for comparison (the faces) are to be distributed over multiple entries of a hash table; a single-valued hash function applied to an item provides a hash index defining the table entry to which the item belongs; and finally, the comparisons have only to be performed with the items of same hash index, i.e. referenced in the respective hash table entry, as shown in Fig. 3 and also below using a symbolic language. If the number of hash table entries is comparable to the number of grid objects N, and the hash function distributes the grid objects over the table entries uniformly, then the time complexity of our algorithm becomes as low as $O(N)$.

```
For each INDEX do {TABLE(INDEX) = 'empty'} end do, Nface = 0
For each tetrahedron itetr from 1 to Ntetr do
    For each tetrahedron face itetrface from 1 to 4
        { Vt1=vertetr(itetr,1), Vt2=vertetr(itetr,2), Vt3=vertetr(itetr,3);
          Vt1=vertetr(itetr,1), Vt2=vertetr(itetr,2), Vt3=vertetr(itetr,4);
          Vt1=vertetr(itetr,1), Vt2=vertetr(itetr,3), Vt3=vertetr(itetr,4);
          Vt1=vertetr(itetr,2), Vt2=vertetr(itetr,3), Vt2=vertetr(itetr,4) } do
            INDEX = hash_function(Vt1,Vt2,Vt3)
            if( TABLE(INDEX) = 'empty' )  then
                Nface = Nface + 1
    verface(Nface,1)=Vt1, verface(Nface,2)=Vt2, verface(Nface,3)=Vt3
            facetype(Nface)='external'
                TABLE(INDEX) add Nface
            else
        For each grid face iface from 1 to kface of TABLE(INDEX) do
            Vf1=verface(iface,1), Vf2=verface(iface,2), Vf3=verface(iface,3)
            If (min(Vf1,Vf2,Vf3)=min(Vt1,Vt2,Vt3) and
                mid(Vf1,Vf2,Vf3)=mid(Vt1,Vt2,Vt3) and
                max(Vf1,Vf2,Vf3)=max(Vt1,Vt2,Vt3) ) then
              facetype(Nface)='internal', go to LABEL
            end if
        end do
                Nface = Nface + 1
    verface(Nface,1)=Vt1, verface(Nface,2)=Vt2, verface(Nface,3)=Vt3
            facetype(Nface)='boundary'
                TABLE(INDEX) add Nface
        end do
    LABEL:
    end do
end do
```

A coordinate-based hash function. An obvious solution to the selection of a hash function for the grid objects is based on the coordinates of the latters. If X_{min} and X_{max} are the minimum and maximum coordinates of the computational domain in X direction, and the hash table has entries from 1 to N_{table}, then a function like

$$index = int[(X_c - X_{min})/(X_{max} - X_{min})\, N_{table}] + 1$$

will distribute the grid objects among the table entries according to the coordinates X_c of their centers. In three spatial dimensions, it is natural introducing a 3-D hash table and three hash functions operating with the grid object centers in each coordinate direction: X_c, Y_c, and Z_c. A practical implementation of the described approach showed a drastic reduction in computational time compared to the baseline algorithm. However, the 3-D hash table requires large memory resources, while the hash functions do not work perfectly leaving multiple table entries unused, especially in case of essentially non-uniform grids, as typically occurs by solution-adaptive refinement.

A simple and efficient hash function. An alternative hash function has been proposed based on the numbers of grid nodes involved in the objects under consideration:

$$index = mod(V_{f1} + V_{f2} + V_{f3}, N_{table})$$

where V_{f1}, V_{f2}, V_{f3} are the numbers of the grid nodes (i.e. respective entries in the table of nodes) associated with the vertices of a given face, and N_{table} is the hash table size. The practical experience has shown that a good choice for the hash table size is that of the grid nodes table. This offers an additional benefit of efficient memory use, as the hash table can utilize a work array for the grid nodes, which is not in use at the moment of face table construction. The above hash

function has demonstrated an excellent efficiency on the unstructured grids containing millions of nodes by reducing the algorithm execution time from many hours to just a few seconds on a regular personal computer.

Application to the grid edges. A similar approach can be applied to the construction of the grid edges table and setting up the references from the tetrahedra or faces to their edges. This can be a typical pre-processing task, as the edges are often used in computations by CFD solvers, while the grid generation software may not provide such data. The above coordinate based hash indexing can be applied to the edges directly. The node numbers based hash function in this case reduces to

$$index = mod(V_{e1}+V_{e2},N_{table})$$

where V_{e1}, V_{e2} are the numbers of the grid nodes (i.e. respective entries in the table of nodes) associated with the ends of a given edge.

CONCLUSION. During a practical work on CFD simulations using large 3-D unstructured grids we faced a problem of unacceptably long execution time of some pre- and postprocessing routines constructing the tables of grid faces and edges from the data for grid tetrahedra and nodes. The problem stems from the fact that the respective intuitive algorithms have time complexity of $O(N^2)$. Hash indexing the grid data reduced the time complexity to $O(N)$. A simpe and efficient hash function has been proposed based on the grid nodes numbers and resulting in tremendous acceleration of the algorithms.

References

1. Saito T., Voinovich P., Timofeev E., Takayama K. *Development and Application of High-resolution Adaptive Numerical Techniques in Shock Wave Research Center. In: Toro E.F. (ed) Godunov Methods: Theory and Applications.* Kluwer Academic/Plenum Publishers, NY, 2001, pp. 763–784
2. Donald Knuth *The Art of Computer Programming. 3: Sorting and Searching (2nd ed.)*. Addison-Wesley, Reading, Massachusetts, 1998