Available online at www.sciencedirect.com
ScienceDirect
Computer Physics Communications 175 (2006) 721–737
Computer Physics Communications
www.elsevier.com/locate/cpc

Parallel adaptive mesh-refining scheme on a three-dimensional unstructured tetrahedral mesh and its applications

Y.-Y. Lian, K.-H. Hsu, Y.-L. Shao, Y.-M. Lee, Y.-W. Jeng, J.-S. Wu *

Department of Mechanical Engineering, National Chiao-Tung University, Hsinchu 30050, Taiwan

Received 6 February 2006; received in revised form 10 May 2006; accepted 10 May 2006

Available online 26 September 2006

Abstract

The development of a parallel three-dimensional (3-D) adaptive mesh refinement (PAMR) scheme for an unstructured tetrahedral mesh using dynamic domain decomposition on a memory-distributed machine is presented in detail. A memory-saving cell-based data structure is designed such that the resulting mesh information can be readily utilized in both node- or cell-based numerical methods. The general procedures include isotropic refinement from one parent cell into eight child cells and then followed by anisotropic refinement which effectively removes hanging nodes. A simple but effective mesh-quality control mechanism is employed to preserve the mesh quality. The resulting parallel performance of this PAMR is found to scale approximately as $N^{1.5}$ for $N_{\text{proc}} \leqslant 32$. Two test cases, including a particle method (parallel DSMC solver for rarefied gas dynamics) and an equation-based method (parallel Poisson–Boltzmann equation solver for electrostatic field), are used to demonstrate the generality of the PAMR module. It is argued that this PAMR scheme can be applied in any numerical method if the unstructured tetrahedral mesh is adopted.

Keywords: Parallel adaptive mesh refinement; Unstructured tetrahedral mesh; Mesh-quality control

1. Introduction

Unstructured grid topology has recently attracted tremendous attention in several disciplines. Examples of which include computational fluid dynamics (CFD) [1,2], computational molecular gas dynamics [3], computational structural mechanics [4, and references cited therein], computational density functional theory in quantum mechanics [5, and references cited therein], computational electromagnetics [6, and references cited therein], among other disciplines. The advantages of using an unstructured mesh over a structured one include higher flexibility in handling boundaries with complicated geometry, and the possibility of adaptively refining the mesh and efficient parallel computing using the graph-partitioning technique for large-scale problems. The disadvantages include indirect addressing to memory for accessing grid data and a possibly longer simulation time as compared to solvers using a structured mesh for the same number of nodes. However, the fast developing high-speed computer technology outpaces these disadvantages nowadays. This explains the rapid progress in developing the unstructured-mesh solvers within several research fields in the past decade.

Detailed reviews of algorithms for adaptive refinement of triangular and tetrahedral meshes in serial processing are skipped here for brevity since we are interested in developing a parallel adaptive mesh refinement (PAMR). Thorough reviews could be found in, for example, [7] and references cited therein. Furthermore, the combination of the parallel physical solver and the parallel adaptive mesh refinement (PAMR) using a three-dimensional unstructured mesh may represent the most powerful technique for numerically solving physical problems. The parallel implementation of a physical solver (equation or particle) is conceptually easy, although the practice is rather generally involved. The development of a PAMR is relatively popular in the past, but most studies only concentrated on a two-dimensional unstructured mesh, e.g., [8]. Nevertheless, PAMR in a three-dimensional unstructured mesh is required to increase numerical accuracy and reduce the number of grid points (thus,

* Corresponding author.
E-mail address: chongsin@faculty.nctu.edu.tw (J.-S. Wu).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2006.05.010

memory) for a large-scale parallel 3-D physical simulation. Unfortunately, studies done on 3-D PAMR are very rare [9–11] mainly because of its complicated logic and the data structures required to track the grid points that are added or removed during the refining process. There are several ways to classify PAMR methods, and some of them are reviewed in the following.

First, from the viewpoint of practical implementation on computers, PAMR can be generally categorized into two types: vectorized shared memory and distributed memory. In the shared memory type [12], the hardware for parallel implementation is generally very expensive, and the coding is comparably easy, like using OpenMP [13], for instance. Memory access is direct and fast, which makes communication cost among processors minimal. Moreover, load balancing among processors is expected to be good, while memory contention may occur which possibly slows down the speed. However, some particle methods, such as direct simulation Monte Carlo (DSMC) [14], are not suitable for this kind of vectorized parallel implementation because of too many logical decisions involved in the actual code. In contrast, the hardware is more affordable for the distributed memory type [9–11] than the shared memory type, while the coding is rather involved using MPI [15], for instance. Global data access is gained through network communication, which often becomes the bottleneck for better parallel performance if a large number of processors is used. However, this becomes increasingly unimportant because of the rapid increase in the speed of network communication, and the dramatic reduction of costs in the past decade. Deciding on what type of parallel machine to implement greatly impacts the fundamental algorithm design as well as the practical implementation of the PAMR.

Second, from the viewpoint of the AMR algorithm itself, there are generally two kinds of mesh refinement: *h*-refinement and mesh regeneration. Mesh regeneration AMR [16] regenerates mesh from a mesh generator based on the distribution of error estimator obtained from the previous coarse mesh. For a large-scale 3-D computation, there exist two problems. First, the process often takes too much time, and second, the interpolation of previous solutions onto the new mesh is rather inefficient, which may slow down the convergence for the simulation on the new mesh. However, the data structure is relatively simple as compared to the *h-refinement* AMR. The *h-refinement* AMR [9–11] adds grid points onto cell edges based on the error estimator obtained from the previous coarse mesh. It is generally much faster than the mesh regeneration type, and the interpolation onto the new mesh from the previous solution is straightforward, which may speed up the convergence for the computation on the new mesh. Often, the resulting "edge-based" data structure is very complicated and memory demanding, especially for the 3-D PAMR on the memory-distributed type of machine. Nevertheless, this type of AMR also has another advantage in that its data structure may be readily modified for mesh de-refinement, which may become important for a time-dependent simulation. Additionally, the *h-refinement* AMR possesses high spatial locality, which makes possible the parallel implementation on a memory-distributed machine using domain decomposition.

Third, from the viewpoint of the unstructured mesh solver itself, two types of mesh data are required: the finite element type of connectivity (or connectivity) and cell-based connectivity (or cell neighbor-identifying information). The equation solver often needs the former, while the latter is required by some particle method for particle tracking, such as DSMC [14] and PIC [17] for solving the Boltzmann equation with neutral and charged species, respectively. However, the past development of the AMR scheme only produced the finite element type of connectivity. Related cell neighbor-identifying information can of course be obtained by post-processing serially (not in parallel) the node-based connectivity, while it may become time-consuming if the resulting mesh size is large. Thus, an ideal PAMR scheme may be expected to directly generate both these two sets of data.

Previously, we developed an AMR module using the *h-refinement* scheme for an unstructured tetrahedral mesh [18] and applied it to simulate rarefied gas dynamics incorporation with a parallelized DSMC code. However, there are some potential problems with the further application of the module. *First*, it is a serial version which cannot be utilized for large-scale problems due to memory constraints. *Second*, due to some strategies used for mesh-quality control (will be introduced in Section 2), the number of final refined cells often becomes too large to handle for a realistic problem. *Third*, the mesh-refining process is not automatic since the user has to stop the running application program and refine the mesh by manually running the AMR module. In this paper, we intend to simultaneously solve these three problems.

Therefore, in the current study, a *cell-based* PAMR scheme for the three-dimensional unstructured tetrahedral mesh with a better mesh-quality control is proposed and tested. The paper begins with detailed descriptions of the PAMR scheme. The results of parallel performance are then presented, which are followed by the application of the PAMR to both particle-based (DSMC) and equation-based (Poisson–Boltzmann equation) codes. Finally, important conclusions of the current study are summarized at the end of this paper.

2. Parallel adaptive mesh refinement scheme

In this section, three major parts are described, including the basic algorithm for mesh refinement, the parallel implementation of mesh refinement, and the coupling of PAMR with other numerical schemes. In each part, a general description and related specific algorithms are respectively provided.

2.1. Basic algorithm for mesh refinement

2.1.1. General procedures

Considering the complexity of involved logic and data structure, we are only interested in refining the unstructured tetrahedral mesh. The general idea of refining the mesh in the current study is to simply "isotropically" refine the parent cells that require refinement into eight child cells. These child cells

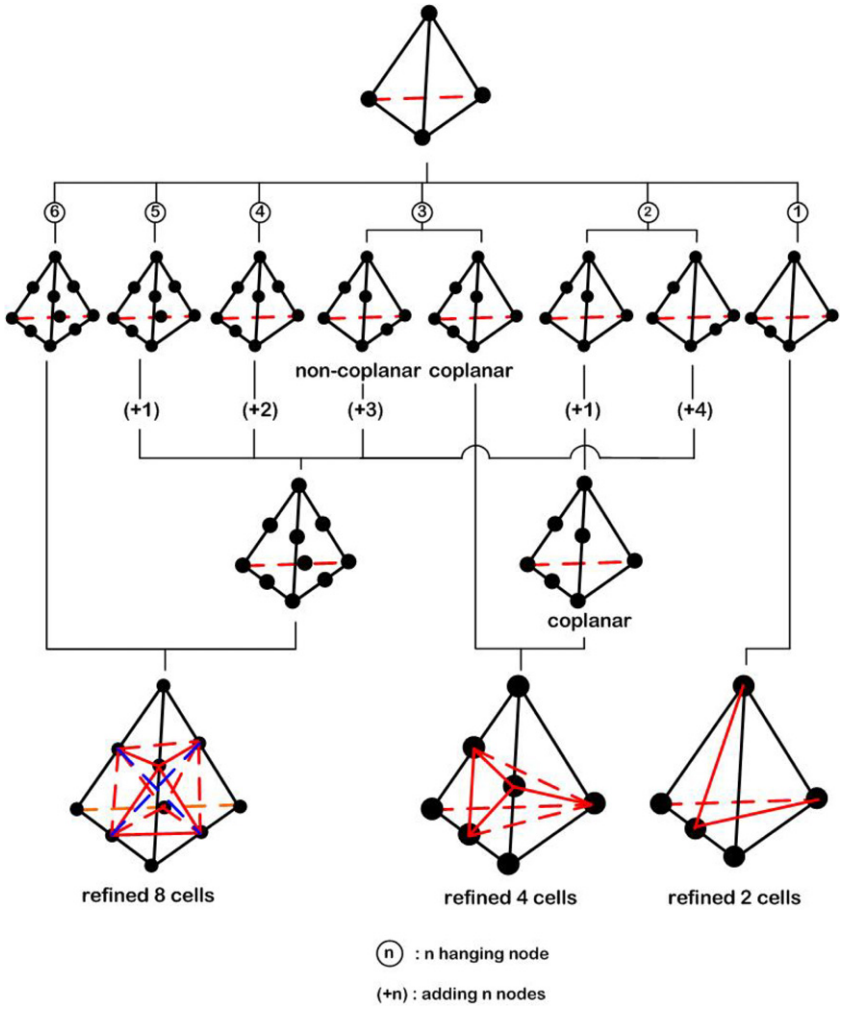


Fig. 1. Schematic diagram showing the procedures of removing hanging nodes for unstructured tetrahedral mesh.

from isotropic refinement are classified as "isotropic cells". In contrast, those cells other than the isotropic cells are then classified as "anisotropic cells". The important steps in refining cells can be summarized as follows. (1) Add a node on the cell edges of "isotropic" and "anisotropic" cells. (2) Refine cells that have added node(s) on the edge(s), and (3) update connectivity and cell neighbor-identifying information. Following the procedures in Fig. 1, the hanging nodes in the anisotropic cell that are adjacent to the isotropic cells are then removed by dividing it into two, four, or eight child cells while considering the cell-quality control mechanism which is introduced next. Note that this process of removing hanging nodes (Fig. 1) also applies to those cells after cell-quality control in all levels of refinement. Mesh refinement along this line is conceptually simple, while the practical implementation is rather involved. The details of the steps outlined above will be described later in the section that introduces the parallel implementation of mesh refinement.

2.1.2. *Cell neighboring connectivity*

In general, the AMR scheme [9–11] often applies the edge-based data structure in storing and searching connectivity-related data. With this edge-based data structure, the cells containing a common refined edge can be easily identified, which is necessary in the AMR scheme. However, the edge-based data structure is relatively *memory demanding*. For example, up to $\sim 7N$ edges can result from a tetrahedral mesh with N nodes, while only $\sim 6N$ cells are formed. With this edge-based data structure, $\sim$10 times of the number of edges are often required since up to $\sim$10 or more cells share a single edge in a typical tetrahedral mesh, while only four times of the number of the cells are required (four faces in a cell) if the cell-based data structure is employed. Thus, the overall memory saved in this regard can be up to 2–3 times if the cell-based data structure is utilized. This justifies the use of the cell neighbor-identifying array in the present study.

Since we do not utilize the edge-based data structure, how to quickly search the cells sharing a common edge becomes very important in the refinement procedure. The basic concept of the cell-based data structure is based on the fact that only two faces share a common edge in a cell, which can be used to efficiently sort the sharing faces. Fig. 2(a) shows a typical example of the common edge shared by the adjacent cells. Edge A-B is shared by Cell A-B-1-2, A-B-2-3, A-B-3-4, A-B-4-5, and A-B-5-1. For brevity, the procedure of sorting the faces sharing the common Edge A-B is used as an example to demonstrate the general

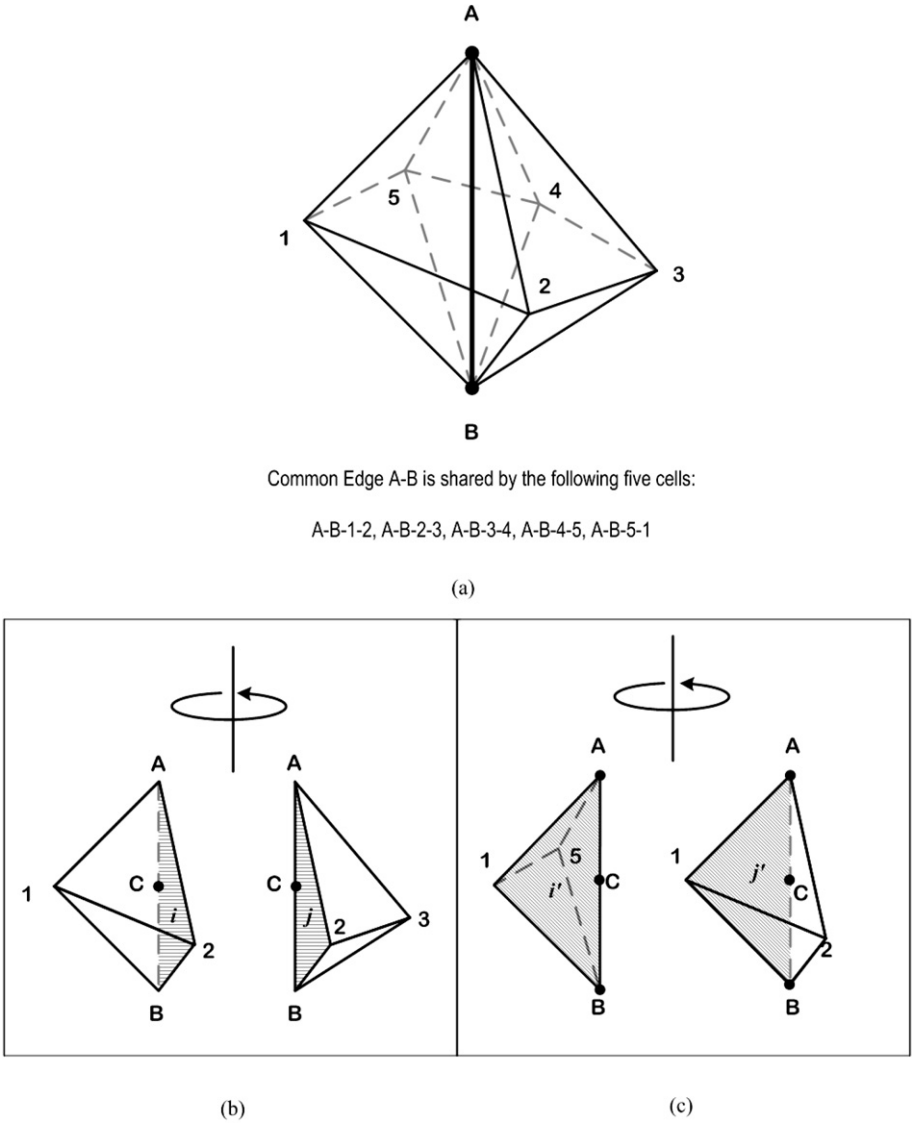


Fig. 2. Example of the common edge shared by the adjacent cells.

ideas of face sorting using cell-based data structure and is summarized as follows:

1. Define Cell A-B-1-2 as the initial searching cell.
2. Search the neighboring cells (A-B-2-3 and A-B-5-1) which share the common edge (Edge A-B) according to the cell neighboring-identifying array and finite element connectivity.
3. Select one of these two neighboring cells (e.g., Cell A-B-2-3 through Cell Interface A-B-2 shown in Fig. 2(b)) as the current searching cell.
4. Mark Edge A-B as a refined edge with local edge numbering in Cell A-B-2-3.
5. Search the next neighboring cell (Cell A-B-3-4 through Cell Interface A-B-3) which shares the common edge (Edge A-B) with Cell A-B-2-3 according to the cell neighboring-identifying array and finite element connectivity.
6. Assign the next neighboring cell as the current searching cell.
7. Repeat the similar process in Procedures 4–5 until the next neighboring cell is the same as the initial searching cell defined in Procedure 1. For example, as shown in Fig. 2(c), the neighboring cell of face i' of Cell A-B-1-5 (the same as the face j' of Cell A-B-1-2) is the Cell A-B-1-2, which is the defined initial searching cell.
8. For cases in which this cell-by-cell searching process meets the physical face boundary or IPB (Interior Processor Boundary), the other neighboring cell (e.g., A-B-5-1 through Cell Interface A-B-1) of the initial searching cell (Cell A-B-1-2) will be selected as the current searching cell. The algorithm will repeat the similar process in Procedures 4–5 until another physical face boundary or IPB is found in the process of identifying the cell group.

2.1.3. Cell-quality control

In the process of adaptive mesh refinement, a refined mesh which has a larger aspect ratio often appears in the final mesh distribution, especially in the "anisotropic" cells. Most mesh-smoothing schemes tend to change the structure of a given mesh to achieve the "smoothing effect" (a better aspect ratio) by rearranging the position of the nodes in the mesh. Changes made by a smoothing scheme, however, could possibly modify the desired distribution of mesh density produced by the AMR procedure. Additionally, the cost of performing a global mesh smoothing could be high. Alternatively and in a simpler way, it is possible to prevent or slow down the degradation of cell quality during a repeated adaptive refinement process. The present cell-quality control scheme classifies the cells based on how they will be refined. This allows us to avoid creating cells with large aspect ratios during the refinement. After identifying those cells, we can then refine them with a better refining strategy.

The basic idea of the proposed mesh-refining scheme is to simply prevent the further deterioration of cell quality while refining the cells that originate from previously "anisotropic" cells. The proposed procedures of the cell-quality control scheme are shown schematically in Fig. 3 along with the corresponding procedures without cell-quality control. The logic looks quite complicated at first glance; however, it is conceptually simple in reality. A general rule of thumb is to identify if the hanging nodes occur at the "normal edge" or "short edge" of the cell. Note that the "normal edge" is defined as one of the six edges of an isotropic cell, while the "short edge" results from the bisection of a "normal edge" in a previous level of refinement. If at least one of the hanging nodes occurs at the short edge, special treatment such as that in Fig. 3 is required to prevent the further deterioration of cell quality in later mesh refinement. Otherwise, the procedure in Fig. 1 applies directly.

In the data structure, we need to record each previously refined cell with an index showing if it is an "isotropic" cell or an "anisotropic" cell (two or four refined cells, respectively). This can then be used for cell-quality control purposes as shown in Figs. 3(a), 3(b). The latter figures show the corresponding process of refining the cell if the parent cell originates from an "anisotropic" cell due to two or four refined cells, respectively. A previously developed cell-quality control scheme (Fig. 3 in Ref. [18]) simply isotropically refined such a cell into eight child cells even if it originates from an anisotropic cell. This rule is simple in implementation; however, it often creates too many isotropic child cells, which tremendously increases the computational load of the solver. For example, in the present implementation as shown in Fig. 4, the original hanging node "8" is directly removed by adding two additional nodes (A and B) to form four child cells, instead of two child cells (1-4-8-7 and 5-4-8-7) from the parent cell 1-5-7-4. The present proposed cell-quality control scheme produces refined cells with a better aspect ratio as compared to the "isotropic" refinement presented previously [18]. Tests verifying the superiority of this new cell-quality control scheme will be shown in a later section.

2.1.4. Surface cell refinement

If the boundary surface of the computational domain is not planar, it is not sufficient to place the added nodes directly on the edge (called "boundary edge" hereafter) of the parent-cell face that belongs to the boundary. Should the refined nodes be placed directly on the boundary edges, this may result in the rough piecewise representation of the original boundary surface. What must be done is to move the position of the added nodes onto the real boundary contour surface. In the present implementation, it is assumed that the boundary surface can be represented in a parametric format (a second order polynomial) in several segments specified by the user. Special cell neighbor identifiers are assigned to these curved boundary cells to distinguish them from straight boundary cells within the computational domain. Whenever the boundary cell which requires refinement is identified as a curved boundary cell, the corresponding parametric function representing the true surface contour is called in to locate the correct node positions along the parametric surface. A typical example of refining the curved boundary surface will be shown in a later section.

2.2. Parallel implementation of mesh refinement

2.2.1. General procedures

Fig. 5 shows the proposed overall procedure for parallel mesh refinement. Basically, the procedures are similar to those presented earlier for serial mesh refinement. However, the detailed procedures and related data structure presented in Fig. 5 are much more complicated than those in the serial mesh refinement because of the parallel processing with domain decomposition. Each spatial subdomain belongs to a specific processor in practice. The algorithm is implemented in the SPMD paradigm (master-slave style) on a memory-distributed parallel machine using MPI as the data communication protocol. The overall procedure, as shown in Fig. 5, can be summarized as follows:

1. Preprocess the input data at the host processor. The detailed preprocessing tasks are explained in detail as follows:
   - 1a. *Initialize MPI.*
   - 1b. *Synchronize all processors.*
   - 1c. *Read the grid data, the solution distribution from the previous mesh, and the refinement criteria in the host processor.* The grid data include the nodal coordinates and the connectivity between nodes and cells. If it is beyond level-0 refinement, then the cell-neighboring identifying array is also read. Solution distribution and refinement criteria are used to determine the proper domain decomposition using the graph-partitioning technique [19] while considering the weighting distribution at graph vertices. For example, the weight of "8" is assigned at the cell center (graph vertex) if the cell will be refined "isotropically"; otherwise, the weight of "1" is assigned at the cell center. This will relieve the load-unbalancing problem often encountered in the PAMR scheme.
   - 1d. *Convert the connectivity data into neighbor-identifying array* in the host processor. Note that **Step 1d** is con-

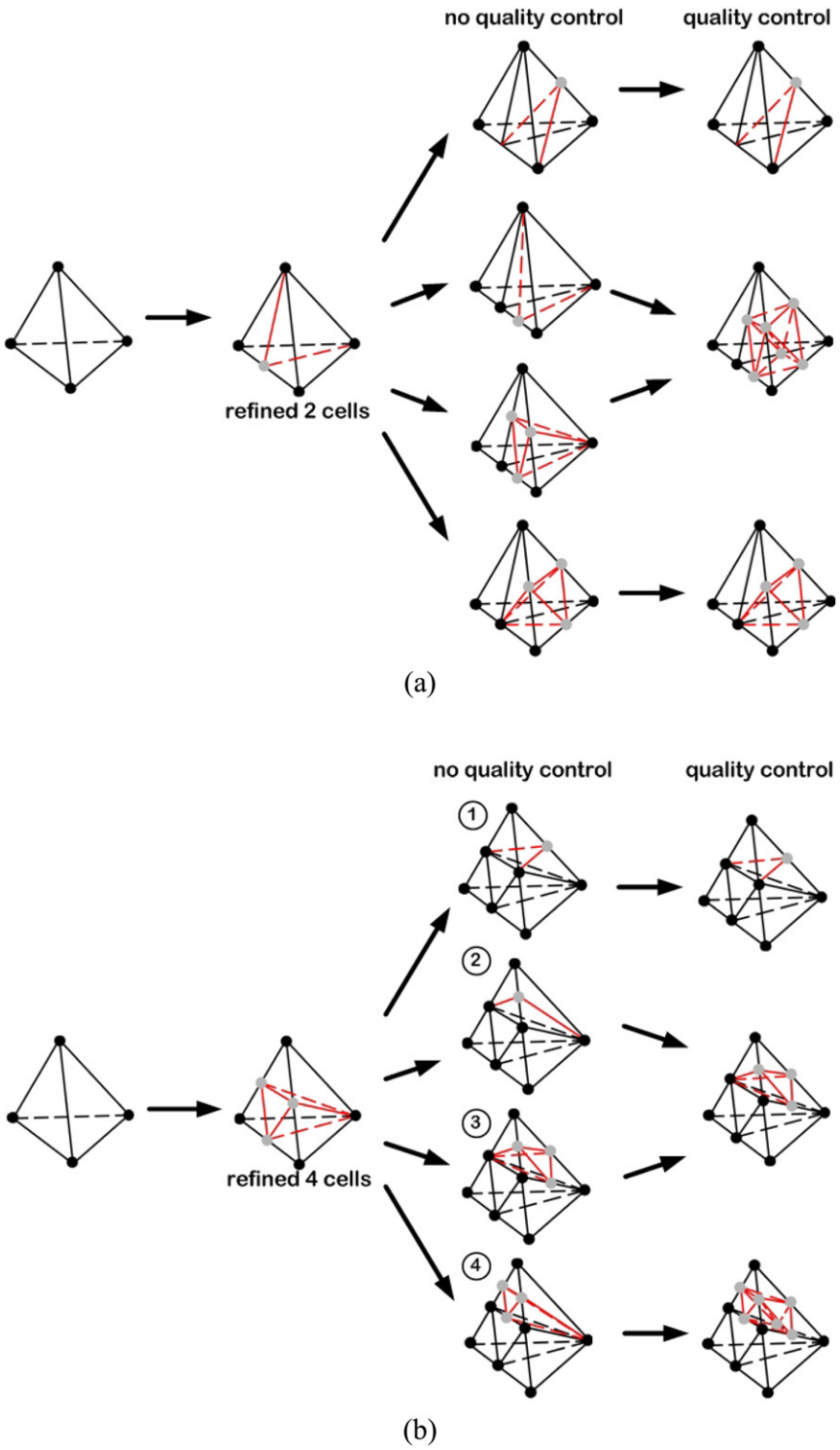


Fig. 3. Schematic diagram showing the procedures of cell-quality control during mesh refinement for unstructured tetrahedral mesh.

ducted only at the level-0 mesh refinement since at further levels, the neighbor-identifying array is available from a previous level of refinement and is read in **Step 1c**. The neighbor-identifying array, whose value is the global cell number, defines what the neighboring cells are across the faces of each cell. It is then used to quickly identify the common edge which is defined by two unique global node numbers. Note that this common edge is shared by many neighboring cells in a 3-D tetrahedral mesh. With this information, there is no need to define the edge-based array, which is often done in other PAMR studies [9–11]. In addition, it is necessary to have this neighbor-identifying array for cell-by-cell particle tracing in a particle-based method, such as the DSMC [18] and the PIC [20] methods.

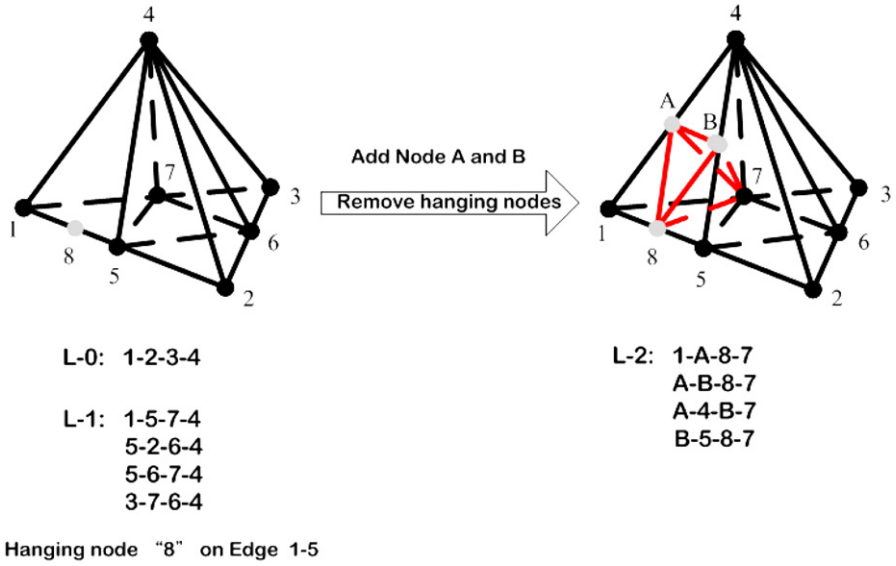


Fig. 4. A typical example for better cell-quality control during refinement.

1e. *Distribute the necessary information to all processors*. Once each processor receives the information, a node-mapping array that defines the conversion of the local to the global node number is defined for all local nodes in each processor. To save memory, the local node and cell numbers range from "one" to the maximum number of nodes and cells in the local processor, respectively. A similar cell-mapping array is also defined in this step. The abovementioned node- and cell-mapping arrays are built mainly to access the grid data in each processor. From **Step 2** and thereon, the procedures become the same for all processors, while the host processor is also used for final data gathering and outputting to files as shown next. Whenever the global data structure needs to be modified, communication among processors becomes inevitable.

2. *Index those cells, in which refinement is necessary, based on the refinement criteria*. The refinement criteria are decided by the type of numerical solvers that can be either of equation or particle type.
3. *Check if further mesh refinement is necessary, and if it is, proceed to the next step. If it is not, proceed to* **Step 9**.
4. *Add new nodes into those cells that required refinement* (Module I in Fig. 5). These cells may include those which are flagged to be refined "isotropically" and those next to the isotropic cells ("anisotropic"). The details of the implementation as shown schematically in Fig. 6 are described as follows:
   4a. *Add new nodes onto all the edges of the isotropic cells*.
   4b. *Add new nodes into the anisotropic cells*, which may require further refinement, as decided in the following steps, in order to remove the hanging nodes created earlier. Note that the node number for each newly added node in **Step 4a** and **Step 4b** is assigned increasingly starting from the original maximum local node number in each processor.
   4c. *Communicate the hanging node data to the corresponding neighboring processor if the hanging nodes are located at IPB*. For each processor, if the hanging node data are received from other processors at this stage, return to **Step 4b** to update the node-related data (local numbering, local connectivity, and coordinates). If no hanging node data are received at this stage, proceed to the next step.
   4d. *Remove the hanging nodes following the procedures shown in Fig. 1*. Return to **Step 4b** if further node addition is necessary in this step. Proceed to the next step if no further node addition is required.
5. *Unify the global node and cell numberings due to the new added nodes among all processors* (Module II in Fig. 5). Note that the global and local node numberings for all original nodes before refinement are kept the same after the refinement. The procedure for this step is explained in detail as follows:
   5a. *Add up the number of newly added nodes in each processor, excluding those located at IPBs*.
   5b. *Gather these numbers from all other processors, add them up to obtain the updated total number of nodes, and distribute it to all other processors*. Note that this updated total number of nodes includes old and new nodes, but excludes the newly added nodes at IPBs. These newly added nodes at IPBs require further treatment in a later step.
   5c. *Build up the updated node- and corresponding cell-mapping arrays for those newly added nodes in the interior part of each subdomain based on the results in* **Step 5b**.
   5d. *Communicate the data of newly added nodes at IPBs among all processors*.
   5e. *Build up the node-mapping array for the received new nodes at IPBs in each processor*.

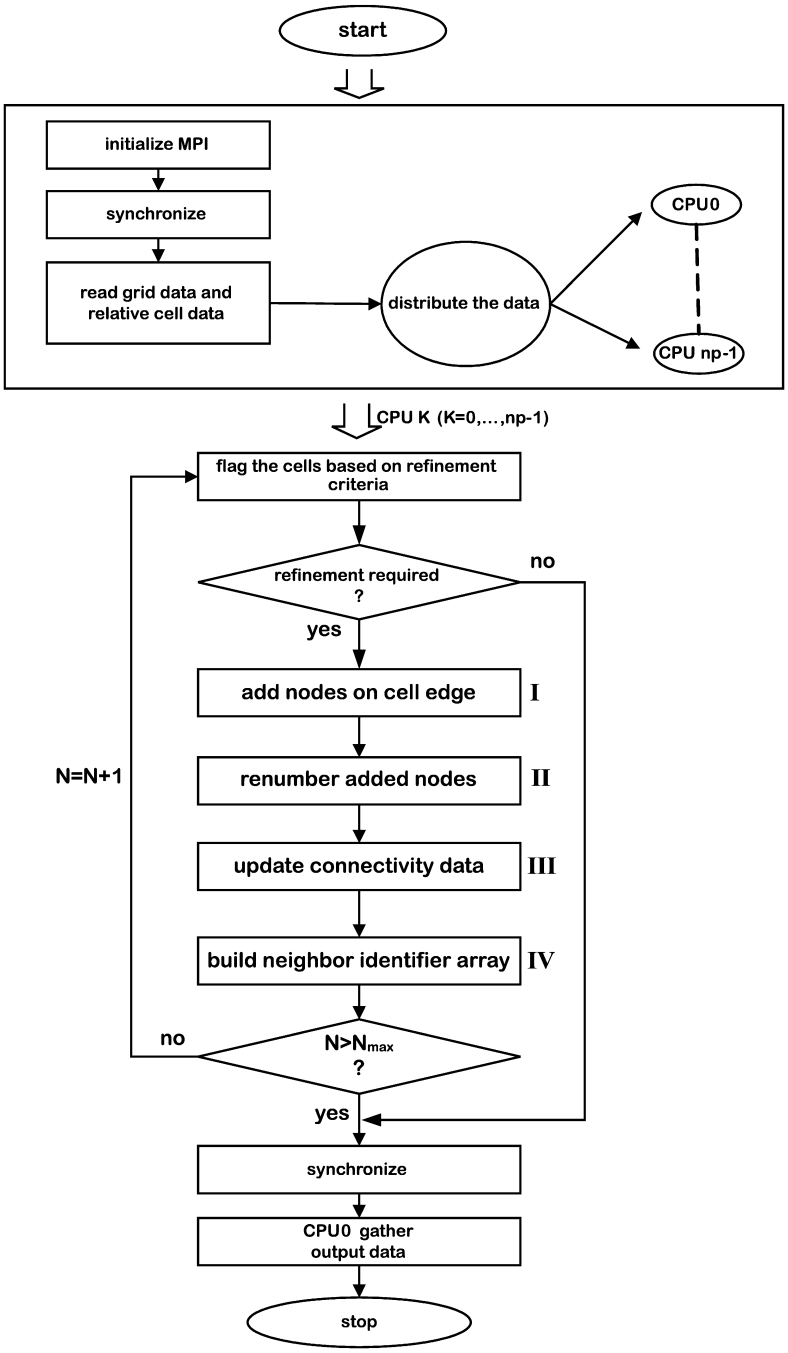


Fig. 5. Proposed algorithm of parallel mesh refinement for unstructured mesh refinement.

6. *Build up new connectivity data for all cells due to the newly added nodes* (Module III in Fig. 5).
7. *Build up the new cell neighbor-identifying array based on the newly obtained connectivity data from* **Step 6** (Module IV in Fig. 5). The procedure for this step is explained in detail as follows:
   - 7a. *Reset the cell neighbor-identifying array*.
   - 7b. *Build up the cell neighbor-identifying arrays for all cells based on the new connectivity data*, excluding the data associated with the faces lying on the IPBs that require the updated information of the global cell number, which has not communicated at this stage.
   - 7c. *Record all the cell neighbor-identifying arrays that are not rebuilt in* **Step 7b**.
   - 7d. *Broadcast all the recorded data in* **Step 7c** *to all processors*.
   - 7e. *Build up the cell neighbor-identifying arrays on the IPBs considering the overall connectivity data structure*.
8. *Decide if it reaches the preset maximum number of refinement levels*. If it does, proceed to the next step; otherwise, return to **Step 3**.
9. *Synchronize all processors*.
10. *The host processor gathers all the data and outputs them*.

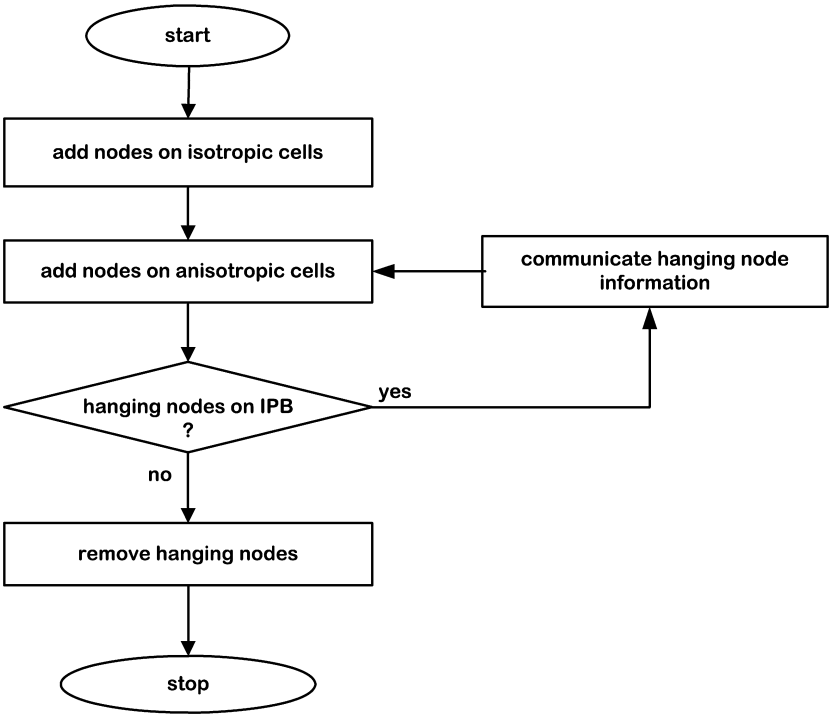


Fig. 6. Proposed algorithm for adding nodes in the cells that require refinement.

In the practical application of coupling the PAMR with other parallel numerical solvers in **Step 3**, the maximum number of refinement levels is set to "one" since the control of whether further refinement is necessary is instead decided upon outside the PAMR, as can be seen in the next section.

2.3. Coupling of PAMR with other parallel numerical solvers

The PAMR scheme presented in the previous section can be easily coupled with any parallel numerical solver that utilizes a 3-D unstructured tetrahedral mesh and MPI for data communication protocol. One can readily wrap up the PAMR as a library and insert it into the source code of any parallel numerical solver one intends to use. However, there exist some possible problems which may occur due to memory conflicts between the inserted library and the numerical solver itself, and which could reduce the problem size one can handle in practice. Instead, a simple coupling procedure as shown schematically in Fig. 7, which is written in a shell script language standard on all Unix-like systems is prepared to link the PAMR and any parallel numerical solver. In the sketch, $N_{\max}$ represents the maximum number of refinement levels preset by the user. The satisfaction of the refinement criteria can be identified by simply checking the status of the index of refinement in each cell obtained from the numerical solver. The choice of refinement criteria depends upon what type of numerical solver is used, and the physical problem it is concerned with. For example, in a particle method such as DSMC, the cell size has to be smaller than the local mean free path that is a function of temperature, number density, and type of species under consideration. Before entering the PAMR module, the mesh is repartitioned based on a new distribution of the weight factors of cells, in which those cells flagged for refinement and are based on problem-dependent mesh-refining criteria, are set as eight, while the others are kept as unity. With this repartition, the workload among processors is approximately balanced during the mesh-refining process. Finally, the adaptively refined mesh is read into the numerical solver for a better solution. The abovementioned procedures are repeated until either the level of refinement exceeds the preset maximum level, or no further mesh refinement is required. One of the immediate advantages of this is that the source codes can be kept intact without any changes. Indeed, it is especially justified if only a steady state of the physical problem is sought, in which normally, only several times of mesh refinement is enough to have a fairly satisfying resolution of the solution. Thus, the total I/O time in switching between two codes, in proportion to the number of couplings, can be reduced to a minimum in practical applications.

3. Results and discussions

The completed PAMR code is tested on a PC-cluster system which is termed as "Cahaba" at the University of Alabama, Birmingham (http://www.eng.uab.edu/me/etlab/cluster.htm#resources). This system is configured in a master–slave network architecture with the following features: 64 dual-processor nodes, 2.4-GHz Xeons for each processor, 2-GB RAM at least for each node, and GB-Ethernet for networking. This PAMR code is expected to have high portability across various parallel machines if they are memory distributed and if they use MPI as the communication protocol. The corresponding parallel performance of PAMR on this machine, and its applications to the parallel direct simulation Monte Carlo method (particle method) and the parallel Poisson–Boltzmann

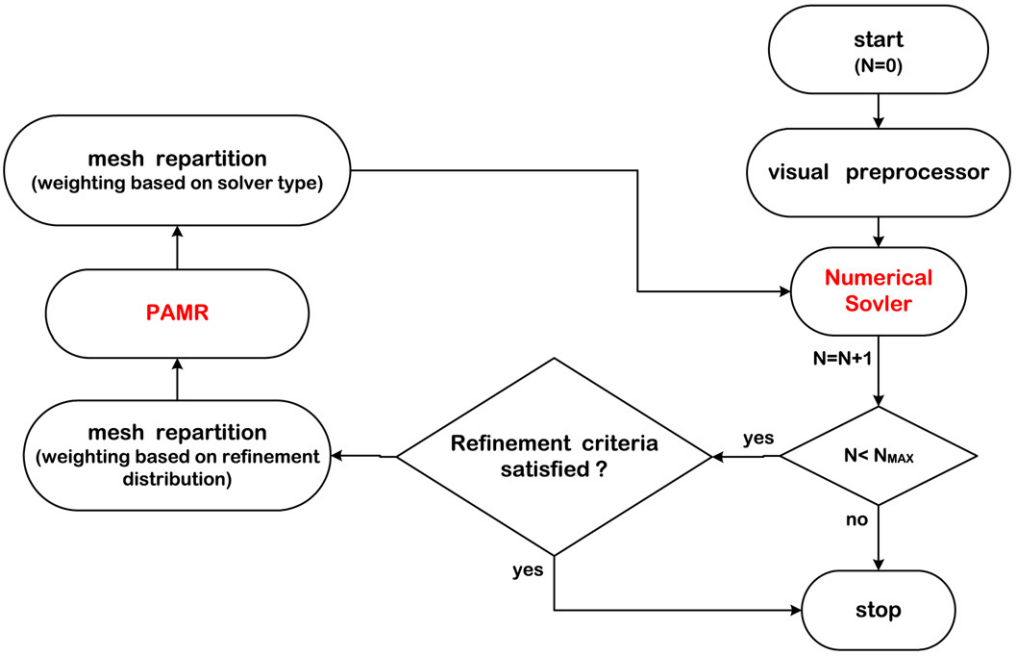


Fig. 7. Proposed algorithm for coupling the PAMR with any parallel numerical solver.

Table 1
Flow conditions for 70° blunt body hypersonic flow with attack angle of 10° ($Kn_\infty = 0.0108$) in SR3 wind tunnel

Ma_∞	Kn_∞	Re_∞	T_∞ (K)	V_∞ (m/s)
20	0.0108	4,116	13.6	1,502

equation solver (equation-based method) are discussed in the following.

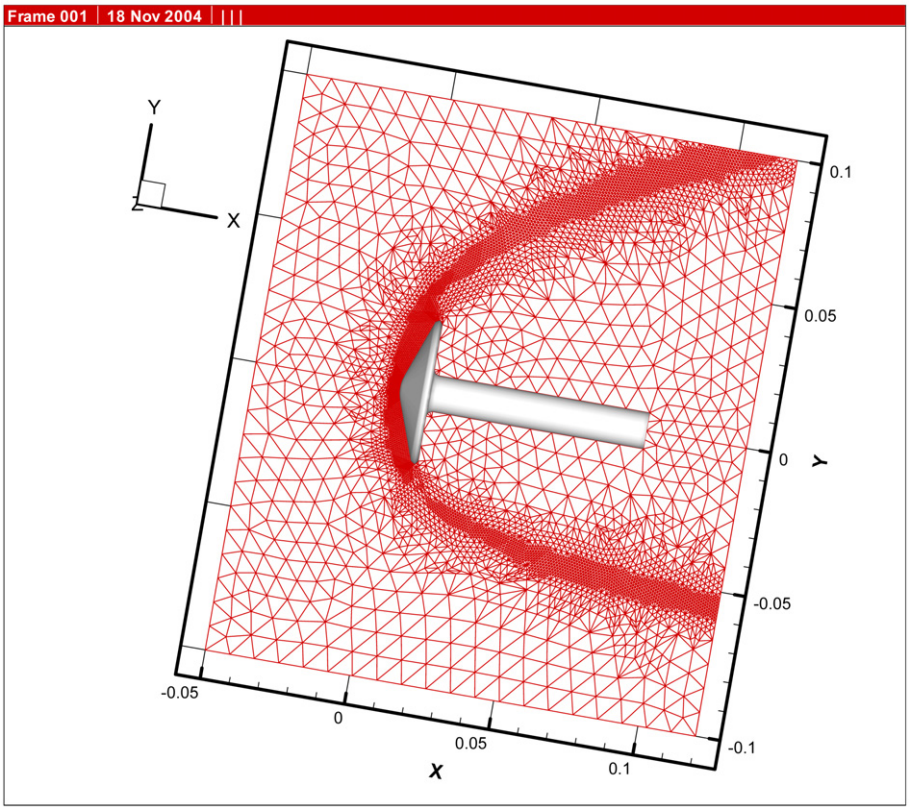
3.1. Parallel performance of PAMR

The parallel performance of the PAMR is studied using a refined mesh generated during the process of AMR (after 3 AMRs) for a hypersonic nitrogen flow past a 70° blunt cone using the DSMC method. The flow conditions are summarized in Table 1. Before the fourth PAMR, the original mesh (level-3) has 1,070,194 tetrahedral cells and 187,649 nodes (Fig. 8(a)), while after the fourth PAMR, the amount increases to 2,701,178 cells and 468,944 nodes (Fig. 8(b)). A test using this fairly large size of mesh lies mainly on the fact that the current PAMR is designed to handle a large mesh size.

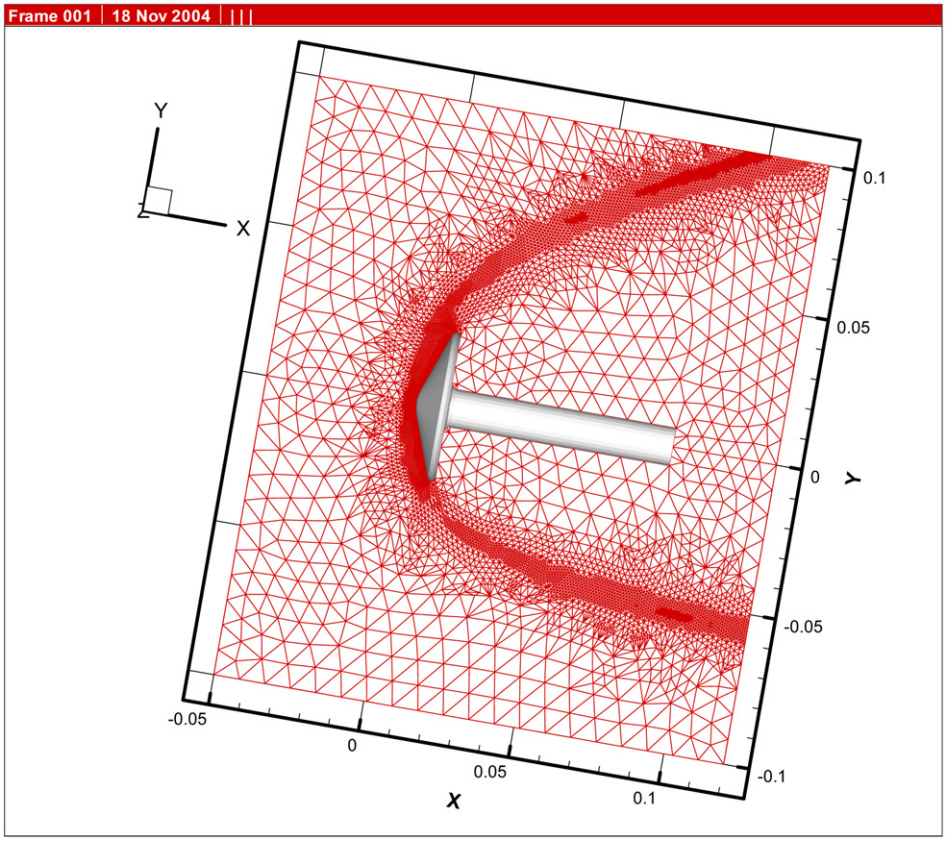
Resulting timings (in seconds) for different components of the PAMR using different numbers of processors (up to 64), respectively, are listed in Table 2. It is clearly shown that the timing for the preprocessing module increases slightly with the increasing number of processors, although the increase seems to level off as the processor number $\geqslant 32$. This mild increase can be attributed to the initial data distribution from the host processor to the increasing number of slave processors. The advantages of the parallel implementation of mesh refinement can be clearly seen from modules I, II, and IV, of which the timings reduce dramatically with the increasing number of processors. For example, as the number of processors increases from 2 to 32, the timing for module I, II, and IV reduces to approximately 61.4, 95.9, and 9.3 times, respectively, while it increases to 2.4 times for module III. The corresponding overall computational time decreases 29.7 times (from 1103 s to 37.1 s). Nevertheless, the timing levels off as the number of processors increases up to 64 due to the increase of communication among processors. In addition, the total computational time decreases even more to 62 times (from 1090.5 s to 17.6 s) if the preprocessing time is excluded. From our experience, reading the grid data (**Step 1c**) takes most of the time in the preprocessing module. Thus, the preprocessing time can be greatly reduced if the parallel processing of **Step 1c** is implemented, although we have not done so in the present study. The resulting increase of computational speed, excluding the preprocessing, is approximately scaled as $N^{1.5}$ for $N_{\text{proc}} \leqslant 32$ which is surprisingly good. This high speedup is obviously due to the dramatic decrease of the time spent in Module II, which is the most time-consuming part of the mesh-refining algorithm. The reasons for this rapid decrease of processing time in Module II (**Steps 5a–5e**) include a possible cache miss of the superscalar machine due to the large problem size and a highly localized algorithm in renumbering the added nodes, except in **Steps 5b** and **5d** which require moderate communication among processors.

3.2. Cell-quality control scheme

The proposed cell-quality control mechanism (Fig. 3) is tested by simulating a supersonic argon gas flow past a sphere ($Kn_\infty = 0.1035$) using the DSMC method. Only 1/16 of the whole domain is taken for simulation because of the axial symmetry in this problem. Fig. 9(a) shows a typical mesh distribution without cell-quality control, while Fig. 9(b) shows the mesh distribution with the present cell-quality control scheme

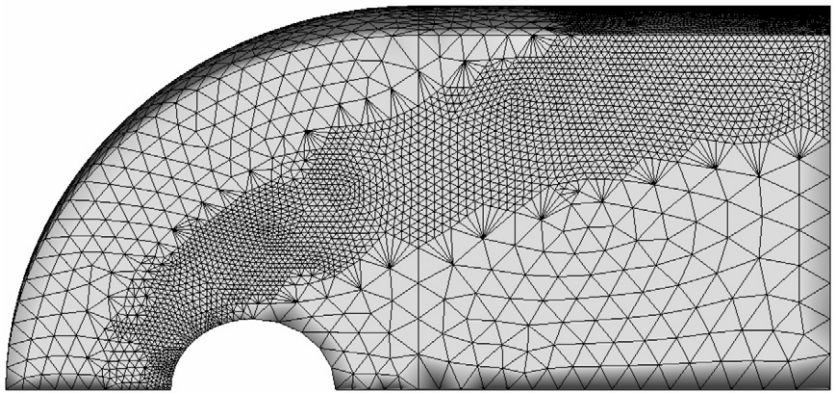


(a)

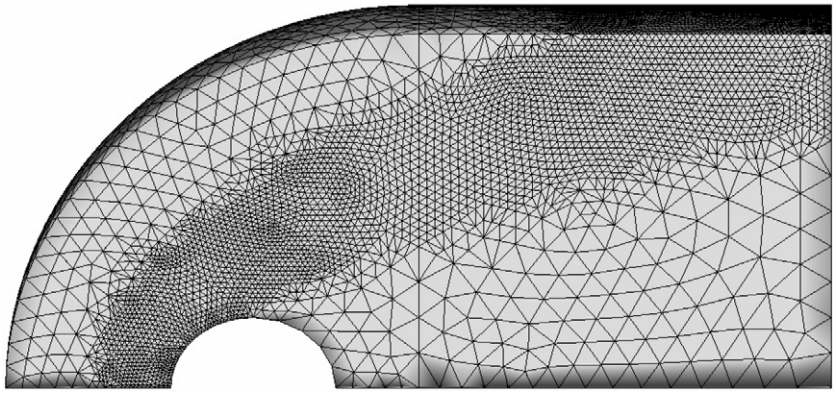


(b)

Fig. 8. Surface mesh distribution of a hypersonic flow over 70° blunt body with angle of attack 10°. (a) Original (level-3) mesh; (b) level-4 refined mesh.



(a)



(b)

Fig. 9. Surface mesh distribution for a supersonic flow past a sphere ($M_\infty = 4.2$, $Kn_\infty = 0.103$, argon gas). (a) With cell quality control, (b) without cell quality control.

Table 2
Timing (seconds) of PAMR module for different processor numbers (Cahaba)

Processor No.	2	4	8	16	32	64
Preprocessing	12.5	13.5	14.4	15.4	19.5	18.2
I	92.1	24.8	6.2	2.0	1.5	2.4
II	977.9	267.4	68.5	23.2	10.2	9.9
III	0.9	1.8	2.0	3.3	3.8	6.9
IV	19.5	9.7	5.4	2.9	2.1	2.4
Total time	1103.0	317.2	96.5	46.7	37.1	39.8

I. Add nodes on cell edges. II. Renumber added nodes. III. Update connectivity data. IV. Build neighbor identifier array.
Mesh refinement test for a hypersonic flow over 70° blunt cone: Original mesh: 1,070,194 cells and 187,649 nodes. Level-1 refined mesh: 2,701,178 cells and 468,944 nodes.

using the same initial mesh. It is clear that the present cell-quality control scheme can effectively reduce the deterioration of the cell quality in the region of "anisotropic" cells. In addition, the number of cells after PAMR with cell-quality control is slightly larger than that after PAMR without cell-quality control.

3.3. Numerical examples

In the following, two numerical examples coupling with the PAMR, including the DSMC method and the Poisson–Boltzmann equation solver, are presented in turn. The underlying physics behind the two numerical examples is not discussed in detail since in the present study, we are only interested in demonstrating the possible applications of the proposed PAMR.

3.3.1. Parallel direct simulation Monte Carlo method

The particle method, which is a direct simulation Monte Carlo (DSMC) method developed by Bird [14], is a standard tool for solving the Boltzmann equation with neutral rarefied gas species. The important steps of the DSMC method include the following: particle moving to a new configuration space, particle sorting into the cell, particle collision to find out the new position in velocity space, and then finally, particle sampling to obtain the macroscopic properties. The details can be found in the study of Bird [14] and are not repeated here for the sake of brevity.

Table 3
Evolution of simulation conditions at different refinement levels using PAMR for 70° blunt body hypersonic ($\alpha = 10°$, $Kn_\infty = 0.0108$)

Level	Cell No.	Particle No.	Sampling time steps	Reference time step (sec)	Transient time steps
0	15,190	~270,000	18,000	1.88E−07	4000
1	43,624	~560,000	18,000	1.55E−07	4000
2	229,040	~5,880,000	18,000	6.02E−08	4000
3	1,070,194	~11,100,000	18,000	2.41E−08	8000
4	2,700,319	~18,100,000	17,000	1.98E−08	10,000

The previously developed parallel direct simulation Monte Carlo (PDSC) in our group [18,21] is coupled with the present PAMR to simulate the flow field of a hypersonic argon gas flow past a 70° blunt cone ($M_\infty = 20$, angle of attack $\alpha = 10°$, free-stream Knudsen number $Kn_\infty = 0.0108$, constant wall temperature of blunt cone $T_w = 300$ K) in the SR3 wind tunnel [22–25], as shown in Fig. 10. The important features of PDSC include parallel implementation using dynamic domain decomposition, a spatially variable time step, a conservative weighting scheme for treating flows with trace species, among others. The detailed dimensions of the blunt cone can be found in [21–24], while the other important flow conditions are briefly summarized in Table 1, as mentioned in the previous section. A semi-cylindrical region enclosing the blunt cone is used as the computational domain considering the symmetry of the physical problem. In addition, a free-stream parameter $\beta_{fr} = 1.05$ [18,26] which is a threshold value of the ratio of local number density to free-stream value, below which a mesh is not refined, is used to reduce the number of cells in the free stream. In the present simulation, both the variable time-step scheme [18] and dynamic domain decomposition [21] are used to reduce the computational time.

The evolution of simulation conditions at different levels of refinement are summarized in Table 3. The number of cells increases from ~15,000 to ~2.7 millions (level 4, Fig. 8(b)), while the number of particles increases from ~270,000 (initial) to 18.1 millions (level 4). The reference time-step size decreases from 1.88E−07 seconds (initial) to 1.98E−08 seconds (level 4), while the total number of sampling time steps is kept at the same value of 18,000. Note that the reference time step spatially represents the smallest time step, which often occurs near the stagnation region with the highest density and the smallest cell size. With roughly ~5–10 particles per cell in most of the regions, this sample size should be good enough to obtain the macroscopic quantities with acceptable statistical uncertainties. In addition, the number of transient time steps in the DSMC computation using the level-4 mesh is extended to 10,000 to ensure that the sampling starts without any transient effect. To reduce the transient time of the simulation at each level of mesh, the solution obtained from the previous level of mesh can be interpolated into the new mesh, although we did not do so in the present study.

Figs. 11–13 respectively illustrate the distribution of normalized density, total temperature, and Mach number in the symmetric plane using a level-4 refined mesh. It is clear that a rather strong bow shock is formed around the blunt cone. Note that in Fig. 11, the streamlines of the flow field are also plotted

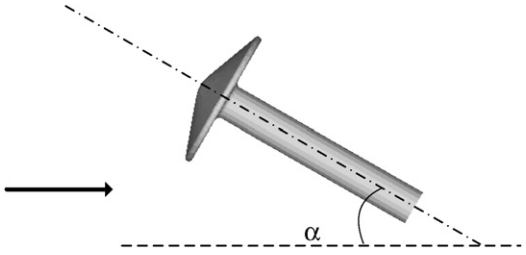


Fig. 10. Sketch of a hypersonic flow over 70° blunt body with angle of attack (Nitrogen gas, $Ma = 20$, $T_\infty = 13.6$ K, $T_w = 300$ K, $\alpha = 10°$, $Kn_\infty = 0.0108$).

Table 4
Aerodynamics coefficients with different refinement level mesh for 70° blunt body hypersonic flow with attack angle of 10° ($Kn_\infty = 0.0108$)

Level	C_D	C_L	C_m
0	1.667 (12.92%)	−0.141 (21.93%)	−0.057 (5.63%)
1	1.621 (9.85%)	−0.154 (14.47%)	−0.058 (7.45%)
2	1.582 (7.16%)	−0.166 (7.69%)	−0.056 (4.59%)
3	1.552 (5.14%)	−0.174 (3.17%)	−0.057 (5.23%)
4	1.543 (4.58%)	−0.182 (0.28%)	−0.055 (0.98%)
Exp. data	1.476	−0.18	−0.054

Value in the parenthesis represents the deviation of simulation with experimental data.

along with the normalized density distribution. Fig. 11 shows that the density increases rapidly across the shock, especially in the region near the nose of the cone ($n/n_\infty > 18$), while a highly rarefied wake region is formed ($n/n_\infty < 0.1$) behind the cone body with a strong recirculation. The resulting maximum density ratio in the flow field is very large ($n_{max}/n_{min} > 180$), which necessitates the use of PAMR. Fig. 12 shows that the total temperature increases up to 76 times of the free-stream value after the bow shock in front of the nose area. In all the regions after the shock, the temperatures are much greater than the free-stream value. Fig. 13 shows that a subsonic flow region is formed around the cone surface due to the nearly normal shock, while the other regions are all supersonic, except in the wake region.

The evolution of the computed aerodynamic coefficients, including C_D (drag), C_L (lift), and C_m (moment), at different levels of mesh refinement along with the experimental data are summarized in Table 4. The results using the level-4 mesh clearly show that C_D, C_L, and C_m agree with the experimental data by 4.58%, 0.28%, and 0.98%, respectively. The deviation of the computed C_D from the experiment seems quite large; however, it is within the experimental uncertainties [24]. Nevertheless, the use of the PAMR does systematically improve the

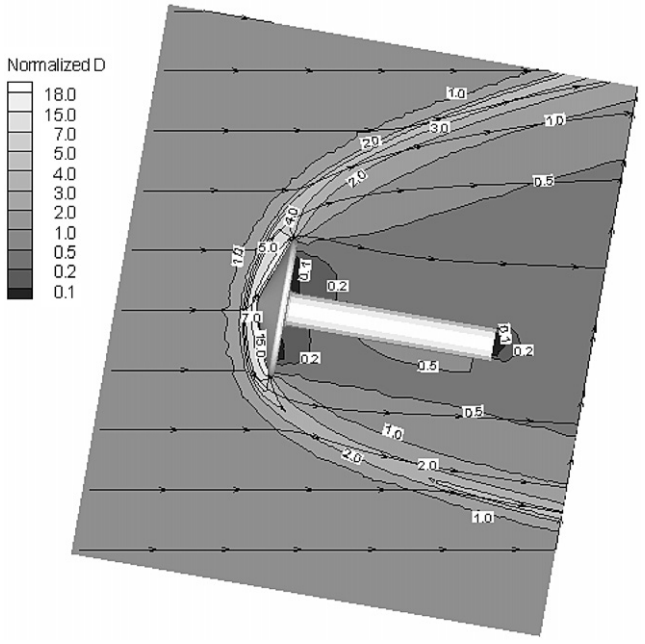


Fig. 11. Normalized density distribution of a hypersonic flow over 70° blunt body with angle of attack 10° with level-4 refined mesh.

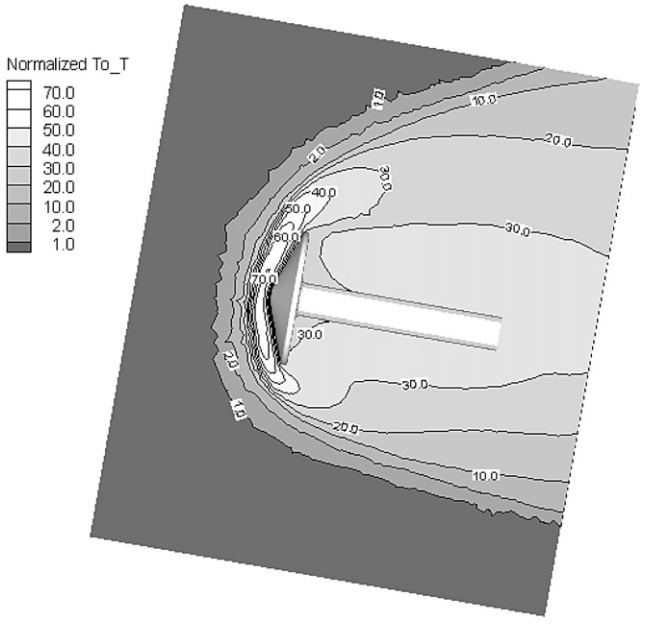


Fig. 12. Normalized total temperature distribution of a hypersonic flow over 70° blunt body with angle of attack 10° with level-4 refined mesh.

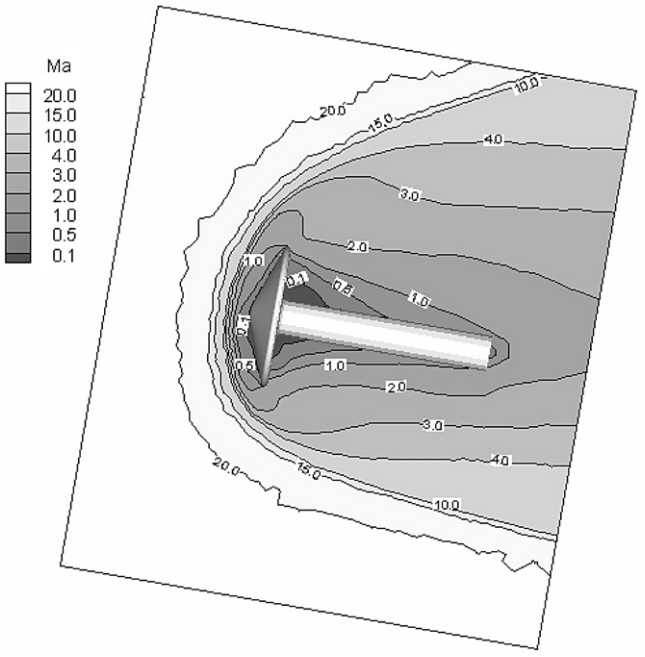


Fig. 13. *Ma* number of a hypersonic flow over 70° blunt body with angle of attack 10° with level-4 refined mesh.

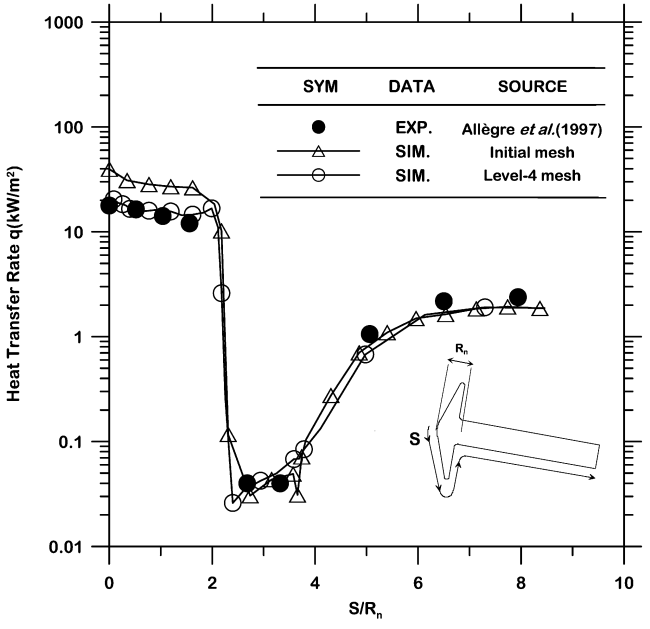


Fig. 14. Heat transfer rate along the specific path S on 70° blunt body surface with different meshes. (R_n: the nose radius of blunt body).

solution accuracy of DSMC. Furthermore, Fig. 14 shows the occurrence of heat transfer along a specified surface line using the initial mesh and level-4 mesh. S is the distance along a specific path on 70° blunt body surface while R_n is the nose radius of the blunt cone. In general, both results show similar tends. Due to the impact of hypersonic flow molecules, an intensely heated region around the nose of the blunt cone is formed. Heat transfer decreases steeply after the shoulders of the blunt cone, and then increases to a relatively low value as compared to that of the nose region. However, the result of using the initial mesh overpredicates (2–3 times) the experimental heat flex around the nose region, while the results using the level-4 mesh agree excellently with the experimental data [22,25].

3.3.2. Parallel Poisson–Boltzmann equation solver

The Poisson–Boltzmann equation plays an important role in describing many physical phenomena, including like-charge particle interaction [27,28], particle–membrane interaction [28,29], and electrokinetic flow [30], to name a few. Thus, its accurate numerical modeling can provide important infor-

mation on effective interaction among the charged surfaces. One of the most important things in modeling these phenomena is to accurately resolve the electrical double layer near a charged surface that is often very thin. In the following, we will briefly describe the parallel Poisson–Boltzmann equation with the finite-element discretization and simulation results by coupling this solver with the current PAMR. The details of the implementation of the parallel Poisson–Boltzmann equation solver could most likely be found elsewhere in the near future.

Consider any fluid consisting of two types of ions with equal, opposite charges ($Z = Z^+ = Z^-$) and equal concentration ($n = n^+ = n^-$), and then the corresponding dimensionless three-dimensional Poisson's equation for electrostatic potential $\Psi(x, y, z)$, assuming Boltzmann relation at equilibrium, can be written as

$$\frac{\partial^2 \bar{\psi}}{\partial \bar{x}^2} + \frac{\partial^2 \bar{\psi}}{\partial \bar{y}^2} + \frac{\partial^2 \bar{\psi}}{\partial \bar{z}^2} = \sinh(\bar{\psi}) \tag{1}$$

where

$$\bar{x} = \frac{x}{\kappa^2}, \quad \bar{y} = \frac{y}{\kappa^2}, \quad \bar{z} = \frac{z}{\kappa^2}, \quad \bar{\psi} = \frac{ze\psi}{k_b T},$$
$$\kappa^2 = 2n_0 Z^2 e^2 / \varepsilon \varepsilon_0 k_b T,$$

with ε and ε_0 as the permittivity ratio and vacuum permittivity, respectively. Note that κ^2 is the so-called Debye–Huckel parameter, and its inverse represents the characteristic length of the electrical double layer.

By applying the Galerkin finite-element method to Eq. (1) in a typical tetrahedral element with a linear shape function to form an element equation, and by subsequently assembling all the element equations throughout the computational domain, the system matrix equation becomes

$$[K]\{U\} = \{F(U)\} \tag{2}$$

where $[K]$ is the coefficient matrix resulting from the Laplacian operator using Gaussian quadrature in the volume integration, $\{U\}$ is the solution vector, and $\{F(U)\}$ is the nonlinear loading vector resulting from the integration of the source term of Eq. (1) in the FE formulation. This nonlinear algebraic system equation is further treated using the Newton scheme, and it becomes

$$[K]\{U^{k+1}\} = \{F(U^k)\} + [J]^k(\{U^{k+1}\} - \{U^k\}) \tag{3}$$

where $[J]^k = F'(U^k)$ is the n by n Jacobian matrix, with n as the number of the total nodes. Note that the variable with superscript k and $k+1$ represents the variable value at previous and current iterative steps, respectively. In addition, it can be easily shown that the only nonzero terms in the Jacobian matrix are the *diagonal* entries if we integrate the nonlinear term of Eq. (1) in the Galerkin FE formulation with the *nodal quadrature*, rather than the Gaussian quadrature. With this integration scheme, Eq. (3) becomes

$$[K]\{U^{k+1}\} = \{F(U^k)\} + [\Lambda]^k(\{U^{k+1}\} - \{U^k\}) \tag{4}$$

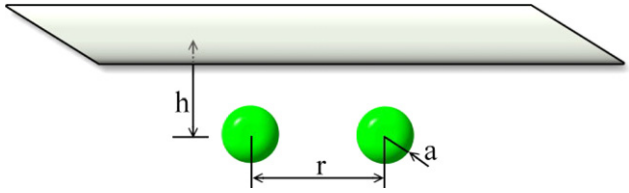


Fig. 15. Sketch of two identical charged spheres near a charged flat plate.

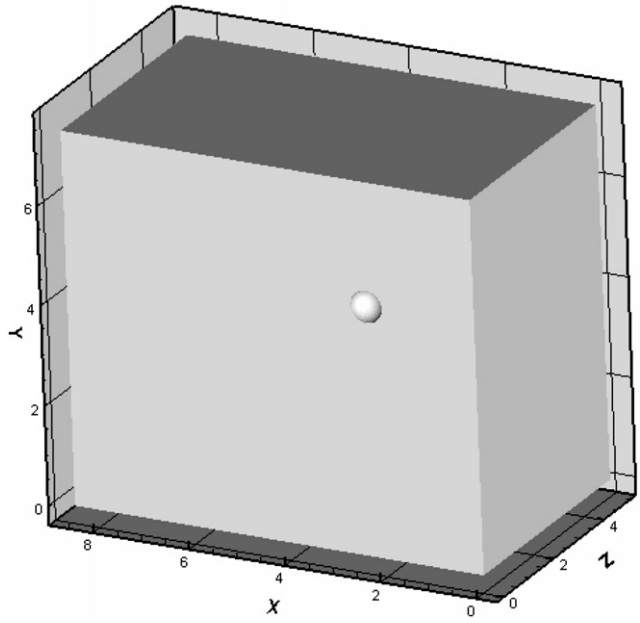


Fig. 16. Computational domain of two identical charged spheres near a charged flat plate.

where $[\Lambda]^k$ is the Jacobian matrix containing only nonzero diagonal terms λ_i^k with the subscript i representing the ith diagonal term. Eq. (4) can be further rearranged as

$$[A]\{U^{k+1}\} = \{B(U^k)\} \tag{5}$$

where

$$[A] = [K] - [\Lambda]^k, \tag{6}$$
$$\{B(U^k)\} = \{F(U^k)\} - [\Lambda]^k\{U^k\}. \tag{7}$$

Then at each iterative step, Eq. (5) is solved using the conjugate gradient method with nonzero entries stored in a compressed sparse row (CSR) format that is computationally efficient both in terms of storage and matrix operations [31]. The Poisson–Boltzmann equation solver is further parallelized using a subdomain-by-subdomain method similar to our previous work [31] in which the domain decomposition method was used.

The completed code is then coupled with the present PAMR to compute the interaction force between two like-charge spheres near a planar surface with the same charge, which has the same experimental configuration as that of Larsen and Grier [32]. This is shown in Fig. 15. In the simulation, $r = 4.5$, $h = 2.5$, and $a = 0.325$. Due to the symmetry of the problem, only 1/4 of the full physical domain is used as the computational domain, as illustrated in Fig. 16. The potentials at the

Table 5
Force of interaction with different refinement level mesh in the simulation of two identical charged spheres near a charged flat plate ($h = 2.5$, $r = 4.5$, $a = 0.325$)

Level	Number of nodes	Number of elements	Force of interaction (N)
0	17,058	83,588	−8.433E−15
1	52,549	275,534	−4.645E−15
2	131,067	698,093	−1.343E−15
3	161,345	872,534	−9.842E−16
4	177,269	963,419	−7.739E−16
5	184,564	1,019,692	−6.178E−16
6	184,611	1,020,243	−5.491E−16

charged sphere surface and outer boundaries are respectively set as $\psi_b = 3.0$ and $\psi_s = 5.0$, while the Neumann boundary conditions are used throughout the symmetric planes. Ten processors are used in the present simulation.

In this simulation, an *a posteriori* error estimator of the type proposed by Zienkiewicz and Zhu [33], and implemented by Dyshlovenko [27] for axisymmetric Poisson–Boltzmann equation is used to estimate the solution error during the mesh-refining process. A postprocessed recovered "exact" solution of the electric field can be obtained from various gradient recovery schemes which are applied to the FE solution of the electric field. The FE solution of the electric field is then compared with this "exact" solution of electric field. Similar to Ref. [26], a very simple gradient recovery scheme, which is based on averaging the cell values instead of the nodal values as used in Ref. [26] of the FE solution of electric field, is employed. This is attributed to the fact that electric fields at the nodes are discontinuous since we only utilize the linear shape function in the current study, rather than the quadratic shape function as used in Ref. [26]. A prescribed global relative error ε_{pre} of 0.0003 is used to control the level of accuracy. The absolute error in each element is then compared with a current average absolute error, which is based on this ε_{pre}, to decide if refinement is required. Note this current average absolute error δ_m is defined as

$$\delta_m = \left[\frac{\varepsilon_{\text{pre}}\|\widehat{E}\|^2}{N}\right],$$

where $\|\hat{E}\|^2$ is the integral of the square of FE electric fields over all elements and N is the number of elements in the computational domain. In addition, the method of calculating interaction force using the potential distribution on the sphere surfaces is described in detail elsewhere such as in Dyshlovenko [27], and is skipped here for brevity.

The evolution of the total number of nodes and elements, and the interaction force between the two spheres at different levels of mesh refinement are summarized in Table 5. The number of nodes increases from 17,058 (initial, Fig. 17(a)) to 184,611 (level-6, Fig. 17(b)), while the resulting force of interaction between the two spheres converges to −5.491E−16, which agrees reasonably well with the experimental data, −8.735E−16, within the experimental uncertainties [32]. To our best knowledge, we believe that we are the first to accurately predict the interaction force under such configuration using the Poisson–Boltzmann equation solver. Note that the original experimental

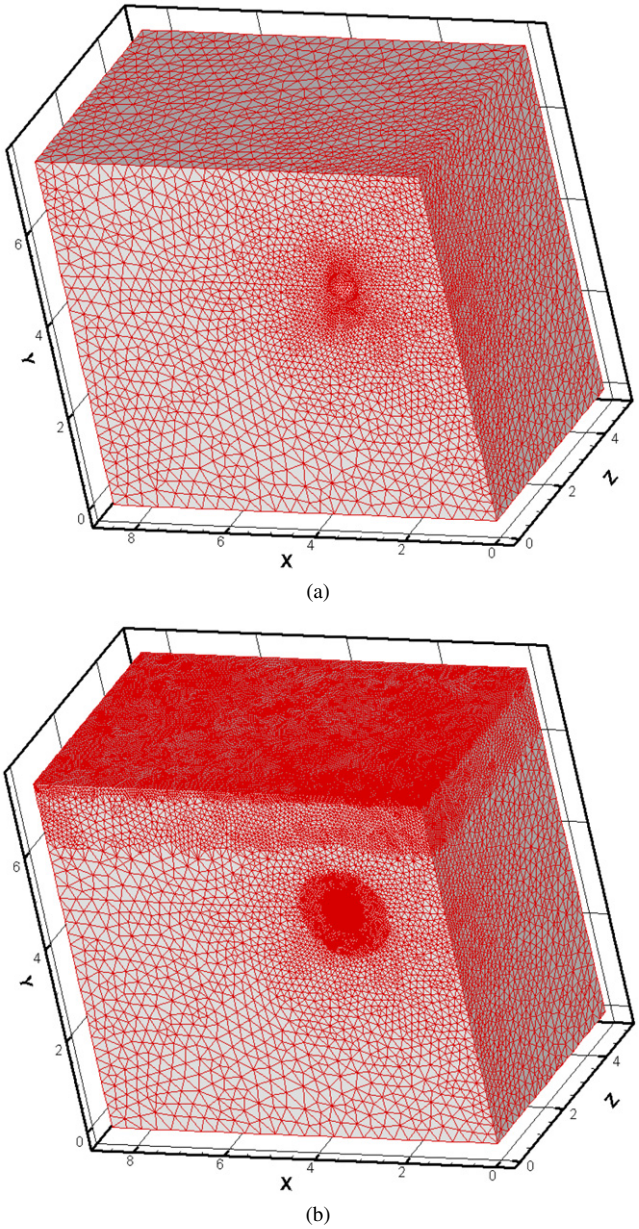


Fig. 17. Surface mesh distribution in the simulation of two identical charged spheres near a charged flat plate. (a) Initial mesh; (b) level-6 refined mesh.

data measured by Ref. [31] is presented in interparticle potential energy. We derive the force by first fitting the curve and obtaining its derivative as the interparticle force. It is also clear that most of the refined mesh is clustered in the electrical double layer near the sphere surface and the planar surface as expected. The parametric study of varying the simulation parameters (such as the distance between the two spheres, the distance between the sphere and the planar surface, etc.) is not pursued in the present study. Research in this direction is currently in progress and will be reported in the very near future.

From the two numerical examples shown above by applying the PAMR to both parallelized particle-based and equation-based solvers, it is shown that the numerical accuracy can be systematically improved without resorting to repeated manual

meshing. This is especially important when we apply the solver to a realistic large-scale 3-D problem with a complicated geometry.

4. Conclusions

In the present study, a parallel adaptive mesh refinement scheme with improved simple cell-quality control for a large-scale unstructured tetrahedral mesh was proposed and presented in detail. A *cell-based* data structure was designed such that the resulting refined mesh information can be readily utilized in both node- or cell-based numerical methods. It was shown from the study of PAMR parallel performance that the parallel speedup scales approximately with $N^{1.5}$ for $N_{\text{proc}} \leqslant 32$ if the number of cells is in the order of millions. Two numerical examples, including a particle-based method (parallel DSMC solver for rarefied gas dynamics) and an equation-based method (parallel Poisson–Boltzmann equation solver for electrostatic field), were used to demonstrate the generality of the present PAMR module.

Acknowledgement

This work was performed under NSC91-2212-E-009-045 of the National Science Council and 92-NSPO(A)-PC-FA06-01 of the National Space Organization office in Taiwan. The computing resource provided by the Department of Mechanical Engineering of the University of Alabama in Birmingham, USA is highly appreciated. The authors would also like to thank Prof. F.-N. Hwang of the National Central University in Taiwan for his insightful discussions during the preparation of this manuscript.

References

[1] J.D. Anderson, Computational Fluid Dynamics: The Basics with Applications, McGraw-Hill Book Co., New York, 1995.
[2] T.J. Chung, Computational Fluid Dynamics, Cambridge University Press, Cambridge, 2003.
[3] J.S. Wu, H.Y. Chou, U.M. Lee, Y.L. Shao, Y.Y. Lian, ASME Journal of Fluids Engineering 127 (2005) 1161.
[4] L. Meirovitch, Computational Methods in Structural Dynamics, Sijthoff & Noordhoff Press, Rockville, MD, 1980.
[5] D.E. Ellis, Density Functional Theory of Molecules, Clusters and Solids, Kluwer Academic, 2002.
[6] J. Jin, The Finite Element Method in Electromagnetics, second ed., John Wiley & Sons, New York, 2002.
[7] M.T. Jones, P.E. Plassmann, Finite Elements in Analysis and Design 25 (1997) 41.
[8] C. Özturan, H.L. de Cougny, M.S. Shephard, J.E. Flaherty, Computer Methods in Applied Mechanics and Engineering 119 (1994) 123.
[9] P. MacNeice, K.M. Olsonb, C. Mobarry, R. de Fainchtein, C. Packer, Computer Physics Communications 126 (2000) 330.
[10] C.D. Norton, J.Z. Lou, T. Cwik, Status and directions for the PYRAMID parallel unstructured AMR library, in: Proceedings of the 15th International Parallel & Distributed Processing Symposium, IEEE Computer Society, Washington, DC, 2001.
[11] L. Oliker, R. Biswas, H.N. Gabow, Parallel Computing 26 (2000) 1583.
[12] J. Waltz, International Journal for Numerical Methods in Fluids 46 (2004) 37.
[13] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, Parallel Programming in OpenMP, Morgan Kaufmann Publishers, San Francisco, CA, 2001.
[14] G.A. Bird, Molecular Gas Dynamics and the Direct Simulation of Gas Flows, Oxford University Press, New York, 1994.
[15] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, Cambridge, MA, 1999.
[16] L. Wang, J.K. Harvey, The application of adaptive unstructured grid technique to the computation of rarefied hypersonic flows using the DSMC method, in: J. Harvey, G. Lord (Eds.), 19th International Symposium on Rarefied Gas Dynamics, 1994, p. 843.
[17] A. Spirkin, N. Gatsonis, Computer Physics Communications 164 (2004) 383.
[18] J.-S. Wu, K.C. Tseng, F.Y. Wu, Computer Physics Communications 162 (2004) 166.
[19] G. Karypis, V. Kumar, ParMETIS 3.1: An MPI-based parallel library for partitioning unstructured graphs, meshes, and computing fill-reducing orderings of sparse matrices, 2003. Available from: http://www-users.cs.umn.edu/~karypis/metis/parmetis.
[20] J.S. Wu, K.H. Hsu, Parallel implementation of a 3-D electrostatic PIC method using unstructured mesh, in: ICOPS-2005, 32nd IEEE Plasma Conference, Montery, CA, June 18–23, 2005.
[21] J.-S. Wu, K.-C. Tseng, International Journal for Numerical Methods in Engineering 63 (2005) 37.
[22] J.N. Moss, J.M. Price, Journal of Thermophysics & Heat Transfer 11 (1997) 321.
[23] J. Allegre, D. Bisch, J.C. Lengrand, Journal of Spacecraft and Rockets 34 (6) (1997) 714.
[24] J. Allegre, D. Bisch, J.C. Lengrand, Journal of Spacecraft and Rockets 34 (6) (1997) 719.
[25] J. Allegre, D. Bisch, J.C. Lengrand, Journal of Spacecraft and Rockets 34 (6) (1997) 724.
[26] J.S. Wu, K.C. Tseng, C.H. Kuo, International Journal for Numerical Methods in Fluids 38 (2002) 351.
[27] P.E. Dyshlovenko, Journal of Computation Physics 172 (2001) 198.
[28] R. Tuinier, Journal of Colloid and Interface Science 258 (2003) 45.
[29] W.R. Bowen, A.O. Sharif, Nature 393 (1998) 663.
[30] R.J. Yang, L.M. Fu, Y.C. Lin, Journal of Colloid and Interface Science 239 (2001) 98.
[31] Y. Saad, Iterative Method for Sparse Linear System, second ed., Society for Industrial and Applied Mathematics, 2003.
[32] A.E. Larsen, D.G. Grier, Nature 385 (1997) 230.
[33] O.C. Zienkiewicz, J.Z. Zhu, International Journal for Numerical Methods in Engineering 24 (1987) 337.