九十二年度太空科技任務導向 委託研究計畫

(期末報告)

中文計畫名稱：衛星推進器引致污染及衝擊影響分析軟體發展

英文計畫名稱：Development of a DSMC Code for Analyzing the Contamination and Plume Impingement on Spacecraft (1/2)

計畫編號：92-NSPO(A)-PC-FA06-01

計畫執行期限：92.07.16~93.07.15

申請機構：國立交通大學機械工程學系

執行單位：國立交通大學機械工程學系

計畫主持人：吳宗信

日　　期：中　華　民　國　93　年　9　月　15　日

Development of a DSMC Code for Analyzing the Contamination and Plume Impingement on Spacecraft

J.-S. Wu

[1]Department of Mechanical Engineering, National Chiao-Tung University, Taiwan

SUMMARY

A general parallel three-dimensional direct simulation Monte Carlo method using unstructured mesh is introduced in this report, which incorporates a multi-level graph-partitioning technique to dynamically decompose the computational domain. Particle ray-tracing technique is used to track the particle on unstructured mesh using cell connectivity information. In addition, various strategies of applying the Stop at Rise (SAR) [30] scheme are studied to determine how frequent the domain should be re-decomposed. The completed code is verified by computing a two-dimensional hypersonic flow, a three-dimensional hypersonic cylinder flow and a sphere flow to demonstrate its superior computational capability. Results are then compared with experimental data and previous simulation data wherever available. Finally, preliminary results simulating plume impingement of ROCSAT-3 using the completed DSMC code is presented.

KEY WORDS: direct simulation Monte Carlo, parallel, graph partition, dynamic domain decomposition, hypersonic flow, ROCSAT-3

LIST OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The DSMC method has become a widely used computational tool for the simulation of gas flows in which molecular effects become important [1]. Specific examples include the plume impingement from attitude-control thrusters on satellite [2], the pumping characteristics of high vacuum pump [3], the low-pressure plasma-etching and chemical vapor deposition (LPCVD) [4], the computer hard disk slider air bearing [5] and the micro-electro-mechanical-system (MEMS) [6-8], to name a few. The advantage of using a particle method under these circumstances is that molecular model can be applied directly to the calculation of particle collisions and particle-wall interactions, while the continuum methods use macroscopic averages to account for such effects. Therefore, particle method can in general predict these effects with much higher accuracy under rarefied condition. With the advancement of computing capability, not only is the DSMC method the practical tool for analyzing the gas flows in the transitional regime, but also it is potentially a numerical method for studying gas flows from continuum to free-molecular regime. However, the main drawback of such direct physical method is its high computational cost, especially in the near-continuum regime.

Computing requirements for near-continuum flows can often render a meaningful DSMC simulation unpractical on scalar machines. Since the DSMC method is a particle-based numerical method, the movement of each particle is inherently independent of each other. The DSMC method is highly suitable for parallel processing since the coupling between particles is only made through collision in the cells. Therefore, the parallel DSMC method represents an opportunity to simulate flows in the near-continuum regime with an acceptable runtime [9] and to dramatically decrease the computational time in other regimes.

In the past, several studies on parallel implementation of DSMC have been published using static domain decomposition on structured/unstructured mesh; see [e.g., 10-14] and references cited therein. Message passing was often used to transfer molecules and associated data between processors and to provide the synchronization necessary for the correct physical simulation. The results show reasonable speedup and efficiency could be obtained if the problem is sized properly to the number of processors. However, the speedup often levels off very quickly due to the load unbalancing and increase of communication among the processors. Besides, there are several important studies in parallel DSMC method, which is worthy of detailed review as follows.

Recently, Boyd's group [15,16] designed parallel DSMC software named *MONACO*, which emphasized high data locality to match the hardware structure of modern workstations, while maintains the code efficiency on vectorized supercomputers. In this code, unstructured grids were used to take the advantage of flexibility of handling complex object geometry. Static domain decomposition technique was used to distribute cells among processors. Interactive human interruption is required to redistribute the cells among processors to maintain workload balance among processors, which is indeed unsatisfactory from practical viewpoint. Timing results show the performance improvement on workstations and the necessity of load balancing for achieving high performance on parallel computers. Maximum 400 IBM-SP2 processors have been used to simulate flow around a planetary probe with approximately 100 million particles, which parallel efficiency of 90% has been reached by manually redistributing the cells among processors during simulation. However, the parallel efficiency for *n* processors is unusually defined as the ratio of computational time to the sum of computational and communicational time, rather than it is normally defined as the ratio of the true speedup to the ideal speedup (n) for *n* processors.

Ivanov's group [17] has developed a parallel DSMC code called *SMILE*, which implements both the static and dynamic load balancing techniques. *SMILE* has united the background cells into groups, so-called "clusters", which are the minimum spatial unit, and are distributed and transferred between the processors. The dynamic domain decomposition algorithm is scalable and requires only local knowledge of the load distribution in a system. In addition, the direction and the amount of workload transfer are determined by the concept of heat diffusion process [18]. In addition, an automatic granularity control is used to determine when to communicate the data among processors [18].

Around the same period of time, dynamic load balancing technique, using Stop At Rise (SAR) [30], which compares the cost of re-mapping the decomposition with the cost of not re-mapping, based on a degradation function, was used in conjunction with the parallel implementation of the DSMC method [9,14]. In the study [9], they used a runtime library, *CHAOS*, for data communication and data structure manipulation on a structured mesh. Results show that it yields significantly faster execution times than the scalar code, although only 25% of parallel efficiency is achieved for 64 processors. LeBeau [19] reported that parallel efficiency up to 90% is achieved for 128 processors for the flow over a sphere. It is not clear how they implemented the dynamic load balancing, although they did mention they have used the concept of heat diffusion [18]. In LeBeau's study [19], surface geometry is discretized using an unstructured triangular grid representation. A two-level embedded Cartesian grid is employed for the discretization of the computational domain.

In summary, studies about DSMC using both purely unstructured mesh and dynamic domain decomposition were relatively few in the past [20-22], although using unstructured mesh exhibits higher flexibility in handling objects with complicated geometry and boundary conditions. Robinson [20-22] has first developed a heuristic, diffusive, hybrid graph-geometric, localized, concurrent scheme, *ADDER*, for repartitioning the domain on an unstructured mesh. Dramatic increase of parallel efficiency was reported as compared with that of static domain decomposition. However, Robinson [20-22] has shown that the parallel efficiency begins to fall dramatically as the number of processors increases to some extent due to the large runtime of the repartitioning the domain relative to the DSMC computation. Thus, the utilization of a more efficient repartitioning runtime library is essential to improve the performance of a parallel DSMC method.

To decompose an unstructured mesh across *NP* processors is a critical but difficult issue in many applications [23]. It is usually approached as a graph-partitioning problem where each node in the mesh represents a vertex in the graph. A edge cut is formed when it connects two vertices across the inter-processor boundary. Each vertex and edge in the graph can be given a weight, which represents an amount of work. A conventional graph-partitioning problem is to subdivide the n vertices between the *NP* sub-domains while minimizing the number of edge cuts, E_c, and balancing the weight in each sub-domain. However, it is well known that it is *NP* complete, which means that the optimal solution of this problem is impossible to compute in polynomial bounded time. Instead, it is relaxed to seek near-optimal solutions within reasonable time. In computer science, there are several methods developed for achieving near-optimal solutions to this problem. Among these, spectral bisection has been widely used [23] in the past. Recently, a multi-level partitioning method has become more popular [24,25], in which the graph is coarsened and partitioned. This new partition is then mapped back to the original graph. These methods utilized substantial heuristic approaches, which has been proven as a powerful graph-partitioning tool. "Pure" heuristic method proposed by Kernighan and Lin [26] has been often incorporated into the local refinement phase of the multi-level schemes. These partitioning tools are shown to have superior performance and are relatively easy to parallelize. In addition, the extension of the graph partitioning to three-dimensional case is straightforward in essence.

One of the advantages in expressing the problem in terms of a graph is that each of the edges and vertices can be assigned a weight to account for the specific numerical application. For example, in DSMC, the vertex (i.e., cell center) can be weighted with the number of particles with all edges that connects cell centers, having unitary weight. A truly dynamic load balancing technique is required for DSMC because the load (approximately proportional to the number of particles) in each sub-domain changes frequently, especially during the transient period. Domain decomposition in DSMC may become very efficient by taking the advantage of successful development in graph partitioning. For example, the multi-level scheme, *PJOSTLE* [27], uses initial domain decomposition (generated by greedy partitioning) and successively adjusts the partition by moving vertices lying on partition boundaries. In this method, vertex shedding is localized since only the vertices along the partition boundaries are allowed to move, not the vertices anywhere in the domain. Hence, this method possesses a high degree of concurrency and has been written as a package of runtime libraries on many modern computer platforms [27]. Thus far, there seems no report that matured graph-partitioning tool has been incorporated in the parallel DSMC method on an unstructured mesh. Thus, it is interesting and technically important to learn that if the graph partition tools can be used efficiently in conjunction with the DSMC method. Thus, in the current study, we will use *PJOSTLE* to dynamically decompose the computational domain for the parallel DSMC simulation.

Therefore, the objectives of the current study are summarized as follows.

1. To complete a parallel three-dimensional DSMC code on an unstructured mesh incorporating the multi-level graph-partitioning technique to dynamically decompose the computational domain.
2. To verify the parallel DSMC implementation by computing a realistic, near-continuum two-dimensional hypersonic flows over a cylinder, and a three-dimensional hypersonic flow past a sphere and compare with previous experimental and DSMC data wherever available.
3. To apply the completed code to compute the plume impingement using ROCSAT-3 as an example.

The report begins with descriptions of the parallel DSMC method and the strategies of repartitioning the domain. Verifications of the current parallel are present by two realistic flows, and finally the roughly simulation results of reaction control system of spacecraft are present in turn.

II. NUMERICAL METHOD

Direct Simulation Monte Carlo Method

The direct simulation Monte Carlo method (DSMC) is a particle method for the simulation of gas flows. The gas is modeled at the microscopic level using simulated particles which each represents a large number of physical molecules or atoms. The physics of the gas is modeled through uncoupling of the motion of particles and collisions between them. Mass, momentum and energy transports are considered at the particle level. The method is statistical in nature. Physical events such as collisions are handled probabilistically using largely phenomenological models, which are designed to reproduce real fluid behavior when examined at the macroscopic level.

Since Bird [1] has documented in detail the conventional DSMC method in his monograph, it is only briefly described here. Important steps of the DSMC method include setting up the initial conditions, moving all the simulated particles, indexing (or sorting) all the particles, colliding between particles, and sampling the molecules within cells to determine the macroscopic quantities. This method is essentially a computer simulation of gas molecular dynamics and depends heavily upon pseudo-random number sequences for simulating the statistical nature of the underlying physical processes. The data variables are often randomly accessed from computer memory. Thus, it is very difficult to vectorize the DSMC code. However, since the movement of each particle and the collision in each cell is treated independently, this makes DSMC perfectly suitable for parallel computation, which is introduced next.

Parallel Implementation of DSMC

The DSMC algorithm is readily parallelized through the physical domain decomposition. The cells of the computational grid are distributed among the processors. Each processor executes the DSMC algorithm in serial for all particles and cells in its own domain. Parallel communication occurs when particles cross the domain (processor) boundaries and are then transferred between processors. High parallel performance can only be achieved if communication is minimized and the computational load is evenly distributed among processors. To minimize the communication for domain decomposition, the boundaries between sub-domains should more or less lie along the streamlines of the flow field; however, it is nearly impossible to achieve this partition for most practical flows. In practice, we can only minimize the number of edge cuts E_c, under the framework of graph theory. Fortunately, the advancement of networking speed has reduced the communication time between processors to an acceptable level. For the DSMC algorithm, the workload (or equivalently the number of particles) in each processor changes frequently, especially during the transient period of a simulation; while the workload attains a roughly constant value during the steady-state sampling. Thus, a truly dynamic (or adaptive) domain decomposition technique is required to perfectly balance the workload among the processors.

Fig. 1 shows a simplified flow chart of the parallel DSMC method proposed in the current study, which incorporates the multi-level graph-partitioning technique. In general, this algorithm not only works for the DSMC method, but also it is suitable for other particle-based methods, such as Molecular Dynamics (MD), Particle-In-Cell (PIC) and Monte Carlo methods in plasma physics, which will be reported in the very near future. Note that processors are numbered from **0** to **np-1** in the figure. Before detailing the proposed procedures (Fig. 1), we will instead discuss the preprocessing required for this parallel implementation. In this implementation, an unstructured mesh is first constructed by a commercial code, *HyperMesh™* [28] or other equivalent meshing tool. Then, a preprocessing code is used to reorder the fully unstructured mesh data into the *globally sequential but locally unstructured* mesh data [10] for each processor in conformation with the partitioning information from graph partitioning tool (*JOSTLE*) [29], as schematically presented in Fig. 2. In addition to the above, another important information output from this preprocessor is the cell-neighboring information, which is needed for particle tracing on an unstructured mesh. Original algorithm [10] used to obtain the information of cell neighbors, using the concept of loops over cells by identifying repeated node number, has been found to be very inefficient as the total number of cells increases up to several tens of thousand. Instead, we have replaced it by a very efficient algorithm, using the concept of loops over nodes by searching through the cells sharing the node, which turns out to be very efficient. For example, for preprocessing 3 million unstructured 3-D cells, it takes less than 20 minutes on a 1.6-GHz (Intel) personal computer. Preliminary results show that the preprocessing time increases approximately linearly with the number of cells. Parallel processing to speed up this preprocessing is currently in progress and will be incorporated into the parallel DSMC code in the very near future.

Note that the partition information from *JOSTLE* provides the cell numbers ($\mathbf{m_n}$ for the nth sub-domain, where **n=0** to **np-1**) and mapping of cells in each partitioned sub-domain. After the cell-number reordering, the cells in each sub-domain are renumbered such that the corresponding global starting and ending cell numbers for the **n**th sub-domain are $\sum_{i=0}^{n-1} m_i + 1$ and $\sum_{i=0}^{n} m_i$, respectively. In each processor, the cell numbering is unordered (unstructured), but both the starting (smallest) and ending (largest) cell numbers increase with processor numbers. We term this as "*globally sequential but locally unstructured*" [10]. Thus, in each processor the memory is only needed to record the starting and ending cell numbers for all processors, in addition to the cell related data in each processor. The mapping between global and local cell data, however, can be easily obtained by a simple arithmetic operation due to this special cell-numbering design. The required array size for cell related data is approximately the same as the number of cells in each sub-domain. For example, if there are one million cells totally in the simulation with 100 processors, each processor will only be required to store the array on the order of 10,000. The memory cost reduction will be approximately 100 times in this case. This simple reordering of cell numbers dramatically reduces the memory cost otherwise required for storing the mapping between the local cell number in each processor and the global cell number in the computational domain if un-reordering unstructured cells are used.

In addition, a processor neighbor-identifying array is created for each processor from the output of the preprocessor, which is used to identify the surrounding processors for those particles crossing the inter-processor boundaries during simulation. From our practical experience, the maximum number of processor-neighbor is on the order of 10 at most; therefore, the increase of memory cost due to this processor neighbor-identifying array is negligible. The resulting *globally sequential but locally unstructured* mesh data with the partition information is then imported into the parallel DSMC code as the initial mesh distribution.

Again referring to Fig. 1, after reading the preprocessed cell data on a master processor (cpu **0**), the cell data are then distributed to all other processors according to the designated initial domain decomposition. All the particles in each processor then start to move as in sequential DSMC algorithm. The particle related data are sent to a buffer and are numbered sequentially when hitting the inter-processor boundary (IPB) during its journey within a simulation time step. After all the particles in a processor are moved, the destination processor for each particle in the buffer is identified via a simple arithmetic computation, owing to the previously mentioned approach for the cell-numbering scheme, and are then packed into arrays. Considering communication efficiency, the packed arrays are sent as a whole to its surrounding processors in turn based on the tagged numbers recorded earlier. Once a processor sends out all the packed arrays, it waits to receive the packed arrays from its surrounding processors in turn. This "send" and "receive" operation serves practically as a synchronization step during each simulation time step. Received particle data are then unpacked and each particle continues to finish its journey for the remaining time step. The above procedures are repeated twice since there might be some particles cross the IPB twice during a simulation time step. Theoretically it could be more than twice, but in our practical experience it is generally at most twice for "normal" domain decomposition and by carefully choosing the simulation time step.

After all particles on each processors have come to their final destinations at the end of a time step, the program then carries out the indexing of all particles and the collisions of particles in each computational cell in each processor as usual in a sequential DSMC code. The particles in each cell are then sampled at the appropriate time. The program then checks whether the remapping (or repartitioning) is required based on some decision policy, e.g., Stop At Rise (SAR) [30] in the current study, which will be described shortly for completeness. If it does, then the program begins to re-decompose the computational domain, using multi-level graph-partitioning technique, after which the cell- and particle-related data are transferred between processors. Finally, the received particles and cells are re-numbered to reflect the new partition in each processor. In brief summary, major difference between the parallel DSMC using dynamic domain decomposition and the original DSMC method lies in the addition of decision policy for repartitioning, repartitioning, migration and renumbering of cell/particle data among processors in the procedures, which will be described, respectively, in detail as follows.

Decision Policy for Repartitioning

DSMC represents a typical dynamic (or adaptive) irregular problem, i.e., workload distributions are known only at runtime, and can change dramatically as simulation proceeds, leading to a high degree of load imbalance among the processors. Thus, some decision policy is required to determine when to repartition the computational domain, since the repartition is often expensive computationally. It has been shown that, for some problems using DSMC, remapping the domain at fixed intervals leads to poor parallel performance [7,9]. Therefore, it is highly desirable to either pre-determine the optimal interval for repartitioning, or using a clever monitoring policy to decide when to repartition. The former choice is definitely impractical since pre-runtime analysis is generally required to determine this optimal choice. Therefore, in the current study, a decision policy, Stop At Rise (SAR) [30], is employed to determine when to repartition the domain. SAR, a "greedy" repartitioning policy, attempts to minimize the long-term processor idle time since the last repartitioning. This decision policy chooses to repartition the computational domain based on the value of a degradation function $W(t)$ at the t$^{\underline{th}}$ time step, which is defined as follows:

$$W(t)=\frac{\sum_{j=1}^{t}[T_{\max}(j)-T_{avg}(j)]+C}{t} \tag{1}$$

where $T_{\max}(j)$ is the maximum amount of time required by any processor to complete the j^{th} time step, $T_{\mathrm{avg}}(j)$ is the average time required by a processor to complete the j^{th} time step, and C is the amount of time required to complete the repartitioning operation.

This degradation function represents the average idle time for each processor including the cost of repartition. In general, $W(t)$ tends to decrease with the increasing value of t. The summation term in Eq. (1) will eventually increase as the workload unbalance develops, while the repartitioning cost, C, is approximately constant during simulation. Repartitioning is not performed until the time that $W(t) > W(t\text{-}1)$, i.e., when the first local minimum of degradation function is detected. This decision policy for repartitioning the domain is inherently advantageous over the fixed-interval scheme in that no prior knowledge of the evolution of the problem is necessary for the determination of the repartitioning interval, and the repartitioning can be expected to follow the dynamics of the problem without wasting computing resources.

Repartitioning Technique

In the current study, we have incorporated the parallel runtime library, *PJOSTLE* [27], as the repartitioning module in our parallel DSMC code. *JOSTLE* [29], a serial version of *PJOSTLE* [27], uses the multilevel implementations that match and combine pairs of adjacent vertices to define a new graph and recursively iterate this procedure until the graph size falls under some threshold. The coarsest graph is then partitioned and the partition is successively refined on all the graphs starting with the coarsest and ending with the original. At evolution of levels, the final partition of the coarser graph is used to give the initial partition for the next finer level. *PJOSTLE* [27], a parallel version of *JOSTLE* [29], uses an iterative optimization technique known as relative gain optimization, which both balances the workload and attempts to minimize the inter-processor communication overhead. This parallel algorithm runs on single program multiple data (*SPMD*) paradigm with message passing in the expectation that the underlying unstructured mesh will do the same. Each processor is assigned to a sub-domain and stores a double-linked list of the vertices (cell centers in DSMC) within that sub-domain. However, each processor also maintains a "halo" of neighboring vertices in other sub-domains. For the serial version, the migration of vertices simply involves transferring data from one linked-list to another. In parallel implementation, this process is far more complicated than just migrating vertices. The newly created halo vertices must be packed into messages as well, sent off to the destination processors, unpacked, and the pointer based data structure recreated there. This provides an extremely fast solution to the problem of dynamically load-balancing unstructured mesh [27].

In DSMC simulation, the workload of each processor is approximately proportional to the number of particles in the corresponding sub-domain. Thus, we can assign the weight of each vertex in graph as particle numbers in the corresponding cell in estimating the workload during simulation. *PJSOTLE* [27] will try to maintain perfect load balance while optimizing the partitions based on pre-determined balance factor. This factor, which affects the partitioning quality and cost, is defined by $B=S_{max}/S_{opt}$, where S_{max} is the largest allowable weight of the sub-domains and S_{opt} is the optimum sub-domain size which equal to the average weight of these sub-domains. B is 1.03 in the current study, unless otherwise specified. Simulated results have shown a fairly even particle distribution among processors is obtained using the above setting, which can be seen later.

Cell/Particle Migration

After repartitioning the domain, relationship between cells and sub-domains has to be updated according to the new partition. Any cell may be assigned to a processor, which is different from the original processor it belongs to. Thus, cell/particle associated data need to migrate to their new parental processor properly. Theoretically, the multi-level graph-partitioning scheme is much faster than the hybrid graph-geometric partitioning scheme developed by Robinson [20-22], in which only the "halo" cells of each sub-domain are allowed to move among processors after each repartition.

In addition, the original neighbor-identifying array, *nbr*(face_number, local_cell_number)=local_cell_number, in a sequential code has been changed to *nbr*(face_number, local_cell_number)=global_cell_number in the parallel code. Thus, an conversion array between local and global cell numbers is required to access the data efficiently. Note that the global cell numbers associated with each cell is not changed throughout the simulation. Only the local cell numbers for each cell in each processor has to be updated according to the new partition. Of course, the conversion array between local and global cell numbers has to be changed accordingly for those cells involved in migrating among processors. Thus, the update of neighbor-identifying array after cell data transferred between processors becomes very easy. Only the local cell numbers for the transferred cells have to be changed with negligible computational cost.

The cell/particle migration after the repartition is briefly summarized as follows.

1. Pack into buffer arrays the to-be-transferred particle related data particle by particle. Fig. 3 illustrates this procedure using CPU0 as the example, which requires data (column in shaded area) to be sent to CPU3 and receive data (row in shaded area) from CPU1 due to repartitioning of the computational

domain. Data include positions, velocities, internal energies and the new local cell numbers in the destination processor, to which the particle shall reside. The new local cell numbers is assigned as the value, which is the sum of one and the most updated pre-partitioned maximum local cell numbers in pre-partitioned destination processor. Having packed the to-be-transferred particle data, they are then removed from the source processor they belong to.

2. Pack into buffer arrays the to-be-transferred cell related data cell by cell. Similar procedure is also shown in Fig. 3. The procedures for each cell are described in detail as follows. *First*, record the data of the to-be-transferred cell, including new local cell number in the destination processor (as in step 1), cell/node coordinates and sampled data in the cell and related cell face. *Second*, update the relation between the local and global numbers, i.e., the data of the to-be-transferred cell numbers are then replaced by the data of the maximum cell numbers in the source processor. Then, subtract one from the maximum local cell numbers in the source processor.
3. Migrate both the particle- and cell-data in the buffer arrays as a whole to the destination processors.
4. Receive and unpack these data from the buffer.
5. Reorder and change the neighboring cell numbers accordingly.
6. Reconstruct the processor neighbor-identifying array for each processor.

The current parallel code incorporating the above procedures, in SPMD (Single Program Multiple Data) paradigm, is implemented on the IBM-SP2 and IBM-SMP machines (distributed memory system) using message passing interface (MPI) to communicate information among processors. It is thus essentially no code modification required to adapt to other parallel machines (e.g., PC-cluster system) with similar distributed memory system once they use the same MPI libraries for data communication.

III. VERIFICAITONS

To verify of the current implementation of parallel DSMC method using dynamic domain decomposition, we have applied it to compute several realistic flows, including a two-dimensional hypersonic flow past a cylinder and a three-dimensional hypersonic flow past sphere. Results are then compared with experimental data and previous simulation wherever available, while the description of flow physics of the test problems will be as brief as possible since it only serves to verify the applicability and its accuracy of the proposed method. Flow conditions and results for each case are described in the following in turn.

Two-Dimensional Hypersonic Flow Past a Cylinder

Flow and Simulation Conditions

Flow conditions are the same as those of Koura and Takahira [32] and represent the experimental conditions of Bütefsch [31]. For completeness, they are briefly described here as follows: VHS nitrogen gas, free-stream Mach number M_∞=20, free-stream number density n_∞=5.1775E19 particles/m^3, free-stream temperature

T_∞=20K, fully thermal accommodated and diffusive cylinder wall with T_w/T_0=0.18, where T_w (=291.6 K) and T_0 (=1620 K) are the wall and stagnation temperatures, respectively. Temperature dependent rotational energy exchange model of Parker [33] is used to model the diatomic nitrogen gas with the following parametric setting: limiting rotational collision number Zr_∞=21, potential well-depth temperature T*=79.8 K. Resulting Knudsen number is 0.025, based on the free-stream mean free path and diameter of the cylinder. An *h-refined* mesh with mesh quality control, resulting from the cell size (less than local mean free path) and density gradient requirements [34] is used in this simulation (64 processors) to increase the accuracy of the solution. The simulation particles are about 1.3 million at steady state and the number of cells is approximately 75,000 after 4 levels of mesh refinement (Fig. 4). Constant time-step method is used throughout the computational domain. 20,000 time steps are used to sample for obtaining averaged flow properties.

Domain Decomposition

Figure 5 shows the initial and final domain decomposition for this simulation. Large variation of sub-domain area in the initial domain decomposition results from the use of a solution-based adaptive mesh, which is obtained from a mesh adaptation module [34] based on a preliminary parallel simulation. For the initial domain decomposition, we have assigned the uniform weight of each cell to operate the initial domain decomposition. Number of cells in each sub-domain is approximately the same initially, although the size of each sub-domain is highly different. Figure 5b shows that the final decomposition, which adapts to the flow dynamics as simulation continues, is totally different from the initial decomposition.

Centerline Properties Distribution

Figures 6 and 7 illustrate the computed centerline densities and temperatures (rotational and translational) using dynamic domain decomposition, respectively, along with the simulation data without dynamic domain decomposition and previous experimental data [31]. Density increases rapidly along the centerline in front of the cylinder and becomes relatively small in the wake region. Temperature also increases rapidly along the centerline but decreases rapidly after the bow shock. Strong non-equilibrium between rotational and translational temperatures is found after the cylinder due to the highly rarefied conditions in the wake region. Nevertheless, agreement between the current simulation with/without dynamic domain decomposition and experimental data wherever available is excellent, considering the experimental uncertainties. In addition, simulation using dynamic domain decomposition reduces the running time up to 60% in this case, as compared with that using static domain decomposition.

Three-dimensional Flow Past a Sphere

Flow and Simulation Conditions

A hypersonic flow past a sphere is simulated to demonstrate the applicability of the current parallel implementation to three-dimensional flow problem. Simulation is conducted for 1/16 of a sphere by taking advantage of the inherent axial symmetry of this problem. The reasons to choose this as the test problems are, first, there exist experimental data and, second, it is a good test for checking if the simulation can reproduce the flow symmetry. Third, it can prove that the dynamic domain decomposition method can be easily extended to three-dimensional flow. Related flow conditions, which represent the experimental conditions of Russel [35], are listed as follows: VHS nitrogen gas; free-stream Mach number M_∞= 4.2; free-stream number density n_∞ = 9.77E20 *particles/m*^{3}; free-stream temperature T_∞= 66.25K; stagnation temperature T_o = 300K; fully thermal accommodated and diffusive sphere wall with the temperature T_w (equal to stagnation temperature T_o). The corresponding free-stream Knudsen number Kn_∞ is 0.1035, based on the free-stream mean free path and diameter of the sphere. The diameter of sphere is 1.28cm.

An *h-refined*, three-dimensional mesh with mesh quality control [36] is used in this simulation (8 IBM-SMP processors) to increase the accuracy of the solution. In addition, variable time-step method [36,37] is implemented in the three-dimensional code to further reduce the computational time, in which the local time step in each cell is proportional to the size of adaptive cell. The simulation particles are about 1.7 million at steady state and the number of cells is approximately 164,000 after 2 levels of mesh refinement (Fig. 8). 20,000 time steps are used to sample for obtaining averaged flow properties.

Dynamic Domain Decomposition

Figure 9 illustrates the initial and final domain decomposition for the hypersonic flow past a sphere on a reduced (1/16) computational domain surface. The initial domain decomposition (Fig. 9a) is obtained assigning equal weight to each cell, which is different from previous two cases. At the final domain decomposition (Fig. 9b), the sub-main size in front of the sphere enlarges as compared with the initial sub-domain size, due to the application of variable time-step method and increased density in the stagnation and bow shock region. In this case, the computational time is saved up to 35% using dynamic domain decomposition, as compared with that using static domain decomposition. We would expect much higher time saving of more processors are used.

Centerline Density Distribution

Figure 10 presents the computed centerline density distribution using dynamic domain decomposition, along with that computed using static domain decomposition and experimental data of Russel [35]. The current computed results agree excellently with experimental data in front of the sphere, in which the experimental data behind the sphere is not available. Also the computed results between dynamic and static domain decomposition is indistinguishable in this case, which again proves the correct implementation of dynamic domain decomposition in three-dimensional flow.

IV. RESULTS AND DISCUSSIONS

When spacecraft flies in space, the spacecraft thrusters are used to provide the attitude control of the spacecraft. However, improper design of the thrust location and thrust angle can induce unwanted effects such as contamination, disturbance torques and possibly erosion on the spacecraft surface. Prediction of the plume impingement on spacecraft is thus very important during the design phase of the reaction control system, which mainly consists of small thrusters. Exhaust jets issuing from the thrusters of spacecraft produce a complicated flow field. In this flow field, the flow is continuum in the nozzle inlet and near the throat, then becomes transitional further downstream near the exit and rarefied as the flow pass through the nozzle to the space. In the current study, we compute the flow field of a realistic full-scale ROCSAT-3 model to demonstrate the feasibility of prediction using the PDSC (parallel direct simulation Monte Carlo Code) developed in the project.

Flow and Simulation Conditions

This satellite is designed for observing meteorology, ionosphere and climate, and operated on an orbit about 700~800 kilometers. The details of the project of ROCSAT-3/COSMIC can be found in the following website, http://www.nspo.gov.tw/rcweb/chinese/rs3_intro.html. The flow conditions of ROCSAT2, which are provided by National Space Program Office (NSPO), are used for ROCSAT3 simulation. Computer-generated photograph of ROCSAT-3 is shown in Fig. 11. Simulation is conducted for 1/2 of the solar panel and the body of the satellite by taking advantage of the inherent axial symmetry of this problem. Simulation conditions are summarized in Table 1. Related flow conditions can be briefly listed as follows: the cant angle of four thrusts is 10°; the flow consists of N_2H_4, NH_3 and H_2 with corresponding mole fraction 0.58:0.71:1.13, respectively. Total number density n_∞ = 2.E22 *particles/m*^{3}; Fig. 12 illustrates the temperature and velocities of different directions at the thruster exit, which is the inlet of computational domain. Note the data at this ocation are obtained using commercial CFD solver with computation starting from the reservoir. In addition, fully thermal accommodation on all solid walls is assumed. Full-scale simulation model of ROCSAT-3 and computational mesh (only surface is shown) is shown in Fig. 13 and Fig. 14, respectively.

Surface properties contours

In this report, most data are presented using surface properties, which are the most concerned properties in the plume impingement study. Some iso-surface data in the space above the main body are shown to help understand the underlying physics of plume impingement.

Figs. 15-26 illustrate the surfaces properties contours and the iso-surface for different cases. As Fig. 16(a), Fig. 18(a), Fig. 20(a), Fig. 22(a), Fig. 24(a), and Fig. 26(a) show, the number densities is generally larger in the regions near the thruster exit and at the location near the thruster jet interaction above the main body. In addition, there is one region on the solar panel with larger number density due to the impingement of the plume. Also the plume is divided into twp parts by the presence of the solar panel. The flow conditions behind the solar panel strongly depends the angle of the solar array. The larger the angle of the solar panel, the less flow goes into the regions behind the solar panel.

The solid surface temperature seems having very little effect on number density distribution. Fig. 16(b), Fig. 18(b), Fig. 20(b), Fig. 22(b), Fig. 24(b), and Fig. 26(b) show the temperature distribution. There are some regions with obvious higher temperatures, including where the jets issue and jets interact. Fig. 15, Fig. 17, Fig. 19, Fig. 21, Fig. 23, and Fig. 25 illustrate the number flux on the satellite surfaces, which shows that much higher surface number flux is obtained for H2 gas due to its high diffusivity. The smaller angle the solar panel is, the larger number flux impinges on the solar panel. It is easy to understand that the closer the distance between the jets and solar panel more particles come to impinge on the solar panel.

V. CONCLUSIONS

In the current study, a parallel DSMC method that dynamically re-decomposes the computational domain using graph-partitioning technique is presented. Proposed method is then applied to compute several realistic cases, including a two-dimensional hypersonic cylinder flow and a three-dimensional hypersonic sphere flow. Computed results are compared with experimental data and previous simulation data wherever available. In summary, the progress of the current research are listed as follows:

1. A general parallel 3-D DSMC method combining variable time-step scheme and dynamic domain decomposition on unstructured mesh are implemented and verified successfully.

2. Verification by computing several realistic flow problems shows the accuracy and computational efficiency of the current proposed parallel DSMC method using dynamic domain decomposition.

3. Completed PDSC applied to compute a full-scale ROCSAT-3 show that it is feasible to predict the complicated plume impingement on spacecraft.

REFERENCES

1. Bird GA. Molecular Gas Dynamics and the Direct Simulation of Gas Flows. Oxford University Press, New York, 1994.
2. Boyd ID, Jafry Y, Beukel JW. Particle simulation of helium microthruster flows. Journal of Spacecraft Rockets 1994; 31:271-281.
3. Lee YK, Lee JW. Direct simulation of pumping characteristics for a model diffusion pump. Vacuum 1996; 47:297-306.
4. Plimpton S, Bartel T. Parallel Particle Simulation of Low-Density Fluid Flows. U.S. Department of Energy Report No. DE94-007858, 1993.
5. Alexander FJ, Garcia AL, Alder BJ. Direct Simulation Monte Carlo for Thin-film Bearings. Physics of Fluids 1994; 6:3854-3860.
6. Piekos ES, Breuer KS. Numerical modeling of micromechanical devices using the direct simulation Monte Carlo method. Transaction for ASME Journal of Fluids Engineers 1996; 118:464-469.
7. Nance RP, Hash DB, Hassan HA. Role of boundary conditions in Monte Carlo simulation of microelectromechanical systems. Journal of Thermophysics and Heat Transfer 1998; 12(3): Technical Notes:447-449.
8. Wu JS, Tseng KC. Analysis of Micro-scale Gas Flows With Pressure Boundaries Using The Direct Simulation Monte Carlo Method. Computers and Fluids 2001; 30: 711-725.
9. Nance RP, Wilmoth RG, Moon B, Hassan HA, Saltz JH. Parallel Solution Monte Carlo Simulation of Three Dimensional Flow Over a Flat Plate. Journal of Thermophysics and Heat Transfer 1995; 9(3):471-477.
10. Wu JS, Tseng KC, Yang TJ. Parallel Implementation of the Direct Simulation Monte Carlo Method Using Unstructured Mesh and Its Application. International Journal of Computational Fluid Dynamics. Vol. 17, No. 3, pp. 405-422, 2003.
11. Furlani TR, Lordi JA. Implementation of the Direct Simulation Monte Carlo Method for an Exhaust Plume Flowfield in a Parallel Computing Environment. 1988, AIAA Paper 88-2736.
12. Matsumoto Y, Tokumasu T. Parallel Computing of Diatomic Molecular Rarefied Gas Flows. Parallel Computing 1997; 23:1249-1260.
13. Nance RP, Wilmoth RG, Moon B, Hassan HA, Saltz J. Parallel DSMC Solution of

Tıree-Dimensional Flow Over a Finite Flat Plate. AIAA/ASME Sixth Joint Tıermophysics and Heat Transfer Conference, Colorado Springs, CO, 1994, AIAA Paper 94-0219.

14. O:a M, Taniguchi H, Aritomi M. Parallel Processing for Direct Simulation Monte C:rlo Method. The Japan Society of Mechanical Engineering (B) 1995; 61(582):496-502.

15. Dietrich S, Boyd ID. Scalar and Parallel Optimized Implementation of the Direct Si nulation Monte Carlo Method. Journal of Computational Physics 1996; 126:328-342.

16. Kannenberg KC. Computational method for the Direct Simulation Monte Carlo technique with application to plume impingement. Ph.D. Thesis, Cornell University, Ithaca, NY, USA, 1998.

17. Ivanov M, Markelov G, Taylor S, Watts J. Parallel DSMC strategies for 3D computations. In Proceedings of Parallel CFD'96, edited by P. Schiano et al., North Holland/Amsterdam, 1997; 485-492.

18. Taylor S, Watts J, Rieffel M, Palmer M. The Concurrent Graph: Basic Technology for Irregular Problems. IEEE Parallel and Distributed Technology 1996; 4:15-25.

19. LeBeau GJ. A Parallel Implementation of the Direct Simulation Monte Carlo Method. Compute Methods in Applied Mechanics and Engineering 1999; 174:319-337.

20. Robinson CD, Harvey JK. A Parallel DSMC Implementation on Unstructured Meshes with Adaptive Domain Decomposition. In Proceedings of 20th International Symposium on Rarefied Gas Dynamics, 1996: 227-232.

21. Robinson CD, Harvey JK. Adaptive Domain Decomposition for Unstructured Meshes Applied to the Direct Simulation Monte Carlo Method. Parallel Computational Fluid Dynamics: Algorithms and Results using Advanced Computers 1997: 469-476.

22. Robinson CD. Particle Simulation on Parallel Computers with Dynamic Load Balancing. PhD thesis, Imperial College of Science, Technology and Medicine, U.K, 1998.

23. Simon H. Partitioning of Unstructured Problems for Parallel Processing. Computing Systems in Engineering 1991; 2:135-148.

24. Walshaw C, Cross M, Everett M. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. Journal Parallel Distributed Computing 1997;

4'(2):102-108.

25. K ırypis G, Kumar V. Metis: Unstructured Graph Partitioning and Sparse Matrix O·dering, Version 2.0 User Manual. Minneapolis MN55455, Computer Science D‹partment, University of Minnesota, U.S.A, 1995.

26. K‹rnighan BW, Lin S. An efficient heuristic procedure for partitioning graphs, B‹ ll Systems Technical Journal 1970; 49:291-307.

27. W.ılshaw C. Parallel Jostle Library Interface: Version 1.2.1. School of C‹ mputation & Mathematical Sciences, University of Greenwich, London, SE10 9LS, UK, 2000.

28. H› permesh, Version 2.0, Altair Computing, Inc., 1757 Maplelawn, USA.

29. W‹ lshaw C. The jostle user manual: Version 2.1. School of Computation & M‹ thematical Sciences, University of Greenwich, London, SE10 9LS, UK, 1999.

30. Ni‹:ol DM, Saltz JH. Dynamic Remapping of Parallel Computations with Varying Re:ource Demands. IEEE Transactions on Computers 1988; 39:1073-1087.

31. Bü:efsch K. Investigation of Hypersonic Non-equilibrium Rarefied Gas Flow Ar‹und a Circular Cylinder by the Electron Beam Technique. Rarefied Gas Dy‹namics II. Academic Press: New York, 1969; 1739-1748.

32. Ko‹ıra K, Takahira M. Monte Carlo Simulation of Hypersonic Rarefied Nitrogen Flo v Around a Circular cylinder. In Proceedings of 20th International Symposium on Rarefied Gas Dynamics, 1996; 1236-1242.

33. Parl:er JG. Rotational and vibrational relaxation in diatomic gases, Physics of Fluids 1959; 2:449-462.

34. Wu JS, Tseng KC, Kuo CH. The Direct Simulation Monte Carlo Method Using Uns:ructured Adaptive Mesh and Its Application. International Journal for Nun‹erical Methods in Fluids 2002; 38(4):351-375.

35. Rus:‹ell DA. Density Disturbance ahead of a Sphere in Rarefied Supersonic Flow. Phy‹ ics of Fluids 1968; 11(8):1679-1685.

36. Wu .'S, Tseng KC, Wu FY. The Three Dimensional Direct Simulation Monte Carl‹ Method Using Unstructured Adaptive Mesh with Variable Time Step Scheme. Computer Physics Communications, Vol. 162/3, pp. 166-187, 2004.

37. Marl elov GN, Ivanov MS. Kinetic Analysis of Hypersonic Laminar Separated Flow‹ s for Hollow Cylinder Flare Configurations, 2000, AIAA paper 2000-2223.

Table 1. Test conditions for plume-impingement simulations of full-scale ROCSAT-3 model

Case	1	2	3	4	5	6
solar array angle[*1]	90°	90°	121°	121°	150°	150°
Mesh size (cell)	142597	142597	145852	145852	143618	143618
Simulated particles #[*2]	1.09M[*3]	1.08M	1M	1M	0.68M	0.68M
Temperature (solar panel)	173K	373K	173K	373K	173K	373K
Temperature (main body)	123K	423K	123K	423K	123K	423K
Temperature (antenna panel)	273K	273K	273K	273K	273K	273K

*1 The angle between solar panel and main body.

*2 # : number

*3 M : million

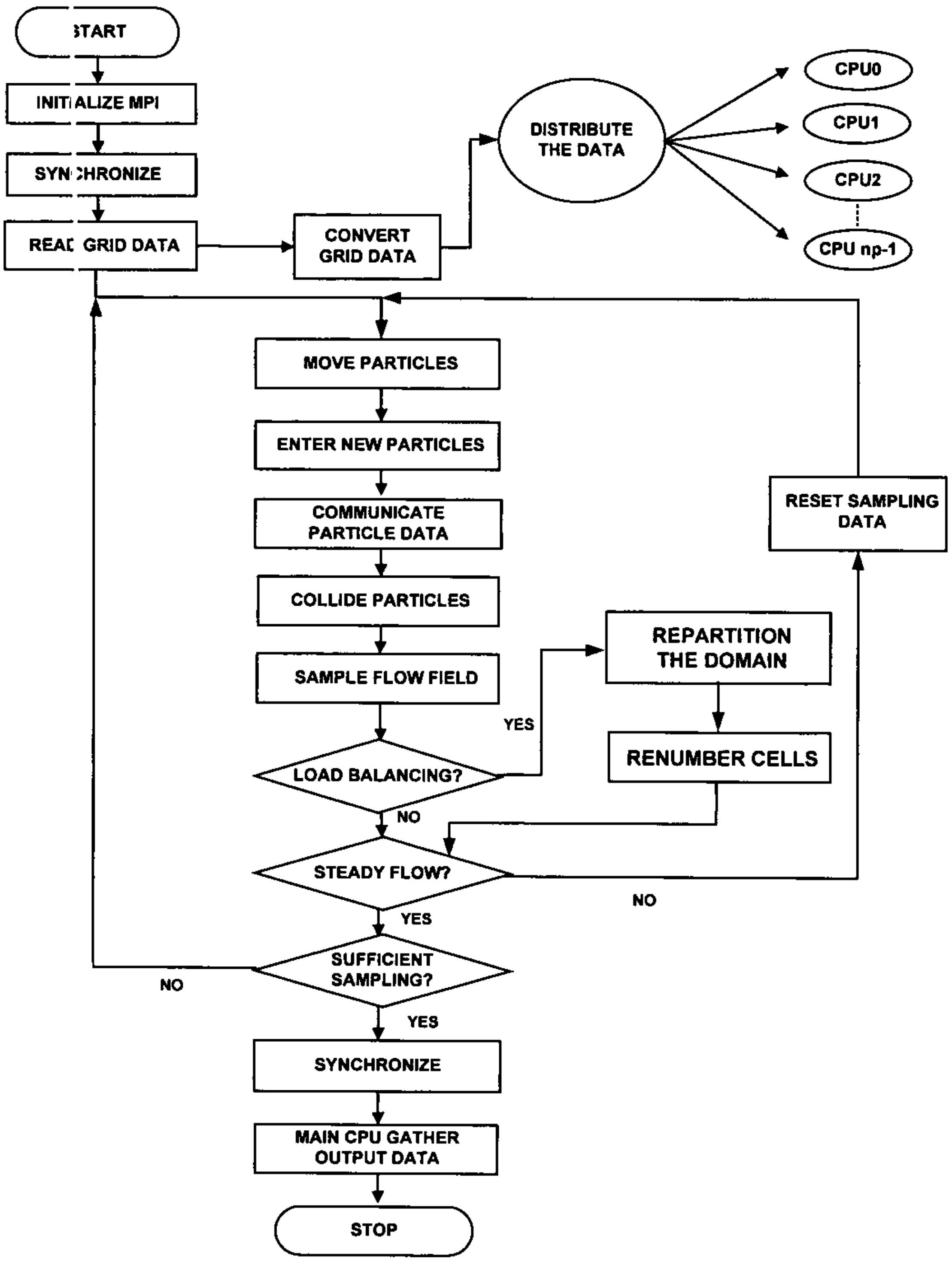


Fig. 1. Proposed flow chart for the parallel DSMC method using dynamic domain decomposition.

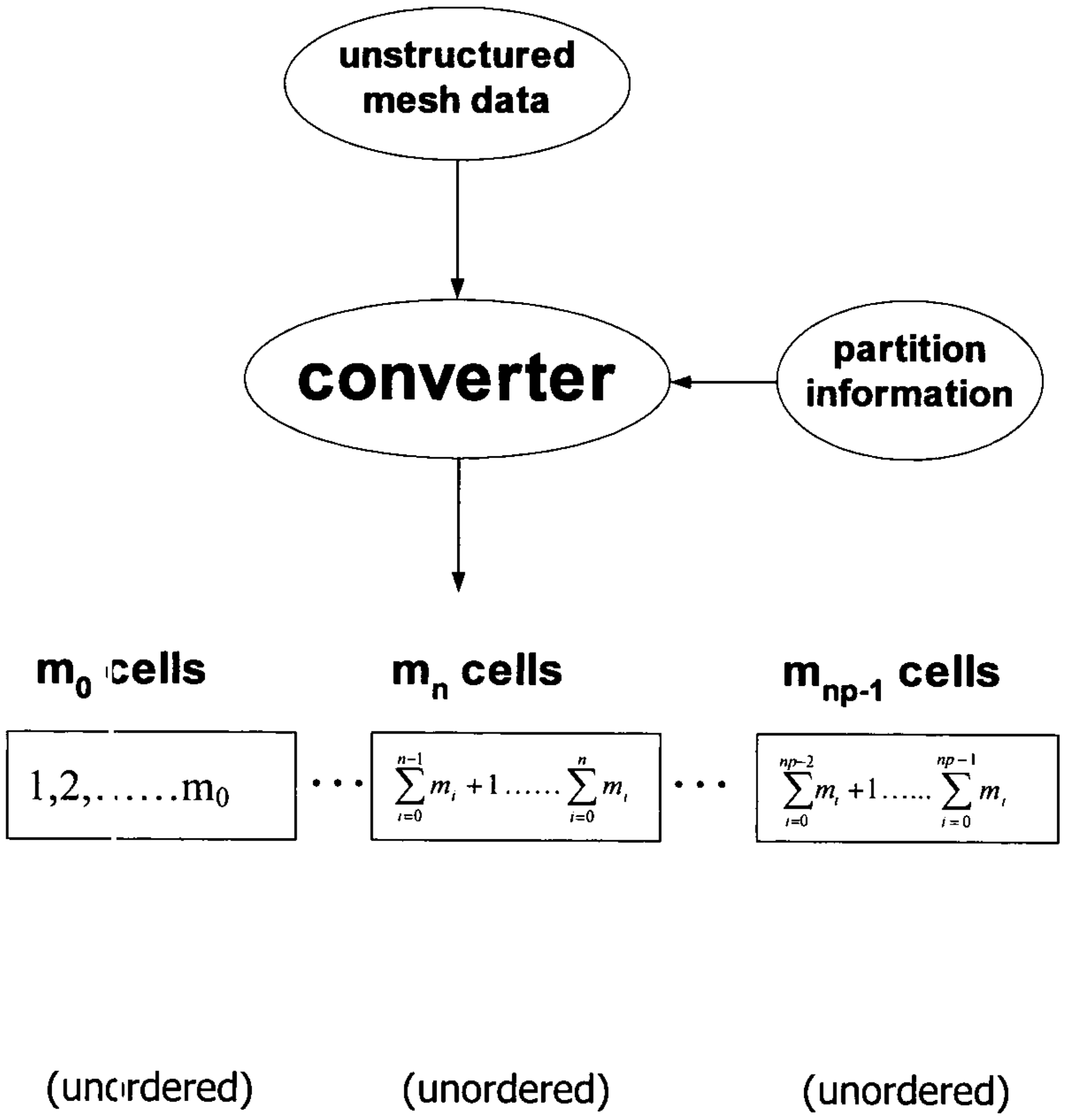


Fig. 2. Sketch of procedures for preprocessing unstructured mesh data into *globally sequential but locally unstructured* mesh data.

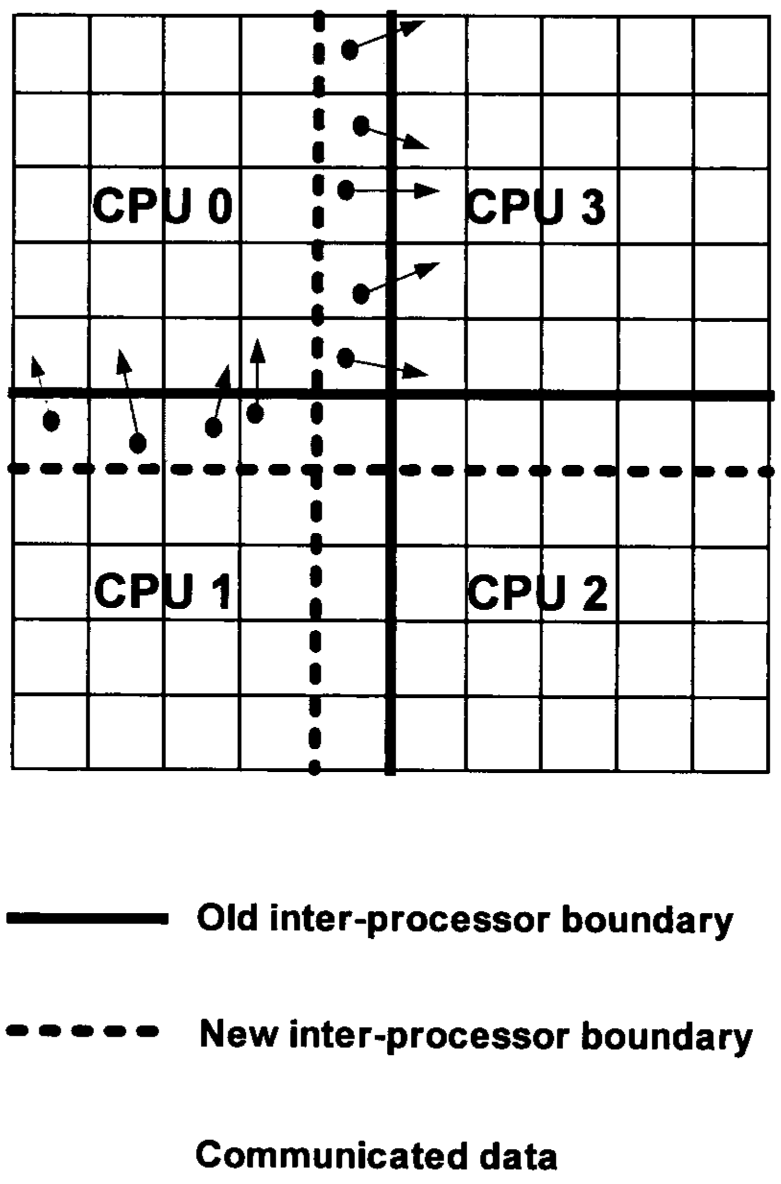


Fig. 3. Sketch of the cell/particle data migration after the repartition.

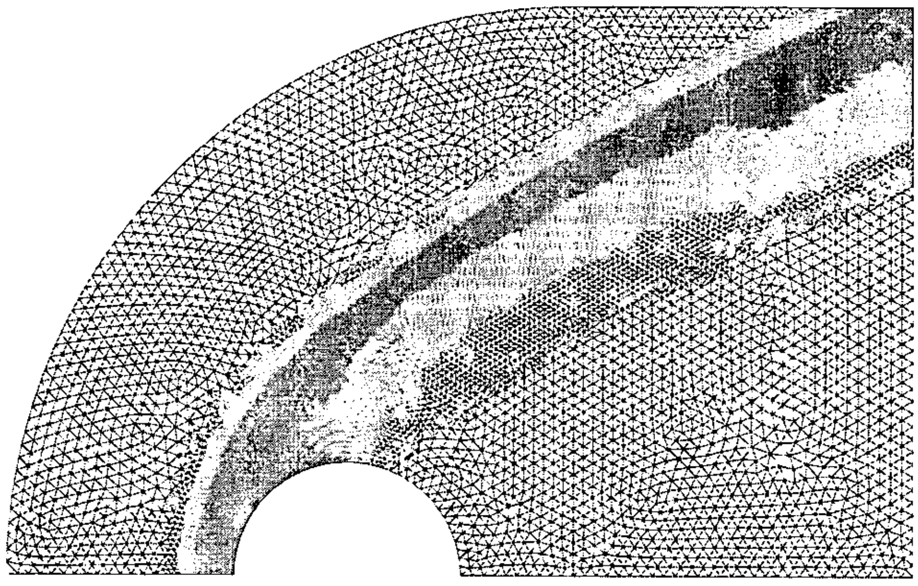
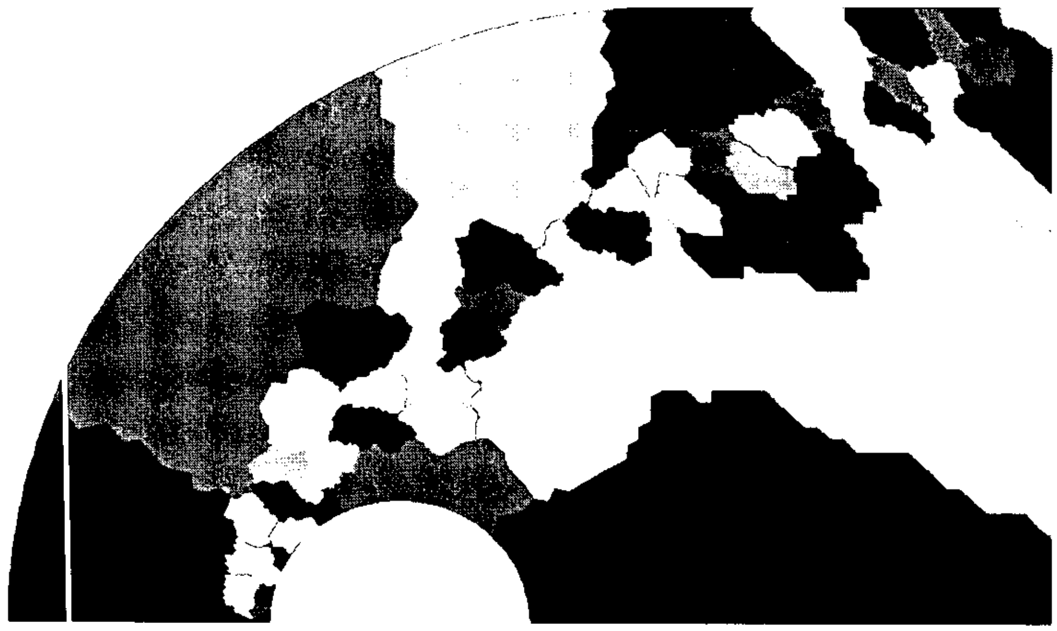
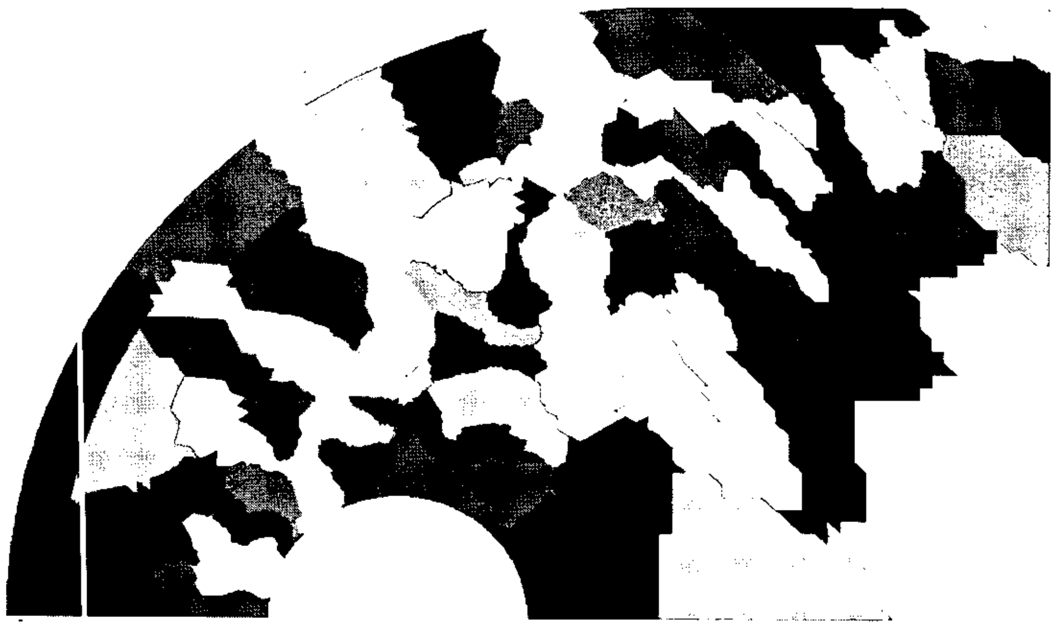


Fig. 4 The mesh of 4 levels adaptation for a two-dimensional hypersonic cylinder flow.



(a)



(b)

Fig. 5. Ir itial and final domain decomposition for 64 processors for a two dimensional hypersonic cylinder flow. (a) initial; (b) final.

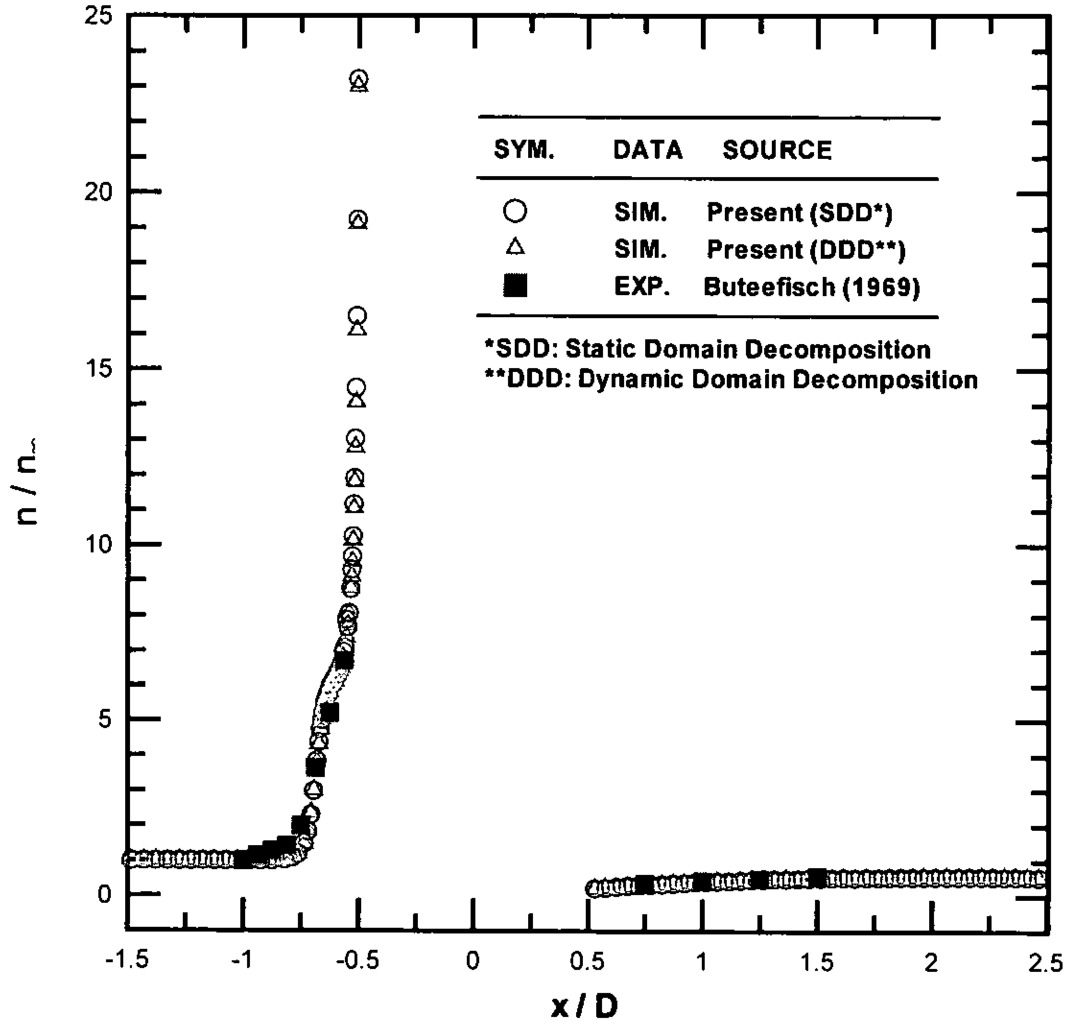


Fig. 6. Normalized density of a two-dimensional hypersonic cylinder flow with/without dynamic domain decomposition.

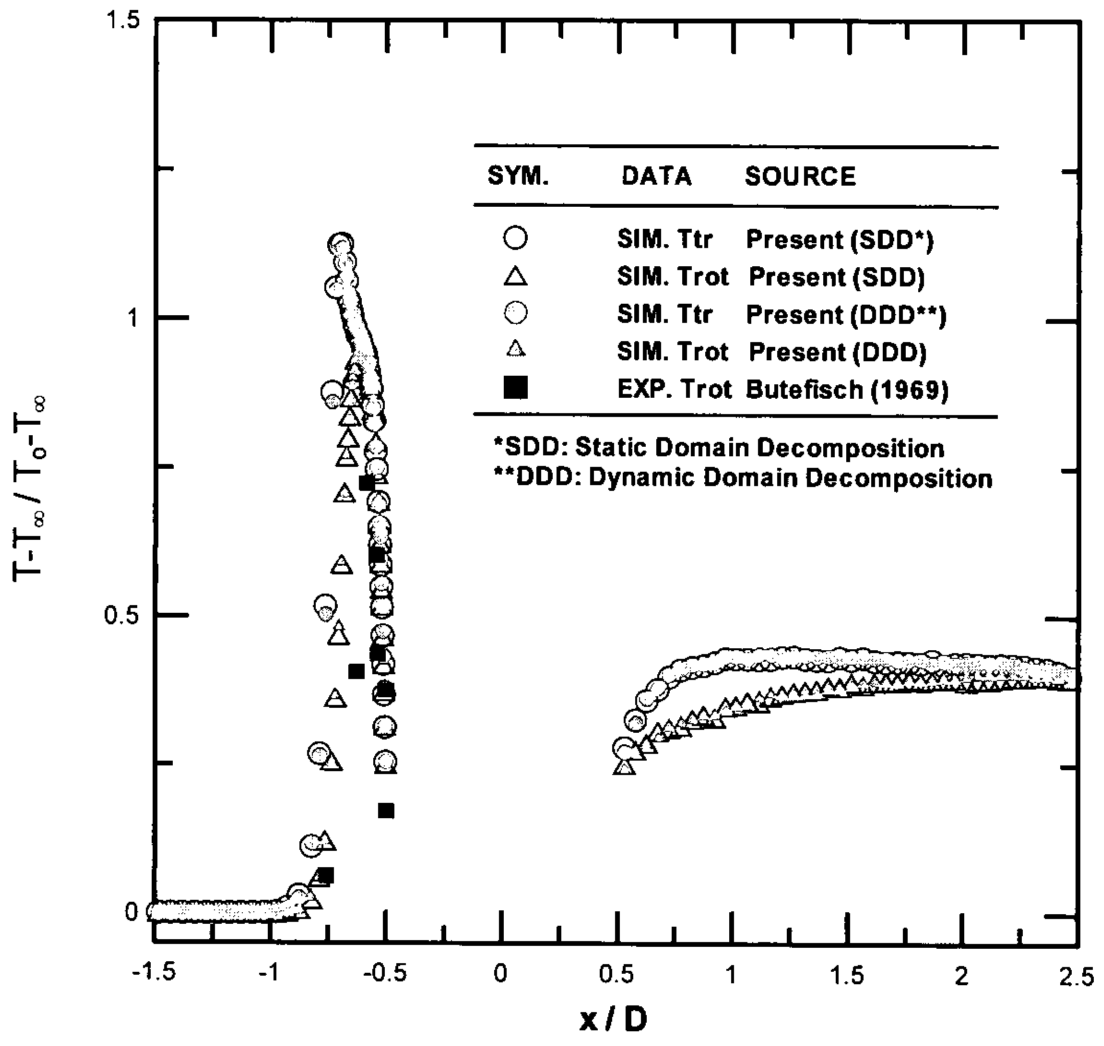


Fig. 7. Temperatures of a two-dimensional hypersonic cylinder flow with/without dynamic domain decomposition.

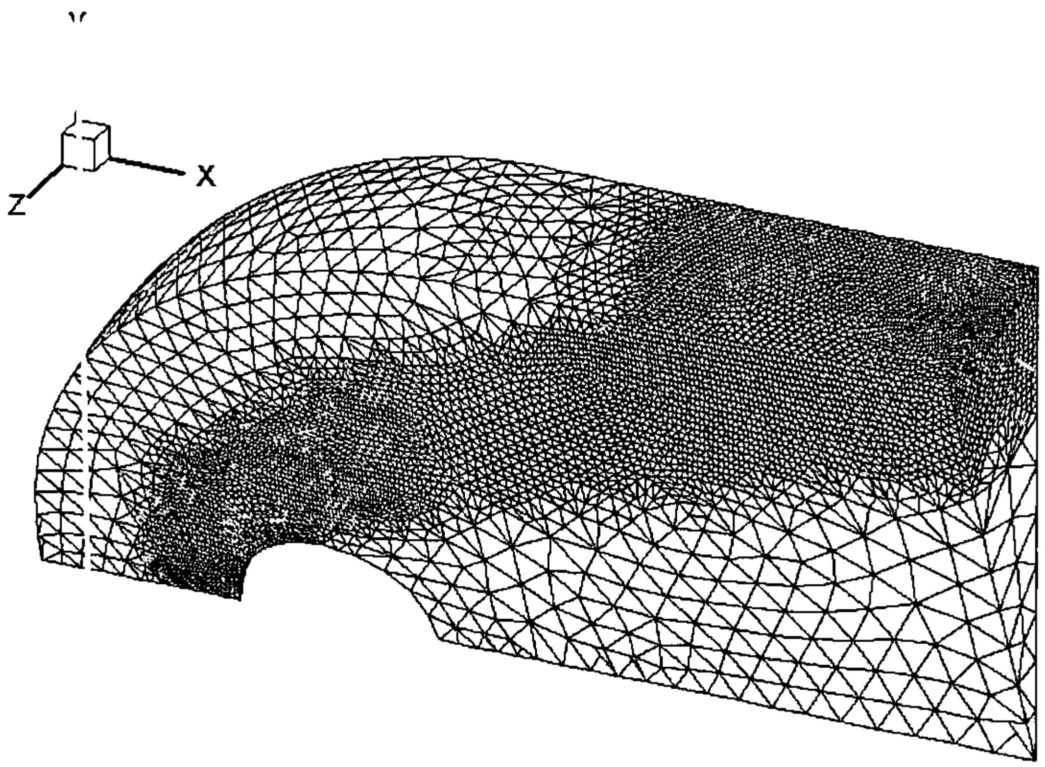
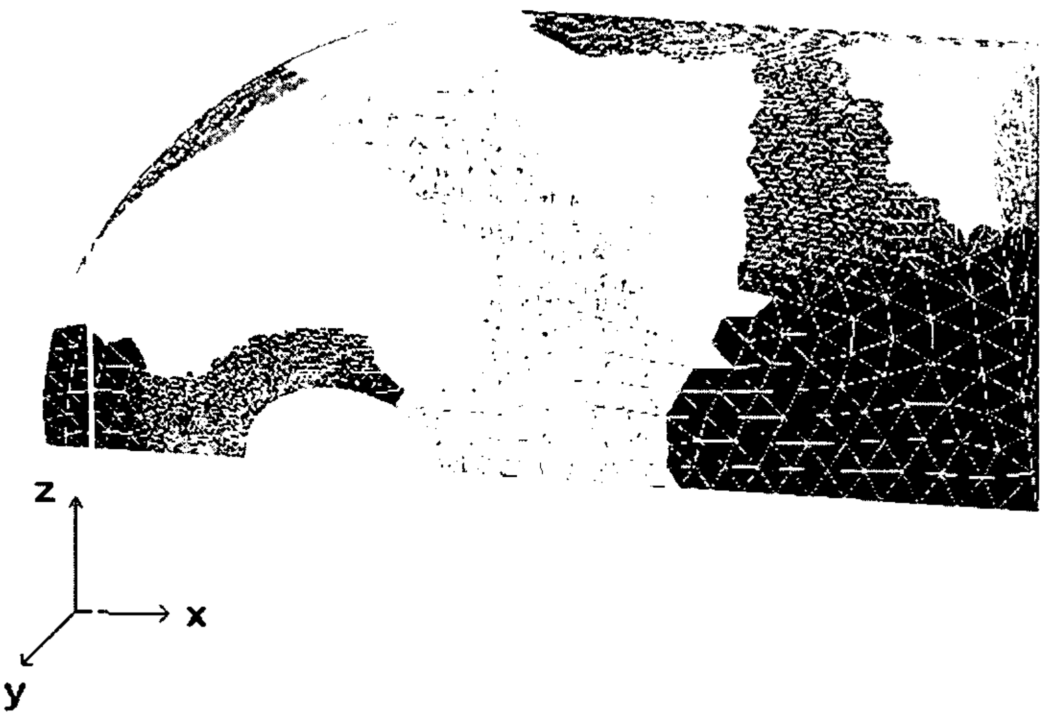
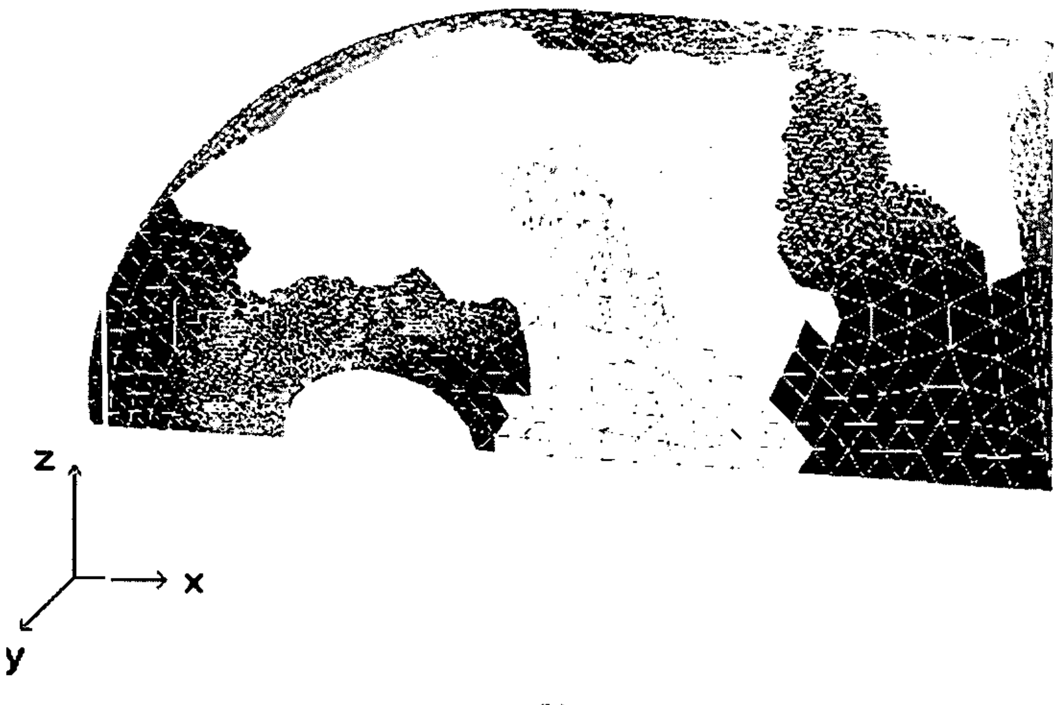


Fig. 8. The mesh of 2 levels adaptation for a three-dimensional hypersonic sphere flow.



(a)



(b)

Fig. 9. Ir itial and final domain decomposition for 8 processors for a three-dimensional hypersonic sphere flow. (a) initial; (b) final.

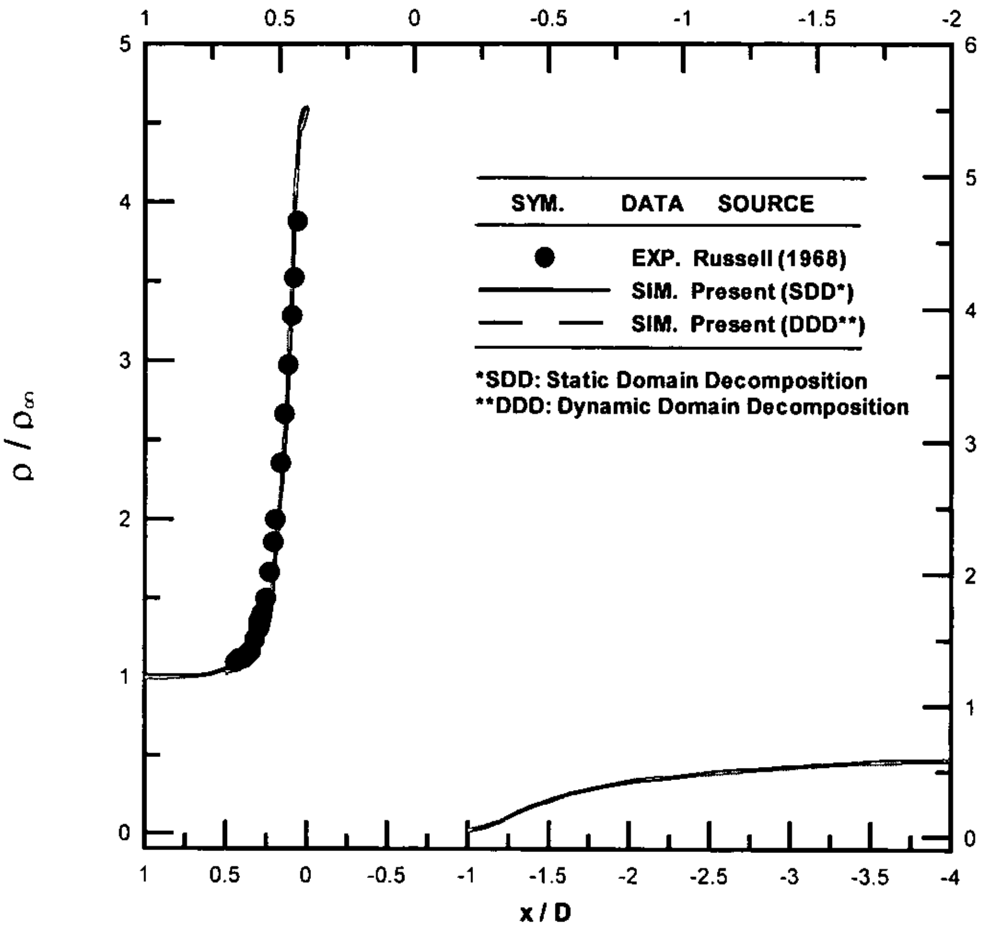


Fig. 10. Normalized density distribution along the stagnation line for a three-dimensional hypersonic sphere flow with/without dynamic domain decomposition.

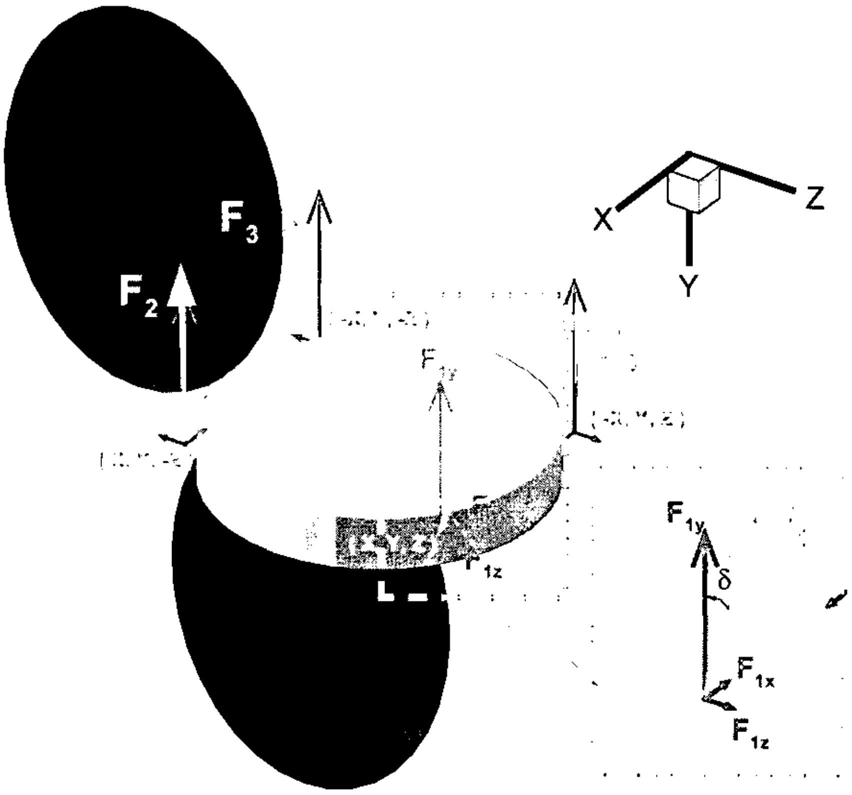


Fig. 11. Imagining photograph of ROCSAT-3

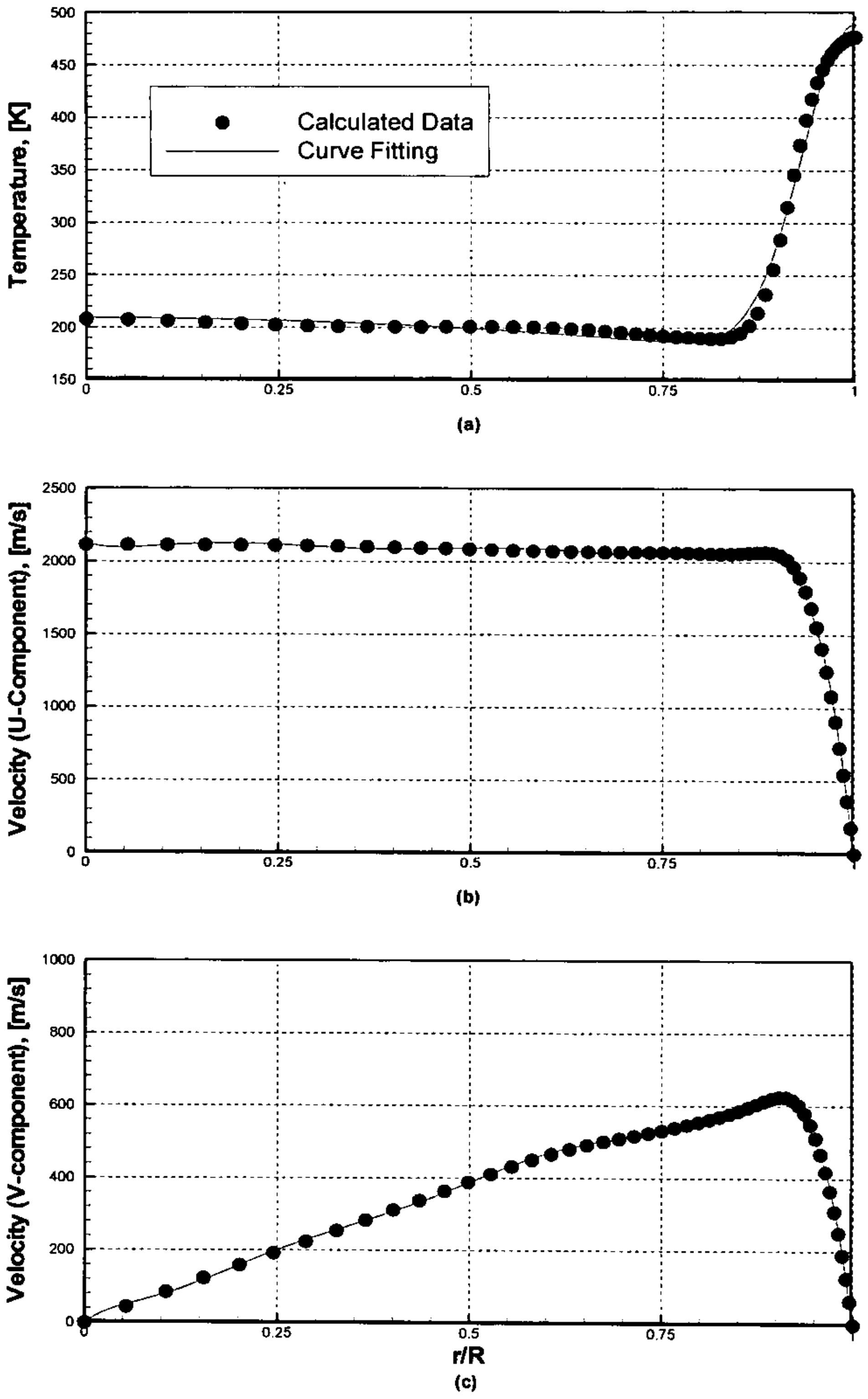


Fig. 12. The input temperature and velocities of different direction of ROCSAT-3.

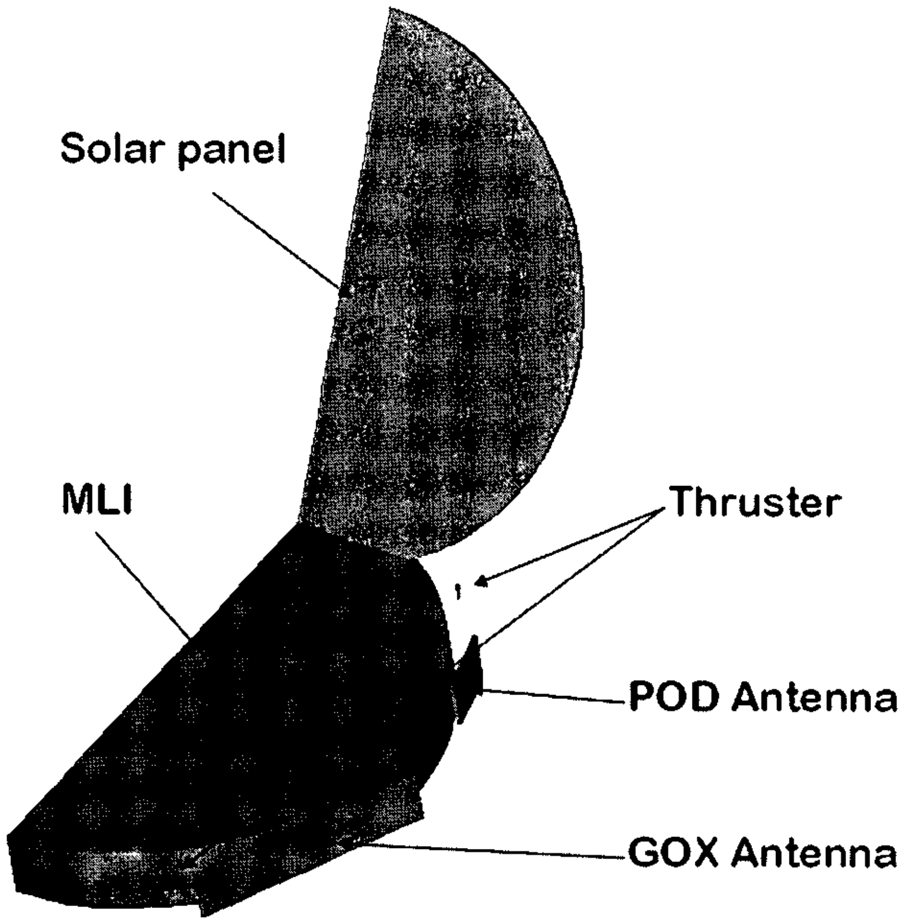


Fig. 13. ROCSAT-3 model

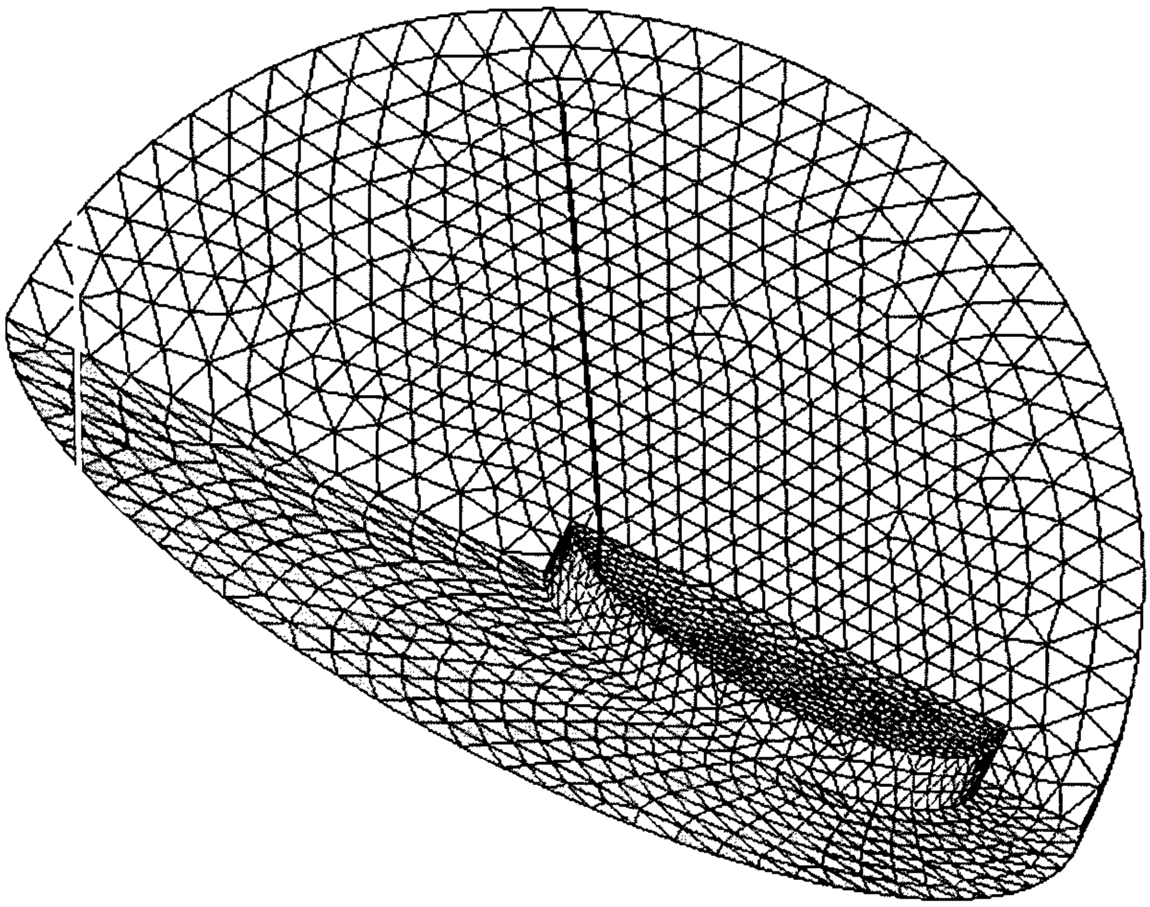
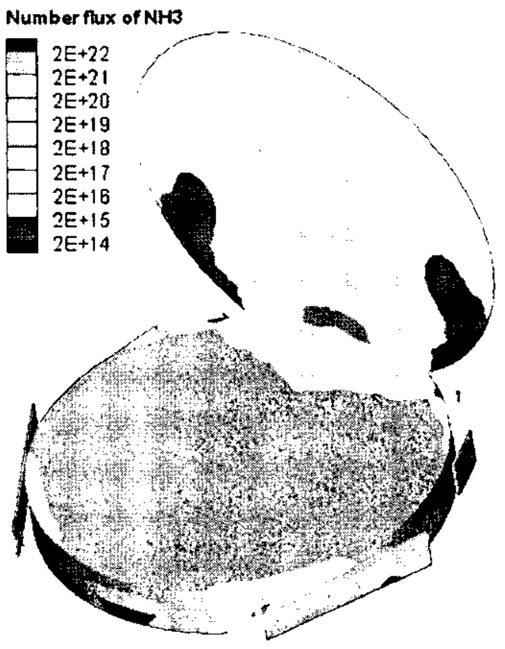
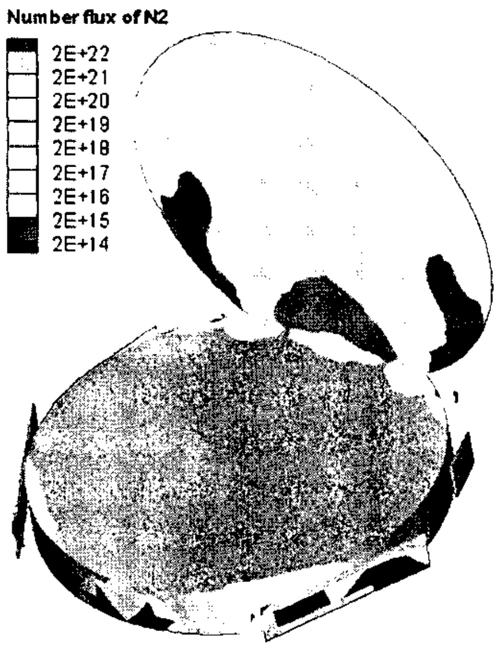


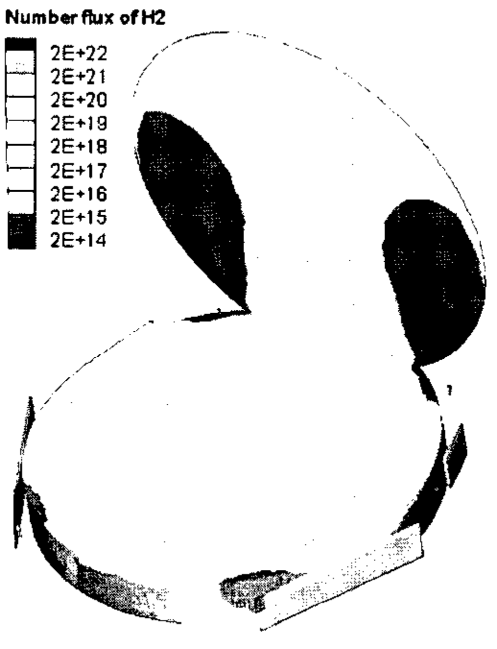
Fig. 14. The surface mesh of ROCSAT-3


Number flux of NH3
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

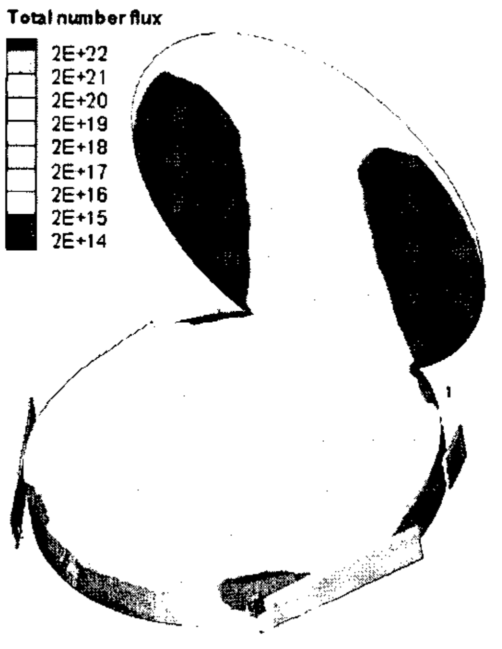
(a)


Number flux of N2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

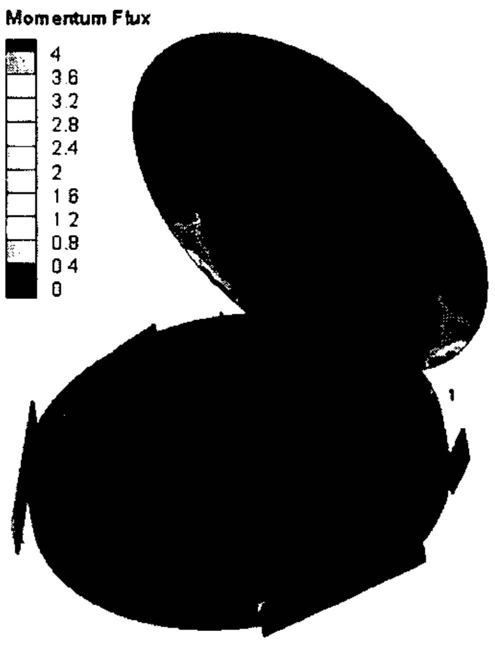
(b)


Number flux of H2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

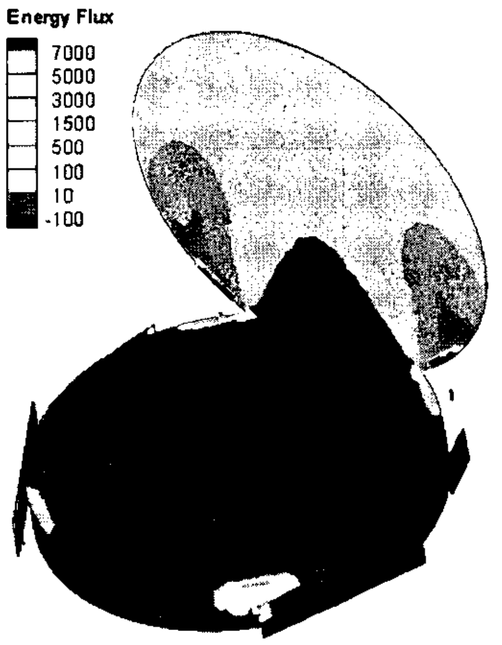
(c)


Total number flux
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

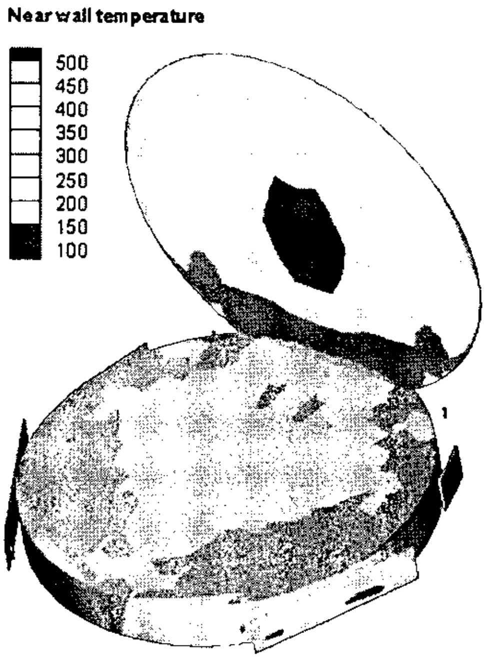
(d)



(e)



(f)



(g)

Fig 15. Surface properties case 1.

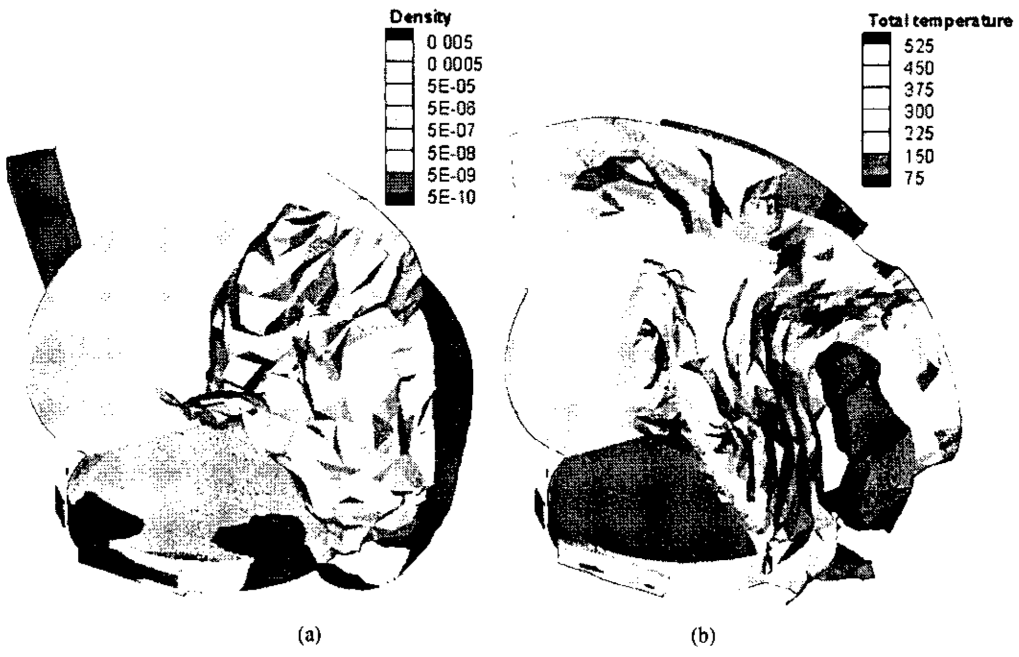
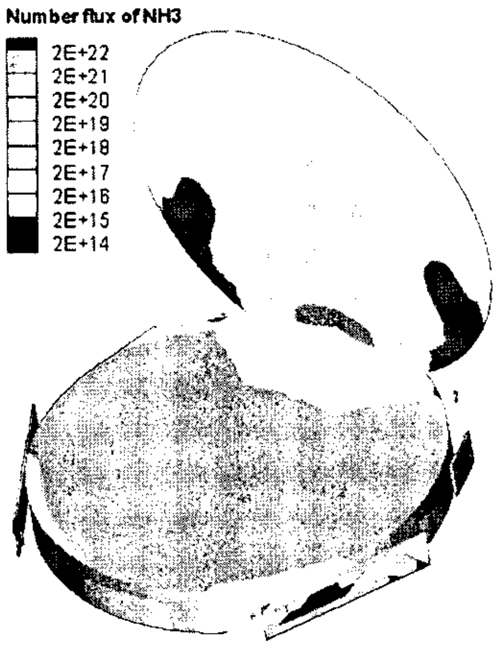
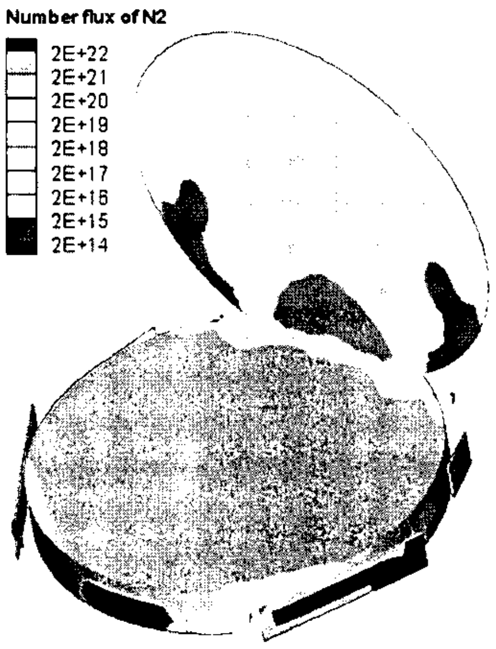


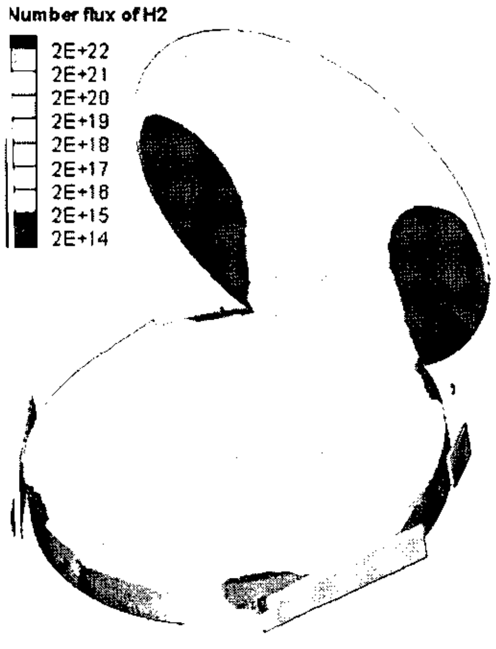
Fig 16. Iso-surface of case 1


Number flux of NH3
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

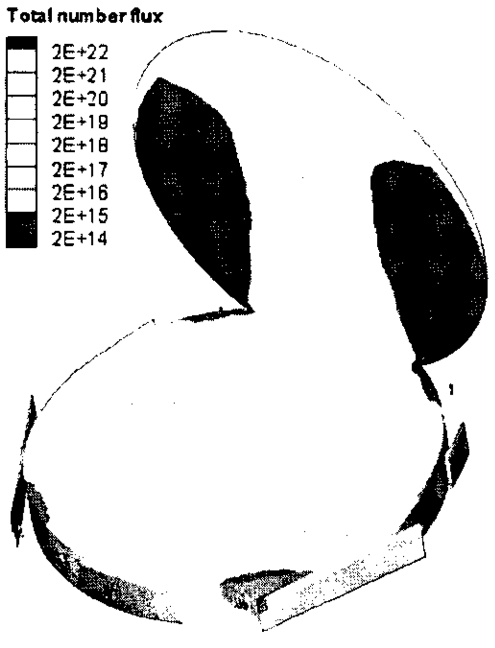
(a)


Number flux of N2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

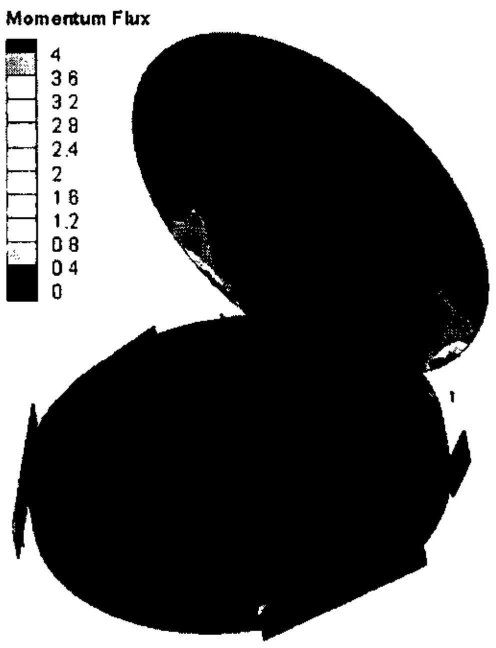
(b)


Number flux of H2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

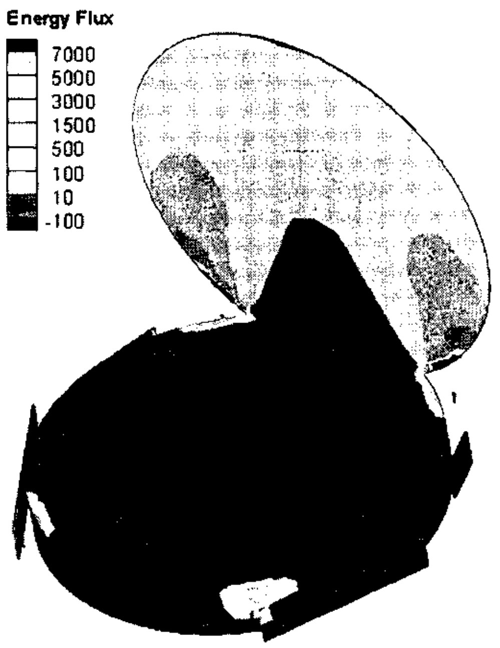
(c)


Total number flux
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

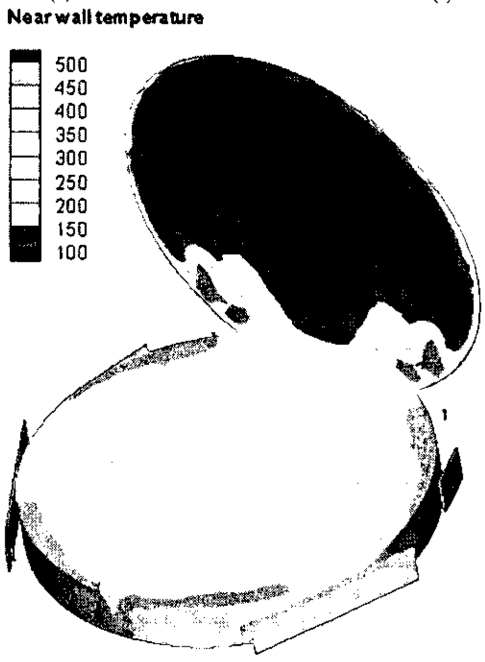
(d)



(e)



(f)



(g)

Fig 17. Surface properties case 2

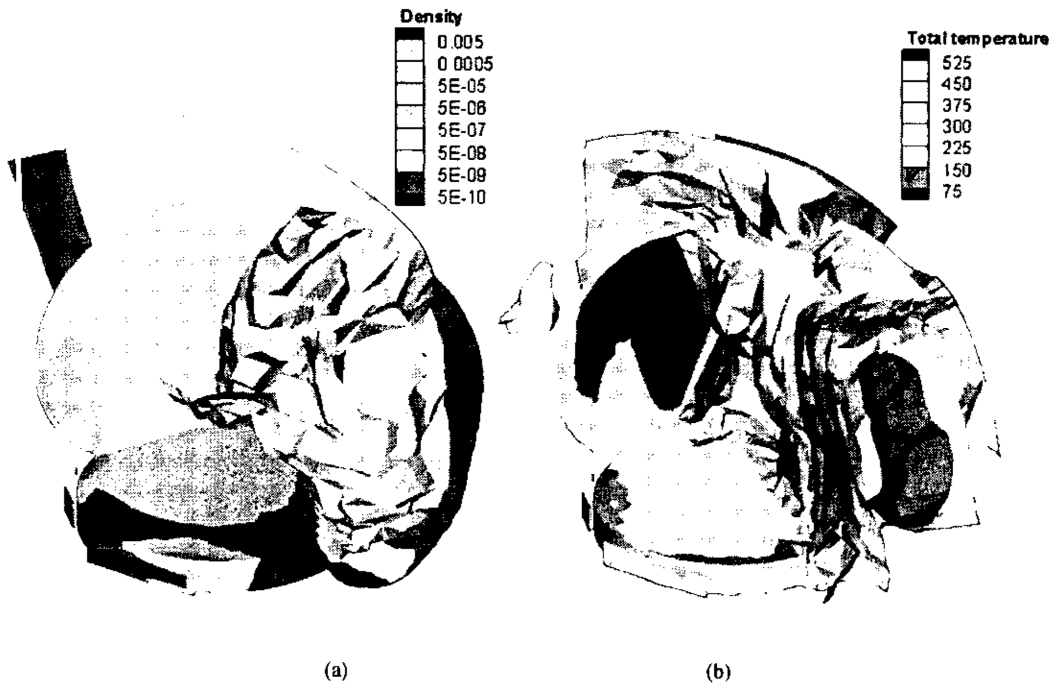
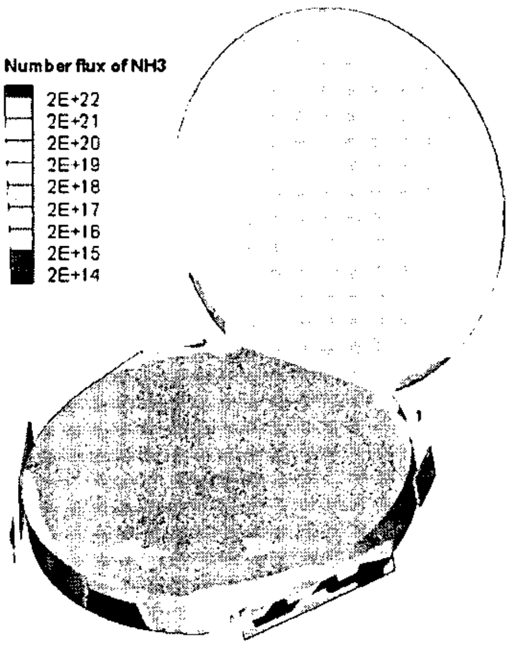
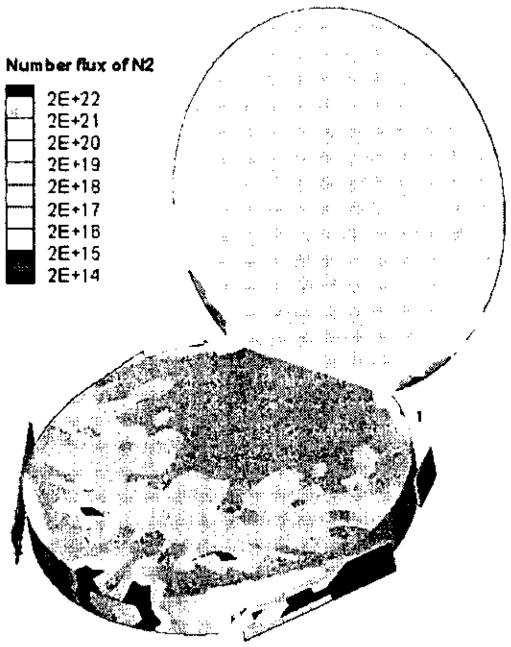
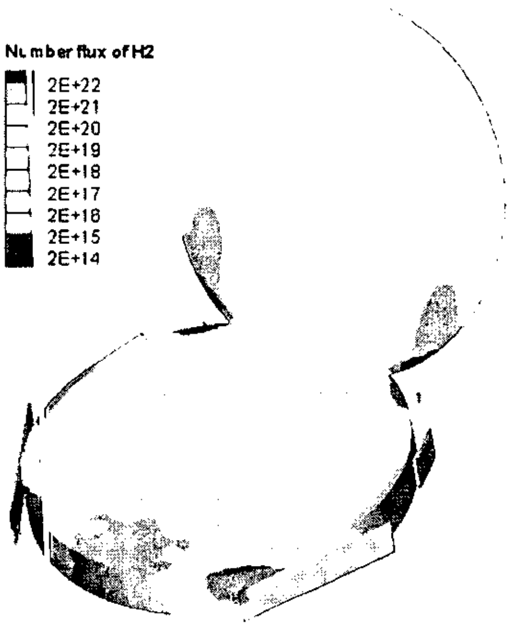
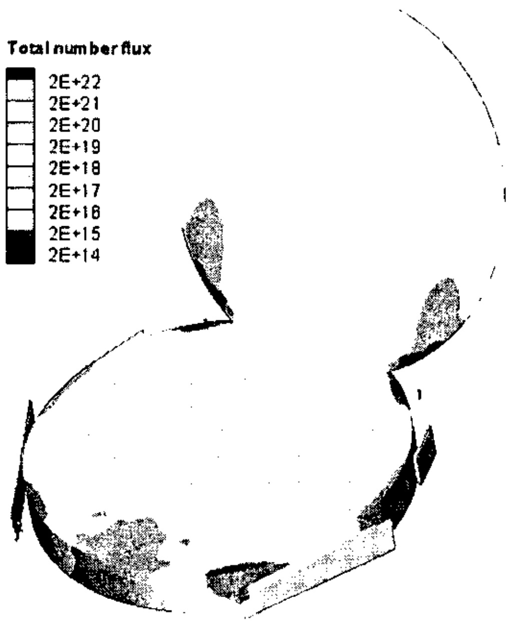


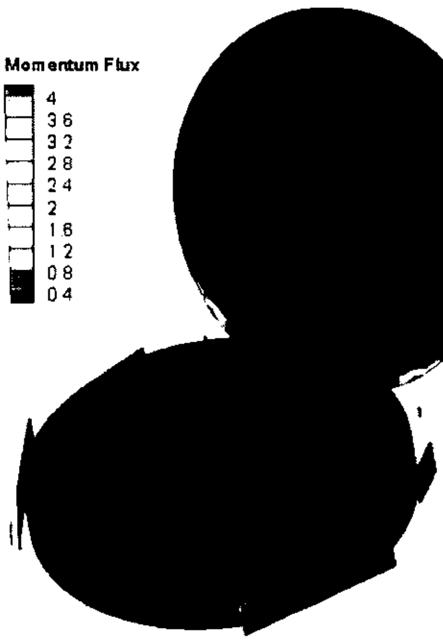
Fig 18. Iso-surface of case 2


Number flux of NH3
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14
(a)

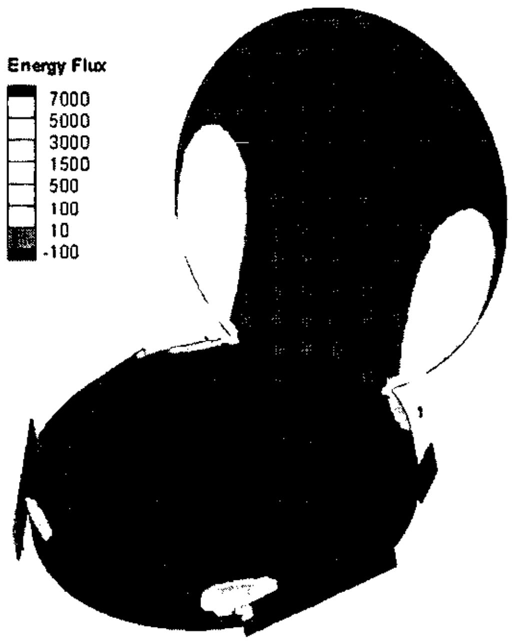

Number flux of N2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14
(b)


Number flux of H2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14
(c)

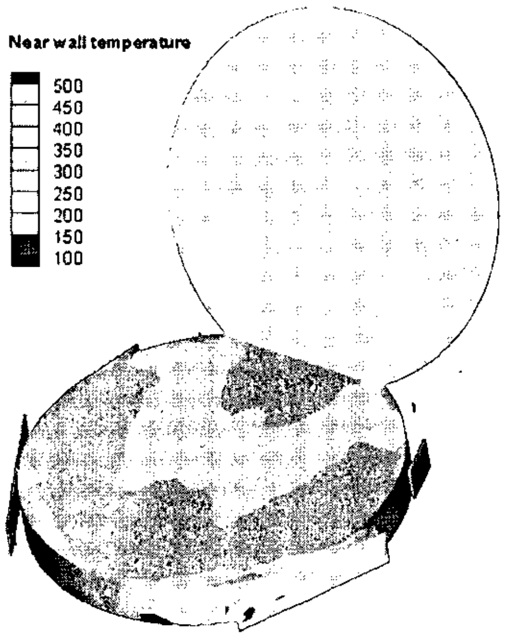

Total number flux
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14
(d)



(e)

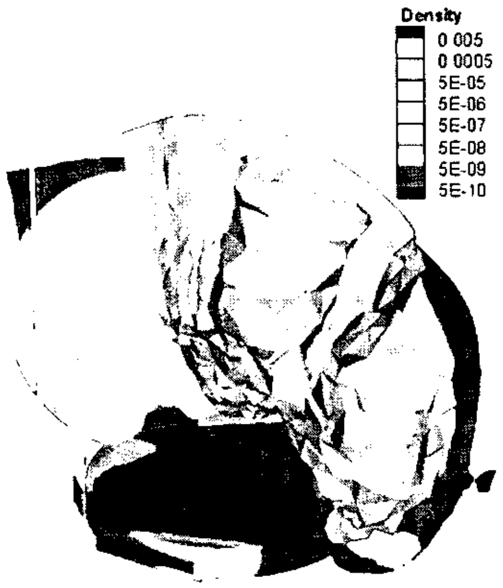


(f)

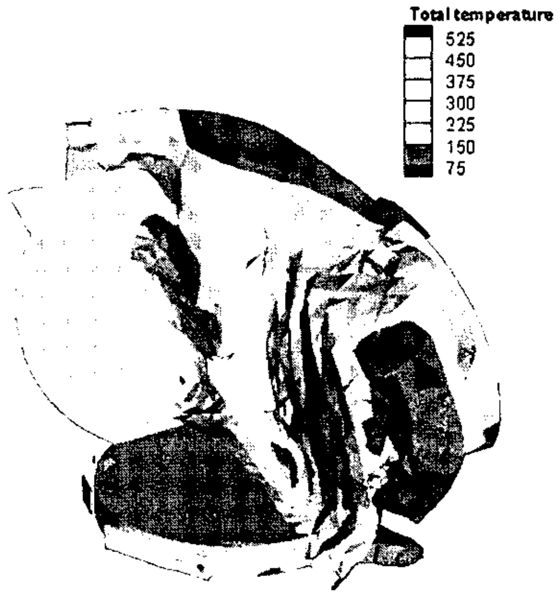


(g)

Fig 19. Surface properties case 3

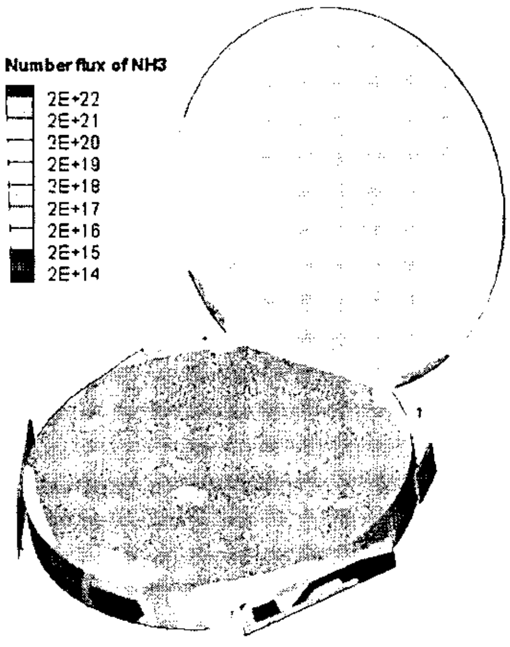


(a)

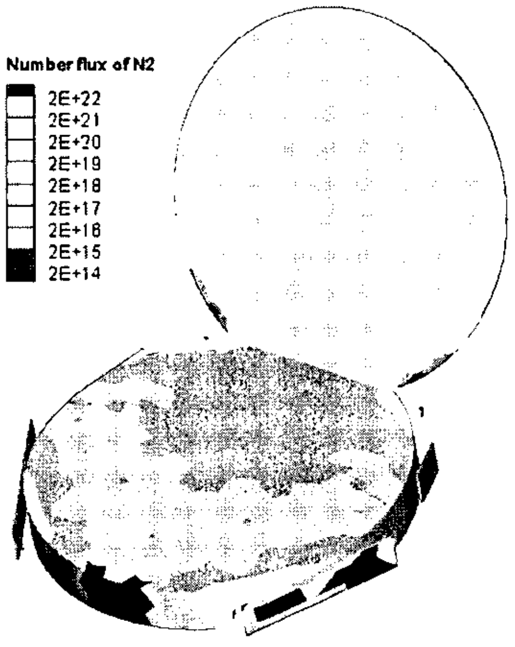


(b)

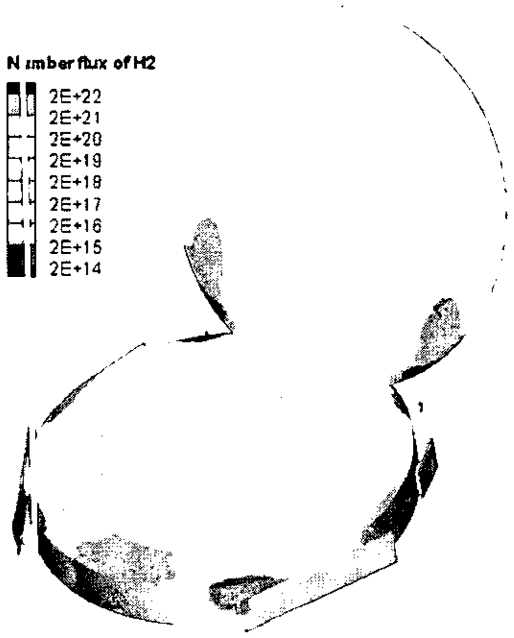
Fig 20. Iso-surface of case3


Number flux of NH3
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

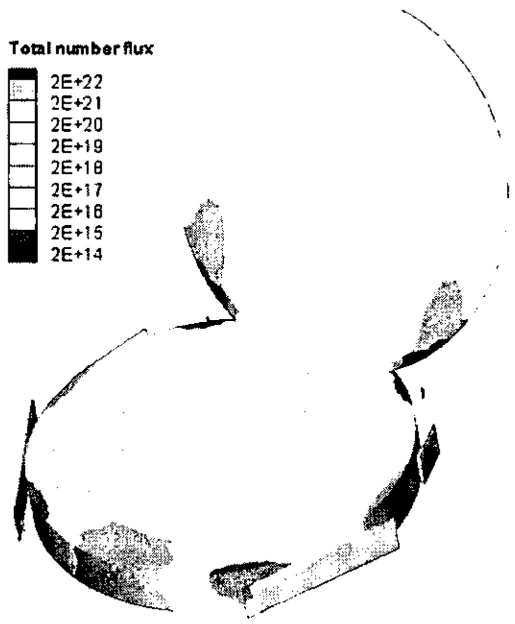
(a)


Number flux of N2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

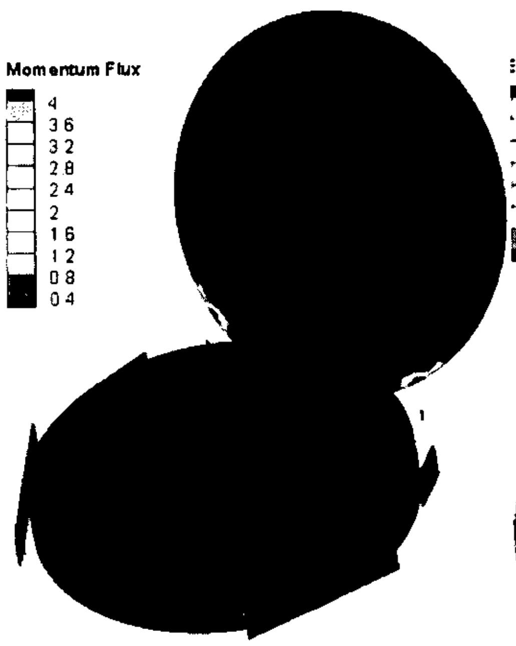
(b)


Number flux of H2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

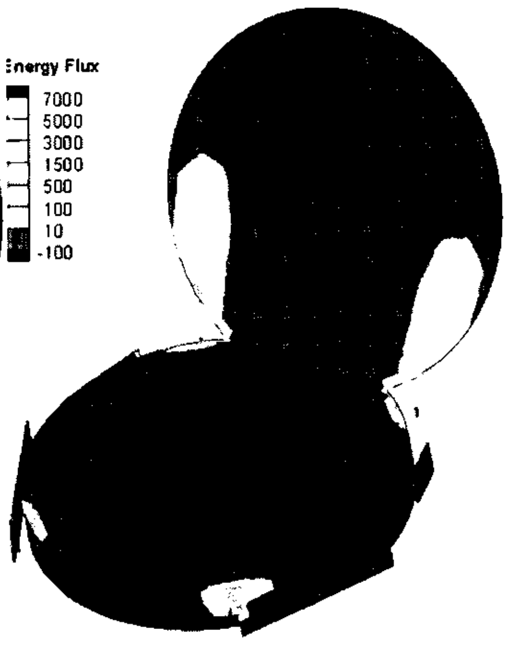
(c)


Total number flux
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

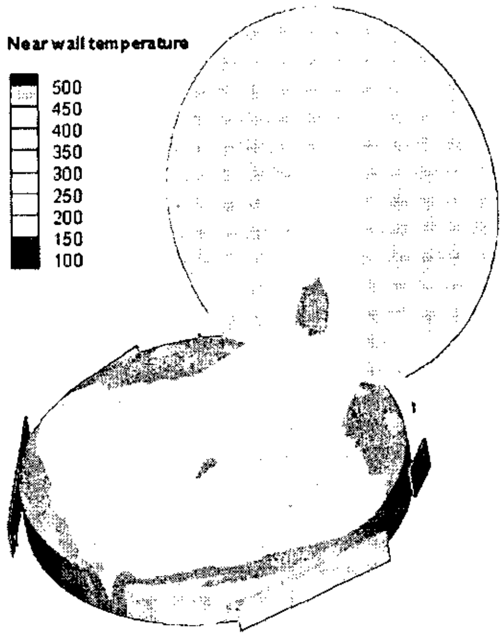
(d)



(e)



(f)



(g)

Fig 21. Surface properties case4

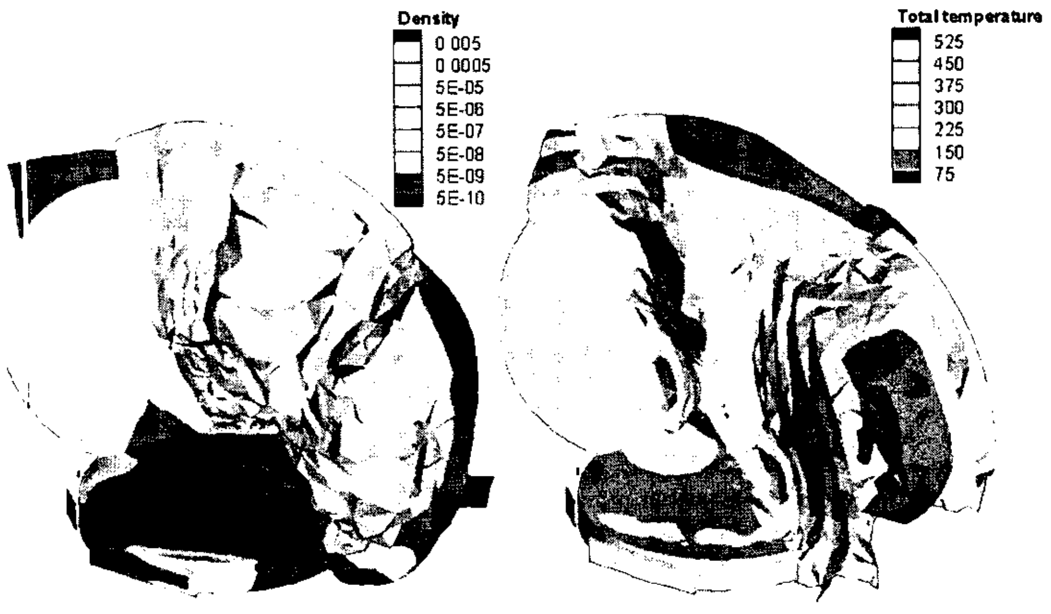
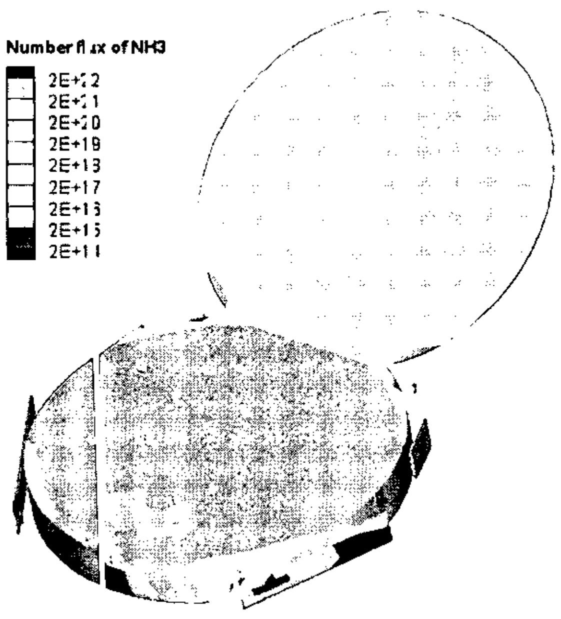
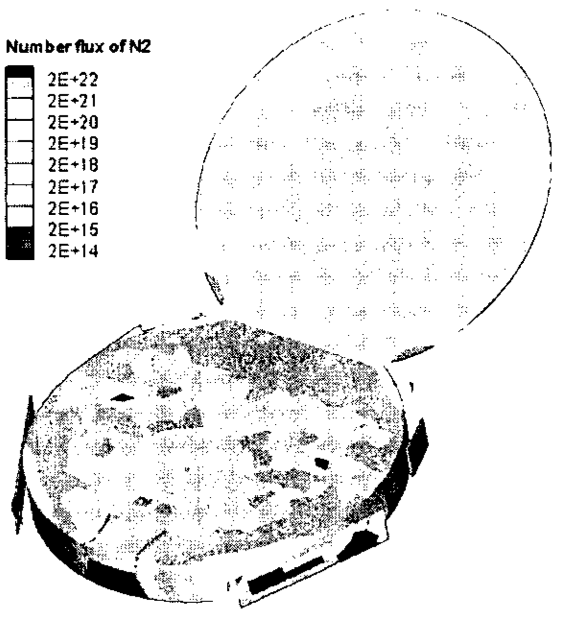


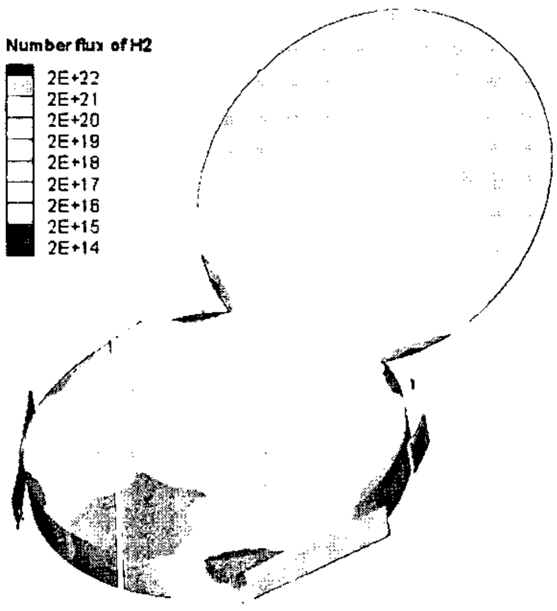
Fig 22. Iso-surface of case 4


Number flux of NH3
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

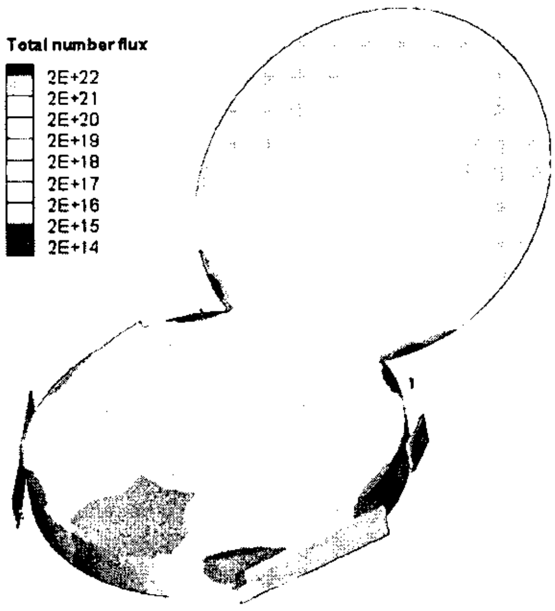
(a)


Number flux of N2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

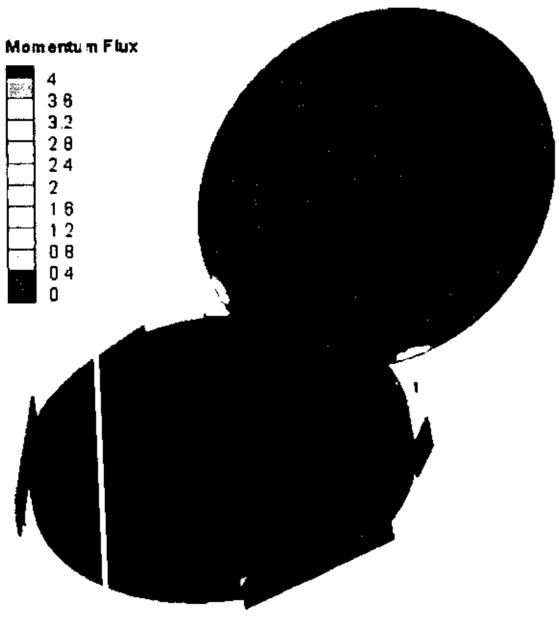
(b)


Number flux of H2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

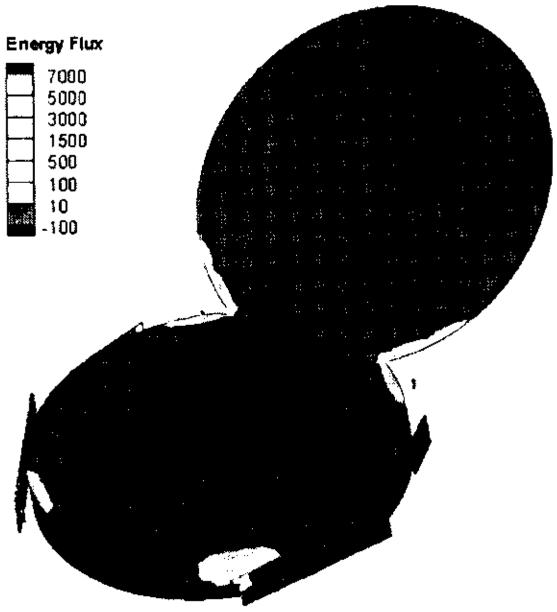
(c)


Total number flux
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

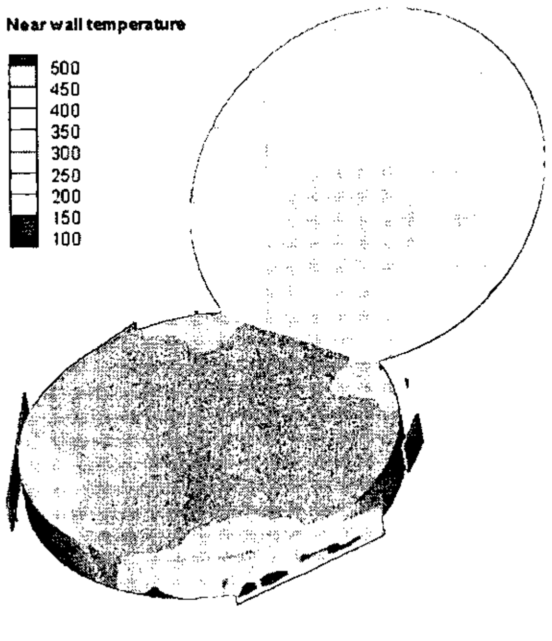
(d)



(e)

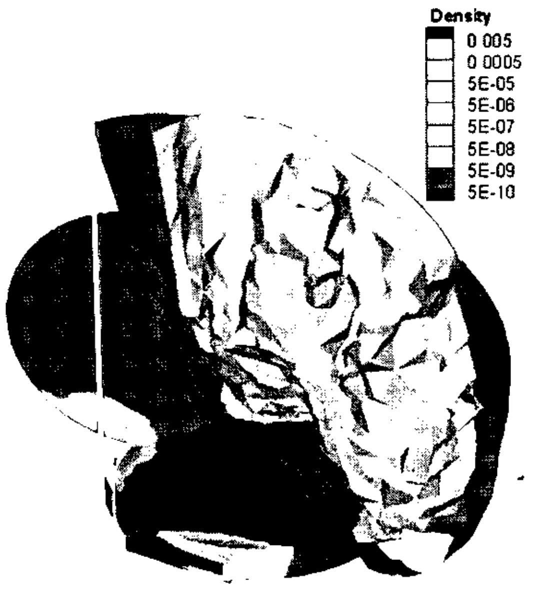


(f)

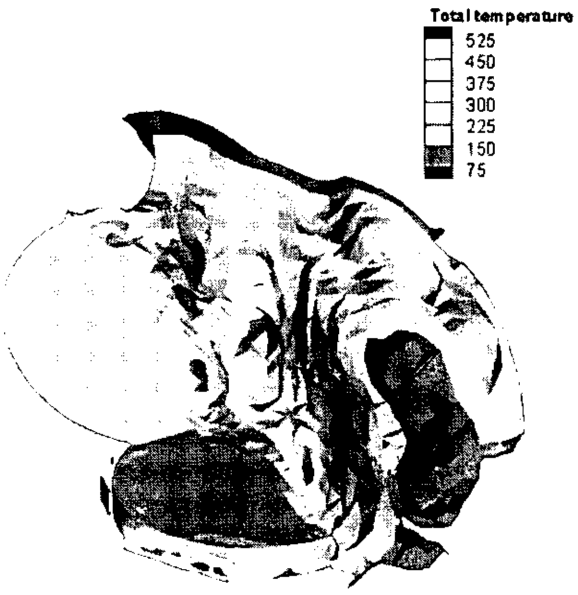


(g)

Fig 23. Surface properties case 5

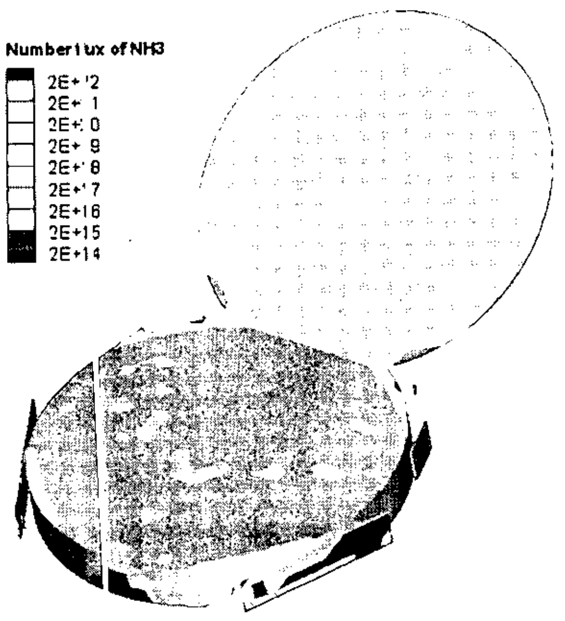


(a)

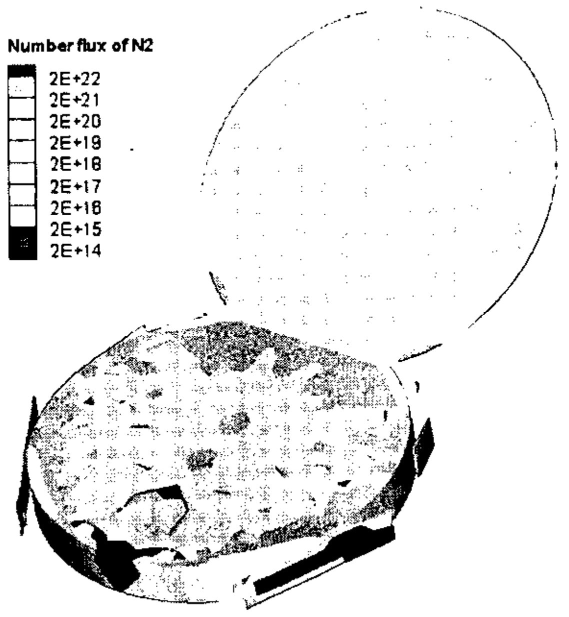


(b)

Fig 24.iso-surface of case 5

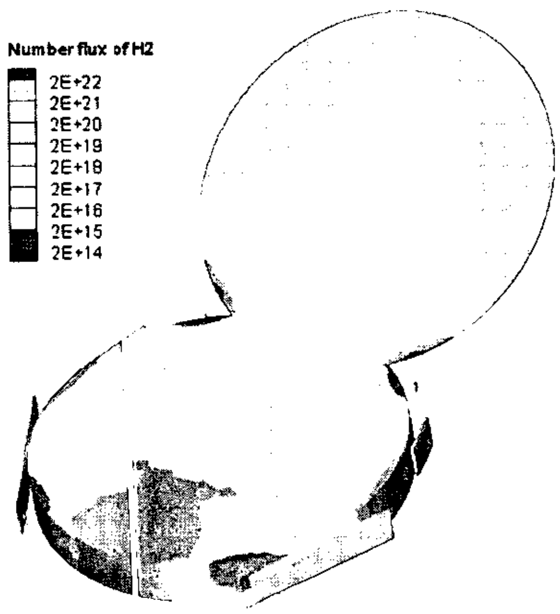


Number flux of NH3
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

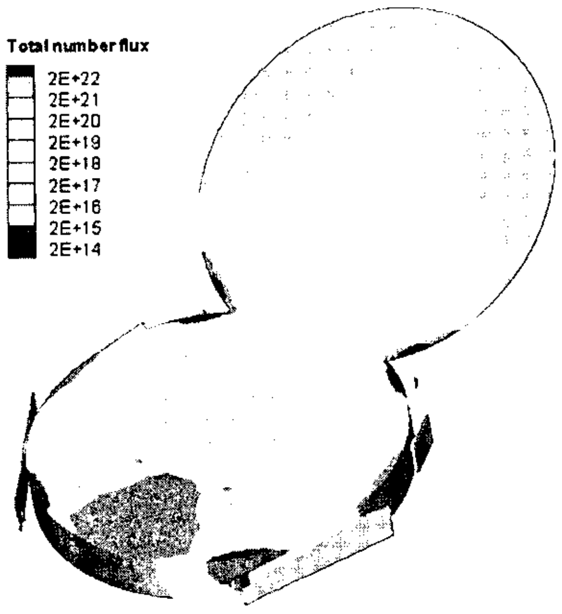


Number flux of N2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

(a) (b)

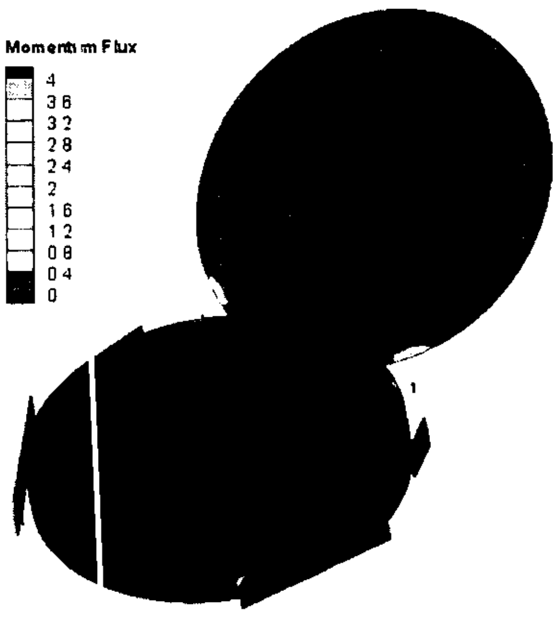


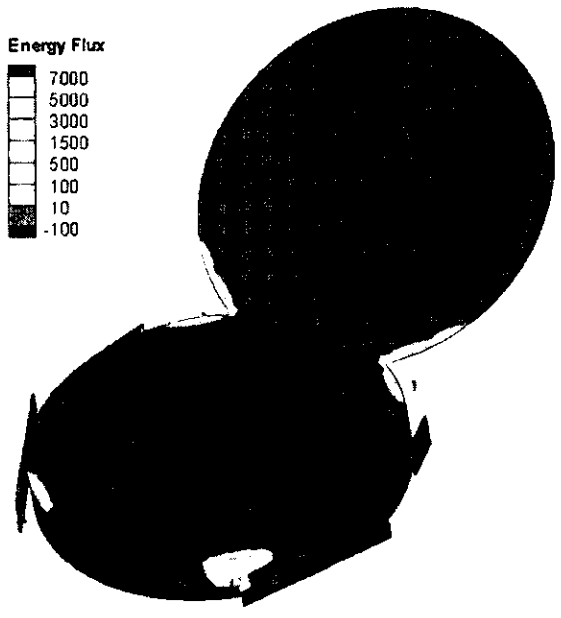
Number flux of H2
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14



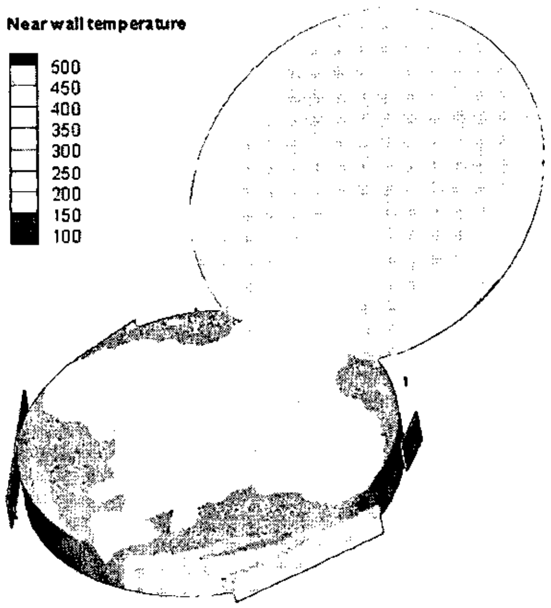
Total number flux
2E+22
2E+21
2E+20
2E+19
2E+18
2E+17
2E+16
2E+15
2E+14

(c) (d)



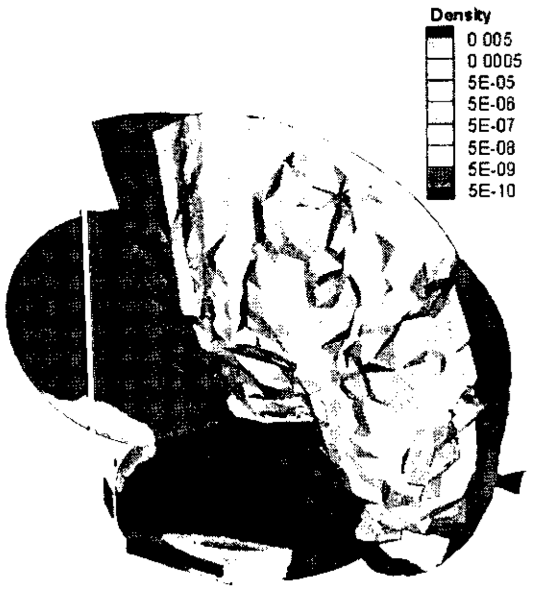


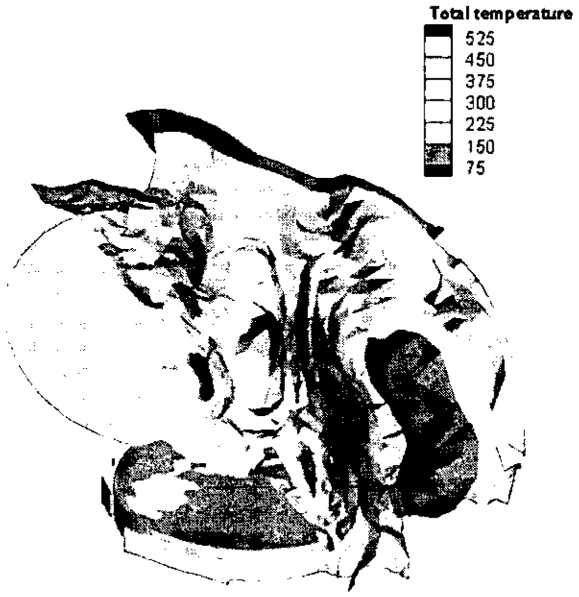
(e) (f)



(g)

Fig 25.Surface properties case 6





(a) (b)

Fig 26. Iso-surface of case 6