INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2005; **63**:37–76
Published online 14 February 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nme.1232

Parallel DSMC method using dynamic domain decomposition

J.-S. Wu*,† and K.-C. Tseng‡

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

SUMMARY

A general parallel direct simulation Monte Carlo method using unstructured mesh is introduced, which incorporates a multi-level graph-partitioning technique to dynamically decompose the computational domain. The current DSMC method is implemented on an unstructured mesh using particle ray-tracing technique, which takes the advantages of the cell connectivity information. In addition, various strategies applying the stop at rise (SAR) (IEEE Trans Comput 1988; **39**:1073–1087) scheme is studied to determine how frequent the domain should be re-decomposed. A high-speed, bottom-driven cavity flow, including small, medium and large problems, based on the number of particles and cells, are simulated. Corresponding analysis of parallel performance is reported on IBM-SP2 parallel machine up to 64 processors. Analysis shows that degree of imbalance among processors with dynamic load balancing is about $\frac{1}{6}$–$\frac{1}{2}$ of that without dynamic load balancing. Detailed time analysis shows that degree of imbalance levels off very rapidly at a relatively low value with increasing number of processors when applying dynamic load balancing, which makes the large problem size fairly scalable for processors more than 64. In general, optimal frequency of activating SAR scheme decreases with problem size. At the end, the method is applied to compute two two-dimensional hypersonic flows, a three-dimensional hypersonic flow and a three-dimensional near-continuum twin-jet gas flow to demonstrate its superior computational capability and compare with experimental data and previous simulation data wherever available. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: direct simulation Monte Carlo; parallel; graph partition; dynamic domain decomposition; hypersonic flow; near-continuum

1. 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

*Correspondence to: J.-S. Wu, Department of Mechanical Engineering, National Chiao-Tung University, 1001 Ta-Hsueh Road, Hsinchu 30050, Taiwan.
†E-mail: chongsin@cc.nctu.edu.tw
‡Graduate student.

Contract/grant sponsor: National Science Council of Taiwan; contract/grant number: NSC92-2623-7-009-003
Contract/grant sponsor: National Space Program Office; contract/grant number: NSC92-2623-7-009-003

Received September 2003
Revised August 2004
Accepted August 2004

impingement from attitude-control thrusters on satellite [2], the pumping characteristics of high vacuum pump [3], the low-pressure plasma-etching and chemical vapour 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. With the advancement of computing capability, not only is the DSMC method the practical tool for analysing 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. References [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 centre) can be weighted with the number of particles with all edges that connects cell centres, 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 two-dimensional parallel DSMC codes on an unstructured mesh incorporating the multi-level graph-partitioning technique to dynamically decompose the computational domain.
2. To utilize the above completed DSMC code to compute a high-speed, bottom-driven cavity flow for different problem sizes and study the related parallel performance using different repartitioning strategies.
3. To apply and extend the parallel DSMC implementation to compute two realistic, near-continuum two-dimensional hypersonic flows over a cylinder and a 15°-compression ramp, respectively, and two three-dimensional flows of sphere and twin-jet interaction, and compare with previous experimental and DSMC data wherever available.

The paper begins with descriptions of the parallel DSMC method and the strategies of repartitioning the domain. Results of parallel performance including speedup (efficiency), time breakdown of parallel implementation and detailed load distribution are then considered for a high-speed cavity flow, and finally applying to compute four realistic flows, in turn.

2. NUMERICAL METHOD

2.1. 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 modelled at the microscopic level using simulated particles which each represents a large number of physical molecules or atoms. The physics of the gas is modelled 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 behaviour 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.

2.2. 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.

Figure 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 $\mathbf{0}$ to $\mathbf{np-1}$ in the figure. Before detailing the proposed procedures (Figure 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^{TM} [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 Figure 2. In addition to the above, another important information output from this preprocessor is the cell-neighbouring information, which is needed for particle tracing on an unstructured mesh. Original algorithm [10] used to obtain the information of cell neighbours, 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

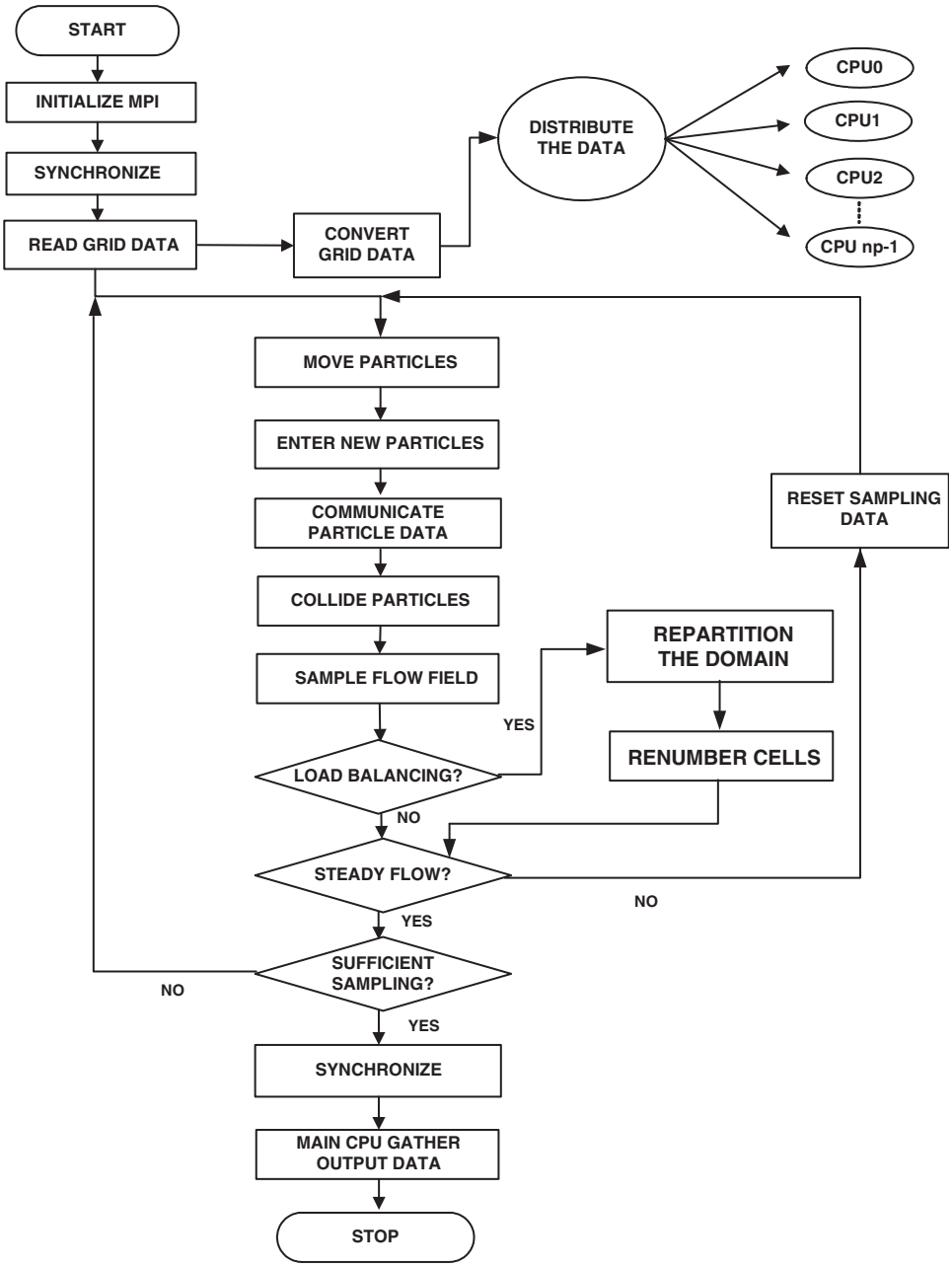


Figure 1. Proposed flow chart for the parallel DSMC method using dynamic domain decomposition.

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

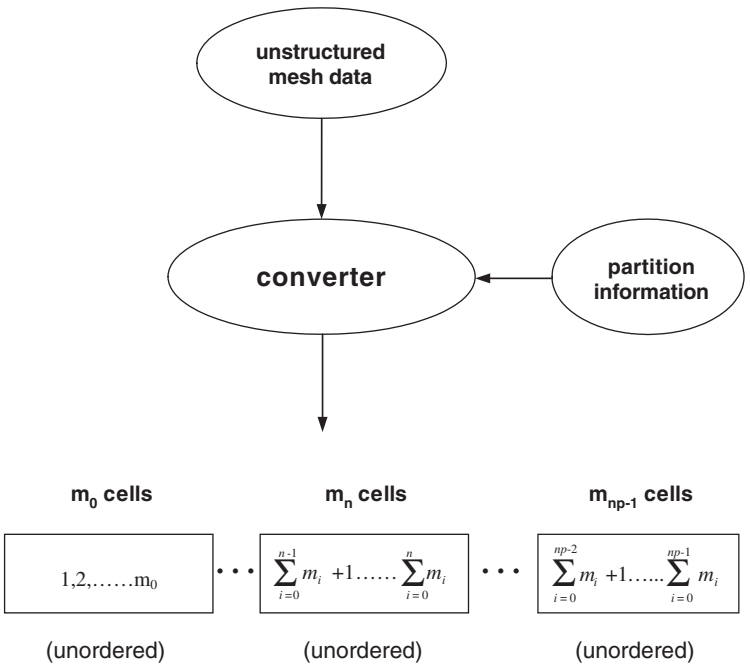


Figure 2. Sketch of procedures for preprocessing unstructured mesh data into *globally sequential but locally unstructured* mesh data.

20 min 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 $\mathbf{n}$th sub-domain, where $\mathbf{n} = \mathbf{0}$ to $\mathbf{np} - \mathbf{1}$) 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 $\mathbf{n}$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 neighbour-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-neighbour is on the order of 10 at most; therefore, the increase of memory cost due to this processor neighbour-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 Figure 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. 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.

2.3. 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, 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 tth time step, which is defined as follows:

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

where $T_{\max}(j)$ is the maximum amount of time required by any processor to complete the jth time step, $T_{\text{avg}}(j)$ is the average time required by a processor to complete the jth 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 Equation (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 $W(t) > W(t-1)$, 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.

2.4. 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 multi-level 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 centres in DSMC) within that sub-domain. However, each processor also maintains a 'halo' of neighbouring 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 $B = S_{\max}/S_{\text{opt}}$, 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.

2.5. 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 neighbour-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, a 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 neighbour-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. Figure 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 Figure 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 co-ordinates and sampled data in the cell

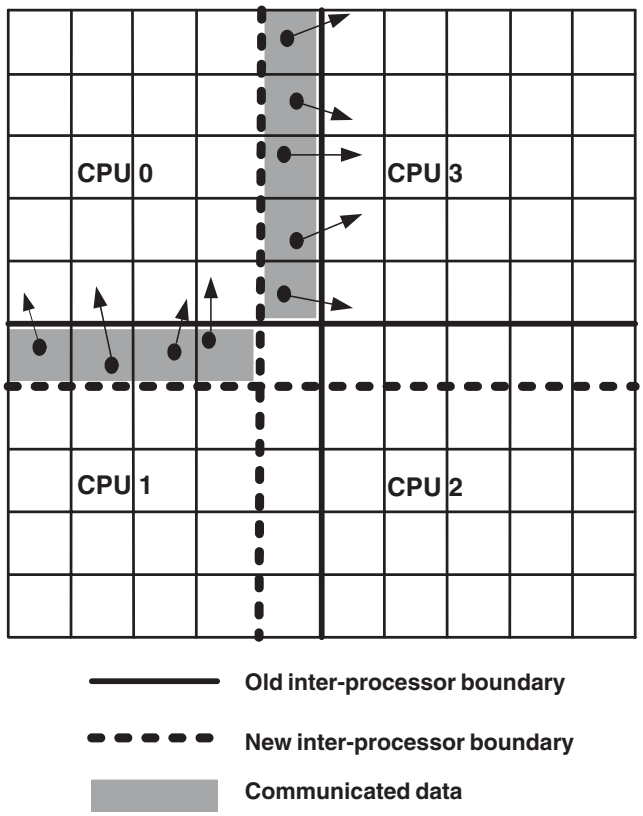


Figure 3. Sketch of the cell/particle data migration after the repartition.

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 neighbouring cell numbers accordingly.
6. Reconstruct the processor neighbour-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.

3. RESULTS AND DISCUSSIONS

In order to test and study the parallel performance of the current parallel implementation of DSMC using dynamic domain decomposition, we have used a two-dimensional, high-speed, bottom lid-driven cavity flow at different problem sizes as the test problem, similar to that used in Reference [22]. We have tested the implementation on different parallel machines, including IBM-SP2, IBM-SMP and PC cluster system, which are all memory-distributed machines. In this report, we will only describe the results on IBM-SP2 parallel machines. Note that there are 64 processors (P2SC-160-MHz) with one processor per node for IBM-SP2 machine. In what follows, the test flow conditions, preliminary simulation results of the lid-driven cavity flow, dynamic domain decomposition, parallel performance and time breakdown analysis of the parallel DSMC code will be reported in turn. Finally, applications of the current parallel implementation, incorporating an adaptive unstructured mesh and variable time-step method, to four realistic, near-continuum flows, including a two-dimensional hypersonic flow over a cylinder, a two-dimensional hypersonic flow over a ramp corner, a three-dimensional hypersonic flow past a sphere and a three-dimensional twin-jet interaction in the very near-continuum regime, are discussed to demonstrate the powerful computational capability of the current parallel implementation.

3.1. Test flow conditions

A square high-speed driven cavity flow with bottom plate moving to the right with the speed of eight times the most probable speed is considered as the test problem (Figure 4). Related flow conditions include argon gas, 300 K of wall temperature and fully diffusive wall boundary conditions. Knudsen number, based on the width of the cavity and the mean free path of the wall temperature, is 0.04. No time counter (NTC) method and variable hard sphere (VHS) molecular model [1] is used for collision kinetics and reproduction of real fluid properties, respectively. Data in the cell are sampled every two time steps in the current study, unless otherwise specified. Different problem sizes, including small, medium and large problem size, are considered for simulation (Table I), where the number of particles and the number of the cells are in the range of 225 000–3 600 000 and 11 250–180 000, respectively. Average number of particles per cell is kept approximately at 20 particles for three test problem sizes. Nearly uniform triangular mesh is used throughout the study. Simulations are run for 50 000 time steps with time step about 1/2 of the initial mean collision time step. The DSMC code is implemented on IBM-SP2 machines with the number of processors in the range of 1–64.

Table I. Number of cells and particles for three different problem sizes for the driven cavity flow.

Problem size	Small	Medium	Large
Cell numbers	11 250	45 000	180 000
Particle numbers	225 000	900 000	3 600 000

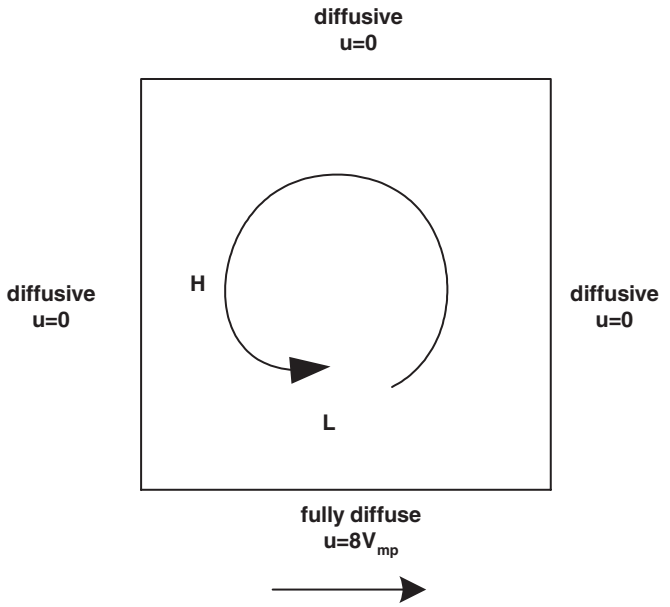


Figure 4. Sketch of the bottom, lid-driven cavity flow ($V_p = 8C_{mp}$, $T_w = 300\,K$, Ar gas, $L/H = 1$, $L = 0.32\,m$, $Kn = 0.04$).

3.2. Preliminary simulation results

Simulation results (Figure 5) show that an ultra high-density region appears at the very right-hand bottom corner due to the high-speed moving plate at the bottom of the cavity. Also the densities at the two top corners are higher than the initial value in the cavity. In addition, most of the region above the moving plate is rarefied as compared with the initial state since the particles are 'entrained' to collide with the moving plate. It is a good test problem for the effectiveness of dynamic domain decomposition in parallel DSMC implementation because it is expected that it would lead to appreciable load unbalancing among processors if static domain decomposition is used, similar to Robinson [22].

3.3. Parallel performance

Static and dynamic domain decomposition methods are both applied in this simulation to investigate the parallel performance of our DSMC code. JOSTLE [29] partition library is used to provide the initial decomposition by assigning constant particle weight on each vertex (cell centre). In other words, the number of cells of each sub-domain is approximately the same initially within the predetermined balance factor, which is 1.03 in the current study, unless otherwise specified. For dynamic domain decomposition, PJOSTLE runtime library [27] is incorporated in the code to repartition the domain based on SAR policy as the DSMC simulation proceeds. In this study, different strategies of activating SAR policy at intervals of $2\Delta t$, $10\Delta t$ and $20\Delta t$ are implemented and compared, which will be shown later.

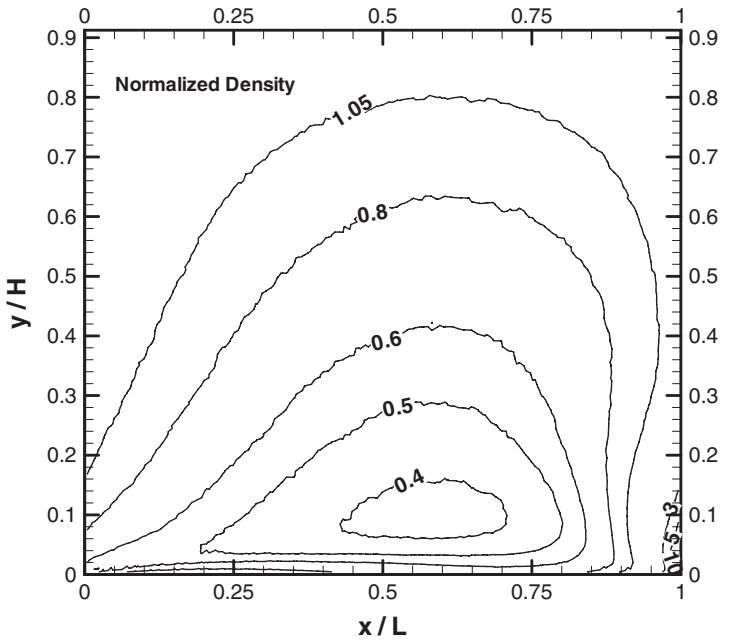


Figure 5. Normalized density contour of in the serial simulation of a high-speed driven cavity flow ($V_p = 8C_{mp}$, $T_w = 300$ K, Ar gas, $L/H = 1$, $L = 0.32$ m, $Kn = 0.04$).

Two parameters often used to measure the performance of the parallel implementation are *speedup* and *efficiency*. The former is defined as the ratio of the required running time for a particular application using *one* processor to that using N processors, i.e.

$$\text{Speedup} \equiv \frac{t_1}{t_N} \tag{2}$$

Efficiency is then defined as

$$\text{Efficiency} = \frac{\text{Speedup}}{N} \tag{3}$$

and is just the ratio of the true speedup to the ideal speedup, N, i.e. the number of processor, and hence its value normally lies between zero and one. However, there is exception when superlinear speedup occurs where the parallel efficiency may exceed unity.

Results of parallel speedup and efficiency of the cavity-flow computation, at different problem sizes on IBM-SP2 machine, as a function of the number of processors are presented in Figure 6. As expected, the parallel performance of those using dynamic domain decomposition is much better than those using static domain decomposition. Several trends for different problem sizes are described in detail as follows.

3.3.1. Small problem size. Super-linear speedup (efficiency $> 100\%$) occurs clearly for number of processors less than or equal to 16, if dynamic domain decomposition is applied

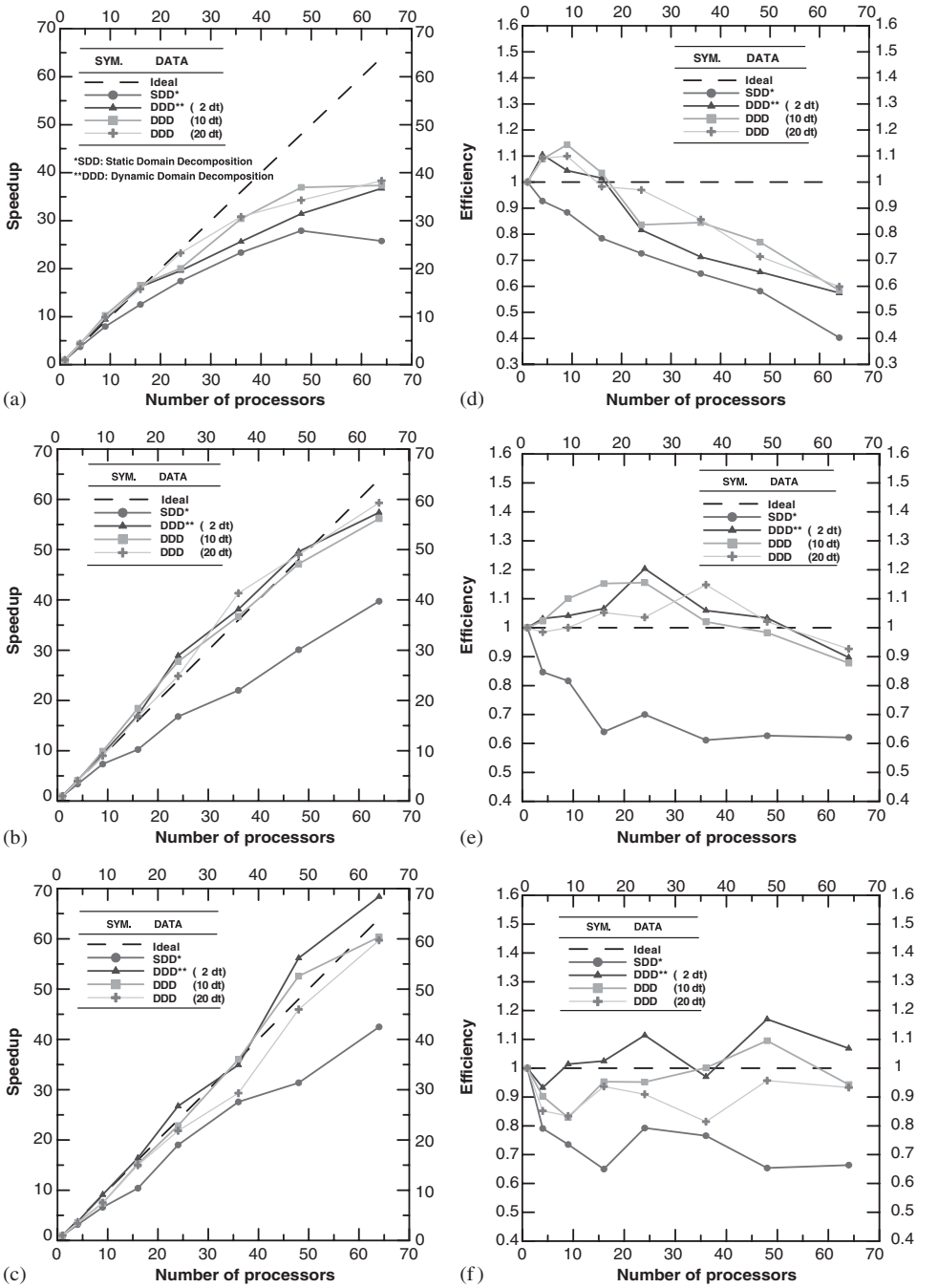


Figure 6. Parallel speedup and efficiency as a function of number of processors for high-speed driven cavity flow at different problem sizes on IBM-SP2 machine (maximum 64 processors).

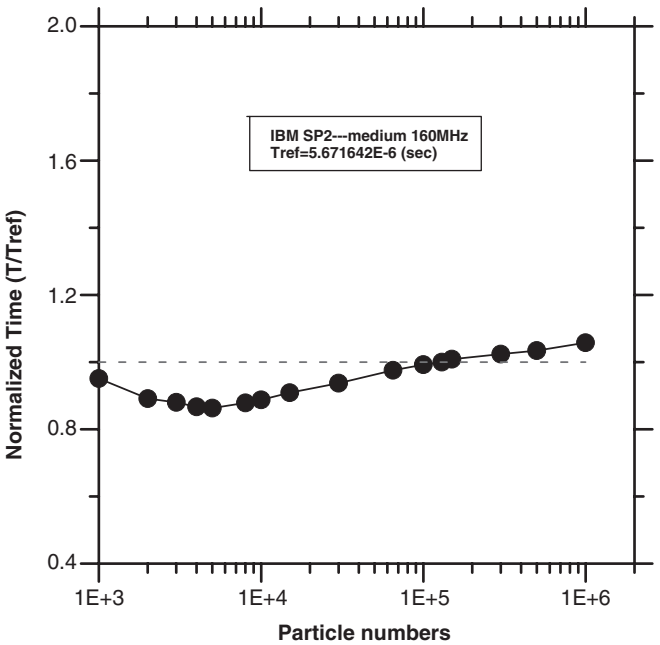


Figure 7. Normalized computational time per particle on a single IBM-SP2 processor.

(Figure 6(a) and (d)). This is mainly attributed to both the cache effects and better load balancing among processors. However, the efficiency decreases with increasing number of processors (up to 64) as expected, due to load unbalancing among processors, if static domain decomposition is applied. Figure 7 shows that the computational time per particle as a function of particle numbers on a single processor (IBM-SP2), in which the minimum computational time per particle occurs at approximately 4000 particles and increases with increasing particle numbers as particle numbers is greater than 4000. As the number of processors increases over 16, negative effects of load unbalancing and communication increase among processors begin to play a more important role than the positive effects of cache effects. Thus, the parallel efficiency decreases monotonously with increasing number of processors (up to 64) even if dynamic domain decomposition is used, as shown in Figure 6(a) and (d).

In addition, results show that parallel performance for applying SAR scheme less frequently is generally better than applying SAR scheme more frequently for number of processors less than 64. As the number of processors increases up to 64, all three strategies of applying SAR scheme result in roughly the same value of parallel efficiency, mainly due to the relative load unbalancing developed among processors levels the advantages gained by repartitioning the domain less frequently. Also, in the small problem size, it might be difficult for the repartitioning library to re-decompose the domain more exactly since too few particles stay within a processor if the number of processors is high, e.g. only approximately 3500 particles per processor with 64 processors. Nevertheless, for the small problem size the parallel efficiency using dynamic domain decomposition improves appreciably in the range of 30–50%, as compared with static domain decomposition.

3.3.2. Medium problem size. Similarly, super-linear speedup exists for the medium problem extending even up to 48 processors, if dynamic domain decomposition is activated (Figure 6(b) and 6(e)). This unusual extended super-linear speedup should be attributed to the relatively small cache size available on the super-scalar workstation, in which the array data can be accessed very fast. However, the super-linear speedup is not seen at all if the dynamic domain decomposition is deactivated, which nevertheless demonstrates the effectiveness of implementing dynamic domain decomposition. As the number of processors is over 48, this super-linear speedup disappears due to increasing communication among processors. For the medium problem size, parallel performance (Figure 6(b) and (e)) using dynamic domain decomposition is generally 50–100% higher than that using static domain decomposition as the number of processors is less than or equal to 64. Note that approximately 90% of parallel efficiency can be reached at processor numbers of 64 for the medium problem size. In addition, advantage of activating SAR scheme less frequently is diminishing for the medium problem size due to the increasing problem size, in which the repartitioning cost becomes comparatively less important than useful particle computation.

3.3.3. Large problem size. For the large problem size, super-linear speedup generally disappears even if the dynamic domain decomposition is activated (Figure 6(c) and 6(f)). The only exception is the case, which activates SAR scheme more frequently (at interval of $2\Delta t$), which balances the workload among processors more efficiently than the other two cases. Parallel efficiency of 107% can still be reached for number of processors of 64 when activating SAR scheme at interval of $2\Delta t$. Thus, the optimal frequency of activating SAR scheme generally increases with increasing problem size.

3.4. Dynamic domain decomposition

Typical evolution of dynamic domain decomposition using graph-partitioning technique is shown in Figure 8 for the large problem size using 64 processors, when activating SAR scheme at intervals of $2\Delta t$. Different colour in each sub-domain is used only for easy identification. It is clear that region covered by each sub-domain (processor) changes as the simulation proceeds due to repartitioning among processors when the initial size of each domain is approximately the same. There exists a smallest sub-domain exists in the right-hand lower corner of the cavity due to the presence of highest density in this region (Figure 8(b)–(c)). In addition, the size of the sub-domains above the moving plate is generally larger as compared with others due to the rarefied conditions caused by the fast moving plate (Figure 8(b)–(c)). It clearly demonstrates that the current implementation of dynamic domain decomposition is very effective in following the dynamics of the flow problem under study.

Figure 9 illustrates both the number of particles in each processor and the partition count as a function of the number of simulation time steps for the large problem size using 16 processors when activating SAR at intervals of $2\Delta t$. It only shows the time history of both quantities in the early stage of simulation for three processors for the clarity of presentation. In this figure, we are not trying to identify the evolution of particle numbers in any specific processor, although different lines represent different processors. Results show that number of particles in each processor approaches to the average number of particles per processor (225 000) right after the repartition, which shows the load balancing takes effect once the repartitioning functions. Note that we have preset the balance tolerance value to be 3% in PJOSTLE library [27], which

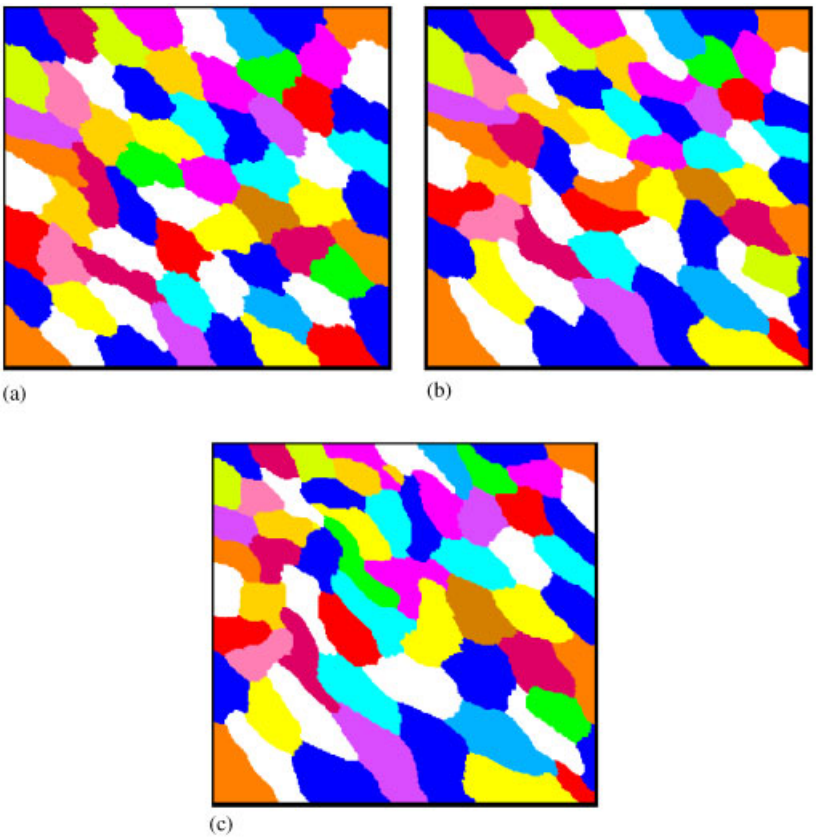


Figure 8. Evolution of domain decomposition for large problem size using 64 processors, when activating SAR scheme at intervals of $2\Delta t$, during the simulation for a bottom, lid-driven cavity flow. (a) initial; (b) intermediate; (c) final.

represents that load imbalance among processors less than this value shall not repartition the domain, even the SAR scheme decides to do so. In addition, the deviation of the number of particles in some processors from the average value deteriorates faster in the early stage right after the repartition, where the flow changes dramatically from initially uniform distribution, as shown in Figure 9. As the flow reaches steady state, the repartitioning is less frequent as expected, which can be seen clearly with smaller value of slope in Figure 10. Figure 10 shows the typical domain repartition counts as a function of simulation time steps for 64 processors. Generally, at first it increases more rapidly (larger value of slope) with simulation time during transient period and then increases less rapidly (smaller value of slope) with simulation time as flow approaches steady state ($\sim$ 10 000 steps in this case). Similar trends can be found for other flow conditions. Note that the repartition history for larger problem size after 20 000 steps is skipped due to the limitation of accessing time to the parallel machine.

Figure 11 shows the final normalized workload distribution (or number of particles per processor) with respect to processor one among the 64 processors for the three different

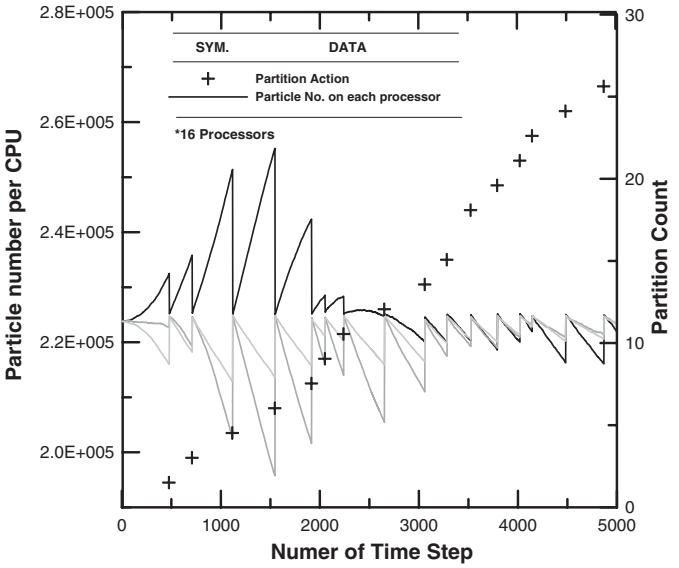


Figure 9. Number of particles in each processor and the number of repartitions as a function of the number of simulation time steps for the large problem size using 16 processors when activating SAR at intervals of $2\Delta t$.

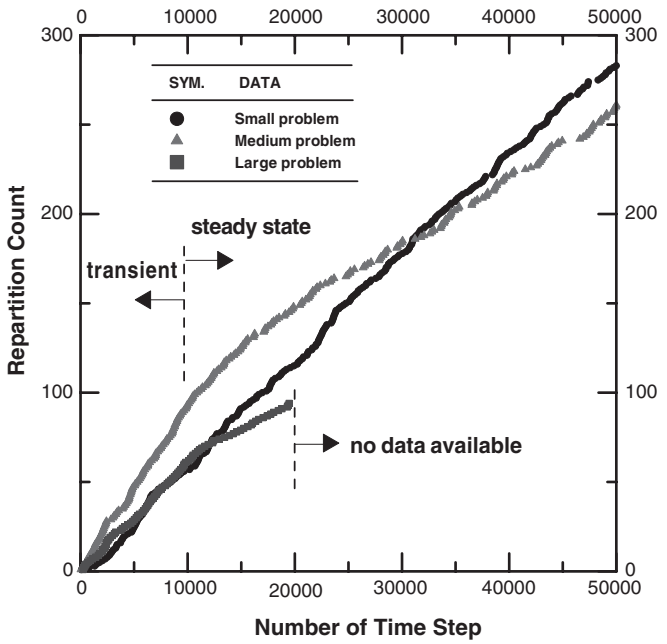


Figure 10. Number of repartitions as a function of simulation time steps at 64 processors.

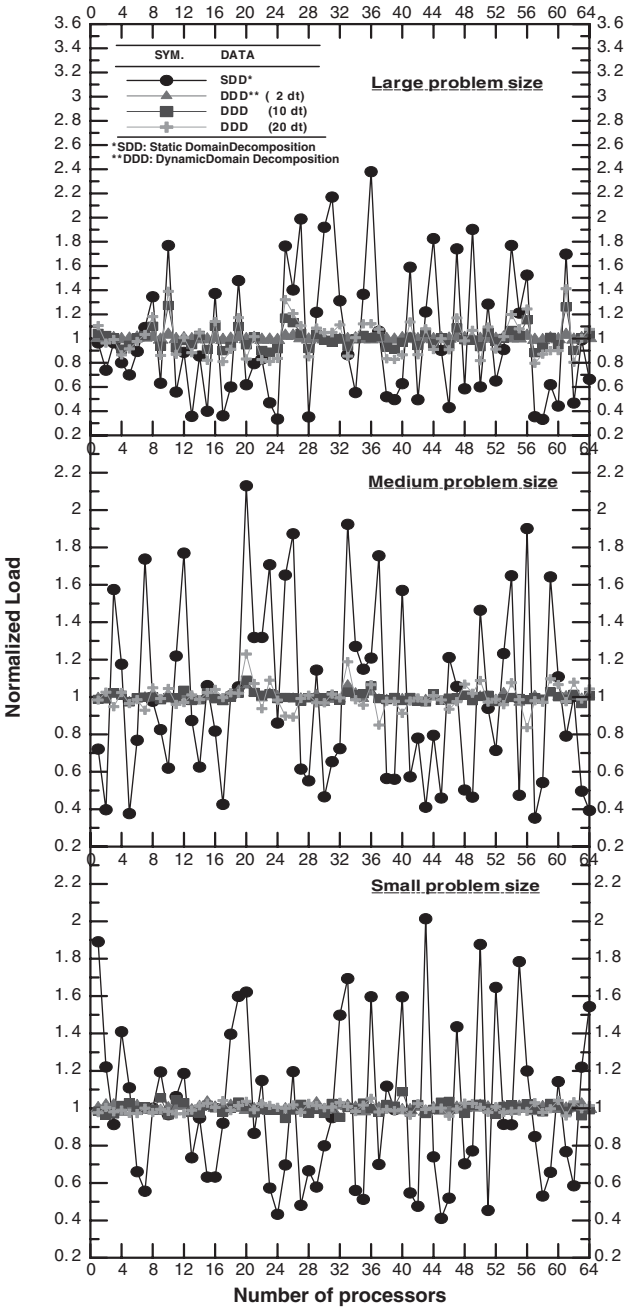


Figure 11. Final normalized particle numbers on each processor for three problem sizes at 64 processors.

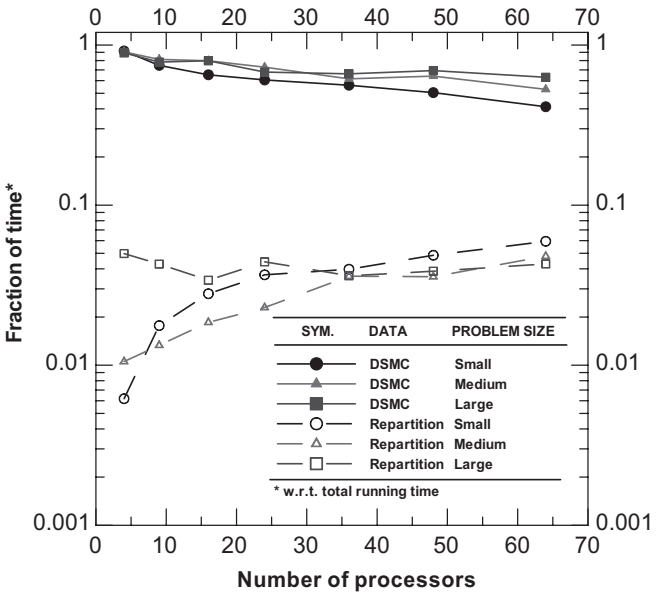


Figure 12. Fraction of time for the DSMC computation and repartition as a function of number of processors.

problem sizes using different strategies of implementing SAR ($2\Delta t$, $10\Delta t$ and $20\Delta t$). It is clear that the workload distribution is much more uniform with dynamic domain decomposition than that without dynamic domain decomposition. In addition, the workload distribution is found to be most uniform for SAR-$2\Delta t$ scheme since it monitors the load imbalance more often than others. This does not guarantee better parallel efficiency (referring to Figure 6(d) and (e)), since frequent repartition is expensive as compared with the normal DSMC computation.

3.5. Time breakdown of parallel implementation

Figure 12 illustrates the typical fraction of time spending in DSMC computation and dynamic domain decomposition per simulation time step as a function of the number of processors by employing SAR scheme at the interval of $2\Delta t$. Note that the DSMC computational time includes the 'useful' DSMC computational time, the idle time and the communicational time during particle movement between adjacent processors. It can be seen that, for the small problem, the average fraction of time spending in repartitioning the domain per time step increases dramatically with the number of processors, which explains the rapid decrease of parallel efficiency at this condition for employing SAR every $2\Delta t$ in Figure 6(d). More or less, similar trend is found for medium problem size. On the contrast, for the large problem, the fraction of time for repartitioning the domain remains approximately the same (~ 0.04) with increasing number of processors up to 64. Correspondingly, the fraction of time for the DSMC computation varies insignificantly as the number of processors is over 24. This shows the current parallel DSMC method may be highly scalable at least for the large problem size.

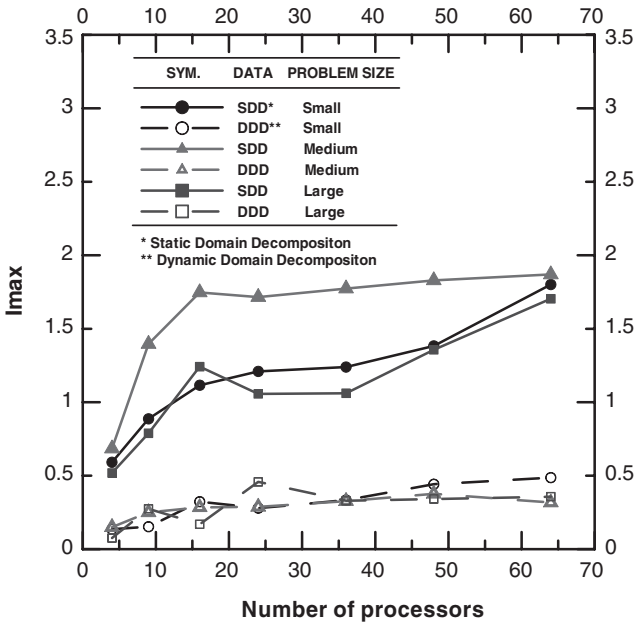


Figure 13. Degree of imbalance as a function of number of processors for static domain decomposition and dynamic domain decomposition.

3.6. Degree of imbalance

Degree of imbalance is a useful indicator for measuring the workload non-uniformity among processors, which is an important parameter for justifying dynamic domain decomposition in the current study. It is interesting to examine the maximal degree of imbalance in the system, $I_{\max}$, which is defined as

$$I_{\max} = \frac{1}{\overline{W}}(W_{\max} - W_{\min}) \tag{4}$$

where $W_{\max}$ and $W_{\min}$ are the maximum and minimum number of particles across the processor, respectively. $\overline{W}$ is the average number of particles in each sub-domain.

Figure 13 shows the variation of maximal imbalance with processor numbers for three different problem sizes. In general, the maximal imbalance developed by static domain decomposition (0.5–2) is much higher (2–6 times) than that by dynamic domain decomposition (0.1–0.5). In addition, workload imbalance developed very fast among processors as the number of processors increases, if dynamic domain decomposition is not used. Although the maximal degree of imbalance for the small problem size deteriorates with increasing number of processors, it is fairly constant within 0.4 for the medium and large problem sizes, which again demonstrates the effectiveness of the dynamic domain decomposition.

In brief summary, computational results show that the current parallel implementation using dynamic domain decomposition can generally increase the computational speed up to 30–100%

for the driven cavity problem, as compared with that using static domain decomposition for processor numbers up to 64.

4. APPLICATIONS

To demonstrate the excellent capability of the current 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, a two-dimensional hypersonic flow past a 15°-compression ramp, a three-dimensional hypersonic flow past sphere and a three-dimensional near-continuum flow of twin-jet interaction. 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.

4.1. Two-dimensional hypersonic flow past a cylinder

4.1.1. Flow and simulation conditions. Flow conditions are the same as those of Koura and Takahira [31] and represent the experimental conditions of Bütefsch [32]. For completeness, they are briefly described here as follows: VHS nitrogen gas, free-stream Mach number $M_\infty = 20$, free-stream number density $n_\infty = 5.1775\text{E}19$ particles/m^3, free-stream temperature $T_\infty = 20\,\text{K}$, fully thermal accommodated and diffusive cylinder wall with $T_\text{w}/T_0 = 0.18$, where $T_\text{w}(= 291.6\,\text{K})$ and $T_0(= 1620\,\text{K})$ 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 $Z_{\text{r}_\infty} = 21$, potential well-depth temperature $T^* = 79.8\,\text{K}$. 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 four levels of mesh refinement (Figure 14). Constant time-step method is used throughout the computational domain. 20 000 time steps are used to sample for obtaining averaged flow properties.

4.1.2. Domain decomposition. Figure 15 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 15(b) shows that the final decomposition, which adapts to the flow dynamics as simulation continues, is totally different from the initial decomposition.

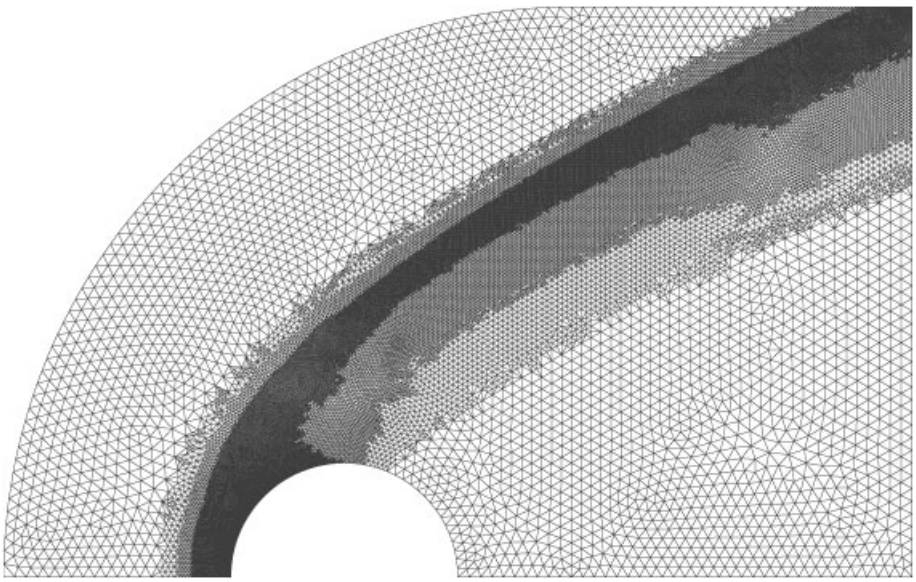


Figure 14. The mesh of four levels adaptation for a two-dimensional hypersonic cylinder flow (73 673).

4.1.3. Centerline properties distribution. Figures 16 and 17 illustrate the computed centreline 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 centreline in front of the cylinder and becomes relatively small in the wake region. Temperature also increases rapidly along the centreline 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.

4.2. Two-dimensional hypersonic flow past a compression ramp

4.2.1. Flow and simulation conditions. A hypersonic flow over a flat plate with a 15°-compression ramp is simulated using the flow conditions provided by the experimental study of Holden and Moselle [35]. The corresponding boundary conditions for simulation are depicted in Figure 18. The simulated flow conditions are brief described in the following: VHS nitrogen gas; free-stream Mach number $M_\infty = 14.36$; free-stream density $\rho_\infty = 5.221\text{E-}4\ \text{kg/m}^3$ and temperature $T_\infty = 84.83\ \text{K}$; the length of the cylinder, x_c, and ramp, x_r, are 43.891 and 36.86 cm, respectively; fully thermally accommodated and diffusive flat and ramp wall with $T_w = 294.4\ \text{K}$. Resulting Knudsen numbers and Reynolds numbers based on x_c are 0.0002 and 1.04E-5, respectively. Constant rotational energy exchange model is used with the rotational

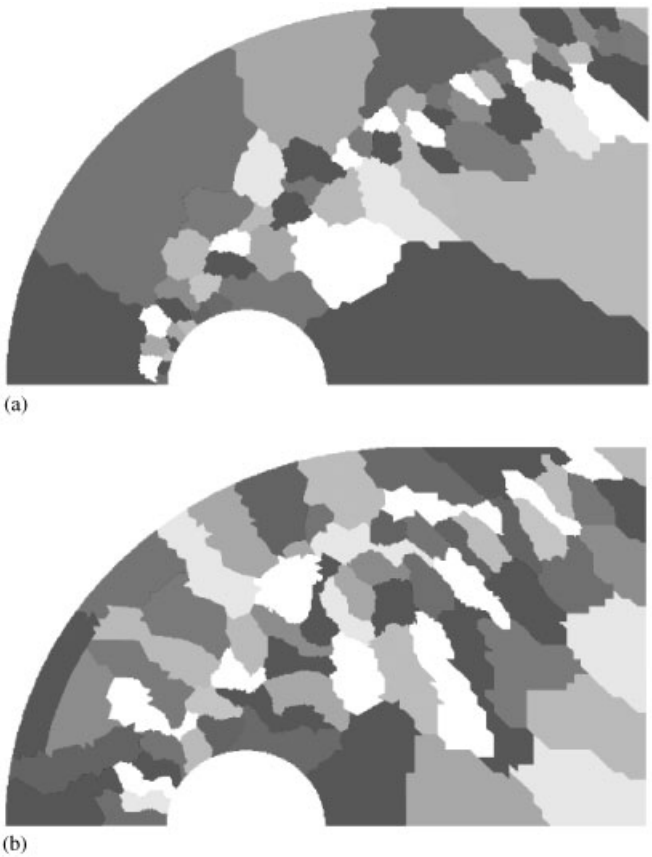


Figure 15. Initial and final domain decomposition for 64 processors for a two-dimensional hypersonic cylinder flow: (a) initial; and (b) final.

collision number $Z_r = 5$. Vibration energy transfer is neglected due to the low temperature involved. Similar to previous case, an *h-refined* mesh with mesh quality control [34] is used in this simulation (64 processors) to increase the accuracy of the solution. The simulation particles are about 2 million at steady state and the number of cells is approximately 84 000 after two levels of mesh refinement (Figure 19). Constant time-step method is used throughout the computational domain. 20 000 time steps are used to sample for obtaining averaged flow properties.

4.2.2. Domain decomposition. Figure 20 shows the initial and final domain decomposition for this simulation. Similar to previous case, the initial computational domain is partitioned by uniform weight. Thus, each subdomain has an approximately cell number. Some large domains above the flat plate are found due to the rarefied conditions in the region (Figure 20(b)). In addition, some small domains at the end of the ramp plate are found, where the density is very high due to the oblique shock originating approximately from the ramp corner.

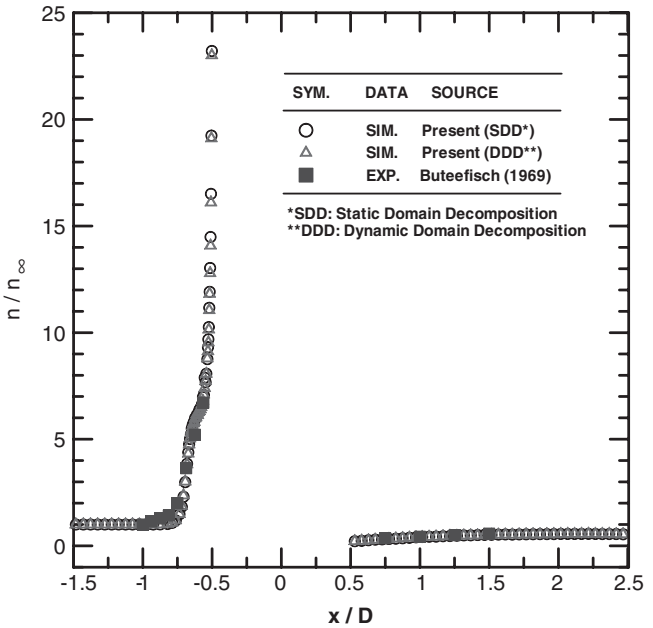


Figure 16. Normalized density of a two-dimensional hypersonic cylinder flow with/without dynamic domain decomposition.

4.2.3. Surface properties. Figure 21 presents the simulated pressure coefficients ($C_p = (p - p_\infty)/1/2\rho u_\infty^2$) along the solid wall using dynamic domain decomposition. Data include computational results without dynamic domain decomposition, previous DSMC results by Robinson [22] and experimental data by Holden [35]. Results show that pressure coefficient increases rapidly near the tip, declines slowly along the flat plate to a lowest value near the ramp corner and finally increases dramatically along the ramp wall. Results generally agree with previous simulation and experimental data, although the current simulated data seems to agree favourably with experimental data along the ramp wall, as compared with those of Robinson's [22]. Final rapid decline of our simulated data should be attributed to the vacuum condition we have imposed at the outflow boundary. In addition, the results of shear stress coefficient and heat transfer coefficient along the solid wall that are obtained with and without dynamic domain decomposition coincides excellently with each other, although they are not shown in this report. The simulation time required for dynamic domain decomposition is about 50% of that required for static domain decomposition, which again justifies the current parallel implementation.

4.3. Three-dimensional flow past a sphere

4.3.1. 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

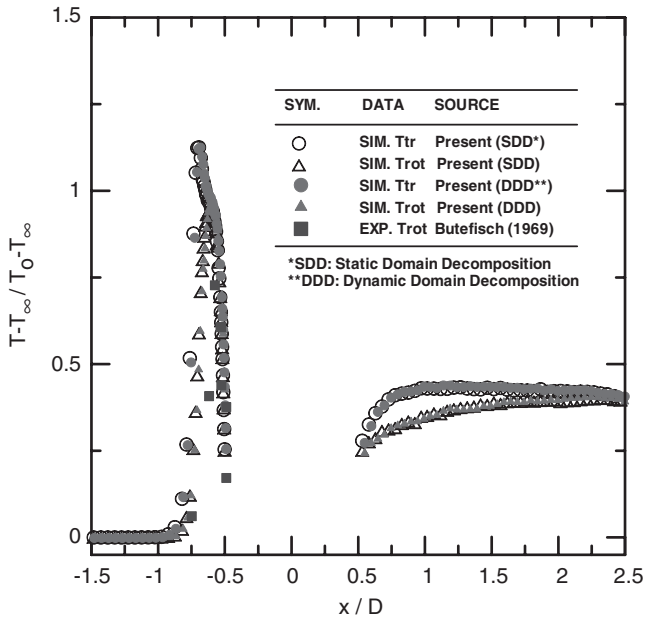


Figure 17. Temperatures of a two-dimensional hypersonic cylinder flow with/without dynamic domain decomposition.

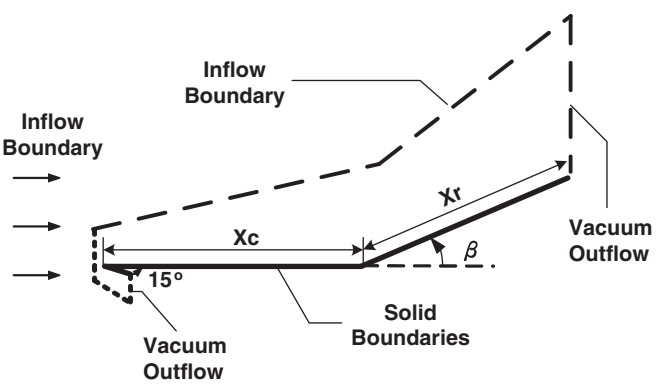


Figure 18. The corresponding boundary conditions for a two-dimensional hypersonic flow past a compression ramp.

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 [36], are listed as follows: VHS nitrogen gas; free-stream Mach number $M_\infty = 4.2$; free-stream number density $n_\infty = 9.77\text{E-}20$ particles/m^3; free-stream temperature $T_\infty = 66.25$ K; stagnation temperature $T_0 = 300$ K; fully thermal accommodated

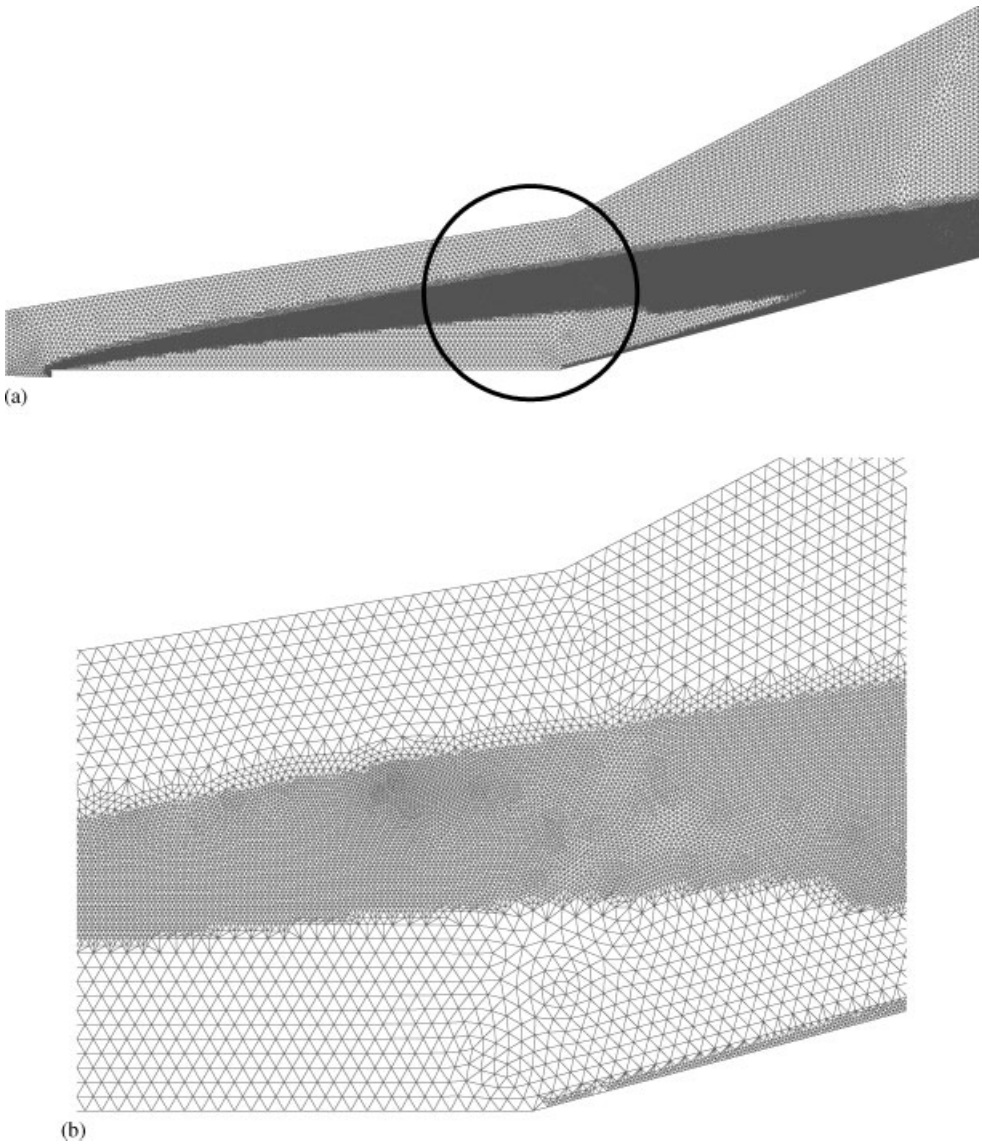


Figure 19. The mesh of two levels adaptation for a two-dimensional hypersonic flow past a compression ramp (83 085).

and diffusive sphere wall with the temperature T_w (equal to stagnation temperature T_0). 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 1.28 cm.

An *h-refined*, three-dimensional mesh with mesh quality control [37] is used in this simulation (8 IBM-SMP processors) to increase the accuracy of the solution. In addition, variable

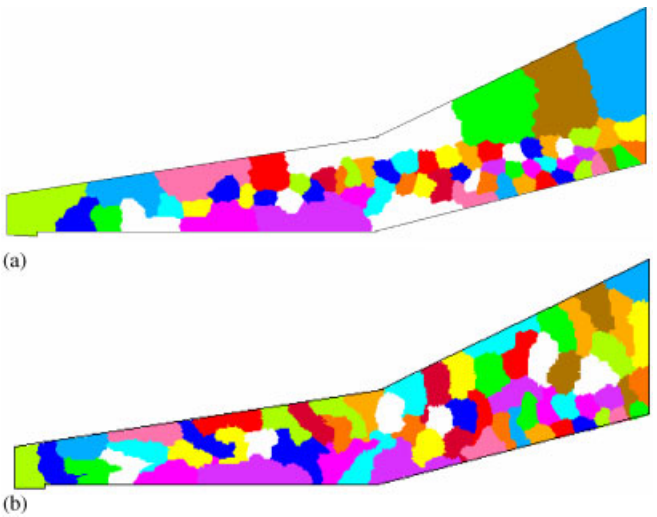


Figure 20. Initial and final domain decomposition for 64 processors for a two-dimensional hypersonic flow past a compression ramp: (a) initial; and (b) final.

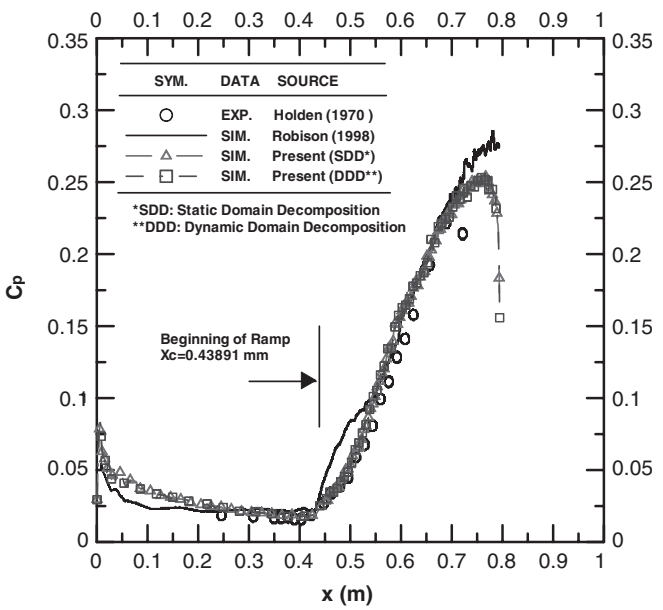


Figure 21. Pressure coefficient along the solid wall for a two-dimensional hypersonic flow past a compression ramp with/without dynamic domain decomposition.

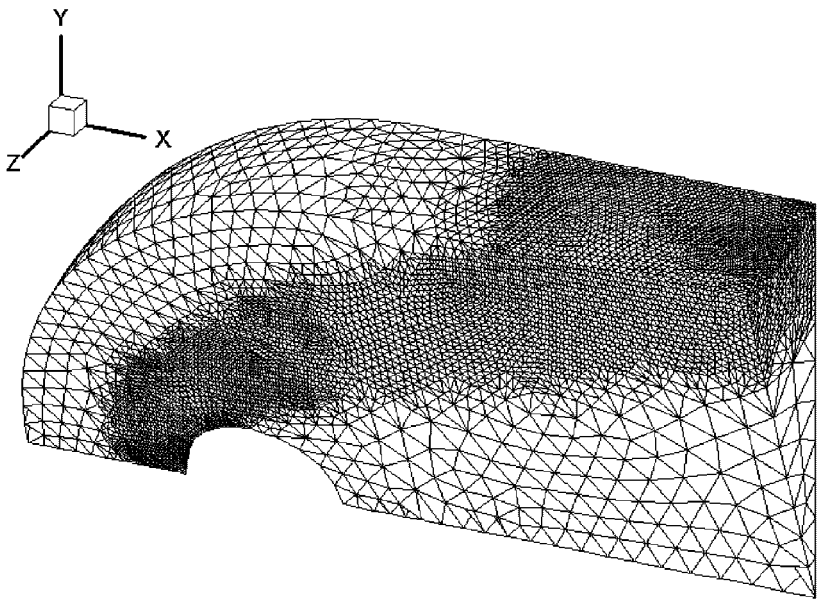


Figure 22. The mesh of 2 levels adaptation for a three-dimensional hypersonic sphere flow (164 276).

time-step method [37, 38] 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 two levels of mesh refinement (Figure 22). 20 000 time steps are used to sample for obtaining averaged flow properties.

4.3.2. Dynamic domain decomposition. Figure 23 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 (Figure 23(a)) is obtained assigning equal weight to each cell, which is different from previous two cases. At the final domain decomposition (Figure 23(b)), 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.

4.3.3. Centreline density distribution. Figure 24 presents the computed centreline density distribution using dynamic domain decomposition, along with that computed using static domain decomposition and experimental data of Russel [36]. 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.

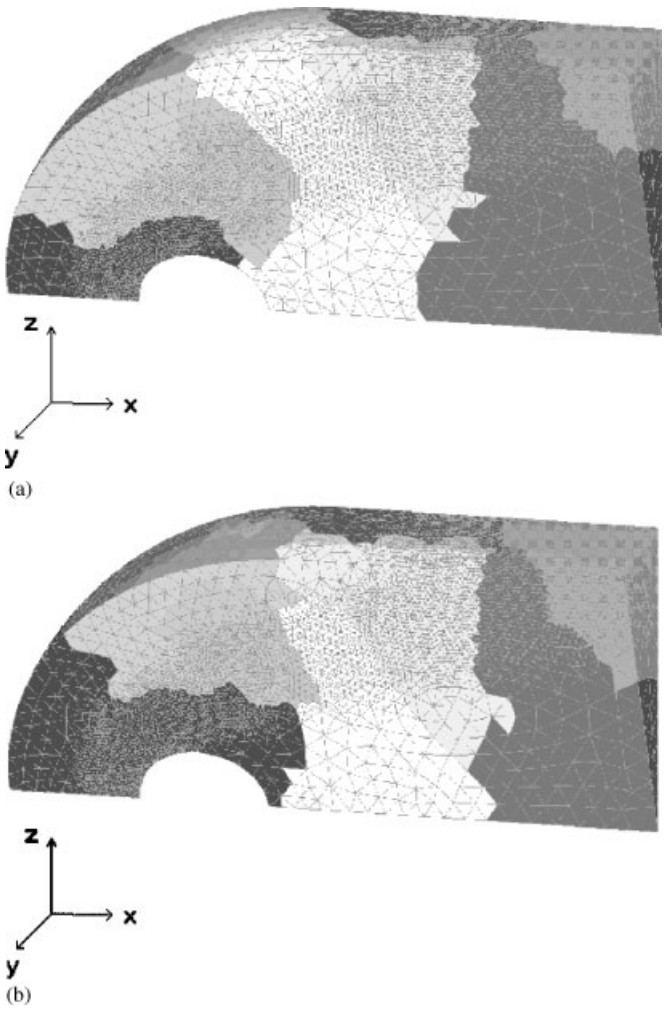


Figure 23. Initial and final domain decomposition for eight processors for a three-dimensional hypersonic sphere flow: (a) initial; and (b) final.

4.4. Near-continuum parallel twin-jet interaction

4.4.1. Flow and simulation conditions. For a truly three-dimensional flow, two parallel, near-continuum, under-expanded jets issuing from sonic orifices into near-vacuum environment is selected as the final test case since the experimental data is available [39]. Sketch of the twin-jet interaction is shown in Figure 25. It is expected that a secondary jet form between the two primary parallel jets under certain flow conditions. Corresponding flow conditions represent a challenging problem since it involves flow regimes from near-continuum at the inlet to near free-molecular flow at the outlet, where the DSMC method may be the only available tool for analysing this problem. Related flow conditions are listed as follows: the test gas is nitrogen gas; stagnation pressure $P_0 = 870\,\text{Pa}$; stagnation temperature $T_0 = 285\,\text{K}$; background

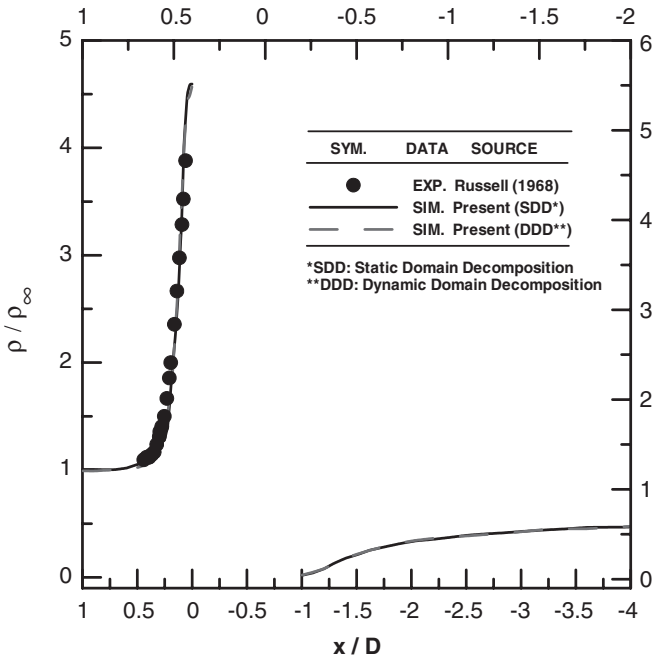


Figure 24. Normalized density distribution along the stagnation line for a three-dimensional hypersonic sphere flow with/without dynamic domain decomposition.

pressure $P_b = 3.7\,\text{Pa}$; resulting pressure ratio $P_0/P_b = 235$. Background pressure effect is either neglected or included in the simulations, in which previous simulations by Dagum and Zhu [40] have been ignored and vacuum boundary is used instead for simplicity. The corresponding Knudsen number $Kn_{th}(= \lambda_{throat}/D$; D is the throat diameter) is 0.00385.

Computational domain is reduced to $\frac{1}{4}$ of the original physical domain by taking advantage of the geometrical symmetry of the flow. Height (H), width (W) and length (L) of the simulation domain are taken long enough as $10D$, $10D$ and $20D$, respectively, as shown in Figure 25, where the positive z-direction is out of the paper. Not that the distance between the centres of two primary jet centreline is 3 times the diameter of the orifice. Internal energy is relaxed through the Borgnakke–Larsen model [41], using a fixed rotational collision number of 5. Variable time-step method [37, 38] and an *h-refined* mesh is used to reduce the computational time further and to increase the accuracy of the solution, respectively. An *h-refined* three-dimensional mesh with mesh quality control [37] is used in this simulation (8 AMD Athlon 1.0 GHz processors on PC-cluster system) to increase the accuracy of the solution. The resulting simulation particles are about 10 millions at steady state and the number of cells is approximately 0.66 million after 2 levels of mesh refinement (Figure 26). Figure 26(a) shows the surface mesh along x–y and y–z planes, while Figure 26(b) illustrates the exploded view of the surface mesh in the same planes near the sonic orifice, where the mesh is refined. 20 000 time steps are again used to sample the particles for obtaining macroscopic properties.

4.4.2. Dynamic domain decomposition. Figure 27 illustrates the initial and final domain decomposition on the surfaces of x–y, y–z and z–x planes for the parallel twin-jet interaction

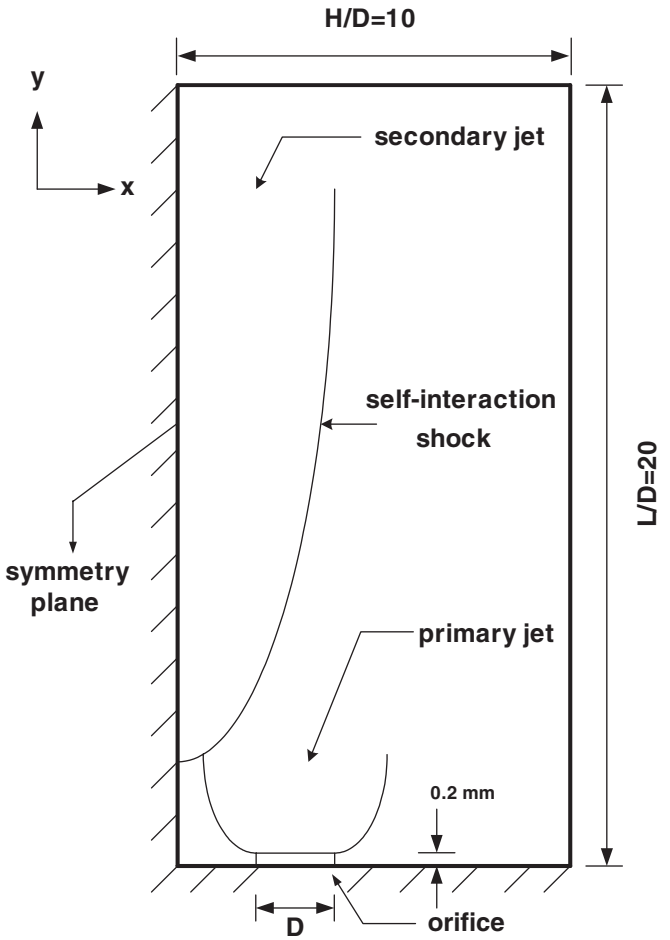
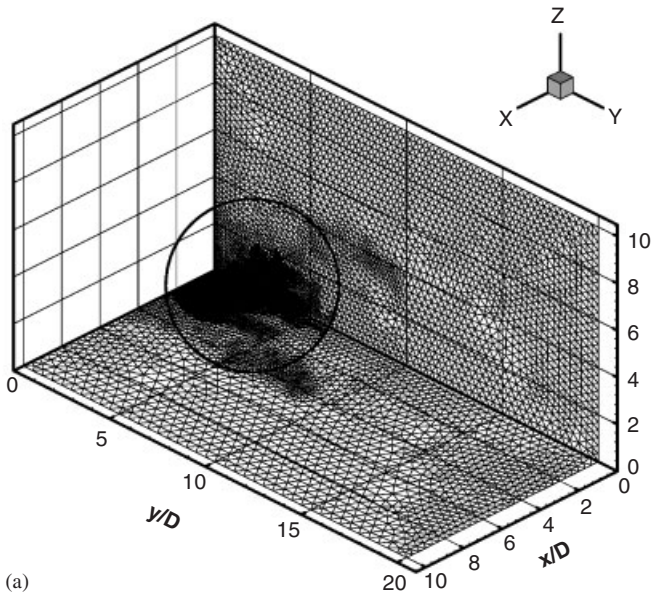


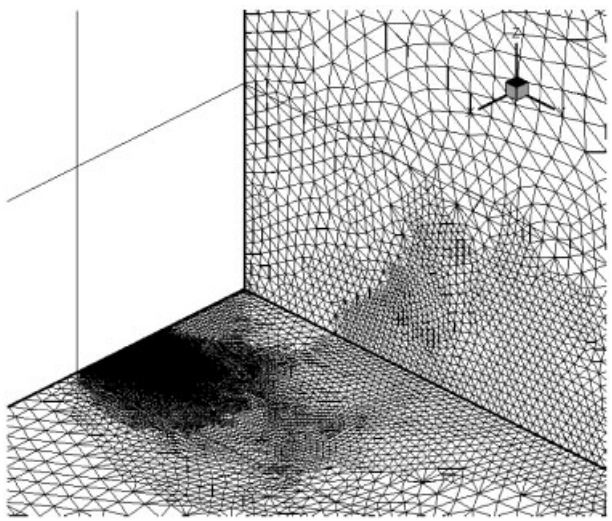
Figure 25. Sketch of the twin-jet interaction.

using vacuum outflow boundary. The initial domain decomposition (Figure 27(a)) is obtained by assigning equal weight to each cell, which is the same as the three-dimensional sphere flow. At the final domain decomposition changes dramatically as compared with the initial domain decomposition. In this case, the computational time is saved up to 100% using dynamic domain decomposition, as compared with that using static domain decomposition. Similar trend is found for the case using pressure outflow boundary. Even more time saving is expected where more processors are used.

4.4.3. Density contour. Figure 28 presents the normalized density contours (with respect to the density at the sonic orifice) on x–y and y–z planes using vacuum outflow boundary. Note that y–z plane is the symmetric plane between the two primary jets. It is clear that a secondary jet is formed and centred along the y-axis between the two primary jets. In addition, the density expends similar to a single jet on the side, where there is no jet interaction.



(a)



(b)

Figure 26. The mesh of 2 levels adaptation along x–y plane and y–z planes for the twin-jet interaction (657 624).

4.4.4. Centreline properties distributions. Figure 29 illustrates the density and rotational temperature distribution along the symmetric centreline (y-axis), respectively, using different outflow boundary conditions (vacuum and pressure). Experimental data measured using electron bean fluorescence by Soga [39] and the DSMC simulation data by Dagum and Zhu [40] using

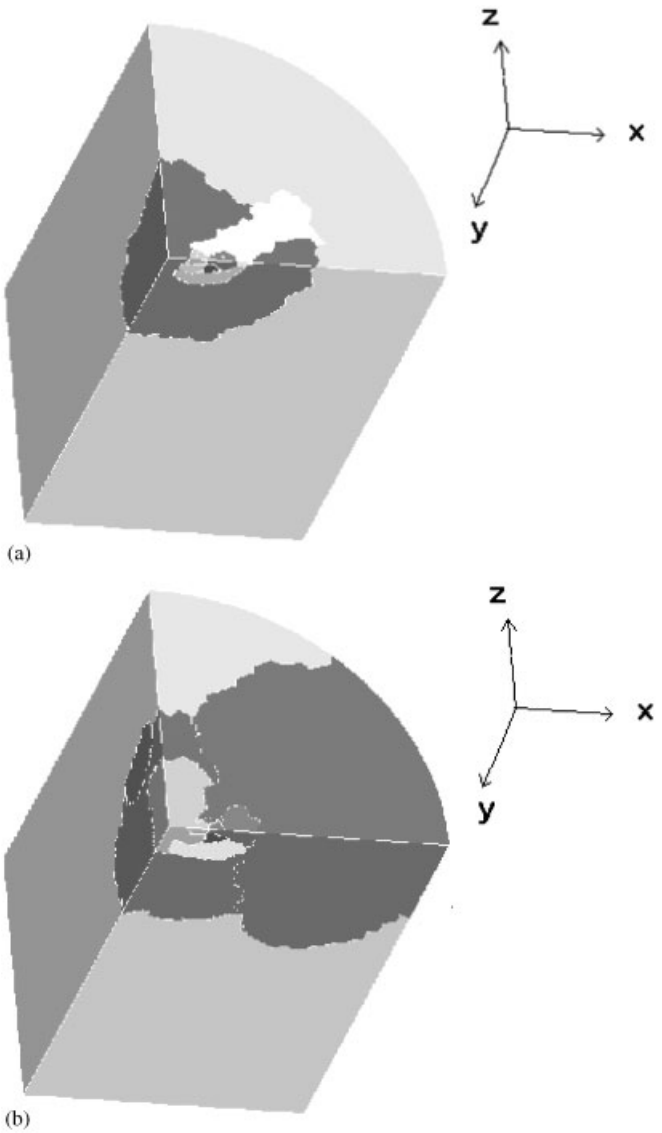


Figure 27. Initial and final domain decomposition for eight processors for twin-jet interaction: (a) initial; and (b) final.

vacuum outflow boundary condition are also included for comparison. It is clear that current simulation data using vacuum outflow boundary condition agree favorably with experimental data [39] and previous DSMC simulation data [40] using the same outflow boundary condition. It is, however, that the centreline density increases again for $y/d > 10$ when pressure outflow boundary condition is used. Similar trend is also found for rotational temperature distribution in Figure 29(b). Note that the background pressure in the vacuum chamber is maintained at 3.7Pa,

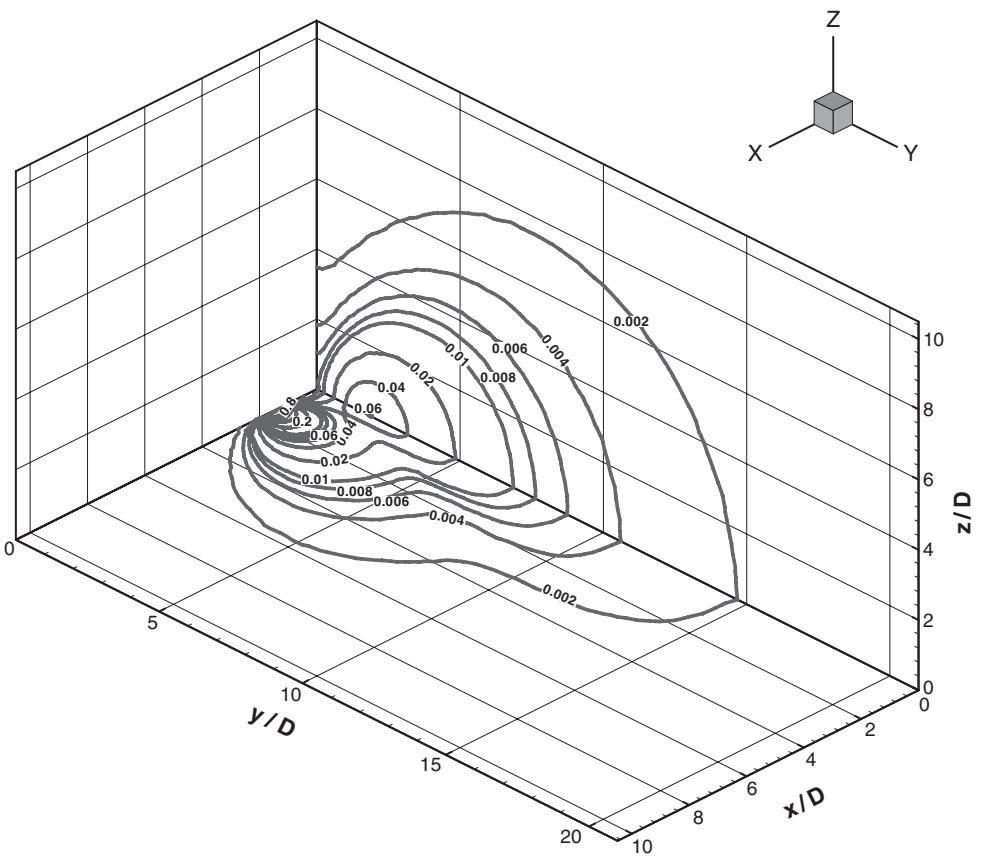


Figure 28. Normalized density contours (with respect to the density at the sonic orifice) on x–y and y–z planes using vacuum outflow boundary for twin-jet interaction.

where no measurement location is clearly specified in the paper [39]. The only possible way to fully duplicate the experimental conditions is to conduct chamber-scale simulation, which is not possible due to the unclear experimental conditions provided in the paper [39].

5. CONCLUSIONS

In the current study, a parallel DSMC method that dynamically re-decomposes the computational domain using graph-partitioning technique is presented. A two-dimensional, high-speed bottom, lid-driven cavity flow is used as the test case, considering three different problem sizes. Related parallel performance of the DSMC implementation on memory-distributed machine (IBM-SP2) is studied. Detailed analysis is performed, including evolution of domain decomposition, time breakdown of the parallel implementation and degree of imbalance among processors. Proposed method is then applied to compute several realistic cases, including two two-dimensional hypersonic flows, a three-dimensional hypersonic flow and a three-dimensional

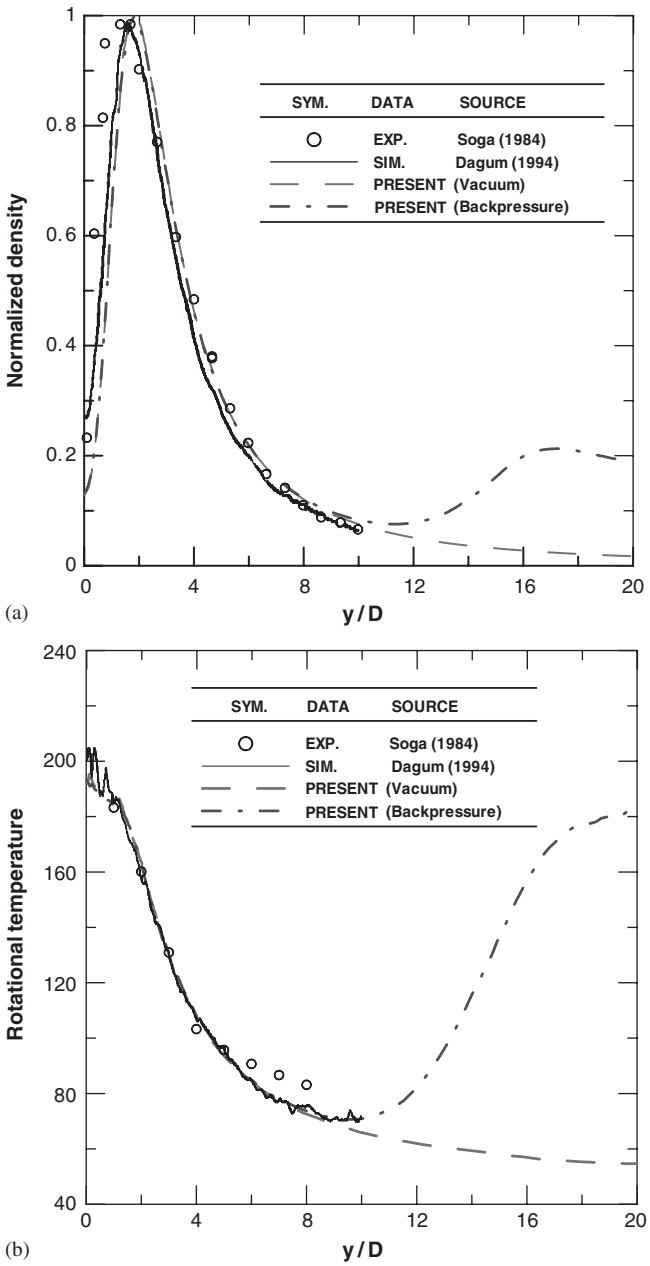


Figure 29. The density and rotational temperature distribution along the symmetric centreline (y-axis) using vacuum outflow boundary for twin-jet interaction.

near-continuum twin-jet interaction. Computed results are compared with experimental data and previous simulation data wherever available. In summary, the major findings of the current research are listed as follows:

1. Parallel DSMC using dynamic domain decomposition can generally save the computational time in the range of 30–100% for a high-speed driven cavity flow, as compared with that using static domain decomposition, for processor numbers up to 64 on IBM-SP2 (160 MHz) machine.
2. Proposed repartitioning scheme using SAR method in the parallel DSMC proves to be efficient in determining the repartition timing from the results of parallel performance. In general, the optimal frequency of activating SAR scheme increases with increasing problem size from the results of high-speed driven cavity flow.
3. Implementation of dynamic domain decomposition can in general lower the maximal degree of imbalance among processors by 2–6 times, resulting in appreciably improved parallel efficiency.
4. For processors less than or equal to 64, the fraction of time for repartitioning the domain is independent of the number of processors for the large problem size, which makes possible the parallel implementation scalable for higher number of processors. While it increases dramatically with increasing number of processors for the small problem size, which makes the parallel implementation non-scalable.
5. Applications, to compute several realistic flow problems and excellent agreement with experimental data, prove the accuracy and computational efficiency of the current proposed parallel DSMC method using dynamic domain decomposition.

From the results of the current study, we can conclude that the current parallel DSMC method using dynamic domain decomposition is generally superior to that using static domain decomposition. However, results using Cartesian structured mesh may have different conclusions as presented here. In addition, to take advantage of mesh refinement, integration of the serial *h-refined* meshing code [34, 37] into the parallel code is currently in progress and will be reported in the near future.

ACKNOWLEDGEMENTS

This investigation was supported by the National Science Council of Taiwan, Grant No. NSC92-2623-7-009-003 and the National Space Program Office, Grant No. NSC92-2623-7-009-003. The authors also would like to express their sincere thanks to the computing resources provided by the National Center for High-Performance Computing of Taiwan. Finally, a thanks to Dr Walshaw for providing the PJOSTLE running library throughout this work.

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. *Journal of Fluid Engineering* 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):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* 2003; **17**(5):405–422.
11. Furlani TR, Lordi JA. Implementation of the direct simulation Monte Carlo method for an exhaust plume flowfield in a parallel computing environment. *AIAA Paper 88-2736*, 1988.
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 three-dimensional flow over a finite flat plate. *AIAA/ASME Sixth Joint Thermophysics and Heat Transfer Conference*, Colorado Springs, CO, (*AIAA Paper 94-0219*) 1994.
14. Ota M, Taniguchi H, Aritomi M. Parallel processing for direct simulation Monte Carlo method. *JSME International Journal Series B* 1995; **61**(582):496–502.
15. Dietrich S, Boyd ID. Scalar and parallel optimized implementation of the direct simulation 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, U.S.A., 1998.
17. Ivanov M, Markelov G, Taylor S, Watts J. Parallel DSMC strategies for 3D computations. In *Proceedings of Parallel CFD'96*, Schiano P *et al* (eds). 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. *Computer 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. *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. *Ph.D. 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; **47**(2):102–108.
25. Karypis G, Kumar V. Metis: Unstructured graph partitioning and sparse matrix ordering. *Version 2.0 User Manual*. Minneapolis MN55455, Computer Science Department, University of Minnesota, U.S.A., 1995.
26. Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal* 1970; **49**:291–307.
27. Walshaw C. *Parallel Jostle Library Interface: Version 1.2.1*. School of Computation and Mathematical Sciences, University of Greenwich, London, SE10 9LS, U.K., 2000.
28. Altair Computing, Inc. *Hypermesh*, *Version 2.0*. Maplelawn, U.S.A., 2002.
29. Walshaw C. *The Jostle User Manual: Version 2.1*. School of Computation and Mathematical Sciences, University of Greenwich, London, SE10 9LS, U.K., 1999.
30. Nicol DM, Saltz JH. Dynamic remapping of parallel computations with varying resource demands. *IEEE Transactions on Computers* 1988; **39**:1073–1087.
31. Bütefsch K. Investigation of hypersonic non-equilibrium rarefied gas flow around a circular cylinder by the electron beam technique. *Rarefied Gas Dynamics II*. Academic Press: New York, 1969; 1739–1748.

32. Koura K, Takahira M. Monte Carlo simulation of hypersonic rarefied nitrogen flow around a circular cylinder. *Proceedings of the 20th International Symposium on Rarefied Gas Dynamics*, 1996; 1236–1242.
33. Parker 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 unstructured adaptive mesh and its application. *International Journal for Numerical Methods in Fluids* 2002; **38**(4):351–375.
35. Holden MS, Moselle JR. Theoretical and experimental studies of the shock wave-boundary layer interaction on compression surfaces in hypersonic flows. *Technical Report 70-0002*, ARL, 1970.
36. Russell DA. Density disturbance ahead of a sphere in rarefied supersonic flow. *Physics of Fluids* 1968; **11**(8):1679–1685.
37. Wu JS, Tseng KC, Wu FY. Parallel three-dimensional DSMC method using mesh refinement and variable time-step scheme. *Computer Physics Communications* 2004; **162**(3):166–187.
38. Markelov GN, Ivanov MS. Kinetic analysis of hypersonic laminar separated flows for hollow cylinder flare configurations, *AIAA paper 2000–2223*, 2000.
39. Soga T, Takanishi M, Yasuhara M. Experimental study of interaction of underexpanded free jets. *Proceedings of 14th International Symposium on Rarefied Gas Dynamics*, 1984; 485–493.
40. Dagum L, Zhu SK. Direct simulation Monte Carlo simulation of the interaction between rarefied free jets. *Journal of Spacecraft and Rockets* 1994; **31**(6):960–964.
41. Borgnakke C, Larsen PS. Statistical collision model for Monte Carlo simulation of polyatomic gas mixture. *Journal of Computational Physics* 1975; **18**:405–420.

