Concurrent DSMC Method Using Dynamic Domain Decomposition

J.-S. Wu, and K.-C. Tseng

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

Abstract. In the current study, a parallel two-dimensional direct simulation Monte Carlo method is reported, 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. Standard Message Passage Interface (MPI) is used to communicate data between processors. In addition, different strategies applying the Stop at Rise (SAR) [7] scheme is utilized to determine when to adapt the workload distribution among processors. Corresponding analysis of parallel performance is reported using the results of a high-speed driven cavity flow on IBM-SP2 parallel machines (memory-distributed, CPU 160 MHz, RAM 256 MB each) up to 64 processors. Small, medium and large problems, based on the number of particles and cells, are simulated. Results, applying SAR scheme every two time steps, show that parallel efficiency is 57%, 90% and 107% for small, medium and large problems, respectively, at 64 processors. In general, benefits of applying SAR scheme at larger periods decrease gradually with increasing problem size. Detailed time analysis shows that degree of imbalance levels off very rapidly at a relatively low value (30%–40%) with increasing number of processors applying dynamic load balancing, while it, at a value of 5–6 times larger, increases with increasing number of processors without dynamic load balancing. At the end, the completed code is applied to compute a near-continuum gas flow to demonstrate its superior computational capability.

INTRODUCTION

For near-continuum gas flows, computing requirements can render a meaningful direct simulation Monte Carlo (DSMC) [1] computation 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 coupling between particles is only made through collision in the cells. As compared with other N-S equation based numerical schemes, the DSMC method possesses much higher parallelism. Thus, the parallel DSMC method represents an opportunity to simulate flows at higher densities with an acceptable runtime [2].

In the past, several studies on parallel implementation of DSMC have been published [3] using static domain decomposition and structured mesh. Message passing was used to transfer molecules 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.

Recently, Boyd's group [4] designed the parallel DSMC software named MONACO. In this code, unstructured grids were used to take the advantage of flexibility of handling complex object geometry. In addition, a new data structured was proposed to meet the specific requirement related to workstation hardware while maintaining high efficiency on supercomputers. Timing results show the performance improvement on workstations and the necessity of load balancing for achieving high performance on parallel computers. Coordinate partitioning technique was used for domain decomposition. The authors also suggested that only decomposition along streamlines should be chosen to keep communication at a minimum. However, it may not always be possible for realistic flow problems.

Ivanov's group [5] 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; these are distributed and transferred between the processors. The algorithm is scalable and requires only local knowledge of the load distribution in a system. In addition, the direction and the amount of work are determined by the concept of heat diffusion process [6].
Around the same period of time, dynamic load balancing technique, using Stop At Rise (SAR) [7], which compares the cost of re-mapping with the cost of not re-mapping, based on a degradation function, was used in conjunction with the parallel implementation of DSMC method using a runtime library, CHAOS, for data communication and data structure manipulations on a structured mesh [2]. 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 [8] reported that parallel efficiency up to 90% is achieved for 128 processors for the flow over a sphere, although it is not clear how they implemented the dynamic load balancing. The 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. Studies about DSMC using both unstructured mesh and dynamic domain decomposition were relatively few in the past, although using unstructured mesh exhibits higher flexibility in handling objects with complicated geometry and boundary conditions. Robinson [9] 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 due to dynamic domain decomposition was reported as compared with that due to static domain decomposition. However, Robinson [9] 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. Degree of imbalance increases with increasing number of processors even applying the dynamic domain decomposition, which explains the not-too-impressive scalability of ADDER. Thus, the utilization of a more efficient repartitioning runtime library is essential to improve the performance of a parallel DSMC method.

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 having unitary weight. Considering DSMC, a truly dynamic load balancing technique is required because the load (approximately proportional to the particle numbers) in each sub-domain changes frequently, especially during the transient period. Domain decomposition in DSMC may become very efficient by taking the advantage of success in graph partitioning. For example, the multi-level scheme, JOSTLE [10] employs 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 [11]. Thus far, there has been no report that matured graph-partitioning tool has been applied in the 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. In the current study, we will use JOSTLE to dynamically decompose the computational domain for the parallel DSMC simulation.

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

1. To complete a two-dimensional parallel DSMC code 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 driven cavity flow for different problem sizes and study the issues related to parallel performance using different repartitioning strategies.
3. To apply the parallel DSMC implementation to compute a realistic, near-continuum hypersonic flow past a cylinder, and compare with previous experimental and DSMC data.

The paper begins with descriptions of the parallel DSMC method and the strategies of repartitioning the domain. Results of parallel performance including speedup (efficiency) and detailed load distribution are then considered using a high-speed cavity flow, and finally applying to a near-continuum hypersonic flow over a cylinder, in turn.

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 are modeled through uncoupling of the motion of particles and collisions between them. Mass, momentum and energy transport 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 particle data 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 lie along the streamlines of the flow field as mentioned [4] previously; however, it is nearly impossible to achieve this partition for most practical flows. 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 initial 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.

In this implementation, an unstructured mesh is first constructed and then a preprocessor (or "converter") is used to convert the fully unstructured mesh data into the *globally sequential but locally unstructured* mesh data [12] for each processor in conformation with the partitioning information from graph partitioner (JOSTLE) [11]. 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. After reading the mesh data on a master processor (cpu 0), the mesh data are then distributed to all other processors according to the designated domain decomposition. All the particles in each processor then start to move as in sequential DSMC algorithm. After all particles on each processor 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) [7] in the current study, which will be described shortly. If it does, then the program begins to re-decompose the computational domain, using the multilevel graph-partitioning technique, in 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 current study and the parallel DSMC method using static domain decomposition lies in the insertion of domain repartitioning and cell renumbering in the procedures.

**Repartitioning Method**

In the current study, we have used the parallel runtime library, PJOSTLE [11], as the repartitioning tool in our parallel DSMC code. JOSTLE, a serial version of graph partitioner, 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, a parallel version of graph partitioner, 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 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 version, 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 processor, unpacked, and the pointer based data structure recreated there. This provides an extremely fast solution to the problem of dynamically load-balancing unstructured mesh [11].

**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, the partitioning runtime library, **PIOSTLE** [11], described in the above is used to repartition the mesh based on some kind of decision policy. It has been shown that, for some problems using DSMC, remapping the domain at fixed intervals leads to poor parallel performance [2]. Therefore, it is desirable to determine a monitoring policy to decide when to repartition. Therefore, in the current study, a decision policy, Stop At Rise (SAR), proposed by Nicol and Saltz [7], is employed to determine when to repartition the domain. SAR, a “greedy” repartitioning policy, which 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. This degradation function represents the average idle time for each processor including the cost of repartition. In general, it tends to decrease with the number of simulation time steps. Repartitioning is not performed until the first local minimum of the degradation function is detected. This decision policy for repartitioning the domain is inherently advantageous in which 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 from previous experience [2,7,9].

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

**RESULTS AND DISCUSSIONS**

**High-speed Driven Cavity Flow**

*Test Flow Conditions*

A square high-speed driven cavity flow with a bottom plate moving to the right ($V=8*C_{mp}$) is considered as the test problem, which has been used by Robinson [9]. Related flow conditions include argon gas, 300K of wall temperature and diffusive wall. Knudsen number is 0.04, based on the width of the cavity and the mean free path of the wall temperature. 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. Different problem sizes, including small, medium and large problem size, are considered for simulation (Table 1), 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 constant for all three problems for the purpose of comparison. Nearly uniform triangular mesh is used throughout the study. Simulations are run for 50,000 time steps with time step about one-half of the mean collision time step. DSMC code is implemented on IBM-SP2 machines with the number of processors in the range of 1–64. Note that the maximum number of processors is 64 only due to the limitation of available computing resources, not the computational method itself.

| Table 1. 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 |
Dynamic Domain Decomposition

Different strategies of activating SAR at intervals of $2\Delta t$, $10\Delta t$ and $20\Delta t$ are implemented in the code to see its effects due to problem size. Some of the effects will be discussed later. Typical evolution of domain decomposition using graph partitioning technique is shown in Fig. 1 for the large problem size using 64 processors at intervals of $2\Delta t$. It is clear that regime of each domain changes as simulation processes due to dynamic load balancing among processors, where the initial size of each domain is approximately the same. It is well known the right-hand lower corner of the cavity exhibit highest-density region and the region above the moving plate shows most rarefied region. It can be seen clearly that a smallest sub-domain exists in the right-hand lower corner of the cavity, while the size of the sub-domains above the moving plate is generally larger as compared with others. This clearly demonstrates that the current scheme of dynamic domain decomposition is effective in following the dynamics of the simulation of the current problem.

**FIGURE 1.** Evolution of domain decomposition at 64 processors for high-speed driven cavity flow

Figure 2 illustrates both the number of particles and the number of repartitions in each processor as function of the number of simulation time steps for the large problem for 16 processors (only for clarity) with activating SAR at intervals of $2\Delta t$. It only shows the time history of both quantities in the early stage of simulation in each processor for the clarity of presentation. In this figure, we are not trying to point out the trend of any specific processor, although different lines represent different processors. Results show that number of particles in each processor approaches the average number of particles (~ 225,000) right after the repartition, which shows the load balancing takes effect once the dynamic domain decomposition activates. Note that we have preset the balance tolerance value to be 3% in PJOSTLE [11], which represents that load imbalance among processors less than this value will not repartition the domain. In addition, the deviation of the number of particles in each processor from the average value deteriorates faster in the early stage where the flow changes dramatically from initially uniform distribution, as shown in Fig. 2.

**FIGURE 2.** Evolution of particle numbers in each processor and partition count as a function of number of time step for the large problem size at 16 processors.
Parallel Performance

Typical results for parallel speedup are presented in Figures 3a-3c, for three problem sizes, respectively, at different strategies of employing SAR at every $2\Delta t$, $10\Delta t$ and $20\Delta t$. Parallel efficiency for 64 processors is 57%, 90% and 107%, respectively, for the small-, medium- and large-size problem applying SAR at $2\Delta t$. Results generally show that parallel efficiency of the current parallel DSMC method with dynamic domain decomposition is 30–60% better than that with static domain decomposition for the number of processors larger than 16. Super-linear speedup occurs for the number of processors up to 48 and 64, respectively, for the medium- and large-size problems. For the small-size problem, parallel speedup applying SAR at $10\Delta t$ and $20\Delta t$ outperforms that at $2\Delta t$ (but not much) due to less time spent in checking if repartitioning is required. On the contrary, for the medium- and large-size problem, parallel speedup applying SAR at $2\Delta t$ becomes comparable with and superior to, respectively, that at $10\Delta t$ and $20\Delta t$. Thus, from the practical point of views, parallel speedup employing SAR at every $2\Delta t$ outperforms other for the current parallel implementation.

**FIGURE 3.** Parallel speedup as a function of the number of processor for three different problem sizes at various SAR strategies (a) small problem (b) medium problem (c) large problem.

Figure 4 illustrates the variation of the degree of imbalance with the problem size applying SAR at every $2\Delta t$. Note that the degree of imbalance is defined as the ratio of difference between maximal and minimal weights to the average weight across the processor array. Figure 4 shows that the degree of imbalance generally increases with increasing number of processors for both static and dynamic domain decompositions. However, the degree of imbalance with dynamic load balancing seems to saturate with increasing number of processors (up to 64) in the range 30%–40% regardless of the problem size. In addition, as one might expect, the degree of imbalance for static domain decomposition is about 5–6 times higher as compared with that for dynamic domain decomposition. Although we do not test the current parallel implement for processors more than 64 due to the limited computing resources available, this saturation more or less explains the superior parallel performance up to 64 processors.

**FIGURE 4.** Degree of imbalance as a function of the number of processors for three different problem sizes employing SAR at every $2\Delta t$. 
Applications of The Parallel DSMC Method

A Hypersonic Flow Over A Cylinder

The proposed parallel DSMC implementation has been verified by the cavity flow stated previously with 64 processors or fewer. We have concluded that the parallel DSMC method with dynamic domain decomposition performs much better than the one with static domain decomposition. Thus, to demonstrate the powerful capability of the current parallel implementation, we have applied it to compute a realistic, near-continuum hypersonic nitrogen flow over a cylinder. Flow conditions are the same as those of Koura and Takahira [14] and represent the experimental conditions of Bütefsch [13]. Resulting Knudsen number based on free-stream conditions and diameter of the cylinder is 0.025. Total 73,673 adaptive triangular cells [15] and about 1.3 million particles are used for the simulation. The initial and final domain decomposition using 64 processors is shown in Fig. 5. It clearly illustrates that the relatively smaller sub-domains near the stagnation region and larger sub-domains in the wake region in the final domain decomposition.

![Initial and final distribution of domain decompositions for a hypersonic flow past a cylinder at 64 processors.](image)

FIGURE 5. Initial and final distribution of domain decompositions for a hypersonic flow past a cylinder at 64 processors.

Results of normalized number density \( \frac{n}{n_\infty} \) along the stagnation line are presented in Fig. 6. Note that the subscript “\( \infty \)” represents free-stream properties, respectively. Previous experimental data of Bütefsch [13] and the DSMC data of Koura and Takahira [14] are also included in these figures. It is clear that the present simulated data are in good agreement with those of Koura and Takahira [14] in the wake region. Other flow properties show good agreement with previous studies as well, which are not shown in the current paper.

![Normalized number density along the stagnation line for a hypersonic flow past a cylinder.](image)

FIGURE 6. Normalized number density along the stagnation line for a hypersonic flow past a cylinder.
CONCLUSIONS

In the current study, a parallel two-dimensional DSMC method using unstructured mesh with dynamic domain decomposition has been proposed. Tests on a high-speed cavity flow show that the parallel speedup using dynamic domain decomposition generally outperforms that using static domain decomposition. The completed code is then applied to compute a near-continuum hypersonic nitrogen flow over a cylinder to demonstrate its superior computational capability.

ACKNOWLEDGEMENTS

This investigation was supported by the National Science of Taiwan, Grant No. NSC 90-2212-009-046. The author would like to thank the National Center for High-performance Computing in Taiwan for providing computing hardware.

REFERENCES